

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/256091009>

# Uma Abordagem Probabilística para Análise Causal de Defeitos de Software

Thesis · March 2011

---

CITATIONS

2

---

READS

248

1 author:



Marcos Kalinowski

Pontifícia Universidade Católica do Rio de Janeiro

93 PUBLICATIONS 435 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



HELENA SURVEY - Hybrid dEveLopmENt Approaches in software systems development [View project](#)



MPS.BR Program, coordinated by SOFTEX (from 2004-2014 I was the executive coordinator of this program) [View project](#)

## UMA ABORDAGEM PROBABILÍSTICA PARA ANÁLISE CAUSAL DE DEFEITOS DE SOFTWARE

Marcos Kalinowski

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Guilherme Horta Travassos

Rio de Janeiro  
Março de 2011

UMA ABORDAGEM PROBABILÍSTICA PARA ANÁLISE CAUSAL DE DEFEITOS DE  
SOFTWARE

Marcos Kalinowski

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ  
COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA  
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS  
NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM  
ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

---

Prof. Guilherme Horta Travassos, D.Sc.

---

Profa. Ana Regina Cavalcanti da Rocha, D.Sc.

---

Profa. Emilia Maria Xavier Mendes, PhD.

---

Prof. Gerson Zaverucha, PhD.

---

Profa. Renata Mendes de Araújo, D.Sc.

RIO DE JANEIRO, RJ – BRASIL  
MARÇO DE 2011

Kalinowski, Marcos

Uma Abordagem Probabilística para Análise Causal de Defeitos de Software / Marcos Kalinowski – Rio de Janeiro: UFRJ/COPPE, 2011.

XVI, 357 p.: il.; 29,7 cm.

Orientador: Guilherme Horta Travassos.

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2011.

Referências Bibliográficas: p. 122-134.

1. Engenharia de Software. 2. Análise Causal de Defeitos de Software. 3. Melhoria de Processos. 4. Engenharia de Software Experimental. I. Travassos, Guilherme Horta II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

À minha família.  
Aos meus amigos.

## Agradecimentos

Ao meu Orientador, Guilherme Travassos, pela dedicação, conselhos e motivação durante este período, o que contribuiu não apenas para minha formação acadêmica, mas também minha formação como pessoa. Agradeço por sua confiança em relação à minha capacidade de realizar este trabalho e por ter me ajudado a desenvolver uma visão mais crítica e profunda a respeito de pesquisa em engenharia de software. Agradeço ainda pela oportunidade de participar de diversos projetos de pesquisa e desenvolvimento, que certamente contribuíram para complementar a minha formação. O caminho que segui sob sua orientação me ajudou a crescer como pesquisador e engenheiro de software, solidificando a base para enfrentar novos desafios.

À professora Ana Regina da Rocha, por participar de minha banca de defesa de doutorado. Pelo aprendizado obtido nas disciplinas cursadas com ela durante o doutorado. Por ter estado sempre disponível para discutir idéias relacionadas à minha pesquisa. Por participar da banca de meu exame de qualificação, tendo fornecido contribuições fundamentais para os ajustes realizados no trabalho desde então. Pela oportunidade de participar de iniciativas da COPPE/UFRJ junto ao programa MPS.BR, me permitindo uma aproximação com a indústria, de extrema importância para meu crescimento profissional e para alinhar a minha pesquisa à realidade das empresas desenvolvedoras de software.

À professora Emilia Mendes, por participar de minha banca de defesa de doutorado e pelos conselhos durante o período entre a minha qualificação e a conclusão deste trabalho, sempre me fornecendo motivação e depositando confiança na minha capacidade de concluir este trabalho. Sua contribuição na escrita de artigos, no uso de redes bayesianas e na aplicação da engenharia de software experimental foi fundamental.

Ao professor Gerson Zaverucha, por participar de minha banca de tese de doutorado, pelo aprendizado obtido nas disciplinas cursadas com ele na COPPE/UFRJ e por ter estado sempre disponível para discutir idéias relacionadas à minha pesquisa.

À professora Renata Mendes de Araújo, por participar de minha banca de defesa de doutorado. Por participar da banca de meu exame de qualificação, tendo fornecido valiosas contribuições para os ajustes realizados no trabalho desde então.

Ao pesquisador David Card, referência em análise causal de defeitos de software com ampla experiência na implantação desta prática em empresas. Pela sua ajuda na concepção inicial da abordagem e pelo acompanhamento ao longo dos anos de pesquisa, sempre fornecendo apoio na escrita de artigos com seus *insights* a respeito da prática industrial.

Aos professores do Programa de Engenharia de Sistemas e Computação (PESC) da COPPE/UFRJ, pela excelência no ensino e pesquisa, tão amplamente reconhecida pela academia e pela indústria.

À COPPE/UFRJ, por dispor de toda a infra-estrutura necessária para a realização deste trabalho.

Aos professores do curso de Bacharelado em Ciência da Computação do IM-DCC/UFRJ, pela excelência do curso, que me forneceu a base para cursar o mestrado e o doutorado, em particular ao professor Marcos Roberto da Silva Borges, que me orientou no projeto final da graduação.

Aos meus amigos e sócios da Kali Software, Marcelo Nascimento Costa e Rodrigo Oliveira Spínola, pela confiança e pela dedicação à empresa nos momentos em que eu tive que me ausentar para focar na tese de doutorado, fazendo com que este período não interrompesse o crescimento da empresa.

Aos demais colaboradores da Kali Software, pela atuação comprometida e responsável, mesmo durante longos períodos de ausência minha na direção. Em especial aos que iniciaram conosco as atividades na fábrica de software de Juiz de Fora, Bernadete Aparecida de Aquino, Daniel Vieira, Fabrício Terra, João Carlos Dal Bianco, Marcella Ferreira da Costa, Marlon Marcos de Andrade e Walter Welenir. A confiança que pude depositar neles foi fundamental para me dar a tranquilidade de concluir este trabalho.

Ao diretor de Competitividade e Qualidade da SOFTEX, José Antônio Antonioni, ao coordenador executivo do programa MPS.BR, Kival Chaves Weber e ao Gerente de Operações do MPS.BR, Nelson Franco. Pelo gratificante convívio e pelas oportunidades de atuação junto ao modelo nacional de qualidade de software, integrando a ETM (equipe técnica do modelo) pelos últimos quatro anos. Estas oportunidades agregaram muito à minha formação.

Ao professor Éber Schmitz, pela confiança depositada em meu trabalho para atuar como professor na pós-graduação e-IS Expert do NCE/UFRJ e pela compreensão durante este período de conclusão do doutorado.

Ao Reitor da Universidade Veiga de Almeida, Mário Veiga de Almeida Júnior, ao Pró-Reitor Acadêmico, Arlindo Cadarett Vianna, ao Diretor Acadêmico, Sérgio Baltar e à Diretora de Pós-Graduação e Pesquisa, Beatriz Balena. Pelo gratificante convívio e pela oportunidade de atuar como coordenador do curso de graduação em Engenharia da Computação. Aos demais coordenadores da Universidade Veiga de Almeida.

Ao professor José Antônio Moreira Xexéo, por seus ensinamentos a respeito de ensino de computação para cursos de graduação e pelas oportunidades dadas para desenvolver atividades de ensino e pesquisa no Instituto Metodista Bennett

durante este período. Pelo gratificante convívio e pela agradável experiência de escrita de artigos sobre educação em computação. A todos os professores que atuaram comigo ou sob a minha coordenação no curso de Ciência da Computação do Instituto Metodista Bennett.

À minha mãe e aos meus irmãos, por acreditarem em mim e pelo apoio constante às minhas iniciativas pessoais e profissionais, muitas vezes de difícil compreensão em curto prazo. Ao meu pai, por manter a expectativa sempre alta e só dar uma alternativa aos filhos, vencer. À minha avó Berta Kalinowski, por se emocionar com o fato de eu estar concluindo doutorado mesmo sem compreender muito bem este processo ou o seu significado.

Aos companheiros da COPPE, em especial aos que colaboraram diretamente com a minha pesquisa, Arilo Claudio Dias Neto, Jobson Luiz Massolar, Lucas Paes, Paulo Sérgio Medeiros, Rodrigo Oliveira Spínola e Vitor Pires Lopes, e aos membros dos grupos ESE e LENS pela amizade, sugestões e ajuda nos momentos que precisei.

Aos meus demais amigos, por compreender os meus períodos de reclusão social e não deixar que estes afetassem nossa amizade.

Aos meus alunos e ex-alunos, tanto de iniciativas acadêmicas quanto de treinamentos na indústria, pelo aprendizado obtido junto a eles e pela sua contribuição indireta na minha formação.

E, finalmente, mas não menos importante, a Deus por permitir a conclusão de mais esta etapa importante da minha vida.



Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

## UMA ABORDAGEM PROBABILÍSTICA PARA ANÁLISE CAUSAL DE DEFEITOS DE SOFTWARE

Marcos Kalinowski

Março/2011

Orientador: Guilherme Horta Travassos

Programa: Engenharia de Sistemas e Computação

Análise causal de defeitos tem se mostrado uma forma eficiente para melhoria de processos com base no produto. Nesta tese uma abordagem de análise causal, chamada DPPI (*Defect Prevention-Based Process Improvement*) é elaborada com base em diretrizes obtidas a partir de revisões sistemáticas da literatura e *feedback* obtido de especialistas da área. DPPI representa uma abordagem inovadora que integra mecanismos de aprendizado de causa e efeito (redes Bayesianas) nos procedimentos de análise causal de defeitos. Adicionalmente, para facilitar o uso das inferências diagnósticas Bayesianas em reuniões de análise causal, o tradicional diagrama de causa e efeito foi estendido para um diagrama de causa e efeito probabilístico.

A experiência de aplicar DPPI a um projeto real indicou sua viabilidade e forneceu uma percepção adicional a respeito de requisitos para apoio ferramental. Nesta experiência, o diagrama de causa e efeito probabilístico diagnosticou as principais causas eficientemente, motivando investigações adicionais. Assim, um estudo experimental foi realizado para investigar os benefícios de utilizar tais diagramas durante as reuniões de análise causal de defeitos.

Os resultados do estudo experimental forneceram indícios de que o uso dos diagramas de causa e efeito probabilísticos de DPPI aumenta a eficácia (capacidade de identificar) e reduz o esforço na identificação de causas de defeitos, quando comparado à identificação de causas de defeitos sem o uso dos diagramas.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

## A PROBABILISTIC SOFTWARE DEFECT CAUSAL ANALYSIS APPROACH

Marcos Kalinowski

March/2011

Advisor: Guilherme Horta Travassos

Department: Computer Science and Systems Engineering

Defect causal analysis (DCA) has shown itself to be an efficient mean to obtain product-focused software process improvement. In this thesis a DCA approach called Defect Prevention-Based Process Improvement (DPPI) is assembled based on guidance acquired through systematic literature reviews and feedback from experts in the field. DPPI represents an innovative approach integrating cause-effect learning mechanisms (Bayesian networks) into DCA procedures. Additionally, in order to facilitate the use of Bayesian diagnostic inferences to support DCA meetings, the traditional cause-effect diagram was extended into a probabilistic cause-effect diagram.

The experience on applying DPPI to a real software project showed its feasibility and provided insights into the requirements for tool support. Moreover, it was possible to observe that DPPI's probabilistic cause-effect diagrams were able to efficiently predict the main defect causes, motivating further investigations. Therefore, an experimental study was conducted to investigate the benefits of such diagrams during DCA meetings.

Results of the experimental study indicate that the use of DPPI's probabilistic cause-effect diagrams can increase the effectiveness (capacity of identification) and reduce the effort of identifying causes of software defects, when compared to the identification of defect causes without using the diagrams.

# ÍNDICE

Índice de Figuras .....	xiii
Índice de Tabelas.....	xvi
Capítulo 1 - Introdução .....	1
1.1    Introdução .....	1
1.2    Contexto.....	4
1.3    Motivação.....	7
1.4    Questões de Pesquisa .....	9
1.5    Objetivos .....	10
1.6    Metodologia de Pesquisa.....	11
1.7    Organização da Tese.....	14
Capítulo 2 - Revisão Sistemática sobre Análise Causal de Defeitos de Software .....	17
2.1    Introdução .....	17
2.2    Planejamento da Revisão Sistemática.....	17
2.3    Execução da Revisão Sistemática.....	19
2.4    Resultados da Revisão Sistemática.....	29
2.5    Considerações Finais.....	37
Capítulo 3 - Diretrizes para Implementar Análise Causal de Defeitos em Organizações de Software .....	38
3.1    Introdução .....	38
3.2    Minha Organização está Pronta para Análise Causal de Defeitos de Software? .....	39
3.3    Que Abordagem deve ser Seguida? .....	39
3.4    Que Métricas devem ser Coletadas? .....	40
3.5    Como Análise Causal de Defeitos pode ser Integrada com Controle Estatístico de Processos?.....	41
3.6    Como Categorizar os Defeitos? .....	44
3.7    Como Categorizar as Causas? .....	45
3.8    Quais os Custos e Resultados Esperados da Análise Causal de Defeitos? .....	45
3.9    Uso das Diretrizes para Realizar Análise Causal de Defeitos de Acordo com os Modelos de Maturidade .....	46

3.10	Considerações Finais.....	49
Capítulo 4 - Abordagem Probabilística para Análise Causal de Defeitos de Software. 51		
4.1	Introdução .....	51
4.2	Visão Geral da Abordagem DPPI .....	52
4.3	DPPI: Análise da Atividade de Desenvolvimento.....	58
4.4	DPPI: Preparação para Análise Causal .....	60
4.5	DPPI: Reunião de Análise Causal .....	62
4.6	DPPI: Melhoria da Atividade de Desenvolvimento.....	67
4.7	Outras Possibilidades de Uso para DPPI .....	67
4.8	Considerações Finais.....	70
Capítulo 5 - Prova de Conceito e Apoio Ferramental .....		
5.1	Introdução .....	73
5.2	Prova de Conceito: Aplicação de DPPI a um Projeto Real.....	73
5.3	Apoio Ferramental: DPPI Framework .....	81
5.4	Considerações Finais.....	89
Capítulo 6 - Avaliação da Abordagem de DPPI para a Identificação de Causas de Defeitos de Software.....		
6.1	Introdução .....	91
6.2	Estudo Experimental: Avaliando o uso da Abordagem de DPPI para a Identificação de Causas de Defeitos de Software. ....	91
6.3	Considerações Finais.....	113
Capítulo 7 - Considerações Finais.....		
7.1	Considerações Finais.....	115
7.2	Resultados Obtidos.....	117
7.3	Contribuições .....	118
7.4	Limitações .....	119
7.5	Trabalhos Futuros.....	120
Referências Bibliográficas .....		
Anexo A – Listagem de Artigos da Execução do Protocolo da Revisão Sistemática . 135		
A1	– Artigos Obtidos em Agosto de 2006 das Máquinas de Busca .....	135
A2	– Artigos Adicionais Obtidos em Setembro de 2007 das Máquinas de Busca .....	147
A3	– Artigos Adicionais Obtidos em Janeiro de 2009 das Máquinas de Busca	149
A4	– Artigos Adicionais Obtidos em Julho de 2010 das Máquinas de Busca....	151
A5	– Artigos Obtidos em Julho de 2010 da Máquina de Busca SCOPUS .....	154
Anexo B – Informações Extraídas dos Artigos Recuperados.....		
		162

B1 – Informações Extraídas dos Artigos Recuperados na Revisão de Agosto de 2006 .....	162
B2 – Informações Extraídas dos Artigos Recuperados na Revisão de Setembro de 2007 .....	235
B3 – Informações Extraídas dos Artigos Recuperados na Revisão de Janeiro de 2009 .....	248
B4 – Informações Extraídas dos Artigos Recuperados na Revisão de Julho de 2010 .....	256
B5 – Informações Extraídas dos Artigos Adicionais Recuperados na Revisão de Julho de 2010 na SCOPUS .....	270
Anexo C – Sumário das Informações Extraídas dos Artigos Recuperados.....	285
C1 – Informações Extraídas sobre Processos e Abordagens .....	285
C2 – Informações Extraídas sobre Esquemas de Classificação de Defeitos ....	290
C3 – Informações Extraídas sobre Esquemas de Classificação de Causas de Defeitos .....	300
C4 – Informações Extraídas sobre Outros Conhecimentos Relacionados a Análise Causal de Defeitos .....	305
Anexo D – Template para Apoiar a Aplicação Manual de DPPI.....	319
D1 – DPPI Template .....	319
Anexo E – A Ferramenta DPPI Framework .....	322
E1 – Descrição Geral do Problema .....	322
E2 – Escopo .....	322
E3 – Requisitos Não Funcionais e Funcionais .....	323
E4 – Casos de Uso .....	324
E5 – Arquitetura da Ferramenta .....	328
E6 – Diagrama de Classes do Domínio .....	330
E7 – Funcionalidade .....	330
Anexo F – Instrumentos do Estudo Experimental.....	338
F1 – Consentimento de Participação.....	338
F2 – Caracterização dos Participantes .....	340
F3 – Instrumentos para a Operação do Experimento.....	342
F4 - Questionário de Acompanhamento .....	356

# ÍNDICE DE FIGURAS

Figura 1.1. Atividades do Processo Tradicional de Prevenção de Defeitos (Mays <i>et al.</i> , 1990). Adaptado de (Rombach e Endres, 2003). .....	6
Figura 1.2. Elementos de um Sistema Causal. Adaptado de (Card, 2005). .....	9
Figura 1.3. Instanciação da Metodologia de Pesquisa Adotada até o Exame de Qualificação. ....	11
Figura 1.4. Instanciação da Metodologia de Pesquisa Adotada entre o Exame de Qualificação e a Defesa da Tese.....	12
Figura 2.1. Artigos Recuperados das Bibliotecas Digitais em Agosto de 2006.....	20
Figura 2.2. Artigos Adicionais Recuperados das Bibliotecas Digitais em Setembro de 2007.....	21
Figura 2.3. Referências entre os Artigos Seleccionados nas Revisões de 2006 e 2007. ....	23
Figura 2.4. Artigos Adicionais Recuperados das Bibliotecas Digitais em Janeiro de 2009.....	24
Figura 2.5. Referências entre os Artigos Seleccionados nas Revisões de Agosto de 2006, Setembro de 2007 e Janeiro de 2009.....	25
Figura 2.6. Artigos Adicionais Recuperados das Bibliotecas Digitais em Julho de 2010. ....	26
Figura 2.7. Referências entre os Artigos Seleccionados nas Revisões de Agosto de 2006, Setembro de 2007, Janeiro de 2009 e Julho de 2010. ....	27
Figura 3.1. Gráfico de Pareto Aplicado à Natureza de Defeitos.....	40
Figura 3.2. Gráficos XmR e U, Considerando Dados Equivalentes. ....	43
Figura 4.1. Momentos de Aplicação de DPPI em um Processo Iterativo Incremental. ....	54
Figura 4.2. Visão Geral da Abordagem DPPI.....	55
Figura 4.3. Distribuição de Probabilidades Conjunta sobre Variáveis Aleatórias de uma Rede Bayesiana. ....	56
Figura 4.4. Ciclo de Retroalimentação da Abordagem DPPI – Os Modelos Causais para as Atividades de Desenvolvimento são Construídos com os Próprios Resultados das Reuniões de Análise Causal. ....	57
Figura 4.5. Exemplos de Gráfico U com Base em Dados de Seis Iterações de um Projeto Real. ....	59
Figura 4.6. Exemplo de Gráfico de Pareto para um Módulo de um Projeto Real.....	61

Figura 4.7. Diagrama de Causa e Efeito Probabilístico para Omissões em Especificações Funcionais, Elaborado com Base em Dados de um Projeto Real. ....	63
Figura 4.8. Estrutura da Rede Bayesiana para o Aprendizado das Causas das Diferentes Categorias de Defeitos. ....	64
Figura 4.9. Rede Bayesiana com Valores e Probabilidades Preenchidas (Gerada a partir de Tuplas de um Projeto Real utilizando a Ferramenta Netica). ....	65
Figura 4.10. Inferência Bayesiana para a Categoria de Defeitos ‘Omissões’. ....	66
Figura 4.11. Rede Bayesiana para Dados Adicionais sobre Defeitos. ....	68
Figura 4.12. Inferência Preditiva para a Causa “Entrevistar o Interessado Inapropriado”. ....	69
Figura 4.13. Inferência Preditiva para a Causa “Falta de Conhecimento de Boas Práticas de Engenharia de Requisitos” ....	70
Figura 5.1. Gráfico U para Defeitos por Hora de Inspeção e Defeitos por Unidade de Tamanho (Pontos de Caso de Uso). ....	74
Figura 5.2. Gráfico de Pareto para os Defeitos do Módulo MPTV. ....	75
Figura 5.3. <i>Brainstorm</i> sobre Possíveis Causas de Defeitos. ....	77
Figura 5.4. Inferência Bayesiana e o Diagrama de Causa e Efeito Probabilístico Correspondente. ....	78
Figura 5.5. Arquitetura da Ferramenta DPPI Framework. ....	82
Figura 5.6. Caracterização da Sessão de Análise Causal. ....	83
Figura 5.7. Apoio Ferramental à Atividade de Análise da Atividade de Desenvolvimento. ....	84
Figura 5.8. Apoio Ferramental à Atividade de Preparação para Análise Causal. ....	85
Figura 5.9. Associação de Defeitos aos Erros Sistemáticos. ....	86
Figura 5.10. Apoio Ferramental à Atividade de Reunião de Análise Causal. ....	87
Figura 5.11. Distribuição de Probabilidades de Acordo com a Inferência Bayesiana. .	88
Figura 5.12. Apoio Ferramental à Atividade de Melhoria da Atividade de Desenvolvimento. ....	89
Figura 6.1. Processo de Experimentação. Adaptado de (Wohlin <i>et al.</i> , 2000a). ....	92
Figura 6.2. Planejamento do Experimento. Adaptado de (Wohlin <i>et al.</i> , 2000a). ....	94
Figura 6.3. Projeto do Estudo Experimental. Três Rodadas, Envolvendo Quatro Participantes, Dois Tratamentos e Dois Objetos de Estudo. ....	99
Figura 6.4. Gráficos de Pareto para os Módulos MAP e MFI. Principais Categorias de Defeitos. ....	100
Figura 6.5. Variação da Eficácia dos Participantes por Erro Sistemático Analisado no Módulo 1. ....	109

Figura 6.6. Variação da Eficácia dos Participantes por Erro Sistemático Analisado no Módulo 2. ....	109
Figura E.1. Diagrama de Casos de Uso para a Ferramenta DPPI Framework. ....	325
Figura E.2. Arquitetura da Ferramenta DPPI Framework. ....	328
Figura E.3. Exemplo de Lista de Defeitos exportada da Ferramenta ISPIS. ....	329
Figura E.4. Mapeamento dos Campos de ISPIS e DPPI. ....	329
Figura E.5. Diagrama de Classes do Domínio para a Ferramenta DPPI Framework. ....	330
Figura E.6. Caracterização da Reunião de Análise Causal. ....	331
Figura E.7. Adição de um Módulo. ....	331
Figura E.8. Importação da Lista de Defeitos. ....	332
Figura E.9. Apoio à Análise da Atividade de Desenvolvimento. ....	333
Figura E.10. Apoio à Preparação para a Análise Causal de Defeitos. ....	334
Figura E.11. Apoio à Reunião de Análise Causal de Defeitos. ....	336
Figura E.12. Apoio à Melhoria da Atividade de Desenvolvimento. ....	337



# ÍNDICE DE TABELAS

Tabela 2.1. Pesquisa e Aplicação de Técnicas de Análise Causal de Defeitos de Software. Adaptado de (Card, 2005). .....	37
Tabela 3.1. Dados Fictícios para Ilustrar a Diferença de Sensibilidade entre Gráficos XmR e U. ....	43
Tabela 4.1. Mapeamento das Práticas Específicas da Área de Processos CAR do CMMI para as Atividades de DPPI. ....	67
Tabela 6.1. Erros Sistemáticos Identificados nos Módulos 1 (MAP) e 2 (MFI). ....	101
Tabela 6.2. Experiência Prática e Nível de Formação dos Participantes.....	103
Tabela 6.3. Resultados Quantitativos do Estudo Experimental.....	104
Tabela 6.4. Resultados Qualitativos do Estudo Experimental.....	105
Tabela 6.5. Resultados para cada um dos Participantes Aplicando os dois Tratamentos ao mesmo Módulo. ....	108
Tabela 6.6. Gabarito de Causas principais para os Erros Sistemáticos do Módulo 1 (MAP).....	112
Tabela 6.7. Gabarito de Causas principais para os Erros Sistemáticos do Módulo 2 (MFI). ....	112
Tabela A.1. Artigos obtidos em Agosto de 2006 das Máquinas de Busca. ....	135
Tabela A.2. Artigos Adicionais Obtidos em Setembro de 2007 das Máquinas de Busca. ....	147
Tabela A.3. Artigos Adicionais Obtidos em Janeiro de 2009 das Máquinas de Busca. ....	149
Tabela A.4. Artigos Adicionais Obtidos em Julho de 2010 das Máquinas de Busca. ....	151
Tabela A.5. Artigos Obtidos em Julho de 2010 da Máquina de Busca SCOPUS. ....	154
Tabela C.1. Processos e Abordagens.....	285
Tabela C.2. Esquemas de Classificação de Defeitos.....	290
Tabela C.3. Esquemas de Classificação de Causas de Defeitos.....	300
Tabela C.4. Conhecimento Genérico a sobre Prevenção de Defeitos. ....	305
Tabela C.5. Conhecimento sobre a Reunião de Análise Causal.....	311
Tabela C.6. Conhecimento sobre a Implementação das Ações.....	316
Tabela C.7. Conhecimento sobre a Atividade de Coleção de Dados de Defeitos e Acompanhamento de Atividades de Engenharia.....	316

# CAPÍTULO 1 - INTRODUÇÃO

*Neste capítulo são apresentados o contexto do trabalho, o que motivou esta pesquisa e as questões de investigação. São também apresentados os objetivos, a metodologia de pesquisa utilizada e como este texto está estruturado.*

## 1.1 Introdução

O aprendizado com base em erros cometidos é inerente ao amadurecimento humano. De acordo com Inhelder e Piaget (1958), entre o momento em que uma criança começa a falar até uma idade de aproximadamente 6 a 7 anos, ela tende a se encontrar no estágio de raciocínio pré-operacional. Neste estágio, o raciocínio passa por diversas limitações, como o egocentrismo, o animismo e a falta de habilidade de observar conservação de propriedades, entre outras.

O egocentrismo se refere à incapacidade de assumir a perspectiva de outra pessoa. Um exemplo do egocentrismo ocorre quando se vê uma criança tapando os próprios olhos e dizendo: "Você não pode me ver!". O animismo, por sua vez, diz respeito à tendência destas crianças de acreditar que todos os objetos são vivos e são capazes de ter sentimentos, intenções e emoções. Um exemplo do animismo ocorre quando se vê uma criança acidentalmente se machucando ao tropeçar em uma pedra e depois voltando para chutar a pedra ou jogá-la para longe, como se a pedra tivesse tido a intenção de machucá-la. A falta de habilidade de observar a conservação de propriedades faz com que uma criança neste estágio tenha dificuldades de observar, por exemplo, que um copo mais largo possa conter mais líquido do que um copo mais fino, mesmo com o líquido apresentando uma altura menor no interior do copo.

As lições aprendidas com os erros cometidos durante a fase pré-operacional ajudam a criança a migrar para a fase operacional concreta, que de acordo com os autores normalmente ocorre entre os 7 e 12 anos, momento em que a criança começa a desenvolver o seu raciocínio lógico, com base em pensamentos concretos, tendo ainda muitas dificuldades com o pensamento abstrato. Assim, por exemplo, após tapar algumas vezes os próprios olhos e dizer que ninguém consegue vê-la e mesmo assim ser encontrada a criança vai aprendendo a levar em consideração a perspectiva de terceiros. Ao observar que sua ação de chutar a pedra não afetou a pedra, ela vai aprendendo a considerar que a pedra talvez possa não ter tido intenção de machucá-

la. Ao passar líquidos de um recipiente para outro, eventualmente transbordando o conteúdo, ela começa a trabalhar sua habilidade de observar a conservação de propriedades.

O aprendizado com base em erros cometidos não ocorre somente com crianças. Ao deixar o celular tocar em uma reunião de negócios, uma pessoa tende a aprender com o erro e a evitar repeti-lo. Outro exemplo interessante de ser considerado é o de um motorista que, ao ver uma multa de trânsito impactando suas finanças pessoais, tende a dirigir de forma mais cautelosa. Da mesma forma, alguém que perde a(o) parceira(o) por atuar de forma excessivamente ciumenta, tende a rever suas atitudes nos próximos relacionamentos. Oscar Wilde sumariza este processo de aprendizado com base em erros, no terceiro ato de seu primeiro sucesso teatral ("O leque de Lady Windermere" - 1893), com a frase: "Experiência é o nome que as pessoas dão aos seus erros" (Wilde, 1940). O moralista e biógrafo grego Plutarch (46-120 D.C.) expressou isto em um de seus pensamentos: "Não cometer erros não está ao alcance dos homens; mas de seus erros os sábios e bons obtêm sabedoria para o futuro" (Blackburn, 2008).

Entretanto, mais do que utilizar os próprios erros para aprendizado pessoal, é possível utilizar lições aprendidas de erros alheios para evitar cometê-los novamente. De fato, erros cometidos ao longo da história da humanidade têm sido analisados para que não sejam repetidos.

O impacto de repetir erros do passado pode ser exemplificado pelo ocorrido durante a tentativa do general cartaginês Aníbal Barca de invadir Roma com suas tropas, militarmente superiores. Durante esta tentativa, Aníbal ultrapassou os limites de alcance de suprimentos sem prover a logística necessária para suas tropas. O general romano Quintus Fabius Maximus estava ciente da superioridade militar dos cartagineses, então tentou fazer do tempo o seu aliado, uma vez que as tropas de Aníbal estavam longe de suas terras. Assim, quando Aníbal tentou invadir Roma, Fabius se recusou a enfrentá-lo em uma única batalha campal de grande porte. Ao contrário, ele manteve suas tropas próximas às de Aníbal, para esgotá-lo, provocando diversos atritos para explorar vulnerabilidades estratégicas de Aníbal, utilizando unidades militares menores, ao mesmo tempo em que impedia que as tropas de Aníbal tivessem acesso a suprimentos. Assim, Fabius foi capaz de defender o império romano dos ataques de Aníbal, utilizando uma estratégia que posteriormente ficaria conhecida como a "Estratégia de Fabius" (Mellor, 1999).

Quase vinte séculos depois, quando Napoleão tentou invadir a Rússia ele não fez uso da lição aprendida do erro de Aníbal. Assim como no caso de Aníbal, o império napoleônico era considerado militarmente superior, mas os generais do alto comando

Russo eram intelectuais estudiosos da história militar e sabiam como aplicar lições aprendidas no campo de batalha. O comandante Mikhail Kutznov decidiu aplicar a “Estratégia de Fabius”, durante o rigoroso inverno russo, e saiu vitorioso do conflito. Dos aproximadamente 450.000 soldados do império napoleônico somente 6.000 retornaram à França. Repetir o erro de Aníbal, ao invés de estudá-lo e analisar suas causas para evitá-lo, representou a primeira grande derrota de Napoleão e o início do declínio de seu império (Abbot, 2005).

Para evitar a repetição de erros do passado, é preciso mais do que saber que eles ocorreram. É preciso conhecer suas causas, para que estas possam ser tratadas evitando que os erros sejam novamente cometidos no futuro (Robitaille, 2004).

No caso do celular que tocou na reunião de negócios, a identificação da causa principal, que ao ser tratada evita que evento se repita, é simples. Esta causa é a pessoa ter se esquecido de alterar o modo de operação do celular para silencioso antes da reunião. É possível ainda observar que, toda vez que um celular tocar em uma reunião a causa será a mesma. Um exemplo de uma ação que poderia ser tomada pela empresa para evitar que esta causa ocorra é colocar um aviso na porta de entrada da sala de reuniões, dizendo “Por favor, lembre-se de colocar o seu celular no modo silencioso antes da reunião”.

No caso da derrota de Aníbal ao tentar invadir Roma, o sistema causal é mais complexo e a identificação das causas principais envolve todo o contexto da guerra, não sendo possível identificar as principais causas apenas com as informações aqui fornecidas. Assim, esta tarefa é deixada para os pesquisadores da área de História. Mas, sendo o contexto de Napoleão parecido, ele poderia ter se beneficiado dos resultados desta análise para saber de antemão sobre possíveis causas que poderiam levar também à sua derrota.

Na área de software, evitar a repetição de erros do passado e aumentar a maturidade dos processos de construção pode se mostrar particularmente interessante. De acordo com Pfleeger e Atlee (2009), software permeia nosso mundo e está presente em praticamente todos os aspectos de nossas vidas. Assim, é preciso assegurar que os softwares, ao serem construídos, funcionem e façam contribuições positivas para as formas como lidamos com nossas vidas. Tian (2005) destaca que estas demandas precisam ser satisfeitas pelas pessoas envolvidas no desenvolvimento e na manutenção destes sistemas de software através de processos que envolvam diferentes atividades de controle da qualidade.

Atividades de controle da qualidade revelam defeitos presentes nos artefatos desenvolvidos ao longo da construção do software, de modo que estes possam ser corrigidos antes da entrega do software. A análise causal destes defeitos (Card, 1993),

por sua vez, envolve identificar os erros cometidos, identificar as causas para estes erros e melhorar o processo para evitar que estas causas ocorram novamente no futuro. Ou seja, o objetivo é evitar a introdução de defeitos, prevenir ao invés de remediar (Mays *et al.*, 1990).

Desta forma, a análise causal de defeitos é uma prática da engenharia de software que representa, em teoria, uma oportunidade para aplicar o pensamento de Plutarch para organizações desenvolvedoras de software. Trazendo o pensamento para este contexto, podemos rephraseá-lo da seguinte forma: “Não introduzir defeitos não está ao alcance de uma organização de software; mas de seus defeitos as organizações sábias e boas obtêm sabedoria para o futuro”.

Entretanto, embora a análise causal seja realizada por algumas organizações de software, as abordagens encontradas envolvem apenas a identificação das causas e a melhoria do processo para evitar a introdução do mesmo tipo de defeito no futuro, e não o acúmulo de conhecimento sobre possíveis causas de defeitos. De acordo com Pearl (2000), o acúmulo de conhecimento sobre relações de causa e efeito é o objetivo central de muitas pesquisas, nas diferentes ciências. Entretanto, a engenharia de software tem negligenciado a elucidação de relações causais referentes a defeitos de software.

Voltando ao paralelo com o comportamento humano em que o aprendizado com base em erros cometidos é inerente ao amadurecimento, é como se uma criança ao pôr a mão sobre a chama de um fogão e se queimar em função deste erro mudasse sua forma de agir (o processo), não colocando mais a mão sobre a chama do fogão, sem obter o conhecimento da relação causal de que o fogo pode causar queimaduras severas.

Assim, a inexistência de abordagens da engenharia de software que permitam a organizações desenvolvedoras de software utilizar os dados de seus defeitos para ao mesmo tempo melhorar seus processos e obter conhecimento a respeito de suas relações causais representa uma lacuna a ser preenchida. Esta lacuna é tratada nesta tese.

## **1.2 Contexto**

Análise causal e resolução é uma maneira de se identificar causas de defeitos e outros problemas e tomar ações para preveni-los de re-ocorrer no futuro. Ela é considerada como importante em muitos modelos e abordagens de melhoria de processos, tais como CMMI (SEI, 2006a), ISO/IEC 12207 (ISO/IEC, 2008a), MPS (SOFTEX, 2009) e Six Sigma (Eckes, 2001). Robitaille (2004) destaca a análise causal

e resolução de problemas como uma forma de identificar oportunidades de melhoria para os ativos de processo da organização e não apenas para projetos isolados.

A análise causal de defeitos de software compreende aplicar análise causal e resolução a um tipo específico de problema: os defeitos dos artefatos de software gerados ao longo do processo de desenvolvimento. O escopo desta tese é relacionado à análise causal de defeitos de software identificados através de inspeções de software. Outros problemas, como desvios no cronograma, não serão tratados diretamente nesta pesquisa.

Adicionalmente, é importante definir de forma precisa o que o termo defeito representa neste trabalho, já que normalmente sua interpretação depende do contexto em que é utilizado. Seguem algumas definições da última atualização do padrão IEEE para a classificação de anomalias de software (IEEE 1044-2009, 2010):

- **Defeito.** Uma imperfeição ou deficiência em um produto de trabalho que faz com que este produto de trabalho não atenda seus requisitos ou especificações e precisa ser consertado ou substituído.
- **Erro.** Uma ação humana que produz um resultado incorreto.
- **Falha.** Fim da habilidade de um produto de realizar uma função requerida ou sua inabilidade de desempenhar dentro de limites previamente especificados.
- **Falta.** É uma manifestação concreta de um erro em um software.
- **Problema.** Dificuldade ou incerteza vivida por uma ou mais pessoas, resultando de um encontro insatisfatório com um sistema em uso.

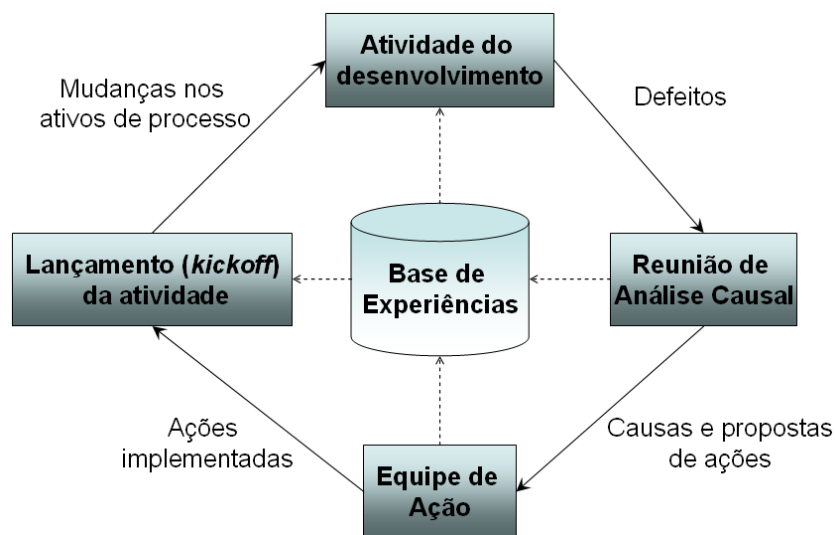
Assim, problemas e falhas estão associados ao uso do software, em que problemas podem ser vividos por usuários e falhas podem ser reveladas (seja em uso operacional, ou, preferencialmente, em atividades de teste). Adicionalmente, de acordo com esta atualização, um defeito pode ser uma falta ou não. Um defeito será uma falta se ele for encontrado através da análise de uma falha do software. Um defeito não é uma falta se ele for encontrado por uma inspeção ou análise estática e removido antes da execução do software.

Neste trabalho consideramos esta terminologia para defeitos. Desta forma, a análise considera apenas defeitos e não problemas, falhas e erros. No caso de falhas será necessário encontrar os defeitos (faltas) correspondentes às falhas através da análise dos artefatos (por exemplo, através de depuração) antes de dar início à análise causal de defeitos de software.

De posse destas definições iniciais, análise causal de defeitos de software pode ser vista como um processo sistemático para identificar e analisar causas

associadas com a ocorrência de tipos específicos de defeitos, permitindo a identificação de oportunidades de melhoria para os ativos de processo da organização e a implementação de ações para prevenir a ocorrência daquele mesmo tipo de defeito em projetos futuros. Desta maneira, a análise causal de defeitos de software visa apoiar a melhoria de processos com base em uma característica concreta do produto: os defeitos contidos em seus artefatos.

A análise causal de defeitos de software pode ser considerada parte do processo de prevenção de defeitos, que trata também a implementação das ações e a comunicação das mudanças implementadas à equipe de desenvolvimento. Uma representação das atividades do processo tradicional de prevenção de defeitos de software (Mays *et al.*, 1990), encontra-se na Figura 1.1.



**Figura 1.1. Atividades do Processo Tradicional de Prevenção de Defeitos (Mays *et al.*, 1990). Adaptado de (Rombach e Endres, 2003).**

Além da reunião de análise causal, a Figura 1.1 destaca a implementação das ações e a comunicação das mudanças à equipe de desenvolvimento. Adicionalmente, a figura indica que a aplicação de atividades de análise causal de defeitos pode colaborar para a formação de uma base de experiências na organização.

Considerando a reunião de análise causal de defeitos em si, Card (2005) a descreve resumidamente através dos seis seguintes passos: (1) selecionar uma amostra dos defeitos, (2) classificar os defeitos selecionados, (3) identificar erros sistemáticos, (4) identificar as principais causas, (5) desenvolver itens de ação e (6) documentar os resultados da reunião de análise causal. Neste contexto, um erro sistemático é um erro que resulta em defeitos similares se repetindo em diferentes ocasiões. Encontrar erros sistemáticos indica a existência de oportunidades de

melhoria para os ativos de processo. Além destes seis passos, Card (2005) destaca a importância de efetivamente gerenciar a implementação dos itens de ação até sua conclusão e de comunicar as modificações à equipe de desenvolvimento.

A análise causal de defeitos de software tem sido discutida desde a década de 70 (Endres, 1975) e é apontada por Boehm (2006) como uma das principais contribuições desta década para a engenharia de software. Desde então, o processo de análise causal de defeitos de software têm sido implantado na indústria, tanto para projetos envolvendo centenas de pessoas (Mays *et al.*, 1990) (Leszak *et al.*, 2002) quanto para projetos menores (Dangerfield *et al.*, 1992) (Yu, 1998) (Jalote e Agrawal, 2005).

Diversos benefícios associados à aplicação de análise causal de defeitos têm sido relatados. Um dos benefícios diretos é a redução na taxa de defeitos, que com análise causal de defeitos pôde ser reduzida em mais de cinquenta por cento em diferentes contextos organizacionais, como o da IBM (Mays *et al.*, 1990), da Computer Science Corporation (Dangerfield *et al.*, 1992), da HP (Grady, 1996) e da InfoSys (Jalote e Agrawal, 2005). Como consequência, a análise causal de defeitos de software tem mostrado diminuir a quantidade de retrabalho (Jalote e Agrawal, 2005) e aumentar a probabilidade de alcançar os objetivos de qualidade e de desempenho do processo de desenvolvimento. Adicionalmente, ela pode servir como instrumento para a difusão de lições aprendidas entre as equipes envolvidas nos diferentes projetos da organização (SEI, 2006a).

### **1.3 Motivação**

Apesar dos benefícios e da adoção na indústria, pouca pesquisa tem sido feita a respeito de análise causal de defeitos de software e pouco conhecimento científico tem sido gerado e publicado (Card, 2005). De fato, talvez por representar uma prática de alta maturidade e que pode significar uma vantagem competitiva para as empresas, também há poucos relatos de experiência, como pode ser observado na revisão sistemática descrita no capítulo 2.

Assim, muitas dúvidas permanecem sobre como implementar análise causal de defeitos de maneira sistemática e eficiente (permitindo a identificação das principais causas a um baixo esforço) em organizações de software, tais como:

- Minha organização está pronta para análise causal de defeitos de software?
- Que abordagem deve ser seguida?
- Que métricas devem ser coletadas?



- Como análise causal de defeitos pode ser integrada ao controle estatístico de processos?
- Como defeitos devem ser categorizados?
- Como as causas devem ser categorizadas?
- Quais são os resultados esperados da aplicação de análise causal de defeitos de software?

Visando fornecer respostas mais isentas e baseadas em evidência para estas dúvidas, uma revisão sistemática foi conduzida, permitindo a compilação de diretrizes para apoiar a implementação de análise causal de defeitos em organizações de software e a identificação de oportunidades para investigação futura (Kalinowski *et al.*, 2008a).

As abordagens encontradas durante a revisão sistemática não forneceram um detalhamento para a realização das tarefas associadas à análise causal de defeitos de software, considerando evidências, identificando e apoiando o uso das melhores práticas de análise causal de defeitos.

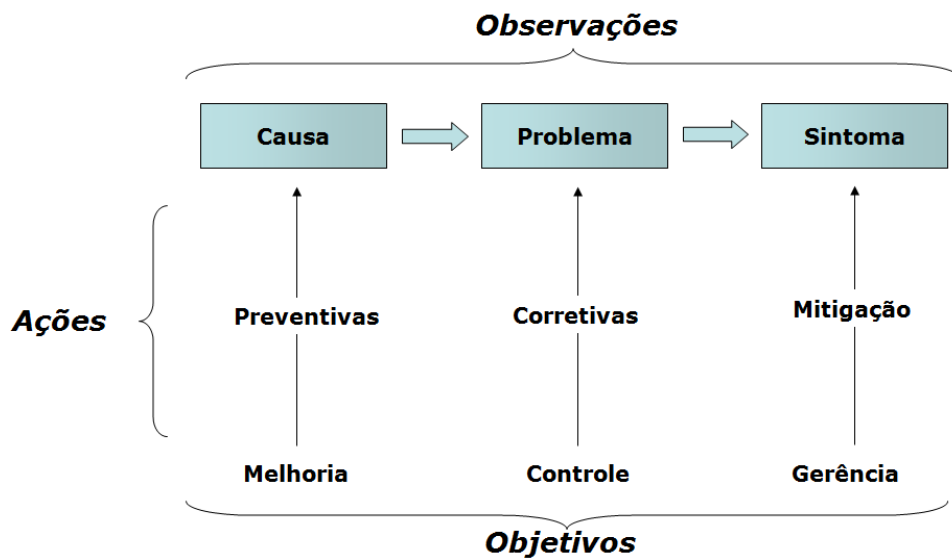
Adicionalmente, nenhuma das abordagens encontradas provê mecanismos para manter e utilizar o conhecimento gerado a respeito de relacionamentos causais<sup>1</sup> ao longo das diferentes sessões de análise causal. Ou seja, o objetivo das abordagens é somente a prevenção dos defeitos ao atuar em sua causa, sendo a elucidação de relacionamentos causais que permitam uma compreensão mais ampla das efetivas causas dos defeitos para aquele contexto organizacional negligenciada.

Para ser mais preciso, a Figura 1.2 ilustra os elementos de um sistema causal (Card, 2005). Esta figura destaca os elementos observáveis, as ações que devem ser tomadas sobre estes elementos e os objetivos em tomar tais ações. Entender os relacionamentos causais entre os elementos de tal sistema para um contexto organizacional específico poderia apoiar uma prevenção de defeitos mais eficiente, fornecendo respostas para perguntas como: “Dado um contexto organizacional, que causas costumam levar a que tipo de defeito”, ou “Dado um contexto organizacional, qual costuma ser o impacto de tomar uma ação para prevenir uma causa específica sobre um tipo específico de defeito”.

---

<sup>1</sup> De acordo com Babbie (1986), três condições precisam ser atendidas para demonstrar um relacionamento causal:

- Condição 1: Precisa haver uma correlação entre a causa e o efeito;
- Condição 2: A causa precisa preceder o efeito;
- Condição 3: O mecanismo ligando a causa ao efeito precisa ser identificado.



**Figura 1.2. Elementos de um Sistema Causal. Adaptado de (Card, 2005).**

De acordo com Pearl (2000), o objetivo central de muitas pesquisas, nas diferentes ciências, é a elucidação de relações de causa e efeito entre variáveis ou eventos. A inexistência de uma abordagem de análise causal de defeitos que permita estabelecer e utilizar conhecimento a respeito de relações causais associadas aos defeitos da organização reflete uma lacuna a ser preenchida de modo que hipóteses associadas a tal abordagem possam ser avaliadas.

## 1.4 Questões de Pesquisa

Tendo em vista o contexto e a motivação descritos nas seções anteriores, esta pesquisa apresenta uma abordagem para análise causal de defeitos que fornece para as organizações um detalhamento das práticas a serem realizadas e permite obter conhecimento a respeito das relações causais dos defeitos identificados através de inspeções de software do contexto organizacional. Assim, as seguintes questões foram formuladas para direcionar a pesquisa:

- Q1: Que conjunto de boas práticas deve ser considerado para fornecer um apoio detalhado para a realização eficiente (permitindo a identificação das principais causas a um baixo esforço) das tarefas associadas à análise causal de defeitos de software?
- Q2: Como organizar este conjunto de práticas no contexto de uma abordagem de análise causal de defeitos de software?

- Q3: Como fazer com que esta abordagem proveja um mecanismo para manter e utilizar o conhecimento gerado a respeito de relacionamentos causais ao longo das diferentes sessões de análise causal?
- Q4: Qual a viabilidade de aplicação da abordagem resultante em projetos de software reais?
- Q5: De que maneira a abordagem resultante pode ser apoiada por ferramentas para facilitar sua utilização em projetos de software reais?
- Q6: Quais os benefícios que esta abordagem traz em relação a diferentes aspectos, como: (i) eficácia na identificação das principais causas, (ii) esforço na identificação das principais causas e (iii) satisfação dos usuários ao realizar análise causal.

## 1.5 Objetivos

O objetivo principal desta tese é definir uma abordagem para análise causal de defeitos de software que forneça para as organizações um detalhamento das práticas a serem realizadas e permita obter conhecimento a respeito das relações causais dos defeitos do contexto organizacional. Este objetivo pode ser desmembrado nos seguintes objetivos específicos, com base nas questões de pesquisa da seção anterior:

- O1: Estabelecer um mecanismo sistemático e dinâmico para a identificação e constante atualização de um conjunto de boas práticas para a realização eficiente das tarefas associadas à análise causal de defeitos de software.
- O2: Organizar este conjunto de práticas no contexto de uma abordagem de análise causal de defeitos de software. Esta abordagem deve prover um mecanismo para manter e utilizar o conhecimento gerado a respeito de relacionamentos causais ao longo das diferentes sessões de análise causal.
- O3: Avaliar a viabilidade de aplicação da abordagem resultante em projetos de software reais e prover apoio ferramental para facilitar sua utilização.
- O4: Avaliar os benefícios que esta abordagem traz em relação a diferentes aspectos, como: (i) eficácia na identificação das principais causas, (ii) esforço na identificação das principais causas e (iii) satisfação dos usuários ao realizar análise causal.

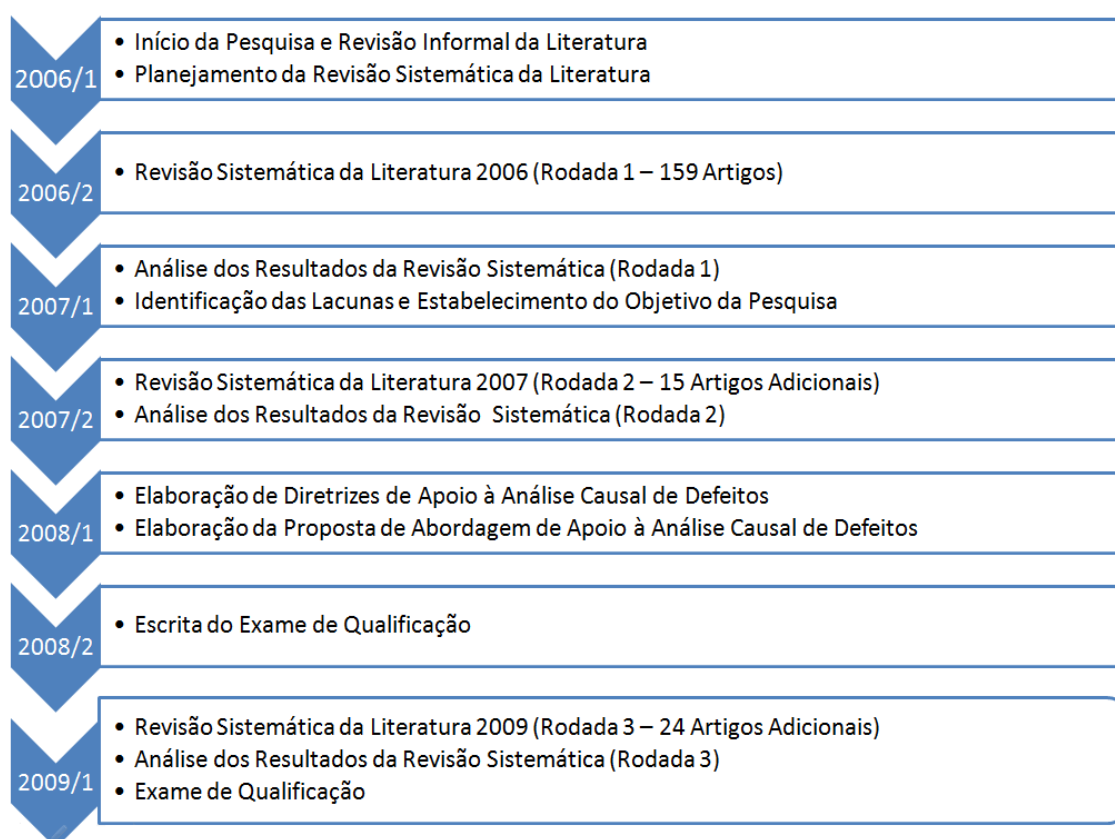
A metodologia adotada visando atingir estes objetivos encontra-se descrita na seção seguinte.

## 1.6 Metodologia de Pesquisa

A metodologia de pesquisa utilizada neste trabalho é fundamentada nos conceitos da engenharia de software experimental (Wohlin *et al.*, 2000a) e na metodologia descrita em (Mafra *et al.*, 2006), sendo apoiada pela condução de estudos secundários e primários. Conforme sugerido em (Mafra *et al.*, 2006), a metodologia é dividida em dois passos: (1) definição e (2) refinamento da tecnologia.

O primeiro destes passos envolve a realização de estudos secundários (revisões sistemáticas) com o intuito de fundamentar a elaboração de uma proposta de uma nova tecnologia. Assim, esta proposta utiliza o conhecimento adquirido e as evidências identificadas através da condução de revisões sistemáticas. O segundo passo, por sua vez, envolve a aplicação de estudos primários (provas de conceito, estudos de caso e estudos experimentais, entre outros) para refinar a tecnologia ou ampliar a compreensão a respeito da mesma.

A Figura 1.3 representa a instanciação desta metodologia, no contexto desta pesquisa, do início do doutorado até o exame de qualificação, destacando as atividades realizadas e os respectivos períodos.



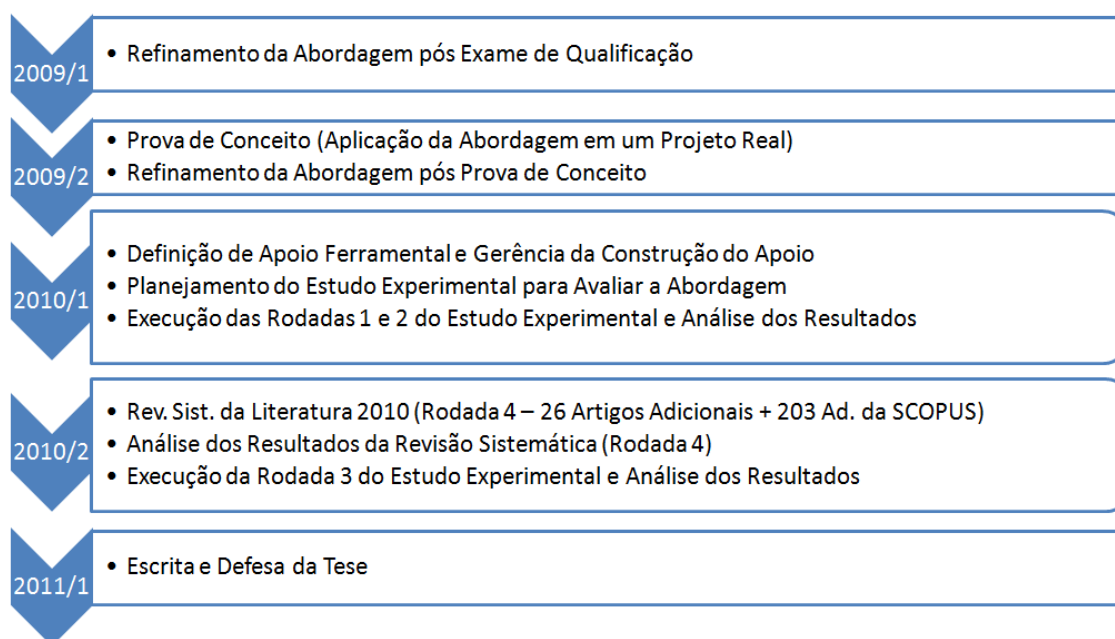
**Figura 1.3. Instanciação da Metodologia de Pesquisa Adotada até o Exame de Qualificação.**

Como pode ser visto na Figura, este período teve foco nos objetivos específicos O1 (identificação do conjunto de boas práticas) e O2 (concepção da abordagem). Desta forma, envolveu principalmente a execução de estudos secundários e a concepção inicial da abordagem de análise causal de defeitos.

Inicialmente, uma revisão informal da literatura foi conduzida antes do planejamento da revisão sistemática, possibilitando ampliar o conhecimento sobre a área a ser pesquisada. Após o planejamento da revisão sistemática a mesma foi conduzida (execução e análise dos resultados) duas vezes, permitindo a identificação de conhecimento e de oportunidades de pesquisa a respeito de análise causal de defeitos, considerando publicações indexadas até Setembro de 2007. Com base neste conhecimento, diretrizes para apoiar a implementação de análise causal de defeitos de software puderam ser elaboradas (Kalinowski *et al.*, 2008a).

Tendo em vista as oportunidades de pesquisa e estas diretrizes, uma proposta de pesquisa pode ser definida e uma abordagem inicial endereçando esta proposta pôde ser elaborada (Kalinowski *et al.*, 2008b). Por fim, para atualizar a revisão bibliográfica para a escrita do exame de qualificação uma nova revisão sistemática foi conduzida considerando publicações indexadas até Janeiro de 2009.

A Figura 1.4 representa a instanciación do restante da metodologia, no contexto desta pesquisa, para o período entre o exame de qualificação e a apresentação desta tese de doutorado, destacando as atividades realizadas e os respectivos períodos.



**Figura 1.4. Instanciación da Metodologia de Pesquisa Adotada entre o Exame de Qualificação e a Defesa da Tese.**

Como pode ser visto na Figura 1.4, este período teve foco em refinar a abordagem e nos objetivos específicos O3 (avaliar a viabilidade de utilizar a abordagem e prover apoio ferramental) e O4 (avaliar benefícios obtidos pelo uso da abordagem). Assim, além do refinamento da abordagem, envolveu uma prova de conceito (Kalinowski *et al.*, 2010), a construção de apoio ferramental e a avaliação dos benefícios através de três rodadas de um estudo experimental. Adicionalmente, para atualizar a revisão bibliográfica para a escrita da tese, uma nova rodada da revisão sistemática foi conduzida considerando publicações indexadas até Julho de 2010.

A partir da realização das atividades apresentadas nas Figuras 1.3 e 1.4, as seguintes contribuições e resultados foram obtidos para cada um dos objetivos.

- O1: Estabelecimento de um mecanismo sistemático e dinâmico para a identificação e constante atualização de um conjunto de boas práticas para a realização eficiente das tarefas associadas à análise causal de defeitos de software.
  - Kalinowski, M., Travassos, G.H., Card, D.N., Guidance for Efficiently Implementing Defect Causal Analysis, VII Simpósio Brasileiro de Qualidade de Software (SBQS), Florianópolis, 2008.
  - Kalinowski, M., Spínola, R.O., Dias-Neto, A.C., Bott, A., Travassos, G.H, Inspeções de Requisitos de Software em Desenvolvimento Incremental: Uma Experiência Prática, VI Simpósio Brasileiro de Qualidade de Software (SBQS), Porto de Galinhas, 2007.
- O2: Organizar este conjunto de práticas no contexto de uma abordagem de análise causal de defeitos de software. Esta abordagem deve prover um mecanismo para manter e utilizar o conhecimento gerado a respeito de relacionamentos causais ao longo das diferentes sessões de análise causal.
  - Kalinowski, M., Travassos, G. H., Card, D. N., Towards a Defect Prevention Based Process Improvement Approach, 34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 199-206, Parma, Italy, 2008.
  - Kalinowski, M., Travassos, G.H., Towards a Defect Causal Analysis Approach for Software Process Improvement and Organizational Learning, Workshop de Teses e Dissertações em Qualidade de Software (WTDQS) do VII Simpósio Brasileiro de Qualidade de Software, Florianópolis, 2008.

- O3: Avaliar a viabilidade de aplicação da abordagem resultante em projetos de software reais e prover apoio ferramental para facilitar sua utilização.
  - Kalinowski, M., Travassos G.H., Mendes, E., Card, D.N., Applying DPPI: A Defect Causal Analysis Approach Using Bayesian Networks, 11th International Conference on Product Focused Software Development and Process Improvement (PROFES 2010), LNCS 6156, pp. 92–106, Limerick, Ireland, 2010.
- O4: Avaliar os benefícios que esta abordagem traz em relação a diferentes aspectos, como: (i) eficácia na identificação das principais causas, (ii) esforço na identificação das principais causas e (iii) satisfação dos usuários ao realizar análise causal.
  - Kalinowski, M., Mendes, E., Travassos G.H., Automating and Evaluating the Use of DPPI's Probabilistic Cause-Effect Diagrams to Improve Defect Causal Analysis, 12th International Conference on Product Focused Software Development and Process Improvement (PROFES 2011), LNCS, Bari, Italy, 2011.

Além destes resultados, existe um artigo adicional definido relatando as contribuições desta pesquisa:

- Título: “DPPI: Improving Defect Causal Analysis with Automated Cause-Effect Learning Mechanisms and Probabilistic Cause-Effect Diagrams”
  - Tema: Descrição da abordagem DPPI, de sua automação e dos resultados obtidos nos estudos experimentais.
  - A ser submetido para o *Journal on Information and Software Technology*.

## 1.7 Organização da Tese

Este capítulo apresentou o contexto, a motivação, as questões de pesquisa, os objetivos e a metodologia desta tese. A organização do restante do texto da tese segue a estrutura de seus objetivos específicos. Com base nesta organização, a estrutura de capítulos é a seguinte:

- **Capítulo 2 – Revisão Sistemática sobre Análise Causal de Defeitos de Software:** apresenta a revisão sistemática da literatura, conduzida visando ajudar a compreender o estado da arte a respeito de análise causal de defeitos de software.

- **Capítulo 3 – Diretrizes para Implementar Análise Causal de Defeitos em Organizações de Software:** descreve a compilação de diretrizes para apoiar a implementação de análise causal de defeitos, elaboradas com base em resultados obtidos da revisão sistemática.
- **Capítulo 4 – Abordagem para Prevenção de Defeitos Provenientes de Inspeções para Apoiar a Melhoria dos Processos de Engenharia do Software:** apresenta a abordagem de análise causal de defeitos provenientes de inspeções para apoiar a melhoria dos processos de engenharia do software, chamada DPPI (*Defect Prevention-based Process Improvement*). DPPI fornece um detalhamento das atividades a serem realizadas, com base no conhecimento obtido na revisão sistemática (capítulo 2) e nas diretrizes (capítulo 3). Adicionalmente, a oportunidade de pesquisa identificada na revisão sistemática foi tratada e DPPI propõe uma forma inovadora de realizar análise causal, integrando um mecanismo de aprendizado de relações de causa e efeito (redes Bayesianas) para apoiar na identificação das causas.
- **Capítulo 5 – Prova de Conceito e Apoio Ferramental:** descreve a experiência de aplicar DPPI a um projeto de desenvolvimento de software real e o apoio ferramental construído para facilitar sua aplicação e para automatizar o seu ciclo de retroalimentação a respeito de relações causais.
- **Capítulo 6 – Avaliação da Abordagem de DPPI para a Identificação de Causas de Defeitos de Software:** descreve o estudo experimental para avaliar se o uso da abordagem DPPI traz benefícios para a identificação de causas de defeitos de software provenientes de inspeções, quando comparada à identificação das causas de forma *ad hoc*.
- **Capítulo 7 – Considerações Finais:** apresenta as considerações finais sobre esta pesquisa, seus resultados obtidos, as limitações do trabalho e futuros direcionamentos para a continuidade da pesquisa.

Além disto, a tese contém 6 anexos:

- **ANEXO A – Listagem de Artigos da Execução do Protocolo da Revisão Sistemática:** apresenta as listagens de artigos obtidos nas diferentes execuções do protocolo da revisão sistemática.
- **ANEXO B – Informações Extraídas dos Artigos Recuperados:** apresenta as informações extraídas dos artigos selecionados como parte da revisão sistemática.



- **ANEXO C – Sumário das Informações Extraídas dos Artigos Recuperados:** apresenta tabelas que resumem as informações extraídas dos artigos recuperados durante as diferentes execuções da revisão sistemática.
- **ANEXO D – Template para DPPI:** apresenta um template que pode ser utilizado para apoiar a aplicação manual da abordagem.
- **ANEXO E – A Ferramenta DPPI Framework:** apresenta resumidamente os requisitos, o projeto e as funcionalidades da ferramenta DPPI Framework, construída para apoiar o uso da abordagem DPPI.
- **ANEXO F – Instrumentos do Estudo Experimental:** apresenta os instrumentos utilizados na condução do estudo experimental realizado para avaliar o uso da abordagem de DPPI para a identificação de causas de defeitos.

## **CAPÍTULO 2 - REVISÃO SISTEMÁTICA SOBRE ANÁLISE CAUSAL DE DEFEITOS DE SOFTWARE**

*Neste capítulo será apresentada a revisão sistemática da literatura, conduzida visando ajudar a compreender o estado da arte a respeito de análise causal de defeitos de software.*

### **2.1 Introdução**

Uma revisão sistemática é uma maneira de identificar, avaliar e interpretar de forma imparcial e justa pesquisas relevantes para questões de pesquisa (Kitchenham, 2004) (Kitchenham *et al.*, 2004). Desta forma, revisões sistemáticas representam um instrumento para prover a fundamentação para posicionar apropriadamente novas atividades de pesquisa (Kitchenham e Charters, 2007). Revisões sistemáticas tendem a ser mais confiáveis por utilizarem uma metodologia rigorosa que pode ser auditada e repetida (Kitchenham, 2004).

No contexto deste trabalho, uma revisão sistemática foi conduzida para identificar o estado da arte a respeito de análise causal de defeitos de software e para prover diretrizes sobre como implementar análise causal de defeitos de forma eficiente em organizações de software. Um protocolo de revisão, focando nas questões de pesquisa foi desenvolvido para guiar a revisão da literatura de acordo com o processo de revisão sistemática. Este processo envolve três atividades principais: planejamento, execução e análise dos resultados (Biolchini *et al.*, 2005) (Kitchenham e Charters, 2007). Nas próximas seções um resumo de como estas atividades foram conduzidas nesta pesquisa é fornecido.

### **2.2 Planejamento da Revisão Sistemática**

O objetivo da revisão sistemática foi conduzir uma revisão imparcial e justa a respeito do estado da arte de análise causal de defeitos de software. Este objetivo pôde ser desmembrado nos seguintes objetivos específicos:

- Resumir os processos, abordagens e diretrizes (conhecimento) que têm sido propostos para análise causal de defeitos de software.
- Adicionalmente, resumir e analisar os esquemas de classificação dos defeitos e das causas utilizados por estes processos e abordagens. É importante deixar claro que este objetivo não trata de analisar todos os

esquemas de classificação existentes para defeitos e causas de defeitos, mas focar naqueles provenientes de fontes relacionadas a análise causal de defeitos de software.

Duas questões de pesquisa, Q0 e Q1, foram formuladas a partir destes objetivos. A questão Q0 compreende um escopo mais amplo, que é analisar causas de defeitos. A questão Q1, por sua vez, está relacionada com análise causal de defeitos de software (incluindo a prevenção destes defeitos em projetos futuros). Esta estratégia foi adotada para que conhecimento genérico a respeito da análise de causas de defeitos pudesse também ser identificado, uma vez que este conhecimento pode ser utilizado como ponto de partida rumo à análise causal de defeitos (incluindo prevenção). A descrição das questões de pesquisa, no formato sugerido em (Biolchini *et al.*, 2005), segue:

- Q0: Que processos, abordagens e conhecimentos têm sido propostos e/ou utilizados para analisar causas de defeitos de software?
  - (P) População (*Population*): publicações científicas e relatos de experiência de projetos, ambientes ou organizações de desenvolvimento de software.
  - (I) Intervenção (*Intervention*): processos, abordagens e conhecimento para analisar causas de defeitos.
  - (C) Comparação (*Comparison*): não existe.
  - (O) Saída (*Output*): Identificação de processos, abordagens e conhecimento para analisar causas de defeitos de software.
- Q1: Que processos, abordagens e conhecimentos têm sido propostos e/ou utilizados para análise causal de defeitos de software (ou prevenção de defeitos)?
  - (P) População (*Population*): publicações científicas e relatos de experiência de projetos, ambientes e organizações de desenvolvimento de software.
  - (I) Intervenção (*Intervention*): processos, abordagens e conhecimentos para análise causal de defeitos de software (ou prevenção de defeitos).
  - (C) Comparação (*Comparison*): não existe.
  - (O) Saída (*Output*): Identificação de processos, abordagens e conhecimentos para análise causal de defeitos de software (ou prevenção de defeitos).

A partir destas questões de pesquisa *strings* de busca puderam ser derivadas e ajustadas para que pudessem ser executadas em diferentes bibliotecas digitais. A

seguir encontra-se a string de busca S0, derivada para a questão Q0 incluindo as palavras chave identificadas para a estrutura (P) *and* (I) *and* (C) *and* (O).

- S0: (P) **and** (I) **and** (O)
  - (“software development” or “software project” or “software process” or “software engineering” or “software organization” or “software environment” or “experience factory” or “software factory”) **and** (“causal analysis” or “cause analysis” or “defect analysis” or “error analysis” or “failure analysis” or “fault analysis”) *and* (defect or error or failure or fault) *and* (causal or cause)) **and** (process or approach or method or methodology or technique or knowledge or tool or paradigm or strategy).

As bibliotecas digitais escolhidas inicialmente foram: ACM Digital Library, EI Compendex, IEEE, Inspec e Web of Science. A *string* de busca foi avaliada contra um conjunto de sete artigos de controle a respeito de análise causal de defeitos de software, lidos antes da revisão sistemática. A *string* foi capaz de recuperar das bibliotecas digitais cinco destes artigos, além de diversos outros artigos de análise causal de defeitos. Os dois artigos de controle que não foram recuperados satisfaziam o critério de busca da *string*, entretanto, não foram recuperados porque um deles não estava indexado nas bibliotecas digitais pesquisadas e o outro provavelmente estava indexado de forma inapropriada. Portanto, a *string* parecia servir para seu propósito.

O critério para incluir um artigo recuperado na revisão era que ele deveria conter processos, abordagens ou conhecimento a respeito de analisar causas de defeitos ou análise causal de defeitos (prevenção de defeitos). Para cada um dos artigos selecionados as seguintes informações foram extraídas: título, referência completa, fonte, esquema de classificação de defeitos, esquema de classificação de causas, identificação do processo ou abordagem, descrição do processo ou abordagem, identificação de conhecimento, descrição do conhecimento e tipo de estudo.

Conforme sugerido em (Kitchenham, 2004) e (Kitchenham e Charters, 2007) o protocolo da revisão descrito nesta seção foi revisto por outros pesquisadores antes de sua execução.

## 2.3 Execução da Revisão Sistemática

Com o protocolo da revisão estabelecido, a execução da revisão sistemática pôde ser iniciada. Ela foi planejada para que pudessem ocorrer repetições periódicas de atualização. As subseções que seguem apresentam um resumo das execuções

realizadas em Agosto de 2006, Setembro de 2007, Janeiro de 2009 e Julho de 2010. A subseção 2.3.5 contém uma consideração sobre as bibliotecas digitais escolhidas e descreve o resumo da execução em uma biblioteca digital adicional, a SCOPUS.

### 2.3.1 Execução de Agosto de 2006

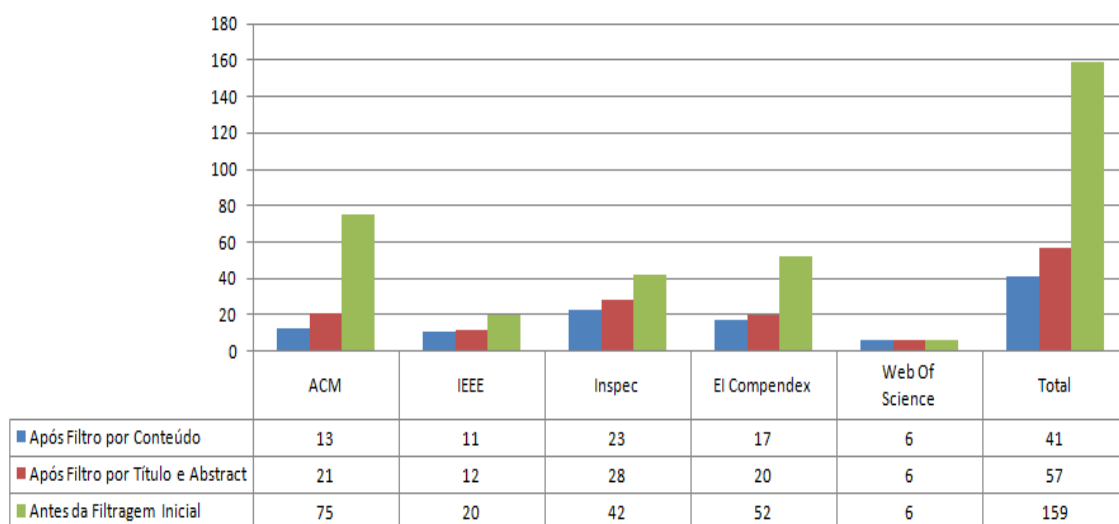
A string de busca foi executada nas bibliotecas digitais e 203 artigos foram retornados. Após eliminar artigos replicados (o mesmo artigo retornado por mais de uma biblioteca digital) o número total de artigos caiu para 159.

Os 159 artigos foram então filtrados. O filtro inicial foi baseado no título e no *abstract*. Apenas os artigos que certamente não estavam relacionados com os objetivos da pesquisa foram eliminados. Como resultado o número de artigos para ler e avaliar reduziu para 57.

Após a leitura e tentativa de extrair as informações dos artigos, alguns artigos adicionais foram excluídos, baseado no critério definido anteriormente. Finalmente, 41 artigos foram selecionados como parte da primeira execução da revisão sistemática.

A lista dos 159 artigos recuperados das bibliotecas digitais, destacando os 57 obtidos pelo filtro por título e abstract e os 41 efetivamente selecionados encontra-se no Anexo A1.

O número de artigos retornados de cada biblioteca digital durante o processo de filtragem pode ser visto na Figura 2.1.



**Figura 2.1. Artigos Recuperados das Bibliotecas Digitais em Agosto de 2006.**

Além dos 41 artigos selecionados pela aplicação do protocolo da revisão sistemática, foram extraídas informações dos seguintes três artigos:

- Card, D., "Defect Causal Analysis Drives Down Error Rates", IEEE Software 1/1993, Volume 10, Issue 4, 7/1993, p.98-99, 1993.

- Card, D., “Defect Analysis: Basic Techniques for Management and Learning”, Advances in Computers, vol. 65, chapter 7, pp. 259-295, 2005.
- Endres, A., “An Analysis of Errors and Their Causes in Systems Programs”, IEEE Transactions on Software Engineering, SE-1, 2, June 1975, pp. 140-149.

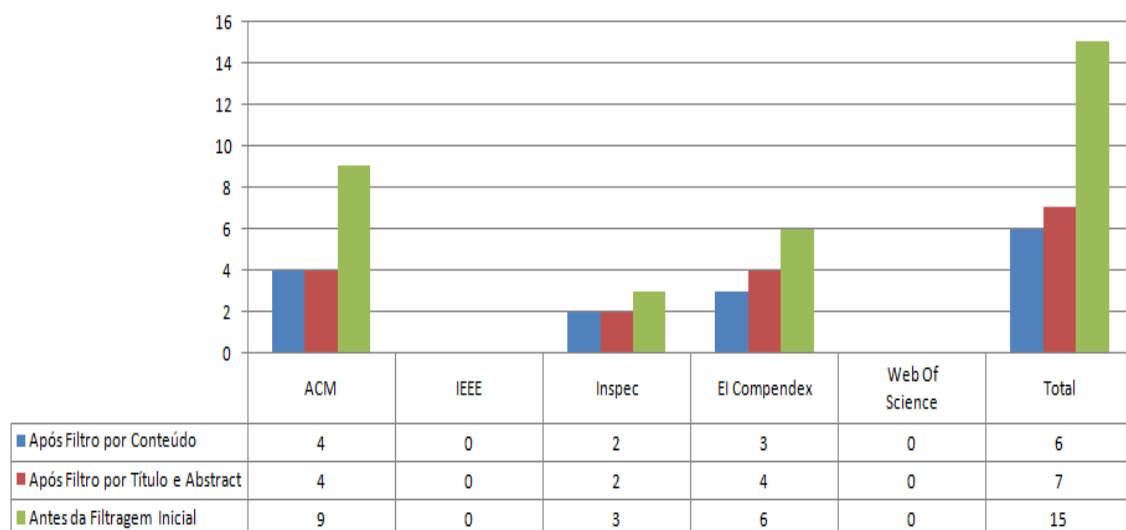
Conforme mencionado anteriormente, os primeiros dois destes artigos faziam parte do controle e não foram recuperados respectivamente por uma falha de indexação ou por não estarem indexados nas bases utilizadas. O artigo (Endres, 1975), por sua vez, foi acrescentado por ter sido referenciado por alguns dos demais artigos analisados como um dos artigos seminais na área. Novamente trata-se de uma falha de indexação na máquina de busca da IEEE, uma vez que se trata de um artigo antigo que contém as palavras chave utilizadas no protocolo.

### 2.3.2 Execução de Setembro de 2007

Em 2007 o mesmo processo de filtragem foi realizado. Como resultado 15 novos artigos foram identificados. Após filtragem por título e *abstract* o número de artigos novos reduziu para 7. Finalmente, após filtragem por conteúdo o número caiu para 6.

A listagem destes 15 artigos novos, destacando os 7 filtrados por título e abstract e os 6 efetivamente selecionados encontra-se no Anexo A2.

A quantidade de artigos novos de cada biblioteca digital durante o processo de filtragem pode ser vista na Figura 2.2.



**Figura 2.2. Artigos Adicionais Recuperados das Bibliotecas Digitais em Setembro de 2007.**

A maneira como os artigos se referenciam, considerando os 50 artigos obtidos nas execuções de 2006 e 2007, é mostrada na Figura 2.3. Esta representação gráfica auxilia na identificação dos artigos mais influentes e de possíveis tendências de pesquisa. Os artigos citados por mais de três dos demais artigos são mostrados em cinza claro. Os artigos adicionais recuperados na execução de 2007 são mostrados em cinza.

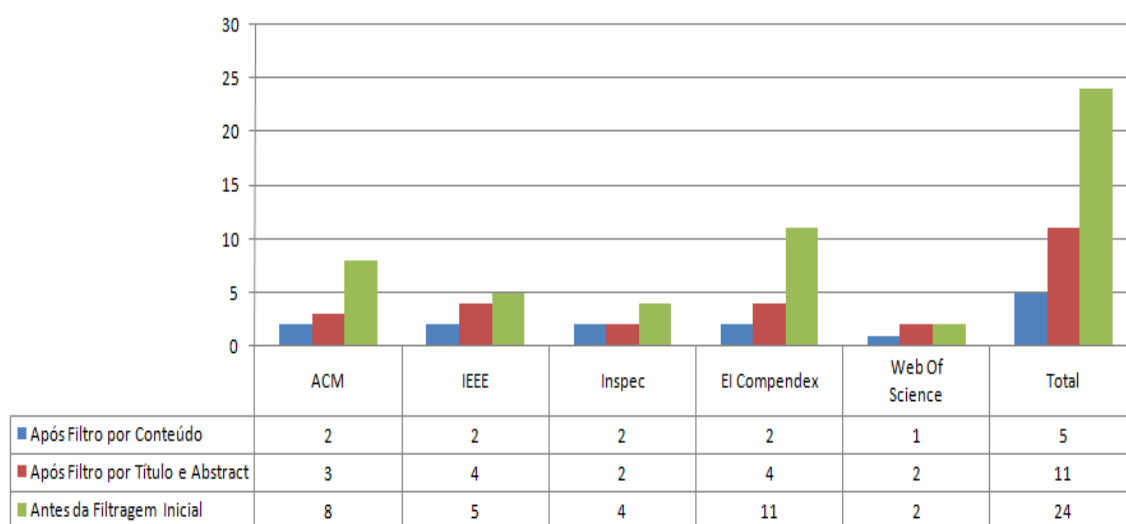




### 2.3.3 Execução de Janeiro de 2009

Após a execução de Setembro de 2007, o mesmo processo de filtragem foi realizado em Janeiro de 2009. Como resultado 24 novos artigos foram identificados. Após filtragem por título e *abstract* o número de artigos novos reduziu para 11. Finalmente, após filtragem por conteúdo o número caiu para 5. A quantidade de artigos novos de cada biblioteca digital durante o processo de filtragem encontra-se na Figura 2.4.

A listagem destes 24 artigos novos, destacando os 11 filtrados por título e *abstract* e os 5 efetivamente selecionados encontra-se no Anexo A3.



**Figura 2.4. Artigos Adicionais Recuperados das Bibliotecas Digitais em Janeiro de 2009.**

A maneira como os artigos se referenciam, considerando os 55 artigos obtidos nas execuções de Agosto de 2006, Setembro de 2007 e Janeiro de 2009, é mostrada na Figura 2.5. Novamente os artigos citados por mais de três dos demais artigos são mostrados em cinza claro, um novo artigo foi incluído neste conjunto: (Leszak *et al.*, 2002). Os artigos adicionais retornados na revisão de Janeiro de 2009 se encontram destacados em cinza. Nesta representação as referências do artigo (Kalinowski *et al.*, 2008b) não foram incluídas, uma vez que o artigo utilizou a revisão sistemática como base e, por consequência, suas referências não reforçam a influência dos artigos, no contexto da própria revisão sistemática.

As informações extraídas de cada um dos artigos das revisões de Agosto de 2006, Setembro de 2007 e Janeiro de 2009 estão registradas respectivamente nos Anexos B1, B2 e B3.

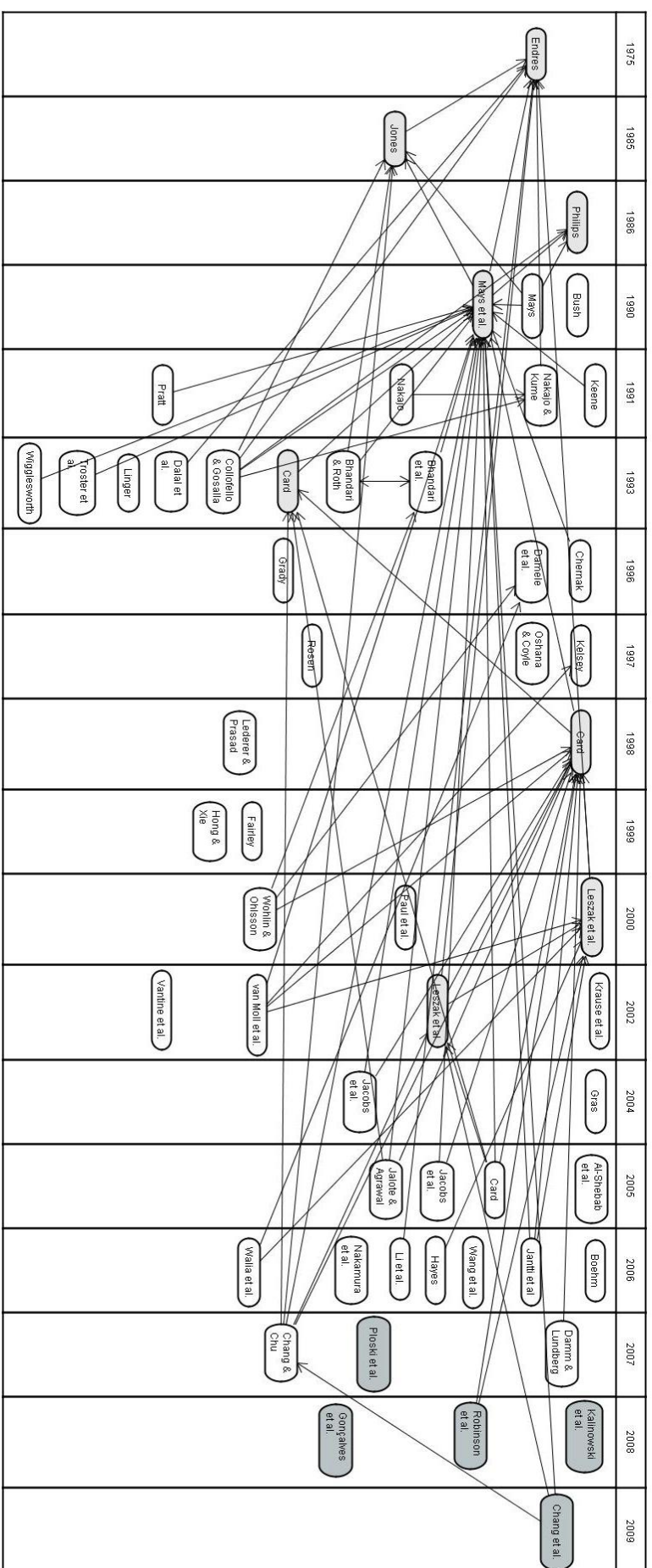
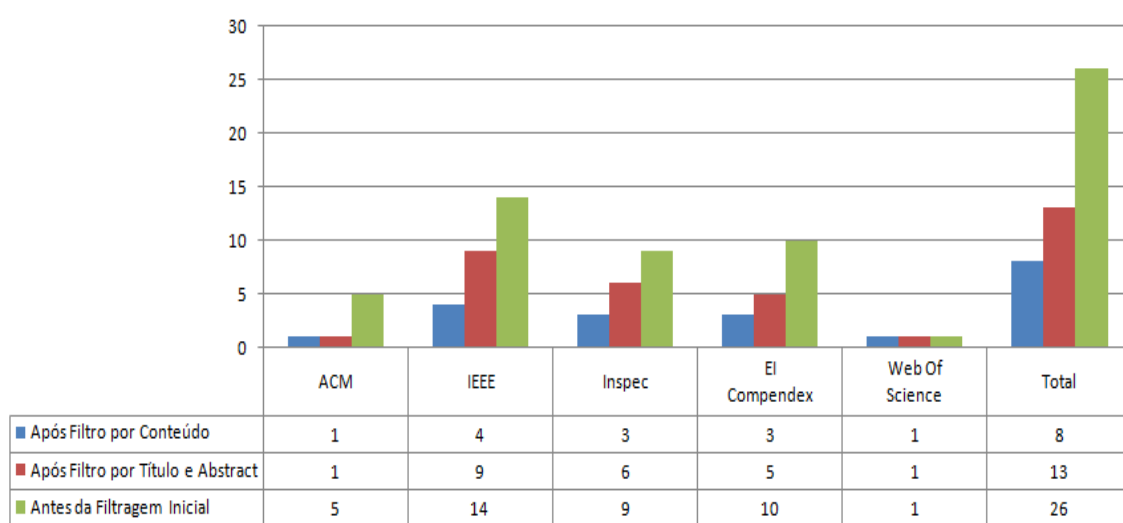


Figura 2.5. Referências entre os Artigos Selecionados nas Revisões de Agosto de 2006, Setembro de 2007 e Janeiro de 2009.

### 2.3.4 Execução de Julho de 2010

Após a execução de Janeiro de 2009, o mesmo processo de filtragem foi realizado em Julho de 2010. Como resultado 26 novos artigos foram identificados. Após filtragem por título e *abstract* o número de artigos novos reduziu para 13. Finalmente, após filtragem por conteúdo o número caiu para 8. A quantidade de artigos novos de cada biblioteca digital durante o processo de filtragem encontra-se na Figura 2.6.

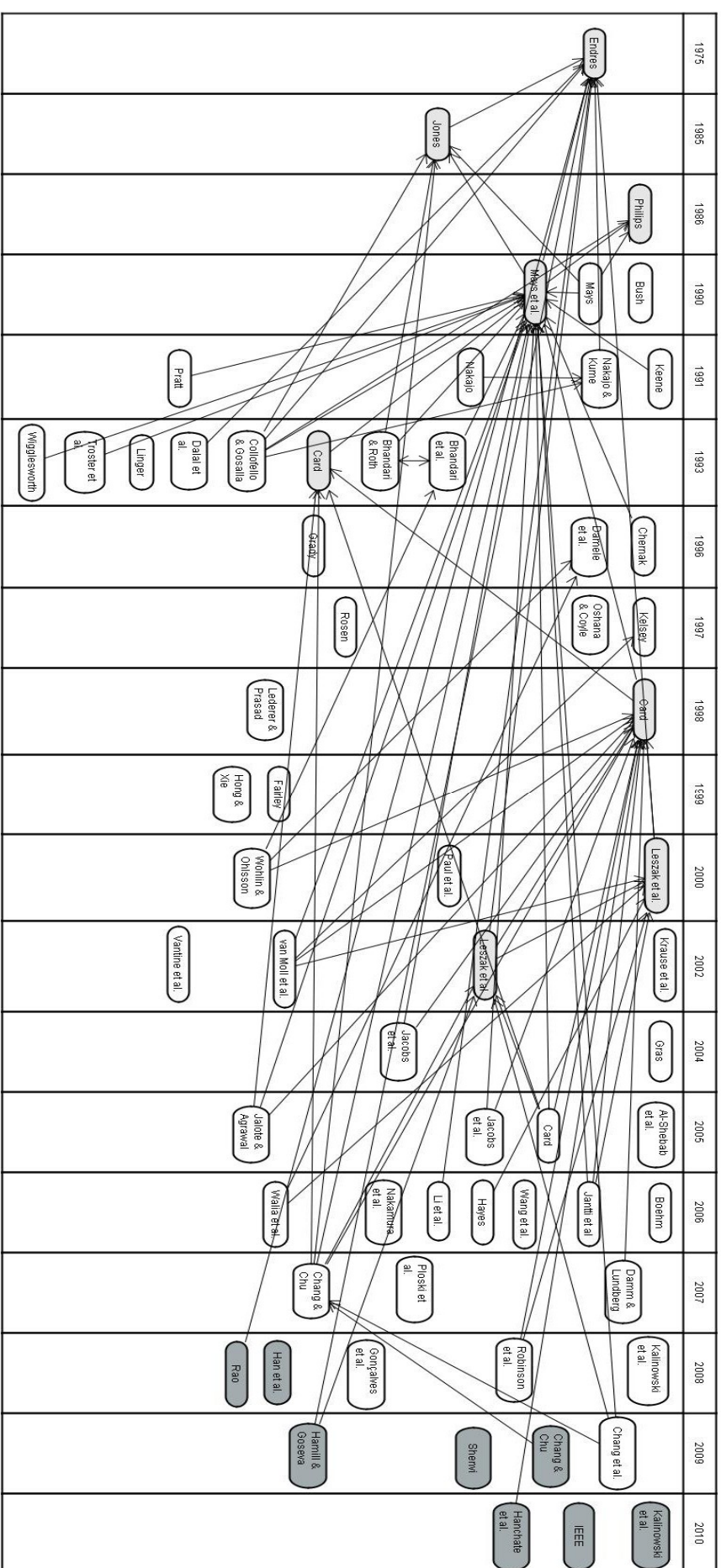
A listagem destes 26 artigos novos, destacando os 13 filtrados por título e *abstract* e os 8 efetivamente selecionados encontra-se no Anexo A4.



**Figura 2.6. Artigos Adicionais Recuperados das Bibliotecas Digitais em Julho de 2010.**

A maneira como os artigos se referenciam, considerando os 63 artigos obtidos nas execuções de Agosto de 2006, Setembro de 2007, Janeiro de 2009 e Julho de 2010, é mostrada na Figura 2.7. Novamente os artigos citados por mais de três dos demais artigos são mostrados em cinza claro, um novo artigo foi incluído neste conjunto: (Leszak *et al.*, 2002). Os artigos adicionais retornados na revisão de Julho de 2010 se encontram destacados em cinza. Nesta representação as referências dos artigos (Kalinowski *et al.*, 2008b) e (Kalinowski *et al.*, 2010) não foram incluídas, uma vez que os artigos utilizaram a revisão sistemática como base e, por consequência, suas referências não reforçam a influência dos artigos, no contexto da própria revisão sistemática.

As informações extraídas de cada um dos artigos das revisões de Agosto de 2006, Setembro de 2007, Janeiro de 2009 e Julho de 2010 estão registradas respectivamente nos Anexos B1, B2, B3 e B4.



**Figura 2.7. Referências entre os Artigos Selecionados nas Revisões de Agosto de 2006, Setembro de 2007, Janeiro de 2009 e Julho de 2010.**

### 2.3.5 Sobre as Bibliotecas Digitais Escolhidas e Execução Adicional na SCOPUS

Durante o período de quatro anos de execuções do protocolo da revisão sistemática, outras revisões sistemáticas foram realizadas em diferentes temas de pesquisa e por diferentes autores (Kitchenham e Charters, 2007). De fato, nem todas estas revisões utilizaram as mesmas bibliotecas digitais. Em uma tentativa de realizar uma busca exaustiva, Brereton *et al.* (2007) identificaram sete bibliotecas digitais de relevância para engenheiros de software:

- IEEEExplore;
- ACM Digital Library;
- Google Scholar;
- Citeseer Library;
- INSPEC;
- Science Direct; e
- EI Compendex.

Estas bibliotecas diferem das utilizadas inicialmente nesta pesquisa (IEEEExplore, ACM Digital Library, INSPEC, EI Compendex e Web of Science). A biblioteca Google Scholar não foi utilizada porque sua atualização com critérios desconhecidos dificulta auditar os resultados das pesquisas e de repetir a execução do protocolo, premissas básicas da execução de revisões sistemáticas (Kitchenham, 2004). A biblioteca Citeseer não tem sido utilizada nas revisões sistemáticas do grupo ESE por aparentes deficiências em sua atualização e por indexar somente artigos de conferências. Por fim, a biblioteca Science Direct é específica da Springer e artigos da Springer são também indexados em outras bases de pesquisa, como a INSPEC, EI Compendex e a Web of Science (esta última não utilizada por Brereton *et al.* (2007)). De fato, cada uma das cinco bibliotecas utilizadas teve contribuições significativas para a revisão, retornando artigos que não haviam sido retornados pelas demais bases.

Kitchenham e Charters (2007) mencionam que além das bibliotecas utilizadas por Brereton *et al.* (2007) pode ser interessante considerar a SCOPUS, que afirma ser a maior base de dados de indexação de resumos e citações. Tendo isto em vista, após as quatro execuções do protocolo nas cinco bases definidas inicialmente, em Agosto de 2010 o protocolo foi executado também na biblioteca SCOPUS.

A execução da string na biblioteca SCOPUS retornou 262 artigos. De fato, a SCOPUS sozinha retornou mais artigos do que as cinco bibliotecas anteriormente

utilizadas juntas (224 artigos). Entretanto, como já era de se esperar, uma vez que a SCOPUS só indexa abstracts e não o artigo em si e ao mesmo tempo indexa muitos abstracts (afirma ser a maior base de abstracts), muitos destes artigos eram de fato falso positivos. Dos 262 artigos, somente 59 haviam sido analisados anteriormente. Dos 203 artigos restantes, através do filtro por título e abstract, apenas 12 artigos foram considerados candidatos a inclusão. Destes 12 artigos 9 atendiam ao critério de inclusão. Assim, informações foram extraídas de 9 artigos adicionais (1 de 1999, 1 de 2002, 2 de 2005, 1 de 2006, 1 de 2007, 2 de 2008 e 1 de 2009). Incluindo estes 9 artigos o número total de artigos incluídos considerando todas as bibliotecas utilizadas passou para 72.

A listagem de todos os artigos retornados pela aplicação da string de busca na SCOPUS, destacando todos desta listagem que foram incluídos na revisão sistemática, pode ser encontrada no Anexo A5. As informações extraídas dos 9 artigos novos incluídos podem ser encontradas no Anexo B5.

Ao todo dos 262 artigos retornados pela SCOPUS apenas 37 foram incluídos. Desta forma, embora a SCOPUS provavelmente seja de fato a biblioteca com o maior número de abstracts e citações indexadas, realizar a revisão somente na SCOPUS pode não ser suficiente.

Tendo utilizado ao todo seis bibliotecas digitais e executado o protocolo da revisão em diferentes anos, os artigos desta revisão foram incluídos de maneira mais imparcial e justa do que em revisões bibliográficas informais, uma vez que sua inclusão se deu em função de sua aderência à própria questão de pesquisa e não buscando artigos de grupos de pesquisa ou autores específicos.

Adicionalmente a importância de outras fontes para análise causal de defeitos de software, não indexadas em bibliotecas digitais, como livros detalhando abordagens como Lean (Poppendieck e Poppendick, 2007) e Six Sigma (Eckes, 2001), normas como a ISO 12207 (ISO/IEC, 2008a) e a ISO 15504 (ISO/IEC, 2008b) e modelos de referência como o CMMI (SEI, 2006a) e o MR-MPS (SOFTTEX, 2009) é reconhecida. Por este motivo, estas fontes adicionais serão utilizadas ao longo desta tese como complementando aos achados da revisão sistemática.

## **2.4 Resultados da Revisão Sistemática**

As seguintes informações, referentes aos objetivos da revisão sistemática, foram extraídas e agrupadas em tabelas, organizadas por data de publicação:

- Processos e abordagens utilizadas para análise causal de defeitos (veja Anexo C1);

- esquemas de classificação de defeitos utilizados nos artigos (veja Anexo C2);
- esquemas de classificação de causas utilizados nos artigos (veja Anexo C3), e;
- outros conhecimentos a respeito de análise causal de defeitos gerado ou citado pelos artigos (veja Anexo C4).

Nas próximas subseções um resumo das análises que puderam ser realizadas com base nestas tabelas é apresentado.

#### **2.4.1 Processos e Abordagens para Análise Causal de Defeitos**

A listagem completa de processos e abordagens encontradas nos artigos encontra-se no Anexo C1. A primeira abordagem encontrada foi a descrita por Endres (1975), na IBM. Esta abordagem lida com a análise individual de defeitos de software, de modo que possam ser categorizados e suas causas identificadas, permitindo a tomada de ações para prevenir sua ocorrência em projetos futuros ou ao menos assegurar sua detecção nestes projetos. A análise dos defeitos nesta abordagem ocorre ocasionalmente, assim como as ações corretivas. Esta abordagem é ainda referenciada em artigos recentes que tratam de análise retrospectiva de defeitos, como (Li *et al.*, 2006) e (Pan *et al.*, 2009).

A diferença entre esta abordagem e o processo de prevenção de defeitos, também estabelecido na IBM (Jones, 1985) (Philips, 1986) (Mays, 1990) (Mays *et al.*, 1990), cerca de dez anos depois, é que o último é integrado ao processo de desenvolvimento. Esta integração ocorre ao estabelecer duas atividades adicionais: (1) uma reunião de análise causal após a atividade de retrabalho (momento em que os defeitos já foram corrigidos) e (2) uma reunião de *kickoff*, onde as ações implementadas desde a última reunião de análise causal são comunicadas ao pessoal envolvido nas atividades de desenvolvimento.

A abordagem apresentada por Endres (1975) e o processo de prevenção de defeitos (Jones, 1985) analisam defeitos exaustivamente e envolvem na análise, se possível, as pessoas responsáveis pela introdução dos defeitos. Quando apropriado, o líder da reunião pode solicitar que alguns defeitos não sejam discutidos pela equipe.

Card (1993) descreve o processo de análise causal de defeitos de software de forma um pouco mais flexível do que o processo de prevenção de defeitos, mas envolvendo os mesmos conceitos. O foco do processo descrito por Card (1993) está na reunião de análise causal. Assim, a análise causal de defeitos de software pode ser

considerada parte do processo de prevenção de defeitos, que endereça também a implementação das ações e a comunicação das mudanças implementadas à equipe de desenvolvimento. A aparente flexibilidade vem do fato de a reunião de análise causal não ser vinculada a uma atividade específica de desenvolvimento, ocorrendo periodicamente. Card (1993) também destaca, com base na experiência da Computer Science Corporation, que não é necessário analisar os defeitos exaustivamente, sendo uma amostra suficiente e que as propostas de ação devem lidar com classes de defeitos e não com defeitos individuais.

Grady (1996) descreve experiências da Hewlett Packard, e cita diferentes periodicidades de aplicação de análise causal de defeitos: análise causal avulsa (realizada eventualmente), análise causal após projeto (realizada ao fim de cada projeto) e ciclo de melhoria contínua de processos (realizada após cada fase de desenvolvimento).

Muitos dos artigos analisados seguem processos congruentes ao processo de prevenção de defeitos (Jones, 1985) ou o de análise causal de defeitos (Card, 1993). Entre os artigos que seguem ou sugerem estes processos estão (Jones, 1985), (Philips, 1986), (Mays, 1990), (Mays *et al.*, 1990), (Pratt, 1991), (Card, 1993), (Collofello e Gosalla, 1993), (Damele *et al.*, 1996), (Grady, 1996), (Card, 1998), (Leszak *et al.*, 2000), (Leszak *et al.*, 2002), (van Moll *et al.*, 2002), (Jalote e Agrawal, 2005), (Card, 2005) e (Shenvi, 2009).

Além das abordagens de análise causal de defeitos, algumas outras abordagens relacionadas com o processo de análise causal de defeitos ou apoiando algumas de suas atividades, puderam ser identificadas. Embora não sejam abordagens de análise causal de defeitos, as consideramos em nossa revisão. Elas representam técnicas para identificar relações de causa e efeito: (Nakajo, 1991), (Nakajo e Kume, 1991), (Wohlin *et al.*, 2000b), (Krause *et al.*, 2002), (Gras, 2004) e (Han *et al.*, 2008); técnicas de controle estatístico: (Hong *et al.*, 1999), (Jalote e Saxena, 2002) e (Wang *et al.*, 2006); e técnicas de análise de defeitos: (Bhandari *et al.*, 1993), (Bhandari e Roth, 1993), (Kelsey, 1997), (Leszak, 2005), (Damm *et al.*, 2006), (Nakamura *et al.*, 2006), (Damm e Lundberg, 2007) e (Damm *et al.*, 2008). Um resumo destas técnicas se encontra no Anexo B, juntamente com a extração das informações dos artigos.

Baseado na revisão da literatura, duas técnicas que se mostraram particularmente úteis para apoiar atividades de análise causal de defeitos são os gráficos de Pareto e os diagramas de causa e efeito (ou diagramas de Ishikawa) (Ishikawa, 1976). Elas apóiam, respectivamente, a identificação das classes de defeitos mais comuns e encontrar as causas para classes específicas de defeitos.



Embora a utilidade dos diagramas de causa e efeito na área de manufatura já tenha sido citada por Mays *et al* (1990), os primeiros artigos analisados a utilizar estes diagramas no contexto de análise causal de defeitos de software datam de 1996. O diagrama de causa e efeito é utilizado e/ou sugerido em muitos dos artigos analisados, incluindo (Chernak, 1996), (Damele *et al.*, 1996), (Grady, 1996), (Card, 1998), (van Moll *et al.*, 2002), (Jacobs *et al.*, 2004), (Card, 2005), (Jacobs *et al.*, 2005), (Jalote e Agrawal, 2005), (Wang *et al.*, 2006), (Gonçalves *et al.*, 2008). Entre os artigos que citam gráficos de Pareto como uma técnica útil estão (Card, 1998), (Hong *et al.*, 1999), (van Moll *et al.*, 2002), (Card, 2005) (Jalote and Agrawal, 2005), (Wang *et al.*, 2006) e (Shenvi, 2009).

## **2.4.2 Esquemas de Classificação de Defeitos**

Dois tipos de informações sobre esquemas de classificação de defeitos foram extraídas dos artigos: a informação a ser coletada a respeito de defeitos e as taxonomias para descrever tipos de defeitos.

### **2.4.2.1 Informações Coletadas**

Foi possível observar um consenso entre os autores a respeito das informações relevantes a serem coletadas a sobre de defeitos de software de modo a apoiar a análise causal de defeitos. Além da descrição dos defeitos, entre os dados considerados relevantes desde os artigos iniciais (Endres, 1975) (Jones, 1985) (Mays, 1990) até artigos mais recentes (Agrawal e Jalote, 2005) (Jantti *et al.*, 2006) (Damm *et al.*, 2006) (Chang e Chu, 2007) (Damm e Lundberg, 2007) (Jalote *et al.*, 2007) (Chang e Chu, 2008) (Damm *et al.*, 2008) (Chang e Chu, 2009) (Chang *et al.*, 2009) estão:

- O momento (ou fase) em que o defeito foi introduzido.
- O momento (ou fase) em que o defeito foi detectado.
- Tipo de defeito.

Estas informações correspondem às três dimensões para classificar defeitos sugeridas em (Card, 1998) e (Card, 2005). A maioria dos esquemas de classificação de defeitos analisados em (Ploski *et al.*, 2007) conta com estas dimensões. Nenhum dos artigos analisados argumentou contra o uso destas dimensões para classificar defeitos. Adicionalmente, estas informações fazem parte dos atributos de defeitos sugeridos no recente padrão IEEE para classificar anomalias de software (IEEE, 2010).

Entretanto, outras informações foram consideradas em alguns artigos, de acordo com os objetivos específicos das abordagens de análise causal de defeitos neles apresentadas:

- Esforço de correção, severidade ou impacto, é considerado em diversos artigos, como, por exemplo, em (Endres, 1975), (Collofello e Gosalla, 1993), (Fairley, 1999), (Nakashima *et al.*, 1999), (Leszak *et al.*, 2000), (Leszak *et al.*, 2002), (Leszak, 2005), (Jantti *et al.*, 2006), (Nakamura *et al.*, 2006), (Chang e Chu, 2007), (Chang e Chu, 2009) (Chang *et al.*, 2009). Adicionalmente, severidade é também considerada em artigos que utilizam o esquema de classificação da IBM (Chillarege *et al.*, 1992), ODC (*Othogonal Defect Classification*): (Bhandari *et al.*, 1993), (Chernak, 1996) e (Shenvi, 2009).
- Localização (ou módulo) é considerada em muitos artigos, tais como (Endres, 1975), (Bhandari *et al.*, 1993), (Bhandari e Roth, 1993), (Chernak, 1996), (Kelsey, 1997), (Leszak *et al.*, 2000), (Leszak *et al.*, 2002), (van Moll *et al.*, 2002), (Leszak, 2005), (Nakamura *et al.*, 2006), (Chang e Chu, 2007), (Chang e Chu, 2008), (Chang e Chu, 2009), (Chang *et al.*, 2009) e (Shenvi, 2009).
- Informações adicionais encontradas nos artigos que utilizam o esquema de classificação ODC (Bhandari *et al.*, 1993) (Chernak, 1996) e (Shenvi, 2009):
  - *Trigger*, que indica como o defeito foi detectado.
  - Se o defeito foi introduzido durante o desenvolvimento de funcionalidades novas ou durante a manutenção de funcionalidade pré-existente.

Estas são as informações que tem se mostrado úteis nos artigos para apoiar análise causal de defeitos, um esquema de classificação mais completo para servir também a outros propósitos da engenharia de software pode ser encontrado em ODC (Chillarege *et al.*, 1992) e no padrão IEEE para classificar anomalias de software (IEEE, 2010).

#### **2.4.2.2 Tipos de Defeitos**

A respeito dos tipos de defeito, diversas taxonomias puderam ser encontradas. A lista completa de taxonomias encontra-se no Anexo C2. Um resumo desta lista, descrevendo quatro entre as taxonomias encontradas nos artigos segue:

- Taxonomia do Padrão IEEE (IEEE, 2010). Contendo os seguintes tipos de defeito: dados, interface, lógica, descrição, sintaxe, padrões, e outros. Este

padrão ainda complementa os tipos com a natureza (*mode*) do defeito, que pode assumir os seguintes valores: incorreto, omitido, e incluído sem ser necessário.

- Taxonomia de ODC (Chillarege *et al.*, 1992), contendo os seguintes tipos de defeitos: interface, funcionalidade, empacotamento (*build/package/merge*), inicialização (atribuição), documentação, verificação (*checking*), algoritmo e temporal (*timing/serialization*). Entre os artigos analisados, esta taxonomia foi utilizada por (Bhandari *et al.*, 1993), (Chernak, 1996) e (Shenvi, 2009). Ela foi ainda mencionada em alguns outros artigos, como (Card, 1998) e (Card, 2005). Adicionalmente, Ploski *et al.* (2007) destacam ODC como um esquema de classificação pouco ambíguo e de fácil utilização. Além dos tipos de defeito, ODC considera ainda a natureza (*qualifier*) do defeito, de maneira consistente com o padrão IEEE, podendo assumir os mesmos valores (errado, omitido e incluído sem ser necessário).
- Taxonomia para defeitos utilizada na Hewlett-Packard (Grady, 1992 apud Grady, 1996), contendo diversos tipos de defeitos organizados por fase de desenvolvimento (veja Anexo C2). Além do tipo de defeito esta taxonomia também considera a natureza do defeito (faltando, ambíguo, errado, modificado, ou melhoria).
- Taxonomia para defeitos de requisitos utilizada na NASA, contendo 13 tipos de defeitos de requisitos, utilizada em (Hayes *et al.*, 2006). Essa taxonomia representa um detalhamento da taxonomia descrita por Shull (1998) (que inclui os seguintes tipos de defeitos: ambigüidade, omissão, informação inconsistente, fato incorreto e informação estranha), refletindo a realidade da NASA. Em (Leszak *et al.*, 2002) tipos similares de defeitos são referenciados como a natureza do defeito e agregados a tipos de defeitos mais específicos.

A variedade de taxonomias encontrada reflete o argumento de Card (2005) que a taxonomia de defeitos deve ser criada de modo que apóie os objetivos específicos de análise da organização que está implementando análise causal de defeitos. Este argumento é reforçado por Jalote e Agrawal (2005) e (Damm *et al.*, 2008) que utilizaram o mesmo argumento para justificar a criação de uma taxonomia específica para a InfoSys e a Ericsson, respectivamente. A utilidade de um esquema específico para a organização é ainda destacada em (Robinson *et al.*, 2008), onde a taxonomia

de Beizer (Beizer, 1990) foi refinada com subcategorias específicas, relevantes para a organização em questão (ABB Corporate Research).

Outra possibilidade para agrupar defeitos em categorias quando não se tem uma taxonomia definida, embora possa envolver um esforço maior, é aplicar o método KJ (diagramas de afinidades) para descobrir categorias de acordo com a amostra de defeitos sendo analisada, conforme feito em (Nakashima *et al.*, 1999).

### 2.4.3 Esquemas de Classificação de Causas

Analisando os diferentes esquemas de classificação de causas de defeitos, listados no Anexo C3, é possível identificar um consenso entre os autores a respeito das categorias a serem utilizadas para análise causal de defeitos de software. As categorias de causas contidas nos diferentes esquemas de classificação encontrados podem ser facilmente mapeadas nas categorias inicialmente propostas por Ishikawa (1976) para a área de manufatura. As categorias de Ishikawa são sugeridas como um ponto de partida para abordagens de análise causal de defeitos de software por Card (1998) (2005). Elas são as seguintes: “ferramentas”, “entradas”, “pessoas” e “métodos”.

Entretanto, é importante destacar que os demais esquemas, considerando o contexto específico de software, podem trazer informações adicionais para refinar o esquema de Ishikawa. Por exemplo, uma categoria adicional considerada em alguns esquemas de classificação de causas é “falta de entendimento dos requisitos”. Esta categoria poderia ser considerada uma subcategoria da categoria “pessoas”<sup>2</sup>, mas como em muitos casos (Endres, 1975) (Nakajo e Kume, 1991) (Hayes, 2006) diversas causas se enquadravam nesta categoria poderia ser interessante considerá-la separadamente. Outra forma de refinar o esquema de classificação de causas é utilizando o método KJ (diagrama de afinidades) para as causas encontradas, conforme feito em (Nakashima *et al.*, 1999).

Outro esquema de classificação interessante, citado em (Jalote and Agrawal, 2005), é o esquema de cinco “M’s” e um “E” (Robitaille, 2004). Os cinco “M’s” referem às categorias “material”, “método”, “pessoas”, “medição” e “maquinário” (*Material, Method, Manpower, Measurement, and Machinery*). O “E” refere à categoria ambiente (*Environment*). De acordo com Robitaille (2004) este esquema é comumente utilizado na manufatura. Entretanto, Jalote e Agrawal (2005) decidiram adaptá-lo ao contexto de software, sugerindo as seguintes categorias: processo, pessoas e tecnologia. Eles

---

<sup>2</sup> Poderia também ser mapeada na categoria “entrada”, dependendo da causa da falta de entendimento.

mencionam que estes são os fatores com o maior impacto na qualidade e produtividade de software (Jalote, 2000). Estas categorias podem por sua vez ser mapeadas para as categorias de Ishikawa, bastando remover a categoria “entradas”.

Por fim, os artigos (Jacobs *et al.*, 2004) e (Jacobs *et al.*, 2005) apresentam uma categoria que não pode ser facilmente mapeada para as categorias de Ishikawa, chamada “organização”. Esta categoria está relacionada a causas organizacionais que transcendem a questão do processo. Por exemplo, políticas organizacionais inadequadas ou inexistentes, distribuição geográfica inapropriada das unidades organizacionais (problemas de comunicação ocasionados por diferentes fusos horários, etc), estrutura organizacional inapropriada (nenhum funcionário responsável pela gerência de configuração, etc), entre outros. É importante ressaltar que estes artigos possuem foco em projetos com equipes geograficamente distribuídas, e que esta categoria parece ser particularmente útil neste contexto.

#### **2.4.4 Conhecimento a Respeito de Análise Causal de Defeitos**

Visando facilitar o uso de demais conhecimentos obtidos através da análise dos artigos, este conhecimento foi estruturado de modo a separar conhecimento genérico a respeito de análise causal de defeitos do conhecimento limitado a suas atividades e tarefas específicas. As seguintes tabelas de conhecimento foram geradas:

- Uma tabela com conhecimento genérico a respeito de análise causal de defeitos.
- Uma tabela para cada atividade do processo de prevenção de defeitos descrito em (Jones, 1985), (Mays, 1990), e (Mays *et al.*, 1990). Conforme descrito anteriormente, estas atividades são: (1) reunião de análise causal, (2) implementação das ações, (3) reuniões de início de fase (*stage kickoff*) e (4) coleção de dados e monitoração. O conhecimento a respeito da reunião de análise causal foi ainda organizado em subgrupos para cada um dos seis passos mencionados em (Card, 2005): (1) selecionar a amostra, (2) classificar os defeitos, (3) encontrar o erro sistemático, (4) encontrar as causas principais, (5) elaborar propostas de ação e (6) documentar os resultados da reunião.

As tabelas completas, organizadas por data de publicação, estão disponíveis no Anexo C4. É importante mencionar que o conhecimento descrito nas tabelas deve ser usado considerando o contexto em que foi gerado. O contexto pode ser visto consultando as tabelas com todas as informações extraídas dos artigos, disponíveis

no Anexo B, onde o conhecimento e o tipo de estudo conduzido são descritos de forma mais detalhada.

## 2.5 Considerações Finais

Este capítulo apresentou os resultados da revisão sistemática conduzida visando ajudar a compreender o estado da arte de análise causal de defeitos de software, com respeito aos objetivos específicos da pesquisa.

A revisão permitiu resumir os processos, abordagens e diretrizes (conhecimento) que têm sido propostos para análise causal de defeitos de software. Foi possível ainda identificar e analisar os esquemas de classificação dos defeitos e das causas utilizados por estes processos e abordagens.

O protocolo elaborado para a revisão permite a execução periódica da revisão para integrar o conhecimento atualizado que esteja sendo gerado por publicações científicas ao longo dos anos.

Os resultados da revisão, que contou com a análise efetiva de 72 artigos, reforçam o argumento de Card (2005), confrontando pouca atividade científica e acadêmica com uma larga adoção industrial. A Tabela 2.1, adaptada de Card (2005), reflete este argumento. A adoção industrial é ainda reforçada pela consideração de atividades de análise causal em modelos de maturidade como o CMMI (SEI, 2006a) e o MPS (SOFTEX, 2009).

**Tabela 2.1. Pesquisa e Aplicação de Técnicas de Análise Causal de Defeitos de Software. Adaptado de (Card, 2005).**

<b>Técnica</b>	<b>Adoção na Indústria</b>	<b>Atividades de Pesquisa</b>
<b>Modelos de captura e recaptura de defeitos</b>	Nenhuma	Moderadas
<b>Perfis estatísticos de defeitos</b>	Ampla	Quase inexistentes
<b>Perfis analíticos de defeitos</b>	Limitada	Baixa
<b>Gráficos de controle</b>	Ampla	Baixa
<b>Análise Causal de Defeitos</b>	Ampla	Quase inexistentes

Este cenário motivou o uso dos resultados da revisão sistemática para a elaboração de diretrizes que pudessem apoiar a implementação de análise causal de defeitos em organizações desenvolvedoras de software. Estas diretrizes são descritas no capítulo seguinte.

## **CAPÍTULO 3 - DIRETRIZES PARA IMPLEMENTAR ANÁLISE CAUSAL DE DEFEITOS EM ORGANIZAÇÕES DE SOFTWARE**

*Apesar das vantagens e da adoção de análise causal de defeitos pouca pesquisa tem sido realizada na área. Desta forma, profissionais encontram pouco apoio e podem enfrentar diversas dificuldades quando implementam análise causal de defeitos em organizações de software. Tendo isto em vista neste capítulo descreve a compilação de diretrizes para apoiar a implementação de análise causal de defeitos, elaboradas com base em resultados obtidos da revisão sistemática descrita no capítulo anterior.*

### **3.1 Introdução**

Com base no conhecimento adquirido e nas análises realizadas como resultado da revisão sistemática, descritas no capítulo 2, apoio para implementar análise causal de defeitos em organizações de software pôde ser elaborado. Este apoio visa oferecer respostas para dúvidas que podem ser enfrentadas por profissionais quando tentam implementar análise causal de defeitos em organizações de software.

As dúvidas tratadas são as seguintes: Minha organização está pronta para análise causal de defeitos? Que abordagem deve ser seguida? Que métricas devem ser coletadas? Como análise causal de defeitos pode ser integrada com controle estatístico de processos? Como defeitos devem ser categorizados? Como as causas devem ser categorizadas? Quais os custos e resultados esperados da implementação de análise causal de defeitos?

Estas respostas podem então ser utilizadas, em conjunto com as tabelas de conhecimentos a respeito do processo de análise causal de defeitos e suas atividades (Anexo C4), como diretrizes para apoiar a implementação eficiente de análise causal de defeitos (Kalinowski *et al.*, 2008a).

É importante ressaltar que estas diretrizes não são a contribuição principal desta tese, mas que sua elaboração visou a obtenção de uma compreensão mais ampla da área para posteriormente propor a abordagem descrita no capítulo 4.

## **3.2 Minha Organização está Pronta para Análise Causal de Defeitos de Software?**

O principal requisito para aplicar análise causal de defeitos é coletar dados a respeito de defeitos. Entretanto, como argumentado por Card (1993) e van Moll *et al* (2002), a análise causal de defeitos se beneficia muito de um processo definido estabelecido para os projetos. O processo definido facilita tanto a identificação quanto a implementação das oportunidades de melhoria.

Dados os potenciais benefícios e alto retorno de investimento de atividades de análise causal de defeitos (Mays *et al.*, 1990) (Card, 1993) (Leszak *et al.*, 2000) (Jalote and Agrawal, 2005) (Shenvi, 2009), a implementação de análise causal de defeitos é recomendada mesmo para organizações de baixa maturidade, embora seja apenas requerida nos mais altos níveis de maturidade de modelos como o CMMI (SEI, 2006a) e o MPS (SOFTEX, 2009). A análise causal de defeitos pode ajudar a melhorar o desempenho e a capacidade dos processos antes que os altos níveis de maturidade sejam alcançados (Card, 1998) (Card, 2005).

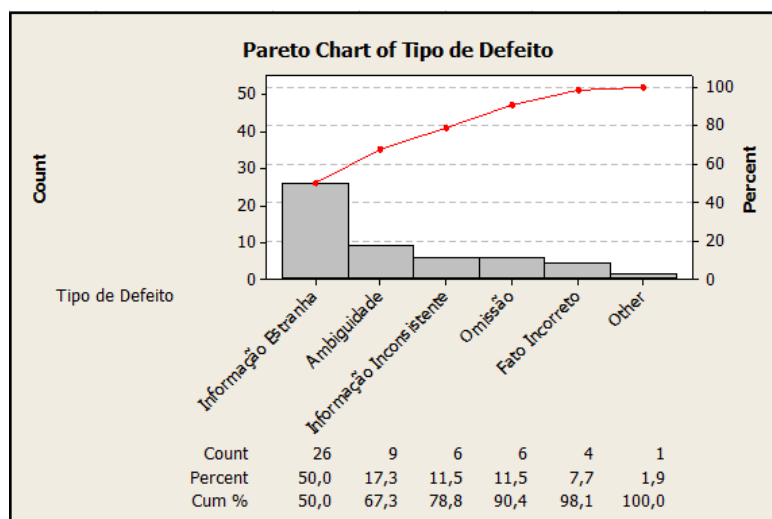
## **3.3 Que Abordagem deve ser Seguida?**

Baseado nos resultados da revisão sistemática apresentados no capítulo anterior, uma abordagem eficiente de análise causal poderia seguir o processo tradicional de prevenção de defeitos (Jones, 1985), que tem sido utilizado tanto nas abordagens iniciais quanto em abordagens mais recentes. Conforme mencionado anteriormente, este processo tem se mostrado de baixo custo e eficiente em reduzir taxas de defeitos em diferentes contextos organizacionais (Card, 2005).

Conforme sugerido por Card (1993) a reunião de análise causal deveria focar em classes de defeitos, com amostras destas classes coletadas antes da reunião de análise causal. Entre as técnicas específicas para apoiar análise causal, os resultados da revisão sistemática indicam o uso de gráficos de Pareto para identificar as classes de defeitos a serem analisadas e diagramas de causa e efeito para apoiar a identificação das principais causas para o erro sistemático que levou aos defeitos daquela classe. O uso de gráficos de Pareto para tratar as classes principais de defeitos é também sugerido na adaptação da abordagem Lean para desenvolvimento de software, descrita em (Poppendieck e Poppendieck, 2007).

A Figura 3.1 ilustra um gráfico de Pareto aplicado à natureza dos defeitos de severidade alta de um projeto real, encontrados em uma inspeção de requisitos.





**Figura 3.1. Gráfico de Pareto Aplicado à Natureza de Defeitos.**

No caso da Figura 3.1 a amostra de defeitos selecionada para análise compreendia os defeitos da natureza “Informação Estranha” (informação que não deveria estar presente no documento de requisitos), que sozinhos eram responsáveis por aproximadamente cinquenta por cento dos defeitos. O erro sistemático que pôde ser identificado nesta amostra era a inclusão de informações que não pertenciam ao domínio do problema. A causa para este erro sistemático era a falta de compreensão do analista sobre o conteúdo que deveria estar presente no documento. Outro exemplo de uma causa de um erro sistemático encontrada através da análise da natureza de defeitos de requisitos pode ser encontrado em (Kalinowski *et al.*, 2007).

### 3.4 Que Métricas devem ser Coletadas?

Conforme observado nos artigos lidos como parte da revisão sistemática, as seguintes métricas podem apoiar a análise causal de defeitos e a interpretação de seus resultados:

- Número de defeitos encontrados por unidade de tamanho;
- número médio de defeitos encontrados por hora de inspeção (se disponível);
- as métricas PIQ (*Phase Input Quality*) e POQ (*Phase Output Quality*) definidas por Damm e Lundberg (2007).

As primeiras duas destas métricas, referentes ao número de defeitos presentes no documento, auxiliam na indicação da eficiência na prevenção de defeitos. As métricas PIQ e POQ, por sua vez, permitem uma compreensão adicional sobre a

eficiência dos mecanismos de detecção de defeitos nas diferentes fases de desenvolvimento.

A métrica PIQ de uma fase do desenvolvimento é computada dividindo o número de defeitos encontrados nesta fase inseridos antes dela pelo número total de defeitos encontrados na fase. Desta forma, a métrica permite identificar as fases de desenvolvimento que sofrem as consequências de defeitos que deveriam ter sido detectados antes. Quanto menor o valor de PIQ (que varia entre 0 e 1), menos a fase é atingida por defeitos propagados de fases anteriores.

A métrica POQ de uma fase do desenvolvimento, por sua vez, é computada dividindo o número de defeitos propagados desta fase e detectados em outras fases pelo número total de defeitos encontrados na fase. Ela complementa a métrica PIQ, permitindo identificar as fases que não são capazes de detectar seus próprios defeitos, ou seja, as fases que mais contribuem para o aumento do PIQ de outras fases.

De acordo com Damm e Lundberg (2007), que fizeram uso das métricas PIQ e POQ em algumas unidades na Ericsson, elas contribuem para identificar a eficiência dos mecanismos de detecção e as fases em que a organização tem as maiores oportunidades de melhoria. Utilizadas em conjunto elas representam uma solução para compreender a propagação de defeitos entre fases, muitas vezes referenciada por porosidade.

Para computar as quatro métricas sugeridas, basta coletar as informações básicas a respeito de defeitos de software (veja seção 3.6 para maiores detalhes) e ter informações sobre o tamanho dos projetos e sobre o esforço gasto nas inspeções. Estas métricas podem ser postas sob controle estatístico integrando análise causal de defeitos com controle estatístico de processos, provendo assim uma maneira mais precisa e quantitativa de determinar como o desempenho e a capacidade dos processos de engenharia do software são afetados pela análise causal de defeitos. A integração com controle estatístico de processos é discutida em maiores detalhes na seção seguinte.

### **3.5 Como Análise Causal de Defeitos pode ser Integrada com Controle Estatístico de Processos?**

A respeito da integração com controle estatístico de processos, análise causal de defeitos pode ser utilizada para duas razões principais: (1) controlar e estabilizar processos e (2) melhorar o desempenho e capacidade dos processos.

Conforme mencionado por Florac e Carleton (1999), o primeiro destes objetivos lida com causas atribuíveis (*assignable causes*). A existência de causas

atribuíveis pode ser diagnosticada ao aplicar testes de estabilidade de processos, como os descritos em (Wheeler e Chambers, 1992), a gráficos de controle estatístico. Se o processo não está estável, então a análise causal de defeitos pode ajudar a encontrar as causas atribuíveis, viabilizando a proposta de ações que eliminem estas causas para estabilizar o processo. Mais informações e um exemplo de controle estatístico de processos podem ser encontrados em (Montoni *et al.*, 2007).

O segundo objetivo trata de causas comuns (*common causes*) responsáveis pelo desempenho e capacidade atual do processo visando melhoria contínua (Florac e Carleton, 1999). Isto pode ser feito aplicando análise causal de defeitos mesmo se o processo se encontra estável, por exemplo, após cada atividade de desenvolvimento ou periodicamente.

Relacionando estes objetivos com os modelos de maturidade CMMI e MPS, controlar e estabilizar processos está relacionado aos níveis de maturidade 4 do CMMI e B do MPS. Melhorar o desempenho e a capacidade, por outro lado, está relacionado aos níveis de maturidade 5 do CMMI e A do MPS.

A revisão sistemática indica que o gráfico de controle mais apropriado para controlar dados de defeitos e apoiar sua análise causal é o gráfico U (Hong *et al.*, 1999) (Jalote e Saxena, 2002) (Card, 2005). Gráficos U se aplicam a dados que seguem uma distribuição de Poisson, com uma área de oportunidade variável (como por exemplo, o tamanho de um documento) para que um evento ocorra (como por exemplo, a detecção de um defeito) (Montgomery, 1991 apud Hong *et al.*, 1999). Gráficos individuais ou XmR podem ser utilizados, mas são menos sensíveis para revelar instabilidades (Jalote e Saxena, 2002).

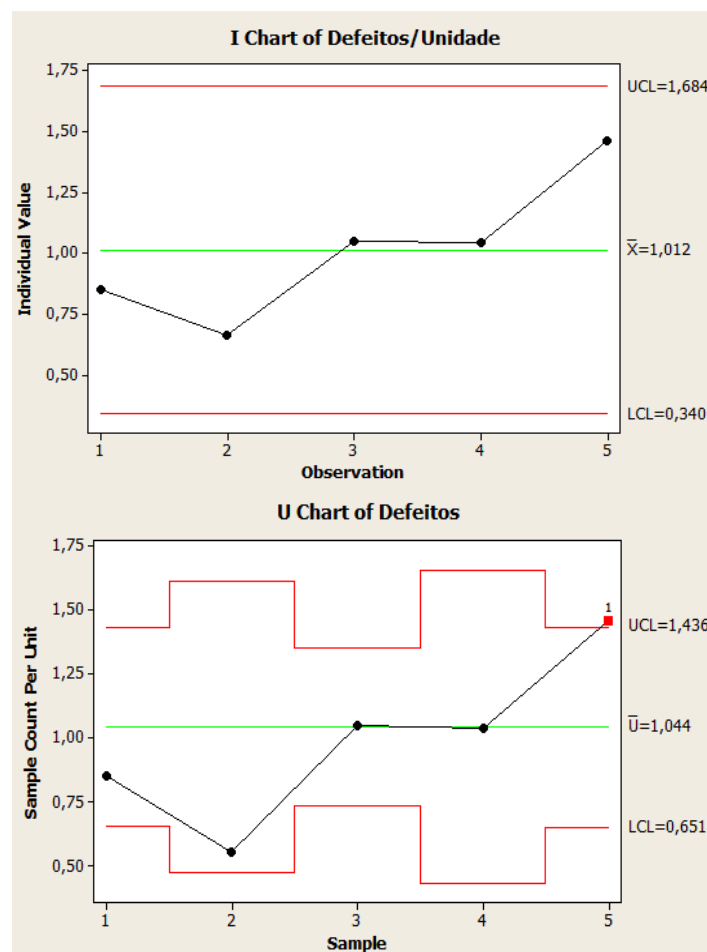
Adicionalmente, limites de controle customizados e ótimos (para controlar atividades de engenharia com base em dados de inspeções de software) podem ser calculados em função de um modelo de custo para o processo de software (Jalote e Saxena, 2002). Para isto, além do modelo de custos, dados sobre a estabilidade do processo são necessários para este cálculo. Entretanto, de acordo com Jalote e Saxena (2002), como ponto de partida, pode ser mais interessante trabalhar com gráficos U do que com os limites de controle customizados e ótimos, uma vez que a sensibilidade alta pode não ser interessante em um momento de institucionalização de gráficos de controle estatístico.

A observação dos dados da Tabela 3.1 através dos gráficos XmR e U ilustra a diferença de sensibilidade entre eles. Os dados são fictícios e representam o número de defeitos encontrados em documentos de requisitos e o tamanho destes documentos.

**Tabela 3.1. Dados Fictícios para Ilustrar a Diferença de Sensibilidade entre Gráficos XmR e U.**

Módulo do Projeto	Defeitos	Pontos de Função	Defeitos / Ponto de Função
1	53	620	0,085
2	16	290	0,066
3	103	980	0,105
4	26	250	0,104
5	89	610	0,146

Os gráficos XmR e U referentes aos dados da Tabela 3.1 encontram-se na Figura 3.2. Considerando estes dados, o gráfico XmR dos defeitos por unidade de tamanho não revela nenhum ponto fora de controle, enquanto o gráfico U revela um ponto fora de controle. Esta diferença se dá porque o gráfico U leva em consideração a área de oportunidade (no caso o tamanho do projeto em pontos de função), quanto maior a área de oportunidade mais rigoroso será o controle do gráfico U.



**Figura 3.2. Gráficos XmR e U, Considerando Dados Equivalentes.**

## 3.6 Como Categorizar os Defeitos?

Esta seção encontra-se subdividida em duas subseções que podem auxiliar profissionais na implementação de análise causal de defeitos. A primeira trata das informações a respeito de defeitos que devem ser coletadas. A segunda discute taxonomias que podem ser utilizadas para atribuir tipos a defeitos.

### 3.6.1 Que Informações Coletar?

De acordo com os artigos analisados, uma abordagem eficiente de análise causal deve fazer uso ao menos das informações de consenso, que são: o momento (ou fase) de introdução, o momento (ou fase) de detecção e o tipo do defeito. Estas informações, além de serem consenso entre os artigos analisados na revisão sistemática são sugeridas também em livros, como (Tian, 2005) e (McDonald *et al.*, 2008).

Adicionalmente, conhecer a severidade dos defeitos (ou seu impacto) e sua localização poderia apoiar análise causal de defeitos apoiando a identificação de *clusters* de defeitos que apóiem a seleção da amostra de defeitos a ser analisada e a monitoração da eficiência do processo de análise causal de defeitos (houve redução na taxa de defeitos severos? Houve redução na taxa de defeitos dos módulos críticos?). Entretanto, é importante mencionar que o significado de severidade é muitas vezes ambíguo uma vez que um mesmo defeito pode ter impactos diferentes dependendo do contexto em que a falha relacionada ao defeito se manifesta (Card, 2005).

Por fim, informações adicionais, tais como o *trigger* (o que levou a defeito a ser revelado) e se o defeito é relacionado com funcionalidades novas, podem ser utilizadas para propósitos específicos, tais como melhorar os mecanismos de detecção (quando a prevenção dos defeitos é difícil de ser atingida), conforme feito em (Chernak, 1996) e (Jalote *et al.*, 2007).

### 3.6.2 Que Taxonomia de Defeitos Utilizar?

Os resultados da revisão sistemática sugerem que uma abordagem eficiente de análise causal pode ser alcançada utilizando uma taxonomia que considere tanto a natureza (ou *mode* ou *qualifier*) quanto o tipo do defeito. A natureza do defeito pode ser expressa utilizando as categorias descritas por Shull (1998), que por sua vez podem ser refinadas para contextos específicos, como em (Hayes, 2006). Estas categorias (omissão, ambiguidade, informação inconsistente, fato incorreto, informação estranha e outros) podem ser utilizadas de maneira genérica para

diferentes artefatos produzidos ao longo do ciclo de vida de desenvolvimento do software, conforme observado por Travassos *et al* (2001).

Conforme indicado na revisão sistemática, a natureza de defeitos é considerada em diversos esquemas de classificação de defeitos, como no de ODC da IBM (Chillarege *et al.*, 1992), onde é referenciada pelo atributo *qualifier* e no da HP (Grady, 1996) e do padrão IEEE (IEEE, 2010), onde é referenciada pelo atributo *mode*.

Para o tipo de defeito, a taxonomia deve ser criada considerando objetivos específicos da abordagem de análise causal da organização, conforme sugerido por Card (2005). Não tendo uma taxonomia para tipos de defeitos estabelecida na organização os tipos considerados na ODC (Chillarege *et al.*, 1992) ou no padrão IEEE (IEEE, 2010) podem servir como ponto de partida. ODC sugere os seguintes tipos de defeitos: interface, funcionalidade, empacotamento (*build/package/merge*), inicialização (atribuição), documentação, verificação (*checking*), algoritmo e temporal (*timing/serialization*). O padrão IEEE, por sua vez, sugere os seguintes tipos de defeitos: dados, interface, lógica, descrição, sintaxe, padrões e outros.

### **3.7 Como Categorizar as Causas?**

Dado o cenário de consenso encontrado na revisão sistemática a respeito das categorias de causa, as categorias de Ishikawa (ferramentas, entradas, pessoas e métodos) podem representar um bom ponto de partida (Ishikawa, 1976), conforme sugerido por Card (1998) (2005), adicionando por questão de completude a categoria “organização” (Jacobs *et al.*, 2004) (Jacobs *et al.*, 2005).

O detalhamento deste esquema de classificação pode ser realizado posteriormente, ao criar subcategories específicas, baseado na realidade da organização (obtida, por exemplo, aplicando o método KJ – diagrama de afinidades - nos resultados das próprias análises causais de defeitos, conforme feito em (Nakashima *et al.*, 1999)).

### **3.8 Quais os Custos e Resultados Esperados da Análise Causal de Defeitos?**

A resposta para esta questão ajuda a estabelecer objetivos quantitativos de melhoria para a implementação da análise causal de defeitos. Baseado no conhecimento genérico obtido a respeito das abordagens de análise causal, listado no Anexo C4, em organizações grandes seu custo varia entre 0,5 e 1,5 por cento do orçamento do projeto (incluindo a implementação das ações propostas), de acordo com dados da IBM e da Computer Science Corporation (Card, 2005). Este custo é

baixo quando comparado aos benefícios, que têm sido uma redução da taxa de inserção de defeitos em mais de 50 por cento (novamente dados da IBM e Computer Science Corporation) (Card, 2005). Como consequência o esforço de retrabalho será reduzido, implicando em uma maior produtividade e menor custo para os projetos.

Embora os benefícios sejam medidos de diferentes formas (por exemplo, redução da taxa de defeitos, redução do esforço de retrabalho, redução de tipos de defeito específicos, entre outros), todos os artigos analisados relataram a obtenção de resultados positivos com a implementação de análise causal de defeitos. Os artigos descrevem experiências em uma grande variedade de organizações de software, incluindo: AG Communication Systems Corporation, Computer Science Corporation, Ericsson, Hewlett-Packard, IBM (diversas unidades organizacionais), InfoSys, Italtel SIT BUCT Línea UT, Lucent Technology, Motorola, Philips Software Centre e Siemens. Maiores detalhes sobre estes benefícios podem ser encontrados nas tabelas do Anexo C4.

### **3.9 Uso das Diretrizes para Realizar Análise Causal de Defeitos de Acordo com os Modelos de Maturidade**

Nesta seção será apresentado como as diretrizes podem ser utilizadas para conduzir a análise causal de defeitos de acordo com as boas práticas recomendadas pelos modelos de maturidade CMMI (SEI, 2006a) e MPS (SOFTEX, 2009). A discussão será focada em um destes modelos, o CMMI, que possui uma área de processos dedicada à análise causal, denominada *Causal Analysis and Resolution (CAR)*, no nível de maturidade 5.

A condução de análise causal de defeitos realizando as práticas específicas da área de processos CAR do CMMI implica no atendimento também de alguns dos resultados dos atributos de processo 5.1 e 5.2 do modelo MPS. Adicionalmente, implica no atendimento de alguns dos resultados de um dos processos complementares da ISO 12207, Melhoria de Processos Quantitativa (*Quantitative Process Improvement*), definido na ISO 15504-7 (ISO/IEC, 2008b). Os resultados atendidos do MPS e da ISO são os referentes à identificação e implementação de oportunidades de melhoria relacionadas a causas comuns de variação do desempenho do processo. De fato, a aplicação de análise causal de defeitos a um processo de engenharia representa uma das principais formas de identificar e tratar causas comuns de variação do desempenho deste processo, e, consequentemente, de permitir a melhoria quantitativa do desempenho deste processo (Card, 2005) (Wang *et al.*, 2006) (SEI, 2006a) (SOFTEX, 2009).

É importante deixar claro que o escopo desta tese é a análise causal de defeitos de software, enquanto o escopo da área de processos CAR do CMMI envolve a análise causal de defeitos de software e de outros problemas que podem ocorrer ao longo do desenvolvimento (como atrasos no cronograma, custos fora do controle, entre outros). Assim, nesta seção estaremos tratando especificamente de utilizar as diretrizes para realizar análise causal de defeitos de software de acordo com os modelos de maturidade. Para a implementação completa da área de processos CAR seria necessário apoiar também a análise causal de outros problemas que podem surgir durante o desenvolvimento. Desta forma, as informações aqui apresentadas complementam as contidas no modelo CMMI (SEI, 2006a) e no guia de implementação do nível A do MPS (SOFTEX, 2009), fornecendo detalhes adicionais sobre como lidar com os defeitos de software.

As subseções seguintes descrevem como cada uma das práticas específicas da área de processos CAR pode ser efetivamente realizada, para a análise causal de defeitos de software, com base nas diretrizes.

### **3.9.1 SP 1.1 Selecionar Dados de Defeitos para Análise.**

De acordo com a tradução do CMMI (SEI, 2006b), esta prática envolve “Selecionar defeitos e outros problemas para análise”. Assim, esta prática é apoiada diretamente pelas respostas às perguntas “Como Categorizar os Defeitos?” e “Que Abordagem deve ser Seguida?”.

O registro, identificação e classificação são abordados na resposta à primeira pergunta, onde esquemas de classificação de defeitos são discutidos. A seleção para análise, por sua vez, pode ser realizada utilizando gráficos de Pareto, conforme sugerido na resposta à pergunta “Que Abordagem deve ser Seguida?”.

### **3.9.2 SP 1.2 Analisar Causas**

De acordo com a tradução do CMMI (SEI, 2006b), esta prática envolve “Realizar a análise de causas de defeitos e de outros problemas selecionados e propor ações para tratá-las”. Assim, esta prática é apoiada pela resposta à pergunta “Que Abordagem deve ser Seguida?”.

A resposta a esta pergunta sugere o uso de diagramas de causa e efeito para que a equipe, composta por responsáveis pela introdução dos defeitos, responsáveis por sua detecção e membros do grupo de processos da organização, identifique as causas principais para o erro sistemático (um mesmo erro que levou a diversos defeitos do mesmo tipo) em questão.



### **3.9.3 SP 2.1 Implementar Propostas de Ação**

De acordo com a tradução do CMMI (SEI, 2006b), esta prática envolve “Implementar propostas de ação selecionadas que foram desenvolvidas durante análise de causa”. Novamente o resultado é apoiado pela resposta à pergunta “Que Abordagem deve ser Seguida?”

A resposta a esta pergunta sugere o uso de um processo aderente ao de prevenção de defeitos (Jones, 1985). Na reunião de análise causal deste processo, uma vez que as causas principais foram encontradas, propostas de ação são desenvolvidas pela equipe para tratar estas causas. Estas ações devem ser priorizadas para que possam ser gerenciadas até sua conclusão (Card, 2005). O processo de prevenção de defeitos compreende ainda uma atividade específica para a implementação das ações de melhoria identificadas.

### **3.9.4 SP 2.2 Avaliar Efeitos de Mudanças**

De acordo com a tradução do CMMI (SEI, 2006b), esta prática envolve “Avaliar os efeitos das mudanças no desempenho do processo”. Uma maneira interessante de implementar este resultado é através da integração da análise causal de defeitos com o controle estatístico de processos, tratado na resposta à pergunta “Como Análise Causal de Defeitos pode ser Integrada com Controle Estatístico de Processos?”.

Assim, objetivos quantitativos de melhoria do processo de engenharia em que os defeitos foram revelados devem ser estabelecidos e acompanhados através do controle estatístico das métricas relacionadas a defeitos sugeridas na resposta à pergunta “Que Métricas devem ser Coletadas?”. As diretrizes sugerem o uso de gráficos U como ponto de partida para o controle estatístico das métricas relacionadas a defeitos (Hong *et al.*, 1999) (Jalote e Saxena, 2002) (Card, 2005).

### **3.9.5 SP 2.3 Registrar Dados**

De acordo com a tradução do CMMI (SEI, 2006b), esta prática envolve “Registrar dados de análise e resolução de causas para uso no projeto e na organização”.

A resposta à pergunta “Que Abordagem deve ser Seguida?” indica o processo de prevenção de defeitos como uma possibilidade de abordagem a seguir, este processo já menciona uma base de experiências alimentada através da documentação da reunião de análise causal e da implementação das ações. A importância de documentar os resultados da reunião de análise causal é também destacada por Card (2005). Fazem parte da documentação da reunião o tipo de defeito analisado, os erros

sistemáticos identificados e analisados, as causas principais encontradas para estes erros e as ações propostas para tratar estas causas (Leszak *et al.*, 2002) (Card, 2005). Em relação à implementação das ações, a documentação é referente ao gerenciamento desta implementação, envolvendo custo, esforço e tempo da implementação de cada uma das ações. O resultado das ações pode ser acompanhado conforme descrito na seção anterior e também deve ser documentado.

### **3.10 Considerações Finais**

Considerando as abordagens encontradas na literatura técnica, foi possível observar que as mesmas descrevem atividades, mas fornecem pouco detalhamento sobre as tarefas que devem ser realizadas no contexto destas atividades e sobre as técnicas que podem apoiar a realização destas tarefas.

Tendo isto em vista, neste capítulo foram apresentadas, com base nos resultados da revisão sistemática, diretrizes para a implementação de análise causal de defeitos de software em organizações desenvolvedoras de software (Kalinowski *et al.*, 2008a). Estas diretrizes são compostas por: (1) respostas baseadas em evidência para dúvidas que podem ser enfrentadas por profissionais ao implementar o processo de análise causal de defeitos de software e (2) pelas tabelas de conhecimentos a respeito do processo de análise causal de defeitos e suas atividades. Adicionalmente, o uso dessas diretrizes para apoiar a realização de análise causal de defeitos de acordo com as práticas específicas da área de processo CAR do CMMI foi brevemente discutido. Estas diretrizes podem ser utilizadas como complemento ao Guia de Implementação do MPS – Parte 7 (SOFTEX, 2009).

Além do pouco detalhamento das abordagens encontradas, a revisão sistemática permitiu identificar outras oportunidades de pesquisa. Uma destas oportunidades de pesquisa, destacada em (Kalinowski *et al.*, 2008a), é que, até a execução da revisão sistemática referente a 2007, não foi possível encontrar uma abordagem que integrasse mecanismos de aprendizado das relações causa e efeito às reuniões de análise causal. Isto significa que, em todas as abordagens analisadas, o conhecimento a respeito de relações causa e efeito obtido durante cada reunião de análise causal era somente utilizado para iniciar ações para melhorar o processo de desenvolvimento e posteriormente era descartado ou simplesmente registrado.

O primeiro esforço neste sentido foi realizado em (Kalinowski e Travassos, 2008) e as primeiras consequências deste esforço (Kalinowski *et al.*, 2008b) (Kalinowski *et al.*, 2010) puderam ser identificadas nas execuções de Janeiro de 2009 e Julho de 2010 da revisão sistemática. A abordagem para análise causal de defeitos resultante será apresentada no capítulo seguinte. Além de tratar a oportunidade de

pesquisa, a abordagem foi elaborada de modo a fornecer um detalhamento das atividades visando facilitar sua efetiva implementação, com base no conhecimento obtido da revisão sistemática e nas diretrizes apresentadas neste capítulo.

## **CAPÍTULO 4 - ABORDAGEM PROBABILÍSTICA PARA ANÁLISE CAUSAL DE DEFEITOS DE SOFTWARE**

*Neste capítulo será apresentada uma abordagem probabilística para análise causal de defeitos provenientes de inspeções visando apoiar a melhoria dos processos de engenharia do software, chamada DPPI (Defect Prevention-based Process Improvement). DPPI fornece um detalhamento das atividades a serem realizadas com base no conhecimento obtido na revisão sistemática e nas diretrizes apresentadas no capítulo anterior. Adicionalmente, uma oportunidade de pesquisa identificada na revisão sistemática foi tratada e DPPI propõe uma forma inovadora de realizar análise causal, integrando um mecanismo de aprendizado de relações de causa e efeito (redes Bayesianas) para apoiar na identificação das causas.*

### **4.1 Introdução**

Com base nos resultados das duas primeiras revisões sistemáticas (2006 e 2007) e nas diretrizes resultantes, uma proposta inicial de uma abordagem para melhoria de processos baseada em prevenção de defeitos pôde ser elaborada (Kalinowski e Travassos, 2008) e refinada (Kalinowski *et al.*, 2008b). Esta proposta representa a primeira sugerindo a integração de um mecanismo de aprendizado das relações de causa e efeito (utilizando Redes Bayesianas<sup>3</sup>) às reuniões de análise causal de defeitos de software para facilitar a identificação das causas.

Posteriormente esta proposta foi evoluída e detalhada, resultando na abordagem DPPI (Defect Prevention-based Process Improvement). A evolução e o refinamento se deram com base em: (i) resultados de uma execução adicional da revisão sistemática (realizada em 2009), (ii) no retorno obtido de especialistas da área

---

<sup>3</sup> O termo rede Bayesiana foi introduzido por Pearl (1985) para enfatizar três aspectos: (1) a natureza subjetiva das informações de entrada; (2) o uso da condição de Bayes como base para atualizar as informações; e (3) a distinção entre o raciocínio causal e o baseado em evidência.

e (iii) da experiência de aplicar a abordagem a um projeto de desenvolvimento de software real (Kalinowski *et al.*, 2010). No restante deste capítulo a abordagem DPPI será descrita.

## **4.2 Visão Geral da Abordagem DPPI**

DPPI representa uma abordagem prática para análise causal de defeitos provenientes de inspeções de software, visando sua prevenção. Quando comparada ao processo tradicional de prevenção de defeitos, descrito em (Jones, 1985), a principal inovação é a integração de conhecimento obtido em sucessivas reuniões de análise causal para prover um maior entendimento a respeito das relações de causa e efeito dos defeitos da organização. Isto trata uma das oportunidades de pesquisa identificadas nas revisões sistemáticas e destacada em (Kalinowski *et al.*, 2008a). O primeiro esforço no sentido de tratar esta oportunidade se deu na concepção inicial da abordagem, descrita em (Kalinowski *et al.*, 2008b), que posteriormente evoluiu para se tornar DPPI (Kalinowski *et al.*, 2010). Até a presente data, de acordo com os resultados da revisão sistemática conduzida, nenhuma outra abordagem considera esta possibilidade de integração.

Tal integração permite estabelecer e manter modelos causais para a organização. Estes modelos podem então ser utilizados para apoiar o raciocínio diagnóstico, facilitando a identificação das principais causas dos defeitos sob análise em uma nova reunião de análise causal. Com estes modelos é possível, por exemplo, apoiar o entendimento, com base no aprendizado obtido de projetos similares da organização, sobre quais causas normalmente contribuem para um determinado tipo de defeitos.

Além desta inovação, DPPI segue as diretrizes para implementar análise causal em organizações de software, descritas em (Kalinowski *et al.*, 2008a) e atualizadas no Capítulo 3 desta tese. O uso das diretrizes permitiu detalhar as atividades de prevenção de defeitos em tarefas mais específicas, fornecendo detalhes adicionais sobre as técnicas a serem utilizadas para realizar estas tarefas eficientemente.

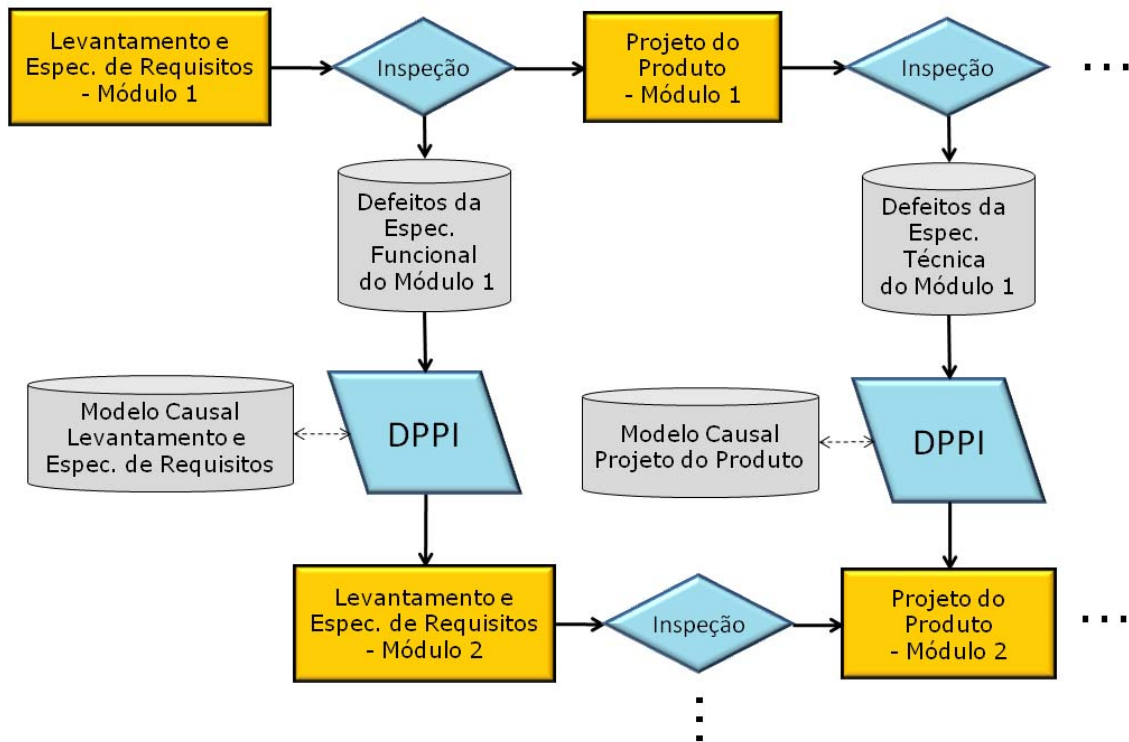
Ainda com base nas diretrizes, DPPI integra a prevenção de defeitos na estratégia de medição e controle das atividades de engenharia do software para as quais a prevenção de defeitos está sendo realizada, permitindo observar se as melhorias implementadas para estas atividades trouxeram benefícios em relação à taxa de inserção de defeitos. Isto é feito ao utilizar as métricas relacionadas a defeitos definidas nas diretrizes para medir e controlar o desempenho destas atividades.

Adicionalmente, DPPI trata todas as práticas específicas da área de processos CAR (*Causal Analysis and Resolution*) do modelo CMMI, descrito em (SEI, 2006a). Desta forma, seguir a abordagem DPPI resulta em aderência a estas práticas em relação à análise causal de defeitos de software provenientes de inspeções. Entretanto, a área de processos CAR do CMMI pode também ser utilizada como referência de práticas para realizar a análise causal de outros problemas, que estão fora do escopo de DPPI.

DPPI visa a realização de análise causal para reduzir continuamente as taxas de inserção de defeitos das atividades de engenharia do software e, conforme sugerido nas diretrizes, ela foi projetada para ocorrer de forma integrada ao processo de desenvolvimento, sempre imediatamente após as inspeções dos artefatos produzidos por estas atividades. Ou seja, DPPI deve ser aplicada tanto quando o desempenho da atividade estiver dentro do esperado (ou estável) ou não. No caso de estabilidade, a aplicação contínua permitirá a identificação de causas comuns para o desempenho (Florac e Carleton, 1999) que poderão subsidiar a implementação de ações que tragam melhorias efetivas para o desempenho relacionado aos defeitos. No caso de um desempenho instável, a aplicação de DPPI permitirá identificar causas atribuíveis para a instabilidade (Florac e Carleton, 1999) que poderão subsidiar a implementação de ações que possibilitem estabilizar o processo.

Assim, o momento de aplicação de DPPI dentro de um processo de desenvolvimento de software é após as inspeções dos artefatos produzidos pelas principais atividades de engenharia do software. É importante observar que o processo de inspeção, conforme definido por Fagan (1976) envolve a correção dos defeitos. Assim, quando DPPI é iniciada, os defeitos encontrados nos artefatos da atividade de desenvolvimento já foram corrigidos. A Figura 4.1 ilustra os momentos de aplicação de DPPI considerando um processo iterativo e incremental. Nesta figura DPPI foi aplicada duas vezes, após a inspeção da especificação funcional produzida na atividade de levantamento e especificação de requisitos, visando melhorar esta atividade para os próximos módulos e após a inspeção da especificação técnica produzida na atividade de projeto do produto, visando melhorar esta atividade para os próximos módulos.

Na Figura 4.1 é possível ainda observar que DPPI tem a lista de defeitos e o modelo causal da atividade como entrada. O modelo causal visa apoiar a identificação das causas dos defeitos e será atualizado a cada aplicação de DPPI. Maiores detalhes sobre o modelo causal e esta atualização serão fornecidos mais adiante.



**Figura 4.1. Momentos de Aplicação de DPPI em um Processo Iterativo Incremental.**

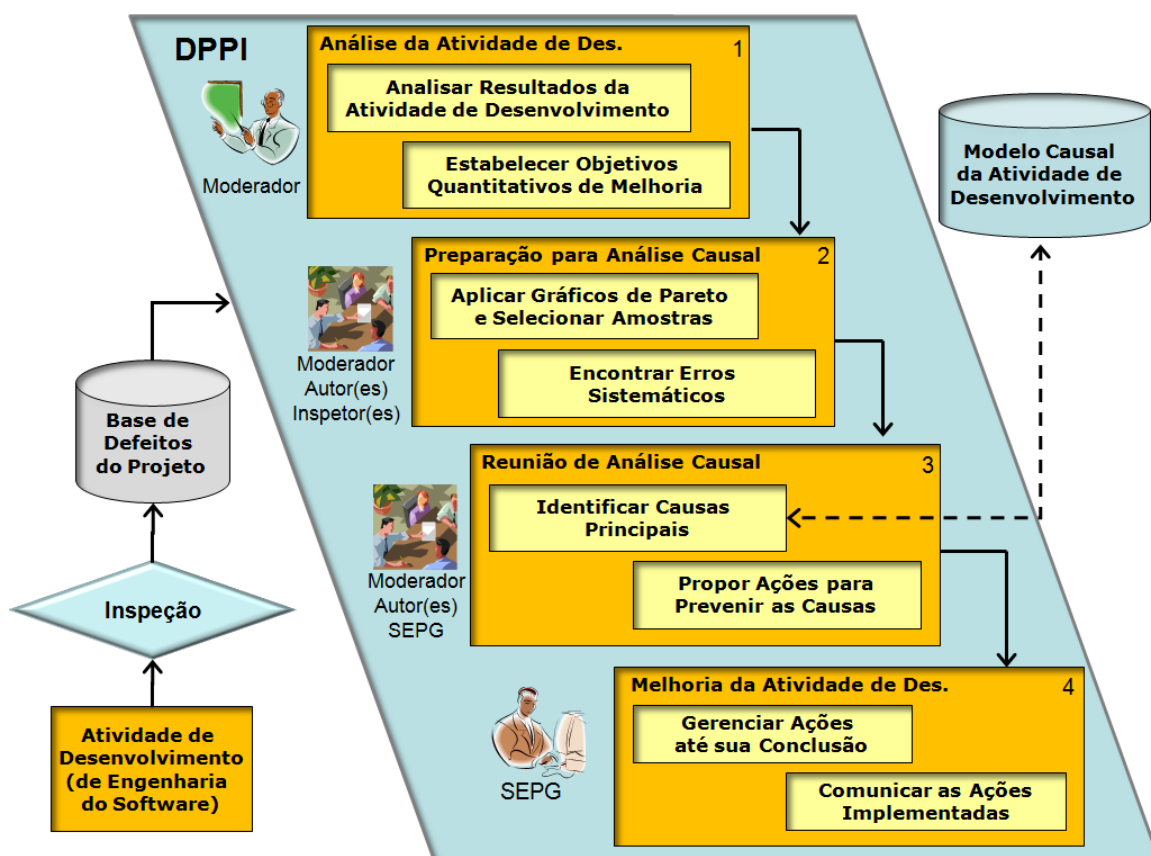
Este contexto de utilização impõe algumas premissas para a aplicação de DPPI. Estas premissas são:

- A existência de um processo definido para o projeto (ou ao menos para as atividades de engenharia do software em que DPPI será aplicada), conforme sugerido nas diretrizes.
- A realização de inspeções de software durante as quais dados sobre os defeitos são armazenados. Os dados a serem armazenados (além da descrição dos defeitos) são os indicados pelas diretrizes como as informações básicas de consenso para a realização de análise causal de defeitos: a atividade em que o defeito foi introduzido, a atividade em que o defeito foi detectado, a natureza e o tipo de defeito.
- DPPI precisa ser utilizada para melhorar atividades de engenharia do software para incrementos (módulos) do mesmo projeto ou de projetos similares. Assim, caso seja utilizada entre diferentes projetos, estes devem compartilhar o mesmo esquema de classificação de defeitos, seguir processos similares, ser relacionados a domínios de complexidade similar, utilizar equipes com experiência similar e utilizar o mesmo tipo de tecnologia. Basicamente, neste caso, as premissas de similaridade entre os projetos são as mesmas que para o agrupamento de projetos para controle

estatístico de processos (Florac e Carleton, 1999). A definição destes critérios de similaridade está fora do escopo desta tese.

- As causas identificadas para os defeitos de cada uma das atividades de engenharia do software em reuniões de análise causal anteriores (considerando projetos similares) precisam ser registradas com um nome, a categoria da causa e o tipo de defeito relacionado. Esta informação será utilizada como retroalimentação, permitindo a montagem do modelo causal.

A abordagem DPPI envolve quatro atividades: (i) Análise da Atividade de Desenvolvimento; (ii) Preparação para Análise Causal; (iii) Reunião de Análise Causal; e (iv) Melhoria da Atividade de Desenvolvimento. A Figura 4.2 fornece uma visão das tarefas e dos papéis envolvidos na execução destas atividades. É importante ressaltar que a atividade de engenharia do software a ser melhorada e a inspeção estão fora do escopo de DPPI.



**Figura 4.2. Visão Geral da Abordagem DPPI.**

Esta figura também destaca a proposta de manter, para cada atividade de engenharia do software, um modelo causal com informações a respeito das relações de causa e efeito dos defeitos da organização. Em DPPI tais modelos causais devem ser estabelecidos e mantidos ao alimentar redes Bayesianas.



Uma rede Bayesiana é um grafo direcionado e acíclico onde os nós correspondem a variáveis aleatórias ( $\text{Vars} = \{X_1, \dots, X_n\}$ ) de interesse e estas tem parâmetros de distribuição de probabilidades associadas que quantificam o efeito dos pais<sup>4</sup> em um determinado nó ( $P(X_i | \text{Pa}(X_i))$ ), onde  $\text{Pa}(X_i)$  denota os pais do nó  $X_i$ ). A rede Bayesiana especifica uma distribuição de probabilidades conjunta sobre variáveis aleatórias, dada pela expressão da Figura 4.3. Assim, segundo Russel e Norvig (2004), uma rede Bayesiana pode fornecer uma descrição completa de um domínio, onde toda entrada na distribuição de probabilidade conjunta total pode ser calculada a partir das informações armazenadas na rede.

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | \text{Pa}(X_i)).$$

**Figura 4.3. Distribuição de Probabilidades Conjunta sobre Variáveis Aleatórias de uma Rede Bayesiana.**

Pearl (2000) afirma que modelos causais probabilísticos podem ser elaborados utilizando redes Bayesianas caso exemplos concretos de relações causais<sup>5</sup> entre variáveis aleatórias possam ser obtidos para alimentar a rede. Assim, a estrutura e os parâmetros de distribuição de probabilidade podem ser aprendidos a partir destes exemplos. Após a alimentação destas redes elas podem ser utilizadas tanto para inferência preditiva quanto para inferência diagnóstica (Russel e Norvig, 2004).

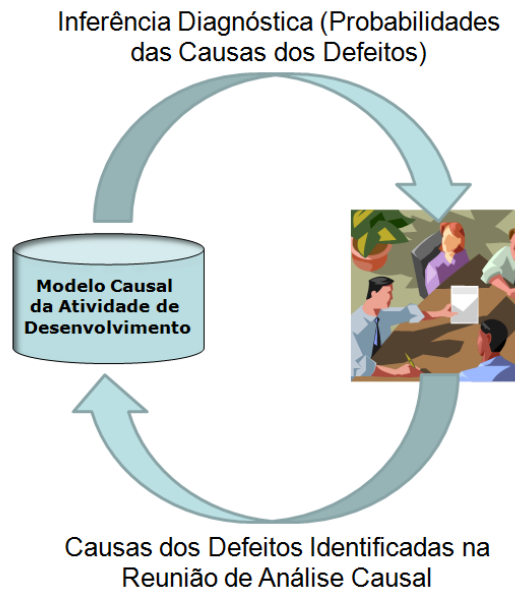
No caso de DPPI, os exemplos de relações causais para alimentar a rede Bayesiana são os próprios resultados de cada uma das reuniões de análise causal (consenso da equipe a respeito das principais causas dos defeitos analisados). Posteriormente, a inferência diagnóstica desta rede deve ser utilizada para apoiar as próximas reuniões de análise causal, fechando explicitamente um ciclo de retroalimentação a respeito das causas de defeitos de software da organização. A Figura 4.4 ilustra esse ciclo de retroalimentação.

---

<sup>4</sup> Os pais de um determinado nó em um grafo direcionado são os nós que possuem arestas direcionadas para este nó.

<sup>5</sup> De acordo com Babbie (1986), três condições precisam ser atendidas para demonstrar um relacionamento causal:

- Condição 1: Precisa haver uma correlação entre a causa e o efeito;
- Condição 2: A causa precisa preceder o efeito;
- Condição 3: O mecanismo ligando a causa ao efeito precisa ser identificado.



**Figura 4.4. Ciclo de Retroalimentação da Abordagem DPPI – Os Modelos Causais para as Atividades de Desenvolvimento são Construídos com os Próprios Resultados das Reuniões de Análise Causal.**

Redes Bayesianas têm sido utilizadas ao longo dos anos para apoiar outras atividades de engenharia de software, como gerência de riscos (Hearty *et al.*, 2005), predição de defeitos (Fenton, 1999) (Gras, 2004) (Fenton *et al.*, 2007) e controle da qualidade (Krause *et al.*, 2002). Nestas abordagens, as redes bayesianas são alimentadas com base em dados históricos sobre os defeitos e/ou com base na opinião de especialistas e posteriormente utilizadas para predição e controle.

Adicionalmente, fora do contexto de engenharia de software, na experiência descrita em (Han *et al.*, 2008), redes Bayesianas construídas com base na opinião de especialistas, aplicadas ao domínio de máquinas para produção de chips, apoiaram o diagnóstico correto de causas de sintomas destas máquinas.

A proposta de DPPI de atualizar as redes bayesianas dinamicamente com relações causais concretas, identificadas nas reuniões de análise causal de defeitos de software, e utilizar a inferência diagnóstica da rede para apoiar a identificação de causas de defeitos nas próximas reuniões de análise causal representa uma proposta nova e anteriormente não explorada.

Tendo fornecido esta visão geral da abordagem DPPI, nas próximas seções as quatro atividades de DPPI e de suas tarefas são descritas em mais detalhes. Quando conveniente, exemplos com base em projetos reais foram fornecidos para facilitar a compreensão da abordagem. Uma prova de conceito, aplicando as atividades abordagem em um projeto real, fornecendo todo o contexto deste projeto será descrita detalhadamente no capítulo seguinte.

### 4.3 DPPI: Análise da Atividade de Desenvolvimento

A análise da atividade de desenvolvimento visa a medição e controle dos subprocessos associados à atividade em relação aos seus defeitos. Esta atividade não é mandatória (a não ser que compatibilidade com o CMMI ou MPS seja desejada). Entretanto, ela é essencial para compreender a eficiência da prevenção de defeitos, conforme destacado em (Jantti *et al.*, 2006). Ela compreende duas tarefas a serem realizadas pelo moderador de DPPI (este papel pode ser desempenhado pela mesma pessoa responsável pela moderação das inspeções, por exemplo). Mais detalhes a respeito destas tarefas seguem nas subseções seguintes.

#### 4.3.1 Analisar Resultados da Atividade

Esta tarefa visa analisar os resultados da atividade de desenvolvimento em relação aos seus defeitos ao compará-los com dados históricos de resultados desta mesma atividade em projetos similares.

Para DPPI esta análise foca em mudanças nas taxas de defeitos e na qualidade da entrada. Assim, conforme sugerido pelas diretrizes, as seguintes métricas devem ser analisadas, em comparação com dados históricos, utilizando um gráfico de controle estatístico:

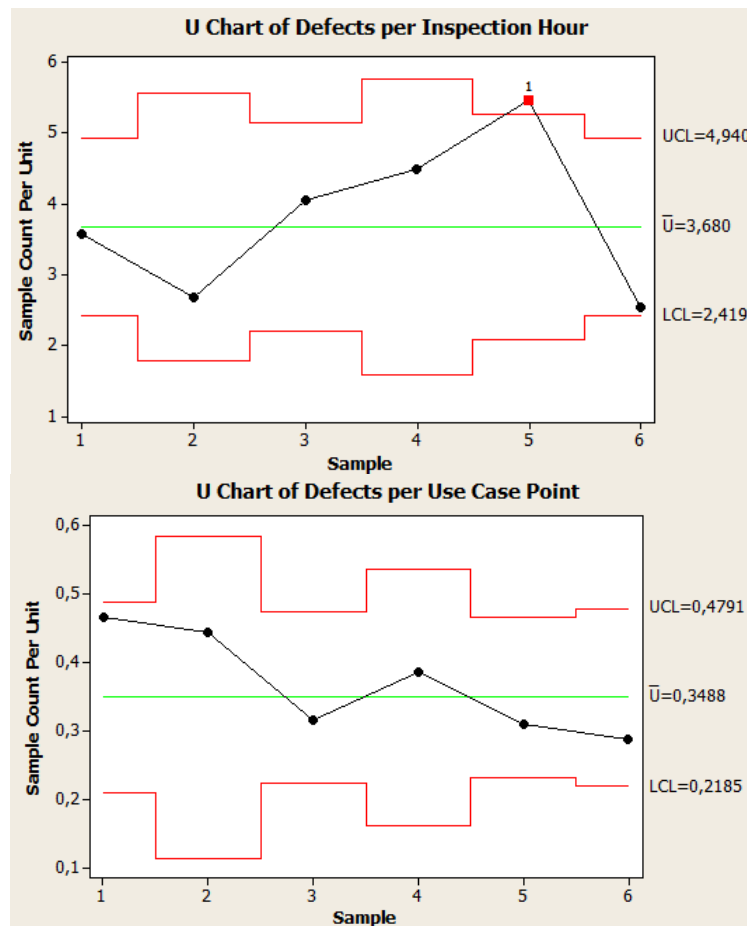
- Numero de defeitos por unidade de tamanho;
- Número de defeitos por hora de inspeção; e
- A métrica PIQ (*Phase Input Quality*), que indica o percentual de defeitos encontrados na atividade de desenvolvimento que foram introduzidos em atividades anteriores (Damm e Lundberg, 2007).

A métrica PIQ ajuda a compreender se os defeitos estão relacionados a problemas na atividade sendo analisada ou a problemas em outro ponto do processo de desenvolvimento. Nos casos em que a métrica PIQ apresenta valores altos (muitos defeitos provenientes de atividades anteriores) pode ser interessante calcular a métrica POQ (*Phase Output Quality*), que indica o percentual de defeitos passando de uma atividade de desenvolvimento para as demais (Damm e Lundberg, 2007), no final do projeto para as principais atividades de desenvolvimento, para fins diagnósticos. Isto permitirá realizar uma aplicação adicional de DPPI nestas atividades.

Adicionalmente, conforme sugerido por (Hong *et al.*, 1999) e indicado pelas diretrizes, DPPI utiliza o gráfico de controle estatístico U para as primeiras duas destas métricas, dado que defeitos tendem a seguir uma distribuição de Poisson. Para a métrica PIQ (que representa um percentual) um gráfico de pontos individuais (XmR) pode ser plotado. Estes gráficos podem indicar se as métricas relacionadas aos

defeitos da atividade estão apresentando resultados estáveis para o subprocesso associado à atividade de desenvolvimento sendo analisada, aplicando alguns testes estatísticos básicos (mais a respeito destes testes pode ser encontrado em (Florac e Carleton, 1999)).

Um exemplo de gráfico de controle U para defeitos por hora de inspeção e defeitos por unidade de tamanho (no caso pontos de caso de uso) de seis iterações da atividade de levantamento e especificação de requisitos de um projeto real pode ser visto na Figura 4.5. Neste exemplo, embora a sexta iteração apresente um comportamento satisfatório (poucos defeitos por hora e poucos defeitos por unidade de tamanho), quando comparado aos demais módulos, DPPI seria iniciada para encontrar as causas comuns do desempenho do processo e identificar oportunidades de melhoria.



**Figura 4.5. Exemplos de Gráfico U com Base em Dados de Seis Iterações de um Projeto Real.**

É importante ressaltar, entretanto, que, para que os gráficos U contendo as métricas de defeitos sejam efetivamente úteis para controlar a atividade de desenvolvimento sendo analisada, as inspeções precisam ser conduzidas seguindo

um processo definido e estável, de modo que os resultados dos defeitos possam ser atribuídos à atividade de desenvolvimento e não às inspeções.

#### **4.3.2 Estabelecer Objetivos Quantitativos de Melhoria**

Esta tarefa envolve estabelecer objetivos quantitativos de melhoria para a atividade de desenvolvimento sendo analisada. Um exemplo típico de objetivo quantitativo de melhoria é: “reduzir a taxa de defeitos por hora de inspeção (ou por unidade de tamanho) em X por cento”.

Se a atividade de engenharia do software sob análise está fora de controle (as métricas aparecem no gráfico de controle fora dos limites de controle ou infringindo outros testes de estabilidade) o foco da reunião de análise causal é identificar as causas atribuíveis e o objetivo de melhoria deve ser estabilizar a atividade.

Se a atividade está sob controle o foco passa a ser identificar as causas comuns para o desempenho atual da atividade e o objetivo de melhoria deve ser melhorar o desempenho e a capacidade da atividade. Neste caso os resultados esperados da aplicação de análise causal de defeitos descritos nas diretrizes do capítulo 3 podem servir como uma *baseline* interessante para estabelecer os objetivos de melhoria iniciais.

### **4.4 DPPI: Preparação para Análise Causal**

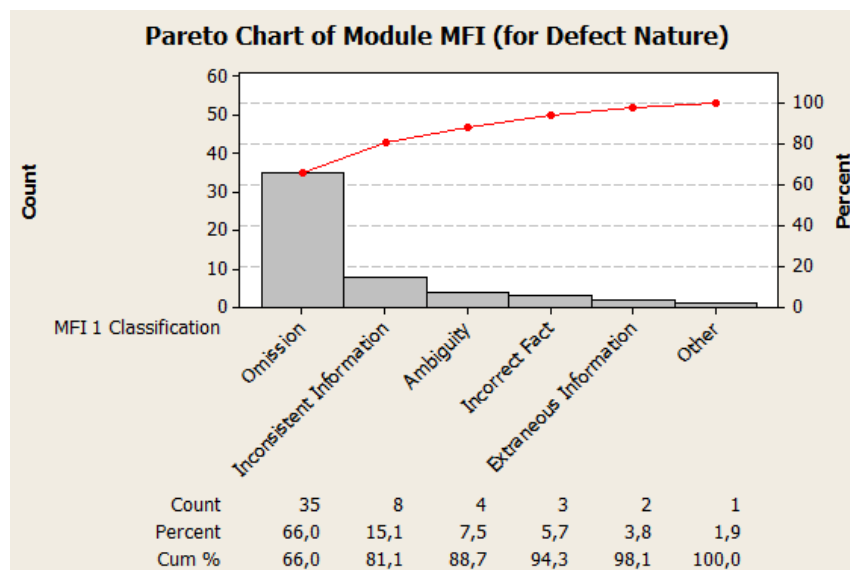
Esta atividade compreende a preparação para a reunião de análise causal de defeitos, selecionando a amostra de defeitos a ser analisada e identificando os erros sistemáticos que levam a diversos destes defeitos.

#### **4.4.1 Aplicar Gráficos de Pareto e Selecionar Amostras**

Esta tarefa se refere a encontrar as amostras de defeitos onde erros sistemáticos têm maior probabilidade de estarem presentes. Como erros sistemáticos levam a diversos defeitos do mesmo tipo, o gráfico de Pareto pode ser utilizado para encontrar estas amostras, utilizando as categorias de defeitos como parâmetro discriminador, conforme indicado pelas diretrizes. Assim, em DPPI, as amostras devem estar relacionadas às categorias de defeitos que contêm a maior parte dos defeitos.

Um exemplo de gráfico de Pareto para a especificação funcional de um módulo de um projeto real, utilizando a natureza do defeito como discriminador, pode ser visto na Figura 4.6. Neste exemplo, como mais de sessenta por cento dos defeitos foram

omissões, as omissões seriam selecionadas como amostra para encontrar erros sistemáticos.



**Figura 4.6. Exemplo de Gráfico de Pareto para um Módulo de um Projeto Real.**

Mais informações sobre como categorizar os defeitos podem ser obtidas nos resultados da revisão sistemática (capítulo 2) ou nas diretrizes (capítulo 3).

#### 4.4.2 Encontrar Erros Sistemáticos

Esta tarefa envolve analisar as amostras de defeitos (leitura da descrição destes defeitos) para encontrar erros sistemáticos. Somente os defeitos associados a estes erros sistemáticos devem ser considerados na reunião de análise causal.

Neste momento o moderador pode receber o apoio de representantes dos autores do documento e dos inspetores envolvidos em encontrar os defeitos. Adicionalmente, a revisão sistemática indicou uma abordagem para abstração de erros a partir de defeitos, EAP (*Error Abstraction Process*) (Walia *et al.*, 2006). Entretanto, para a institucionalização de DPPI, pode ser mais interessante inicialmente realizar esta tarefa com reuniões informais, até que a abordagem DPPI esteja estabelecida na organização.

Exemplos concretos de erros sistemáticos, encontrados no contexto de um projeto real, para a categoria omissão, são: não explicitar regras de negócio, não explicitar campos e informações e não explicitar detalhes de navegação. Outros exemplos podem ser encontrados em (Leszak *et al.*, 2002) e nos capítulos 5 e 6 desta tese.

## 4.5 DPPI: Reunião de Análise Causal

Nesta atividade o moderador deve receber o apoio de representantes dos autores (que conhecem o contexto do projeto) e do grupo de processos de engenharia de software (SEPG). Uma descrição das duas tarefas envolvidas nesta atividade encontra-se nas subseções seguintes.

### 4.5.1 Identificar Causas Principais

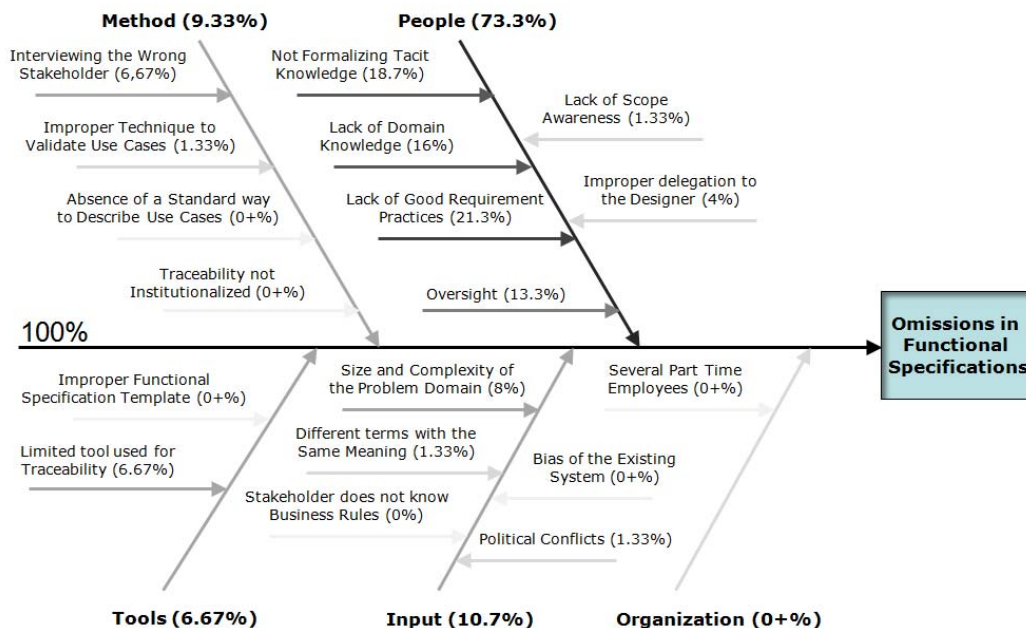
O objetivo desta tarefa é a identificação das principais causas dos erros sistemáticos. Esta tarefa é auxiliada pela existência do modelo causal e pode ser destacada como a principal inovação da abordagem.

Em todas as abordagens analisadas durante as revisões sistemáticas o conhecimento a respeito de relações causais obtido durante cada reunião de análise causal era somente utilizado para iniciar ações para melhorar o processo de desenvolvimento e posteriormente era descartado. Na proposta descrita em (Kalinowski *et al.*, 2008b) as relações causais descobertas em cada reunião de análise causal são utilizadas para alimentar uma rede Bayesiana, permitindo obter um entendimento mais amplo das relações causais dos defeitos de software de uma organização (que causas costumam levar a que tipo de defeito).

Assim, dado o modelo causal elaborado com base em reuniões anteriores de análise causal de projetos similares para a mesma atividade de desenvolvimento, a probabilidade de cada uma das causas ser responsável pelo tipo de defeito relacionado com o erro sistemático sendo analisado pode ser calculada utilizando a inferência diagnóstica da rede Bayesiana. Estas probabilidades podem então ser utilizadas para auxiliar a equipe da reunião de análise causal na identificação das principais causas.

Para facilitar esta tarefa, baseado nas probabilidades de cada causa, um diagrama de causa e efeito probabilístico (Kalinowski *et al.*, 2008b) pode ser construído para o tipo de defeito relacionado com o erro sistemático sendo analisado. Este diagrama foi criado com base no diagrama de causa e efeito (Ishikawa, 1976), sugerido pelas diretrizes para identificar causas. O diagrama de causa e efeito probabilístico estende este diagrama ao (i) mostrar as probabilidades de cada uma das causas estar contribuindo para o tipo de defeito sendo analisado e (ii) representar as causas utilizando tons de cinza, onde os tons mais escuros são utilizados para causas com maior probabilidade. Um exemplo de um diagrama de causa e efeito probabilístico para omissões em especificações funcionais, elaborado com base em dados de um projeto real, encontra-se na Figura 4.7 (os termos foram mantidos em

inglês porque a rede Bayesiana foi alimentada desta forma, para manter a consistência e facilitar o mapeamento entre esta figura e as da rede Bayesiana que se encontram mais adiante).



**Figura 4.7. Diagrama de Causa e Efeito Probabilístico para Omissões em Especificações Funcionais, Elaborado com Base em Dados de um Projeto Real.**

Esta representação pode ser facilmente interpretada pela equipe de análise causal e destaca as causas que possuem maior probabilidade de estar contribuindo para o tipo de defeito analisado. O diagrama é baseado nos resultados de reuniões de análise causal anteriores e será diferente para cada organização, refletindo seu sistema causal específico.

Desta forma, durante a reunião de análise causal, baseado na descrição dos defeitos associados aos erros sistemáticos sendo analisados e com o apoio dos diagramas de causa e efeito probabilísticos (para os tipos de defeito sendo analisados), um novo diagrama de causa e efeito deve ser produzido, contendo as principais causas identificadas para a sessão atual de análise causal. De fato, quando DPPI está sendo aplicada da primeira vez, somente a descrição dos defeitos associados ao erro sistemático estará disponível. A premissa assumida pela abordagem DPPI é que o diagrama de causa e efeito probabilístico pode ajudar a identificar as causas através da utilização do conhecimento armazenado no modelo causal (a rede Bayesiana).

Ao final da reunião, a rede Bayesiana deve ser alimentada com as causas identificadas para o tipo de defeito que foi analisado, de forma que as probabilidades relativas das causas possam ser atualizadas dinamicamente, fechando assim um ciclo

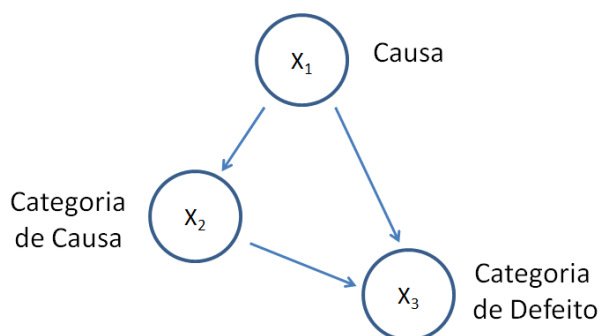


de retroalimentação para a próxima sessão de análise causal de defeitos, com o conhecimento de consenso obtido pela equipe de análise causal na sessão de análise causal atual. Um exemplo concreto de aplicação de DPPI a um projeto real e a descrição do ferramental que permite automatizar seu ciclo dinâmico de retroalimentação encontram-se descritos no capítulo seguinte. Mais detalhes sobre como a rede Bayesiana pode ser montada e utilizada são descritos a seguir.

Ao final de uma sessão de análise causal as causas para cada um dos erros sistemáticos foram determinadas (como consenso de toda a equipe de análise, com diferentes representantes). A rede Bayesiana é então alimentada com este conhecimento, utilizando tuplas de conhecimento contendo a causa, a categoria da causa e a categoria do defeito. Isto é feito para cada um dos defeitos associados ao erro sistemático.

Assim, se um erro sistemático (por exemplo 'não explicitar campos e informações') foi associado a dez defeitos de um determinado tipo (por exemplo 'omissão'), cada uma das causas identificadas para este erro sistemático terá como casos de aprendizado dez tuplas para alimentar a rede. Outro erro sistemático (por exemplo 'inserção de informação ambígua') referente a vinte defeitos de um determinado tipo (por exemplo 'ambiguidade') teria para cada uma de suas causas vinte tuplas como casos de aprendizado. De fato, se um erro sistemático tem mais defeitos associados é porque suas causas ocorreram com maior frequência. Desta forma, a rede é calibrada quantitativamente de acordo com o que efetivamente ocorreu durante a atividade de desenvolvimento.

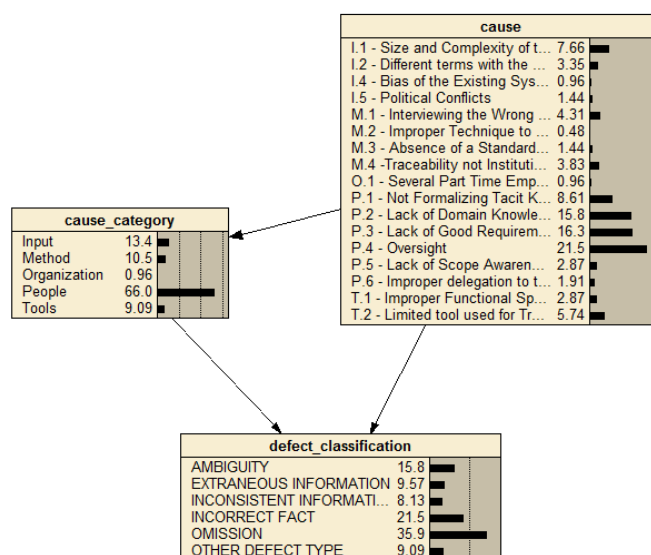
As relações causais de nosso interesse puderam ser facilmente modeladas de forma a definir a estrutura da rede Bayesiana. A rede é composta por três variáveis aleatórias, que são: a causa, a categoria da causa e a categoria de defeito. Pela sua própria definição, a causa e a categoria de causa afetam a categoria de defeito. A causa por sua vez afeta a categoria de causa. Assim, a estrutura da rede ficou conforme representado na Figura 4.8.



**Figura 4.8. Estrutura da Rede Bayesiana para o Aprendizado das Causas das Diferentes Categorias de Defeitos.**

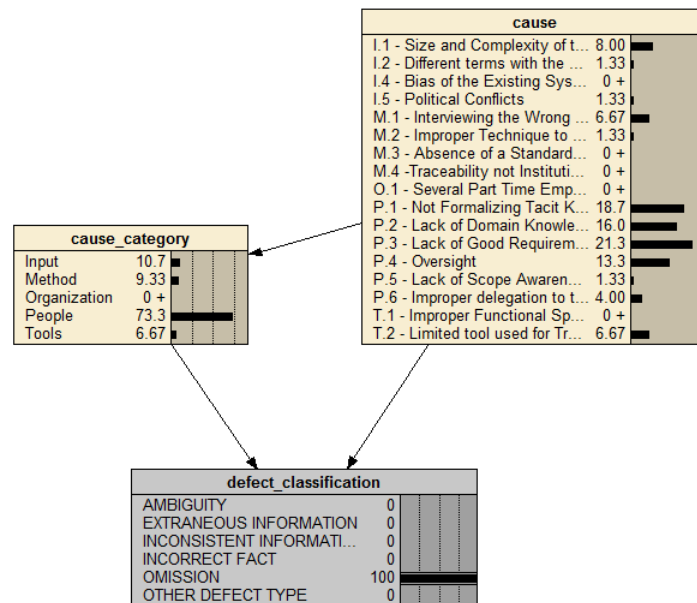
Notem que o erro sistemático não aparece como variável aleatória na modelagem da rede. Isto é proposital, uma vez que o erro sistemático é determinado de maneira informal e serve apenas como instrumento para facilitar a identificação das causas de um subconjunto de defeitos de uma determinada categoria.

Os valores para cada uma das variáveis aleatórias, bem como as probabilidades, são então determinados utilizando as tuplas de dados de cada um dos projetos como casos de aprendizado (considerando a mesma atividade de desenvolvimento e projetos semelhantes). Um exemplo de uma rede Bayesiana (gerado na ferramenta Netica (Netica, 2011)) já preenchida com os valores e probabilidades, no contexto de um projeto real pode ser visto na Figura 4.9. Nesta Figura nenhuma inferência foi realizada e os percentuais ao lado de cada um dos valores representam apenas a distribuição destes valores entre as tuplas utilizadas para o aprendizado. A Figura encontra-se em inglês porque as tuplas utilizadas para o aprendizado estavam em inglês.



**Figura 4.9. Rede Bayesiana com Valores e Probabilidades Preenchidas (Gerada a partir de Tuplas de um Projeto Real utilizando a Ferramenta Netica).**

Tendo obtido a rede Bayesiana, a inferência diagnóstica é facilmente realizada. A Figura 4.10 mostra a inferência diagnóstica para a categoria de defeitos 'omissão'. É possível observar que, no contexto organizacional onde este projeto foi realizado, a maioria das omissões em especificações funcionais é causada por fatores relacionados à categoria de causa 'pessoas' (73,3%) e as causas mais comuns para as omissões são o 'não uso de boas práticas da engenharia de requisitos' (21,3%) e a 'não formalização de conhecimento tácito' (18,7%).



**Figura 4.10. Inferência Bayesiana para a Categoria de Defeitos 'Omissões'.**

Tendo a inferência da rede Bayesiana, o diagrama de causa e efeito probabilístico é obtido através da simples transcrição dos valores utilizando tons mais escuros para as causas com probabilidades mais altas (a causa de maior probabilidade fica com cor preta e as demais em tons de cinza relativos a este valor). A inferência ilustrada na Figura 4.10 resultou no diagrama de causa e efeito probabilístico da Figura 4.7.

#### 4.5.2 Propor Ações para Prevenir as Causas

Conforme destacado por Card (2005), a simples identificação das causas fornece apenas um diagnóstico e não é suficiente para evitar sua recorrência em projetos futuros, é preciso tomar ações de melhoria para prevenir as causas. Ou seja, melhorar o processo da atividade de desenvolvimento sendo analisada.

Assim, nesta tarefa, um *brainstorm* deve ser realizado pela equipe na reunião de análise causal (que inclui representantes do grupo de processos) para identificar ações de melhoria para os ativos de processo visando a prevenção das causas identificadas. Neste ponto é possível ainda disponibilizar uma lista com as ações propostas anteriormente e seus resultados efetivos, para ajudar a equipe no *brainstorm*. Os resultados da reunião (taxa de defeitos atual, objetivos de melhoria, principais categorias de defeito, erros sistemáticos, causas identificadas e ações propostas) devem ser documentados.

## 4.6 DPPI: Melhoria da Atividade de Desenvolvimento

Por fim, as propostas de ação identificadas devem ser implementadas por uma equipe comprometida com as melhorias e gerenciadas até sua conclusão. O status das ações propostas e o esforço para implementá-las deve ser registrado.

Após a implementação das ações, as mudanças feitas no processo (da atividade de desenvolvimento para a qual defeitos foram analisados) devem ser comunicadas à equipe de desenvolvimento. Os resultados da implementação das ações serão visíveis no gráfico de controle da atividade de análise dos resultados da próxima aplicação de DPPI à mesma atividade de desenvolvimento, fechando a retroalimentação a respeito da efetividade das ações implementadas.

Um template que pode ser utilizado para apoiar a aplicação manual de DPPI encontra-se no Anexo D.

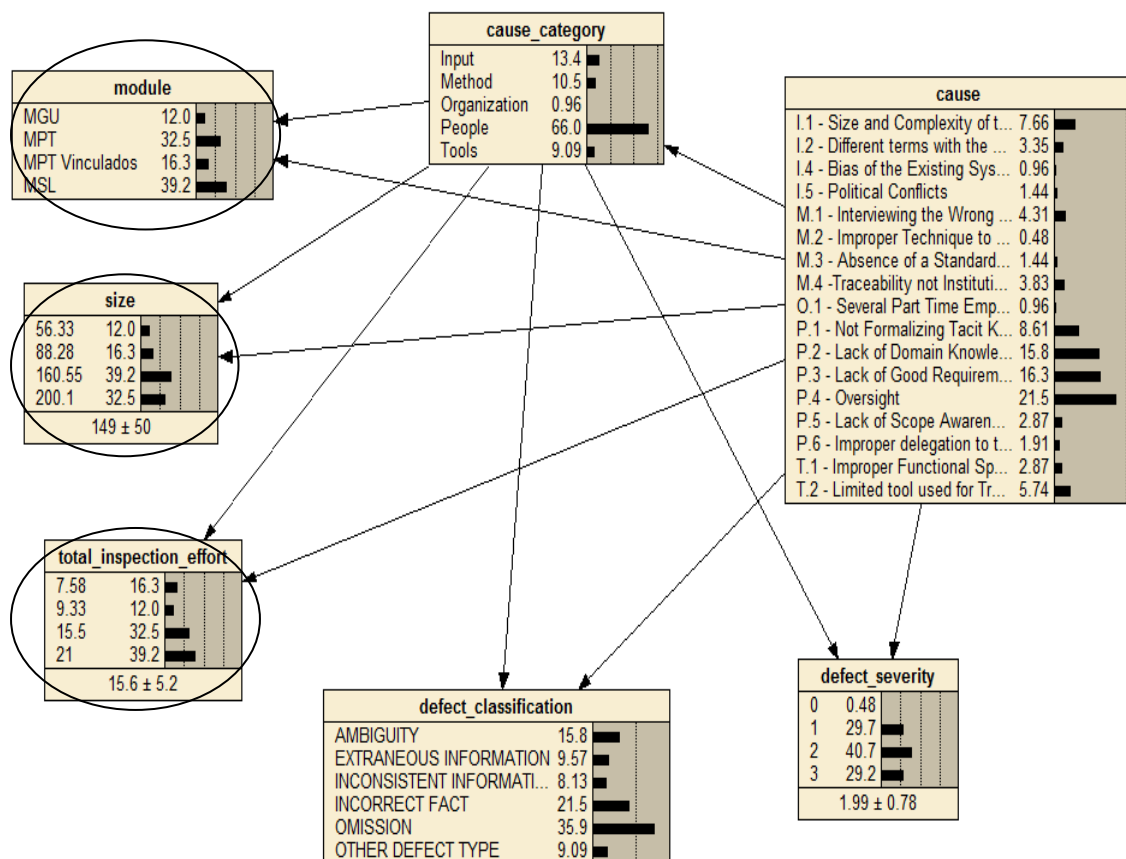
## 4.7 Outras Possibilidades de Uso para DPPI

Engenheiros de software podem utilizar DPPI para analisar as causas de defeitos de software atendendo às práticas da área de processos CAR (*Causal Analysis and Resolution*) do CMMI. A Tabela 1 contém um mapeamento das práticas específicas desta área de processos para as atividades da abordagem DPPI. Orientações adicionais podem ser encontradas nas diretrizes do capítulo 3.

**Tabela 4.1. Mapeamento das Práticas Específicas da Área de Processos CAR do CMMI para as Atividades de DPPI.**

<b>Objetivos e Práticas Específicas da Área de Processo CAR do CMMI</b>	<b>Atividades de DPPI Relacionadas</b>
SG 1 Determinar as Causas de Defeitos	Preparação para Análise Causal e Reunião de Análise Causal
SP 1.1 Selecionar Dados de Defeitos para Análise	Preparação para Análise Causal
SP 1.2 Analisar as Causas	Reunião de Análise Causal
SG 2 Tratar as Causas de Defeitos	Análise da Atividade de Desenvolvimento e Melhoria da Atividade de Desenvolvimento
SP 2.1 Implementar as Propostas de Ação	Melhoria da Atividade de Desenvolvimento
SP 2.2 Avaliar o Efeito de Mudanças	Análise da Atividade de Desenvolvimento
SP 2.3 Registrar Dados	Ao longo de DPPI (taxa de defeitos atual, objetivos de melhoria, principais categorias de defeito, erros sistemáticos, causas identificadas e ações propostas)

Outra possibilidade de uso é para predição de defeitos. Como o modelo causal contendo o conhecimento a respeito de defeitos obtido pela aplicação de DPPI é armazenado como uma rede Bayesiana ele pode ser utilizado para dois propósitos, diagnóstico e predição (Pearl, 2000). Nesta tese exploramos somente a perspectiva de diagnóstico. Entretanto, a rede Bayesiana pode ser atravessada na direção contrária, para apoiar a definição de estratégias de mitigação de riscos realizando simulações de cenários “what-if” sobre as causas. Por exemplo, “O que acontecerá com o perfil de defeitos se os interessados inapropriados foram entrevistados?”. Além disto, normalmente se tem mais informações sobre os defeitos do que somente sua classificação, categoria de causa e causa. A Figura 4.11 mostra uma rede obtida a partir de dados de defeitos de um projeto real que continham informações adicionais sobre severidade dos defeitos, módulo em que foi encontrado, tamanho do módulo e esforço da inspeção que revelou os defeitos (em horas). A rede ilustra como a causa e a categoria de causa afetam estas variáveis.

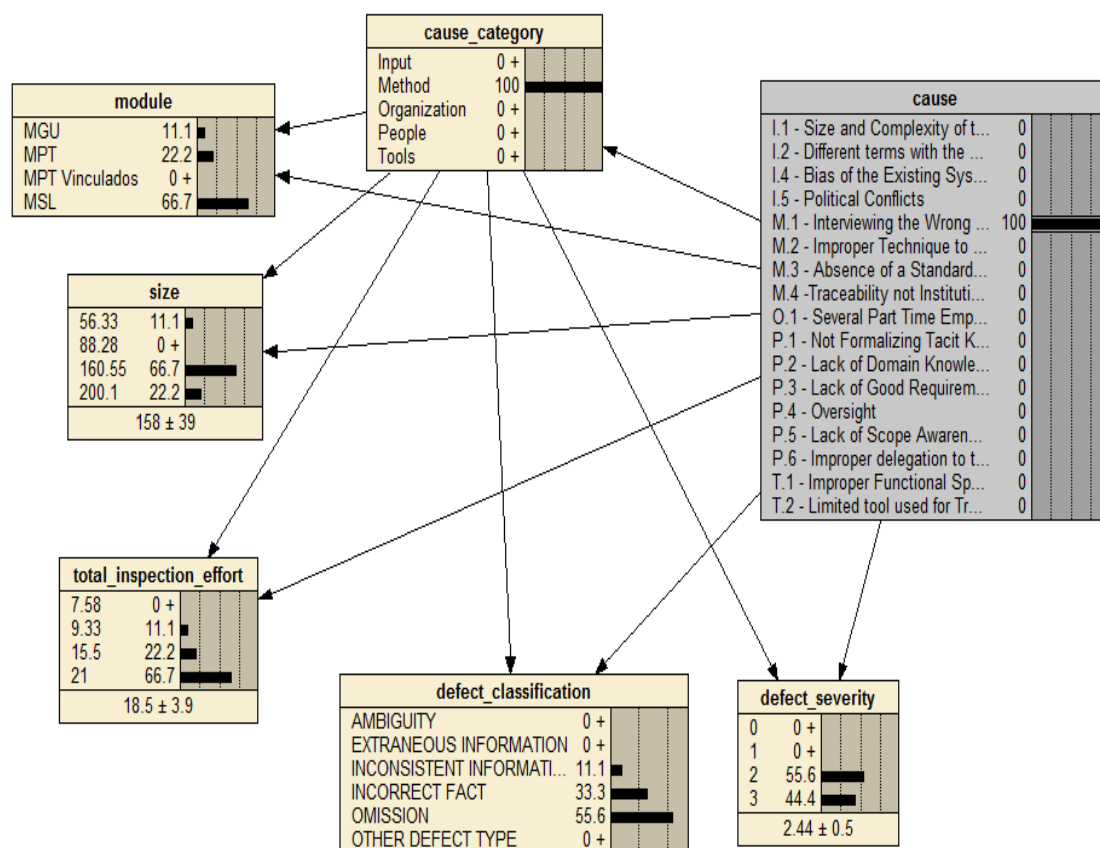


**Figura 4.11. Rede Bayesiana para Dados Adicionais sobre Defeitos.**

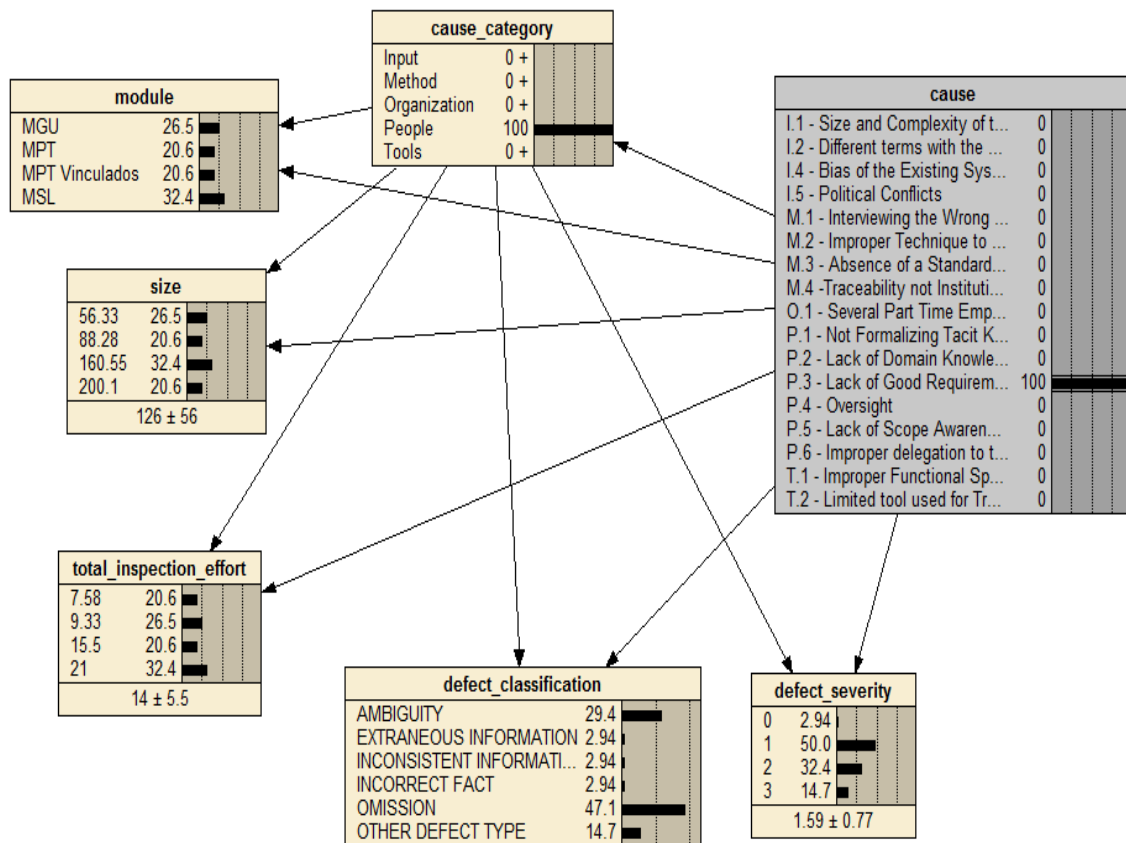
Note que nem todos os relacionamentos representados na rede são de fato relacionamentos causais (os destacados com círculos não são), mas que os números

obtidos da rede representam correlações que podem oferecer interpretações interessantes sobre a realidade da empresa. Os números da Figura 4.12, por exemplo, representam a inferência para a causa “Entrevistar o Interessado Inapropriado”. A Figura 4.13, por sua vez representa a inferência para a causa “Falta de Conhecimento das Boas Práticas de Engenharia de Requisitos”.

É possível observar que, para a realidade desta organização, entrevistar o interessado inapropriado tende a resultar em defeitos de severidade alta, tendo além de omissões (o tipo de defeito mais comum neste projeto) muitos fatos incorretos e informações inconsistentes. A falta de conhecimento de boas práticas de engenharia de requisitos, por sua vez, tende a resultar em defeitos de severidade mais baixa, sendo, em sua maioria, omissões e ambigüidades. É possível também observar que, esta causa está mais concentrada nos primeiros módulos (que aparecem na Figura com as siglas “MSL” e “MGU”). Isto já era esperado, uma vez que inspeções foram empregadas ao longo do projeto e estas implicam no aprendizado das boas práticas por parte da equipe (Fagan, 1976).



**Figura 4.12. Inferência Preditiva para a Causa “Entrevistar o Interessado Inapropriado”.**



**Figura 4.13. Inferência Preditiva para a Causa “Falta de Conhecimento de Boas Práticas de Engenharia de Requisitos”.**

É importante ressaltar que explorar o conhecimento contido na rede Bayesiana para predição representa apenas uma possibilidade adicional e que está fora do escopo desta tese. Por este motivo, o benefício deste uso não será investigado em maiores detalhes.

## 4.8 Considerações Finais

Este capítulo apresentou uma abordagem para melhoria de processos com base em análise causal de defeitos, denominada DPPI (Kalinowski *et al.*, 2010). DPPI foi projetada contendo quatro atividades: (1) Análise da Atividade de Desenvolvimento, (2) Preparação para Análise Causal, (3) Reunião de Análise Causal e (4) Melhoria da Atividade de Desenvolvimento. As principais tarefas destas atividades, bem como os papéis envolvidos em sua execução e as técnicas sugeridas para apoiar sua realização foram detalhados. Entre as principais contribuições da proposta de DPPI destacamos:

- A integração de conhecimento obtido em sucessivas reuniões de análise causal para prover um maior entendimento a respeito das relações de causa e efeito dos defeitos da organização. Isto trata uma das

oportunidades de pesquisa identificadas nas revisões sistemáticas e destacada em (Kalinowski *et al.*, 2008a). O primeiro esforço no sentido de tratar esta oportunidade se deu na concepção inicial da abordagem, descrita em (Kalinowski *et al.*, 2008b), que posteriormente evoluiu para se tornar DPPI (Kalinowski *et al.*, 2010). Até a presente data, de acordo com os resultados da revisão sistemática conduzida, nenhuma outra abordagem considera esta integração. Tal integração permite estabelecer e manter modelos causais para a organização, fechando explicitamente um ciclo de retroalimentação para o conhecimento sobre causas de defeitos, com base nos resultados de cada sessão de análise causal. Estes modelos podem então ser utilizados para apoiar o raciocínio diagnóstico, facilitando a identificação das principais causas dos defeitos sob análise em uma nova reunião de análise causal. Com estes modelos é possível, por exemplo, apoiar o entendimento, com base no aprendizado obtido de projetos similares da organização, sobre quais causas normalmente contribuem para um determinado tipo de defeitos.

- A criação do diagrama de causa e efeito probabilístico (Kalinowski *et al.*, 2008b), que permite representar informações de incerteza (probabilidades) provenientes de redes Bayesianas em uma notação facilmente interpretada por equipes de análise causal.
- O detalhamento do apoio para a realização de análise causal de defeitos, baseado no conhecimento obtido na revisão sistemática e nas diretrizes obtidas a partir de seus resultados (Kalinowski *et al.*, 2008a) e da atualização deste conhecimento (capítulo 3 desta tese). O uso destas diretrizes permitiu detalhar as atividades de prevenção de defeitos em tarefas mais específicas, fornecendo detalhes adicionais sobre as técnicas a serem utilizadas para realizar estas tarefas eficientemente.
- A integração da prevenção de defeitos na estratégia de medição e controle das atividades de engenharia do software para as quais a prevenção de defeitos está sendo realizada, permitindo observar se as melhorias implementadas para estas atividades efetivamente trouxeram benefícios. Isto foi feito utilizando as métricas definidas nas diretrizes para medir e controlar o desempenho destas atividades.
- O apoio para a implementação da área de processos CAR (*Causal Analysis and Resolution*) do CMMI. DPPI trata todas as práticas específicas da área de processos CAR. Desta forma, seguir a abordagem DPPI resulta em



aderência a estas práticas em relação à análise causal de defeitos de software provenientes de inspeções.

- A possibilidade de explorar o conhecimento obtido em DPPI para outros propósitos, como, por exemplo, predição de defeitos, podendo representar um ponto de partida para outras pesquisas.

Tendo elaborado a abordagem DPPI, diversas questões associadas a esta abordagem surgem. Por exemplo:

- O esforço envolvido na utilização de DPPI torna o uso da abordagem (e das boas práticas nela previstas) viável para organizações desenvolvedoras de software?
- O uso da abordagem de DPPI para identificar causas de defeitos traz benefícios quando comparado à identificação de causas de forma ad-hoc (sem o uso do modelo causal)?
  - Resulta em uma identificação mais eficaz das causas?
  - Reduz o esforço da identificação das causas?
  - Resulta em uma maior satisfação da equipe de análise causal?

Para investigar a primeira destas questões, o capítulo seguinte descreve a experiência de aplicar DPPI à atividade de especificação funcional de um projeto de desenvolvimento de software Web real de larga escala e o apoio ferramental construído com base nesta experiência para reduzir o esforço de sua aplicação e automatizar o seu ciclo de retroalimentação (em relação às causas dos defeitos e à efetividade das ações implementadas). A segunda questão foi avaliada através de um estudo experimental descrito no Capítulo 6.

## **CAPÍTULO 5 - PROVA DE CONCEITO E APOIO FERRAMENTAL**

*No decorrer do capítulo a experiência de aplicar DPPI a um projeto de desenvolvimento de software real é descrita e o apoio ferramental construído para facilitar sua aplicação e automatizar o ciclo de retroalimentação proposto é apresentado.*

### **5.1 Introdução**

Visando avaliar a viabilidade de aplicar DPPI a projetos reais, como prova de conceito, DPPI foi aplicada manualmente e retroativamente para realizar análise causal de defeitos na atividade de especificação funcional de um projeto Web real de larga escala.

Esta experiência ajudou a obter *insights* adicionais a respeito dos requisitos para construir um apoio ferramental que pudesse automatizar o seu ciclo de retroalimentação (a respeito de causas de defeitos e da eficiência das ações implementadas).

As seções seguintes descrevem, respectivamente, a prova de conceito e o apoio ferramental construído.

### **5.2 Prova de Conceito: Aplicação de DPPI a um Projeto Real**

Como prova de conceito, DPPI foi aplicada manualmente e retroativamente para realizar análise causal de defeitos na atividade de especificação funcional de um projeto Web real de larga escala. O escopo deste projeto, chamado SIGIC, foi desenvolver um novo sistema de informação para gerenciar as atividades da Fundação COPPETEC. Assim, o projeto envolveu diversos departamentos desta Fundação, tais como: recursos humanos, financeiro, protocolo, entre outros.

O sistema a ser desenvolvido foi modularizado e implementado utilizando um ciclo de vida iterativo e incremental. Baseado neste ciclo de vida, um processo de desenvolvimento foi definido, em que inspeções de software eram realizadas nas especificações funcionais de cada um dos módulos (Kalinowski *et al.*, 2007), utilizando o framework ISPIS (Kalinowski e Travassos, 2004).

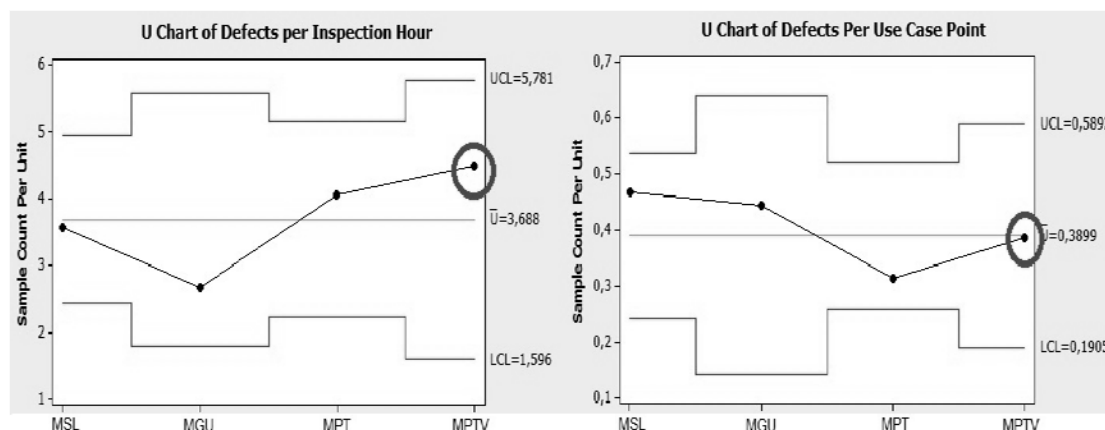
Ao todo, mais de 10 iterações foram realizadas e mais de 200 casos de uso foram especificados, implementados e entregues ao cliente, ao longo de um período de desenvolvimento de mais de três anos. No final do projeto, todos os dados das inspeções estavam disponíveis, incluindo detalhes de mais de 1000 defeitos encontrados e removidos das especificações funcionais antes da efetiva implementação.

Neste contexto, DPPI foi aplicada retroativamente à atividade de especificação funcional do quarto módulo desenvolvido, referente ao protocolo de solicitações de projetos vinculados, chamado MPTV.

Os detalhes de como cada uma das atividades de DPPI (exceto a atividade de melhoria, que não pode ser realizada, uma vez que a aplicação foi retroativa) pôde ser realizada são descritos a seguir.

### 5.2.1 Realizando a Análise da Atividade de Desenvolvimento

Para entender o cenário atual e as mudanças nas taxas de defeitos comparando o módulo MPTV com os módulos anteriores (MSL, MGU e MPT), conforme definido em DPPI, um gráfico U foi plotado, utilizando a ferramenta MiniTab (MiniTab, 2011), para defeitos por hora de inspeção e defeitos por unidade de tamanho (neste caso pontos de caso de uso). Este gráfico está ilustrado na Figura 5.1.



**Figura 5.1. Gráfico U para Defeitos por Hora de Inspeção e Defeitos por Unidade de Tamanho (Pontos de Caso de Uso).**

Neste gráfico foi possível notar que o número de defeitos revelados por hora de inspeção no MPTV foi mais alto do que a média, enquanto o número de defeitos por ponto de caso de uso foi próximo da média. O gráfico U mostra um aparente cenário dentro do esperado (nenhum dos limites de controle foi ultrapassado), embora o número de pontos seja insuficiente para afirmações relativas à estabilidade.

Como a especificação funcional é a primeira atividade do desenvolvimento, a métrica PIQ (*Phase Input Quality* – percentual de defeitos proveniente de fases anteriores) tem valor 0 e a métrica POQ (*Phase Output Quality* – percentual de defeitos propagando desta atividade para as demais), que é diagnóstica em relação ao processo como um todo, não pôde ser calculada, uma vez que dados de defeitos de outras atividades não estavam disponíveis. Mais detalhes sobre as métricas PIQ e POQ podem ser obtidos no capítulo 3.

Dado este cenário, o objetivo quantitativo de melhoria estabelecido foi reduzir a taxa de defeitos por unidade de tamanho de 0,39 (a atual) para 0,31 (a melhor taxa obtida até então).

### 5.2.2 Realizando a Preparação para Análise Causal

Esta atividade envolve aplicar o gráfico de Pareto para selecionar amostras e encontrar erros sistemáticos nestas amostras. O gráfico de Pareto elaborado para a análise dos defeitos do módulo MPTV (gerado utilizando a ferramenta MiniTab) está ilustrado na Figura 5.2.

O projeto considerou as categorias de defeitos descritas em (Shull, 1998). É possível observar que a maioria dos defeitos é do tipo fato incorreto e que a soma de fatos incorretos e omissões representa em torno de 60% do total de defeitos encontrados. Com o apoio do gráfico, os 12 defeitos do tipo fato incorreto (de um total de 34 defeitos) foram selecionados como amostra para identificar os erros sistemáticos.

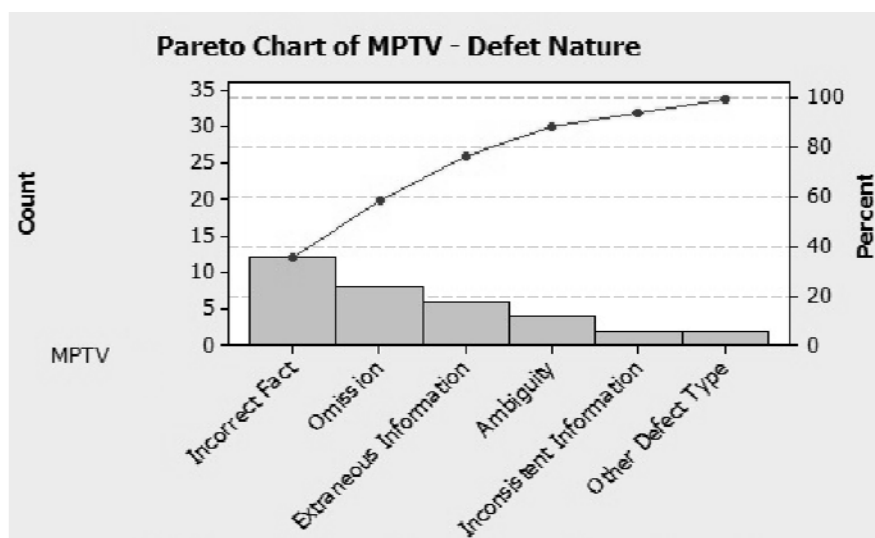


Figura 5.2. Gráfico de Pareto para os Defeitos do Módulo MPTV.

A partir da análise da descrição dos defeitos os seguintes dois erros sistemáticos puderam ser encontrados: “Escrever regras de negócio inválidas”

(referente a 4 dos 12 defeitos) e “Ligar descrições de casos de uso de forma incorreta” (referente aos outros 8 dos 12 defeitos).

Para deixar mais claro o significado destes erros sistemáticos, um exemplo de defeito que foi atribuído ao erro sistemático “Escrever regras de negócio inválidas” é “UC01 - O funcionário do setor de protocolo não pode alterar o projeto/fundo selecionado pelo coordenador! Baseado em quê ele faria essa alteração? Assim, formulário só deveria ter campo read-only”. Um exemplo de defeito atribuído ao erro sistemático “Ligar descrições de casos de uso de forma incorreta” é “UC03 - O fluxo A7 deveria retornar para o passo 2 ao invés do 3”.

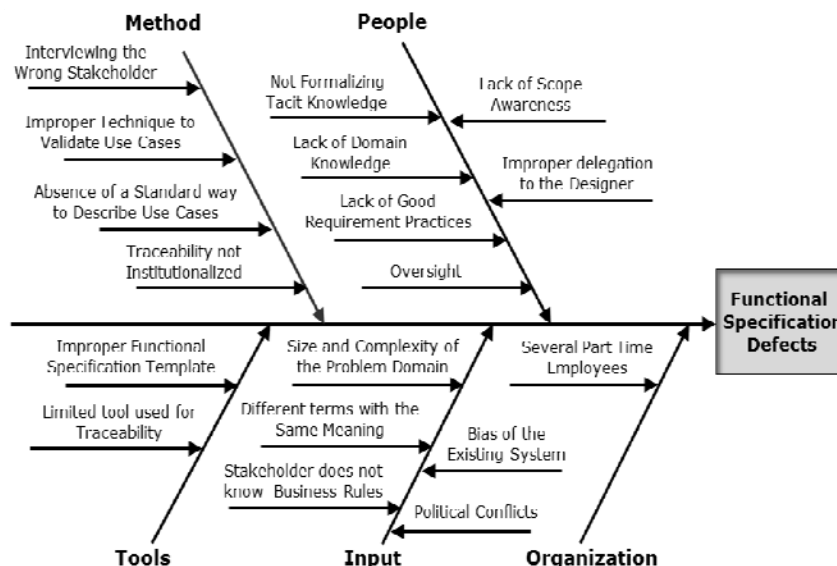
### 5.2.3 Realizando a Reunião de Análise Causal

Antes de descrever como a atividade de reunião de análise causal foi realizada, alguns detalhes serão fornecidos sobre como o modelo causal pôde ser construído retroativamente.

**Construção Retroativa da Rede Bayesiana.** Como nenhuma sessão anterior de análise causal havia sido realizada no projeto SIGIC, o modelo causal Bayesiano teve que ser construído retroativamente com base nos dados de defeitos dos módulos anteriores, para que pudesse ser usado para a sessão do MPTV.

Para isto, primeiramente uma reunião foi conduzida com a equipe responsável pela análise e gerência no projeto SIGIC (um total de 4 participantes do projeto) para realizar um *brainstorm* sobre possíveis causas para defeitos de especificação funcional, baseado nas categorias de causa (método, pessoas, ferramentas, entradas e organização) sugeridas nas diretrizes. O resultado deste *brainstorm* pode ser visto na Figura 5.3 (os termos foram mantidos em inglês, porque a rede foi alimentada desta forma, para obter consistência com as demais figuras apresentados no restante do capítulo).

Após este *brainstorm*, descrições dos 163 defeitos de especificação funcional que haviam sido encontrados em módulos desenvolvidos antes do MPTV foram analisadas com o apoio dos autores destes módulos, de membros do grupo de processos (SEPG) e de inspetores responsáveis por encontrar estes defeitos. Durante este processo, cada um dos defeitos foi associado a uma das causas do *brainstorm*. Outras causas poderiam ter sido adicionadas, mas para os 163 defeitos analisados isto não foi necessário.



**Figura 5.3. Brainstorm sobre Possíveis Causas de Defeitos.**

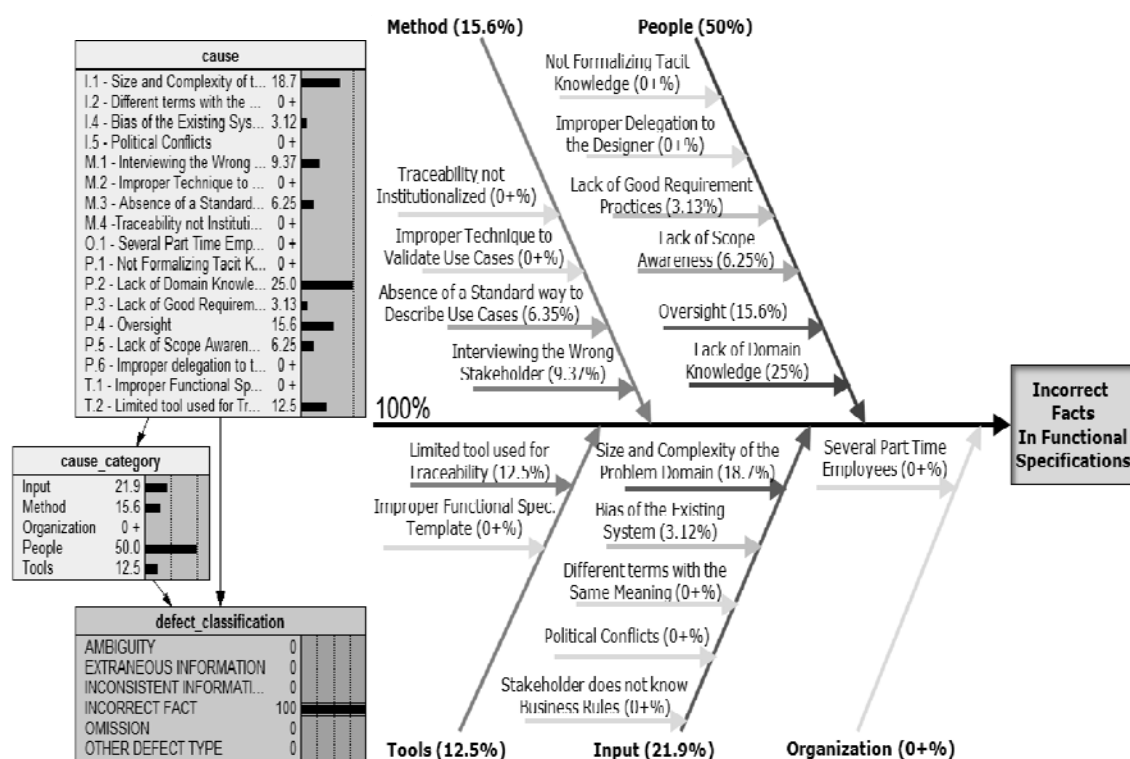
Ao final deste processo, cada defeito continha as seguintes informações: o módulo, o tamanho deste módulo, o esforço de inspeção, a categoria do defeito, a severidade do defeito, a causa e a categoria de causa (estes últimos dois representam os dados novos adicionados). Estas eram as variáveis aleatórias candidatas a fazer parte da rede Bayesiana. Conforme descrito no capítulo anterior, as relações causais de nosso interesse puderam ser facilmente modeladas utilizando três destas variáveis, com a causa e a categoria da causa ambas afetando a categoria do defeito (veja Figura 4.8). Isto definiu a estrutura de nossa rede Bayesiana. Tendo definido a estrutura, os valores e as probabilidades puderam ser obtidos utilizando os dados dos defeitos como casos de aprendizado.

É importante ressaltar que, neste caso, a rede Bayesiana foi construída de forma retroativa, uma vez que nenhuma sessão anterior de análise causal havia sido conduzida ao longo do projeto utilizando DPPI. Caso contrário, os casos de aprendizado poderiam ter sido obtidos a partir dos resultados das sessões anteriores (realizadas em módulos anteriores do mesmo projeto ou em módulos de projetos similares) e a rede Bayesiana poderia ter sido construída dinamicamente, sem a necessidade de classificar os dados dos defeitos um a um. Mais detalhes sobre a construção dinâmica da rede podem ser obtidos no capítulo 4.

**Conduzindo a Reunião de Análise Causal.** No início da atividade de reunião de análise causal o diagrama de causa e efeito probabilístico deveria estar disponível para as categorias de defeitos associadas aos erros sistemáticos sendo analisados. Note que, seguindo a abordagem DPPI, um erro sistemático sempre estará associado a defeitos da mesma categoria.

Este diagrama pode ser construído transcrevendo a inferência Bayesiana para a categoria de defeito em questão. A inferência Bayesiana mostra as probabilidades com que cada causa levou a uma determinada categoria de defeitos, considerando os dados de módulos passados (ou de projetos similares) do mesmo contexto organizacional.

A inferência diagnóstica Bayesiana e o diagrama de causa e efeito probabilístico correspondente para fatos incorretos em especificações funcionais do projeto SIGIC estão ilustrados na Figura 5.4. Esta figura mostra que, neste projeto, as causas para fatos incorretos usualmente são “Falta de Conhecimento do Domínio” (25%), “Tamanho e Complexidade do Domínio do Problema” (18,7%), “Descuido” (15,6%) e “Ferramental limitado para Rastreabilidade” (12,5%).



**Figura 5.4. Inferência Bayesiana e o Diagrama de Causa e Efeito Probabilístico Correspondente.**

De fato, na reunião de análise causal do modulo MPTV as causas identificadas para o erro sistemático “Escrever regras de negócio inválidas” foram “Falta de Conhecimento do Domínio” e “Tamanho e Complexidade do Domínio do Problema” e a causa identificada para o erro sistemático “Ligar descrições de casos de uso de forma incorreta” foi “Descuido”. Estas causas correspondem às três causas do diagrama de causa e efeito probabilístico com maior probabilidade e foram identificadas através da leitura das descrições dos defeitos envolvendo o autor, inspetores e membros do SEPG. Com base nesta experiência, aparentemente, o

diagrama de causa e efeito probabilístico pode se mostrar útil, embora uma investigação adicional desta utilidade seja necessária (capítulo 6).

As ações propostas para tratar as causas “Falta de Conhecimento do Domínio” e “Tamanho e Complexidade do Domínio do Problema” foram: (i) estudar o domínio e o sistema pré-existente e (ii) modificar o template de especificação funcional, criando uma sessão separada para as regras de negócio, listando separadamente a descrição dos casos de uso.

Para a causa “Descuido” a ação foi publicar exemplos relevantes de defeitos da categoria “Fato Incorreto” relacionados a descuidos e apresentá-los à equipe de especificação funcional.

De acordo com DPPI, neste momento a rede Bayesiana seria retroalimentada com 16 novos casos de aprendizado, 4 tuplas contendo o tipo de defeito fato incorreto e associando-o à causa “Falta de Conhecimento do Domínio”, 4 tuplas contendo o tipo de defeito fato incorreto e associando-o à causa “Tamanho e Complexidade do Domínio do Problema” e 8 tuplas contendo o tipo de defeito fato incorreto e associando-o à categoria “Descuido”. Como somente fatos incorretos foram analisados nesta sessão de análise causal, nenhum aprendizado adicional foi obtido a respeito das demais categorias de defeitos. Esta retroalimentação é sistemática e pode ser facilmente automatizada.

A próxima atividade de DPPI seria a melhoria da atividade de desenvolvimento. Entretanto, conforme mencionado anteriormente, esta atividade não foi realizada, uma vez que a prova de conceito foi realizada retroativamente e o próximo módulo já havia sido especificado. Por este mesmo motivo, resultados quantitativos da aplicação de DPPI (que seriam observados nos gráficos U da análise da atividade de desenvolvimento para o próximo módulo) não puderam ser obtidos.

#### **5.2.4 Considerações Finais da Prova de Conceito**

A aplicação manual e retroativa de DPPI a um projeto de desenvolvimento real permitiu a obtenção de *insights* adicionais a respeito de sua viabilidade de aplicação e dos requisitos para apoio ferramental.

As inspeções realizadas no contexto do projeto ocorreram com o apoio da ferramenta ISPIS (Kalinowski e Travassos, 2004), que facilitou a condução de um processo de inspeção bem definido que apresentou resultados estáveis e permitiu o registro dos dados de defeitos. Entretanto, a condução das inspeções está fora do escopo de DPPI e elas poderiam também ter sido realizadas de forma manual.

Durante a aplicação de DPPI descrita na prova de conceito as ferramentas MiniTab e Netica foram utilizadas. A ferramenta MiniTab foi utilizada para gerar os



gráficos de controle U e o gráfico de Pareto. Ela foi escolhida por atender às necessidades de geração destes gráficos. A geração manual destes gráficos iria requerer muito esforço. Entretanto é importante ressaltar que existem outras ferramentas estatísticas que poderiam ser utilizadas para a geração destes gráficos. A ferramenta Netica, por sua vez, foi utilizada para gerar a rede Bayesiana a partir dos casos de aprendizado e para realizar a inferência Bayesiana. Novamente existem outras ferramentas que atendem a este propósito e que poderiam ser utilizadas.

Conforme mencionado anteriormente, DPPI trata todas as práticas específicas da área de processos CAR do CMMI. Entretanto, mesmo com a aplicação sendo moderada pelo criador da abordagem, seguir as atividades de DPPI e manter toda a documentação relacionada (taxas de defeitos, objetivos de melhoria, principais categorias de defeito identificadas, erros sistemáticos, causas identificadas e propostas de ação) utilizando somente templates (como o disponível no Anexo D) sem um apoio ferramental se mostrou uma tarefa complexa durante a experiência de aplicar DPPI ao módulo MPTV.

Adicionalmente, a geração manual e retroativa dos casos de aprendizado para alimentar a rede Bayesiana exigiu grande esforço (além de ser uma tarefa sujeita a erros). Ao todo foram gastas mais de 80 horas da equipe com sessões de brainstorm para identificar as possíveis causas e classificar a base de defeitos para obter os casos de aprendizado (defeitos associados às suas causas). Este esforço seria proporcional ao tamanho da base de defeitos e pode não ser aceitável em aplicações industriais. Entretanto, a abordagem prevê a geração dinâmica destes casos com os próprios resultados de cada uma das sessões de análise causal, que, se automatizada, não exigiria esforço adicional.

De maneira geral, a experiência de aplicar DPPI a um projeto Web real de larga escala mostrou sua viabilidade de aplicação em projetos reais, desde que apoio adequado seja fornecido para sua institucionalização e a retroalimentação da rede Bayesiana seja automatizada.

Assim, requisitos para que um *framework* computacional para apoiar a realização de DPPI pudesse ser construído foram identificados, de modo que este *framework* forneça um passo a passo para a realização das atividades previstas na abordagem, mantendo toda a documentação necessária como simples consequência de sua utilização. Adicionalmente, o *framework* construído estabelece e mantém os casos de aprendizado e realiza o aprendizado e a inferência bayesiana sem esforço adicional, utilizando os próprios resultados de cada sessão de análise causal, automatizando desta forma o ciclo de retroalimentação da abordagem DPPI (Figura 4.4). Os requisitos para o *framework* foram definidos a partir da abordagem DPPI e de

seu template (Anexo D). Sua construção se deu no contexto de um projeto final de Engenharia da Computação da UFRJ (Paes, 2010). Um resumo de seus requisitos, de seu projeto e de suas principais funcionalidades encontra-se no Anexo E.

### **5.3 Apoio Ferramental: DPPI Framework**

O propósito de construir um framework computacional apoiando a aplicação de DPPI foi facilitar seu uso pela indústria, fornecendo um apoio passo a passo que embute o conjunto de práticas de análise causal identificadas pelas diretrizes (Kalinowski *et al.*, 2008a) e consideradas em DPPI (Kalinowski *et al.*, 2010). Adicionalmente, o framework resultante automatiza o ciclo de retroalimentação proposto, permitindo estabelecer e manter os casos de aprendizado da rede Bayesiana e realizando a inferência que permite elaborar o diagrama de causa e efeito probabilístico. Maiores detalhes sobre a ferramenta podem ser obtidos em (Paes, 2010) ou no resumo contido no Anexo E. Uma visão geral de sua arquitetura e de como a ferramenta poderia ser empregada para realizar a prova de conceito, permitindo entender sua utilização, se encontra a seguir.

#### **5.3.1 Arquitetura da Ferramenta DPPI Framework**

Seis componentes foram identificados para a arquitetura: o componente principal, um componente para caracterizar a sessão de análise causal e um componente para cada uma das quatro atividades de DPPI. A Figura 5.5 ilustra a arquitetura proposta.

O componente principal da ferramenta a torna acessível pela Web e assegura que a abordagem seja seguida de forma sistemática. Ao longo de DPPI, ele provê ferramentas estatísticas externas (previamente registradas) para que sejam usadas para a geração dos gráficos U e Pareto. A decisão de utilizar ferramentas externas para esta geração se deu em função da grande variedade de ferramentas estatísticas existentes que já tratam esta geração adequadamente.

Considerando as entradas da ferramenta, os dados de defeitos do módulo atual podem ser importados em formato XML pelo componente de caracterização do projeto, onde o nome do projeto, do módulo e a atividade de desenvolvimento devem ser informados. Para isto, os dados dos defeitos devem seguir o formato especificado em um XML Schema. O XML Schema em questão é o mesmo utilizado em ISPIS para importar defeitos (Kalinowski e Travassos, 2004). Desta forma dados de defeitos gerados em ISPIS podem ser importados diretamente. Para importar dados de



**Defect Causal Analysis Characterization**

Project:	SIGIC ▾	<a href="#">Add Project</a>
Module:	MPTV ▾	<a href="#">Add Module</a>
Development Activity:	Functional Specification ▾	<a href="#">Add Development Activity</a>

[Upload XML Defect List](#)

[Start DPPI](#)

**Figura 5.6. Caracterização da Sessão de Análise Causal.**

Tendo realizado estas definições iniciais é possível iniciar a aplicação de DPPI. Ao selecionar o início de DPPI o componente principal direcionará a sessão de análise causal para a primeira atividade de DPPI, que é a análise da atividade de desenvolvimento.

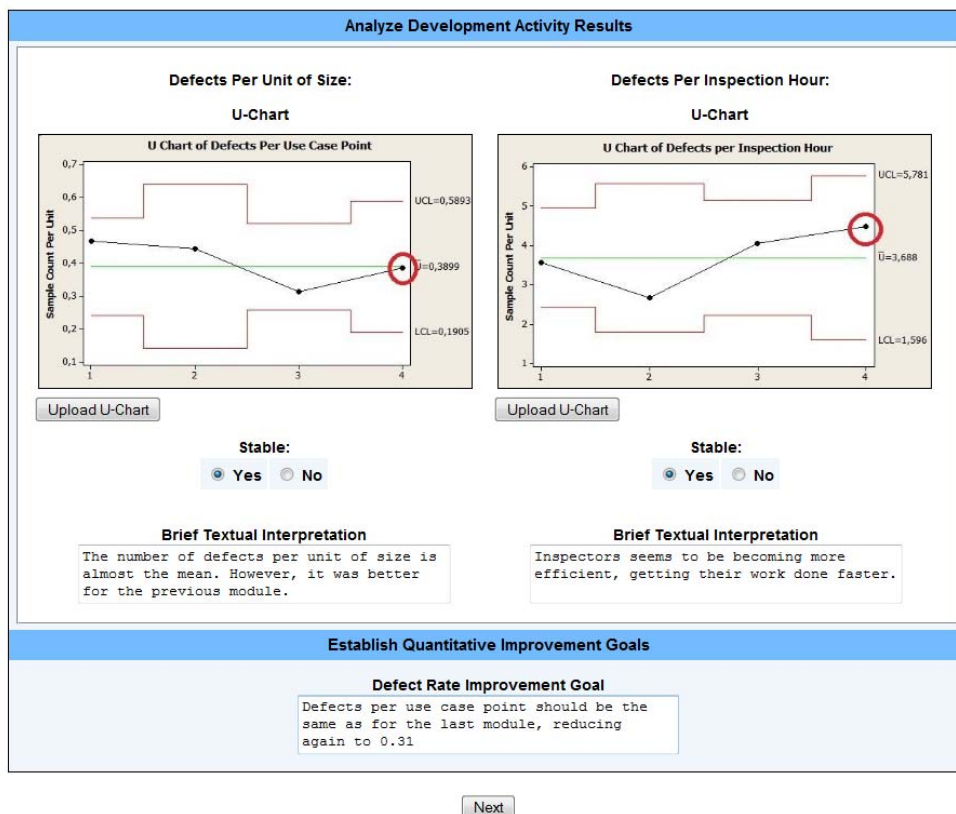
**5.3.3 Análise da Atividade de Desenvolvimento**

Nesta atividade a ferramenta disponibiliza ferramentas estatísticas externas cadastradas previamente para a geração dos gráficos U. Uma vez gerados, os gráficos U podem ser carregados na ferramenta como imagens para que possam ser analisados e a análise registrada. Adicionalmente o objetivo quantitativo de melhoria é registrado. A Figura 5.7 ilustra a atividade sendo realizada na ferramenta para a prova de conceito descrita na seção 5.2. O tratamento das métricas PIQ (*Phase Input Quality*) e POQ (*Phase Output Quality*) ainda não foi implementado na ferramenta.

Após finalizar esta atividade inicia-se a próxima atividade de DPPI, que é a preparação para análise causal.

## Development Activity Analysis

Development Activity:	Functional Specification	Project:	SIGIC
Module:	MPTV	Date:	Feb 22, 2010
Moderator:	Marcos Kalinowski		



**Figura 5.7. Apoio Ferramental à Atividade de Análise da Atividade de Desenvolvimento.**

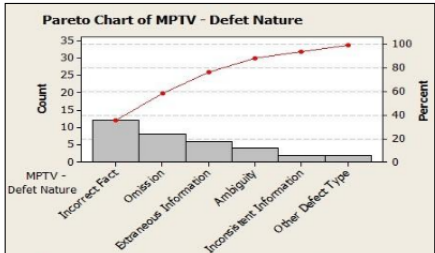
### 5.3.4 Preparação para Análise Causal

Nesta atividade a ferramenta disponibiliza ferramentas estatísticas externas cadastradas previamente para a geração dos gráficos de Pareto. Uma vez gerado, o mesmo pode ser carregado na ferramenta para que possa ser analisado. A ferramenta possibilita ainda visualizar todos os defeitos de determinada categoria para que sua descrição possa ser analisada e os erros sistemáticos cadastrados. A Figura 5.8 ilustra a atividade sendo realizada na ferramenta para a prova de conceito descrita na seção 5.2.

## Defect Causal Analysis Preparation

Development Activity:	Functional Specification	Project:	SIGIC
Module:	MPTV	Date:	Feb 22, 2010
Moderator:	Marcos Kalinowski		
Inspectors:	-		
Authors:	-		

### Apply Pareto Chart and Select Samples



**Pareto chart**

**Pareto Chart of MPTV - Defect Nature**

Count

Percent

MPTV - Defect Nature

Incorrect Fact

Omission

Extraneous Information

Ambiguity

Inconsistent Information

Other Defect Type

Upload Pareto Chart

**Defect Type**

Select...

**Defect Nature**

INCORRECT FACT

**Choose:**

☐ Defect Type

☒ Defect Nature

Show Defects

**Main Defect Categories**

The main identified category was Incorrect Fact, since almost 40% of the defects are of that type. Additionally, reading the defects allowed finding systematic errors.

### Find Systematic Errors

**Defect Category:** INCORRECT FACT

**Systematic Error Name:** Linking Use-Case descriptions incorrectly

Add...

Defect Category :	Systematic Error	Related Defects
INCORRECT FACT	Writing Invalid Business Rules	Associate Defects
INCORRECT FACT	Linking Use-Case descriptions incorrectly	Associate Defects

Next

**Figura 5.8. Apoio Ferramental à Atividade de Preparação para Análise Causal.**

Para cada erro sistemático adicionado, ao clicar em “Associate Defects”, o sistema apresenta uma lista de defeitos (Figura 5.9) filtrada pela categoria do erro sistemático (no caso “Incorrect Fact”), para que sejam associados ao erro sistemático.

Após finalizar esta atividade inicia-se a próxima atividade de DPPI, que é a reunião de análise causal.

### Associate Defects to Systematic Error

Description	Location ^	Severity ^	Defect Nature ^	Associate Defect
Descrição do Caso de Uso 4.2.1, em [A1] o MSL deve apresentar uma mensagem e voltar para o passo 1 do seu fluxo principal.	RS - Página 11	Média	INCORRECT FACT	<input checked="" type="checkbox"/>
Descrição do Caso de Uso 4.2.1, em [A2], o MSL deve apresentar uma mensagem e voltar para o passo 1 do seu fluxo principal.	RS - Página 11	Alta	INCORRECT FACT	<input checked="" type="checkbox"/>
Descrição do caso de uso 4.2.3, pós-condição: A PF é vinculada a ZERO ou mais projetos, já que o passo 5 do fluxo principal permite ao coordenador não selecionar projeto.	RS - Página 13	Média	INCORRECT FACT	<input type="checkbox"/>
O passo 4 do fluxo principal deve referenciar as regras de negócio do UC4.2.1. Além disso, o caso de uso não possui nada relacionado à exclusão; Não se pode apagar uma PF?	RS - Página 13 UC4.2.2	Média	INCORRECT FACT	<input type="checkbox"/>
No fluxo alternativo A2, passos 1 e 3 é citado o campo CÓDIGO DE MODALIDADE, mas o correto seria CÓDIGO DO DOCUMENTO.	RS - Página 28 UC4.2.18	Média	INCORRECT FACT	<input type="checkbox"/>
O requisito referenciado neste caso de uso (RF24) não se relaciona ao caso de uso. Está errado.	RS - Página 31 UC4.2.21	Média	INCORRECT FACT	<input type="checkbox"/>

Save Close

**Figura 5.9. Associação de Defeitos aos Erros Sistemáticos.**


### 5.3.5 Reunião de Análise Causal

Conforme previsto em DPPI, nesta atividade a ferramenta permite registrar causas para os erros sistemáticos e propostas de ação para estas causas. A Figura 5.10 ilustra a atividade sendo realizada na ferramenta para a prova de conceito descrita na seção 5.2.


Além de registrar as causas é possível consultar a distribuição de probabilidades relativas das causas de acordo com a inferência Bayesiana (Figura 5.11). Para o aprendizado e a inferência Bayesiana a API Java WEKA (WEKA, 2011) foi utilizada. Os valores das probabilidades não coincidem com os apresentados da prova de conceito da seção 5.2 porque a base foi populada apenas com uma amostra de defeitos dos módulos anteriores. A representação destas informações no formato de um diagrama de causa e efeito probabilístico ainda não foi implementada, em função dos prazos para a construção da ferramenta. Entretanto, esta representação pode facilmente ser feita manualmente durante a reunião de análise causal a partir das informações, fazendo uso de um quadro branco.

Ao final da atividade, as novas causas identificadas são utilizadas para atualizar o modelo causal com os novos casos de aprendizado, que associam cada uma das causas identificadas com todos os defeitos do erro sistemático sendo analisado. Note que, de acordo com DPPI, todos estes defeitos pertencem a uma

mesma categoria de defeitos. Isto assegura o fechamento do ciclo de retroalimentação de DPPI (Figura 4.4) sem esforço manual adicional.



[Home](#)
[View External Statistical Tools](#)
[About](#)



1
2
3
4

### Defect Causal Analysis Meeting

<b>Development Activity:</b>	Functional Specification	<b>Project:</b>	SIGIC
<b>Module:</b>	MPTV	<b>Date:</b>	Feb 22, 2010
<b>Moderator:</b>	Marcos Kalinowski		
<b>SEPG Members:</b>	<input type="text"/>		
<b>Authors:</b>	<input type="text"/>		

#### Identify Main Causes

Systematic Error ^	Cause-effect Diagram	Main Causes
Linking Use-Case descriptions incorrectly	<input type="button" value="Show Probabilistic Cause-effect Table"/>	<a href="#">Add Cause</a> <ul style="list-style-type: none"> <li>• Oversight</li> </ul>
Writing Invalid Business Rules	<input type="button" value="Show Probabilistic Cause-effect Table"/>	<a href="#">Add Cause</a> <ul style="list-style-type: none"> <li>• Lack of Domain Knowledge</li> <li>• Size and Complexity of the Problem Domain</li> </ul>

#### Propose Actions to Prevent Causes

Cause ^	Action
Lack of Domain Knowledge	<a href="#">Add Action</a> <ul style="list-style-type: none"> <li>• Study the domain and the pre-existing system</li> </ul>
Oversight	<a href="#">Add Action</a> <ul style="list-style-type: none"> <li>• Publish relevant examples of oversight related defects and presenting them to the functional specification team</li> </ul>
Size and Complexity of the Problem Domain	<a href="#">Add Action</a> <ul style="list-style-type: none"> <li>• Modify the functional specification template by creating a separate session for the business rules, listing them all together and separate from the use case description</li> </ul>

**Figura 5.10. Apoio Ferramental à Atividade de Reunião de Análise Causal.**

Assim, na próxima sessão de análise causal um erro sistemático relacionado com o tipo de defeitos “Fato Incorreto” apresentará probabilidades atualizadas. Como estas probabilidades possuem significado apenas para projetos similares, instâncias separadas da ferramenta devem ser mantidas para diferentes tipos de projeto. A avaliação da similaridade entre os projetos é a mesma que é realizada para o controle estatístico de processos e está fora do escopo de DPPI.



Probability Distribution for the Selected Systematic Error		
Cause ↕	Cause Category ↕	Probability ▼
Oversight	People	29.0 %
Lack of Domain Knowledge	People	19.0 %
Size and Complexity of the Problem Domain	Input	17.0 %
Limited Tool Used for Traceability	Tools	9.0 %
Interviewing the Wrong Stakeholder	Methods	7.0 %
Lack of Scope Awareness	People	5.0 %
Absence of a Standard way to Describe Use Cases	Methods	5.0 %
Lack of Good Requirement Practices	People	3.0 %
Political Conflicts	Input	3.0 %
Bias of the Existing System	Input	3.0 %
People: 56.0 % Input: 23.0 % Methods: 12.0 % Tools: 9.0 % Organization: 0.0 %		

**Figura 5.11. Distribuição de Probabilidades de Acordo com a Inferência Bayesiana.**

Após finalizar a reunião de análise causal inicia-se a próxima atividade de DPPI, que é a melhoria da atividade de desenvolvimento.

### 5.3.6 Melhoria da Atividade de Desenvolvimento

Nesta atividade o apoio ferramental permite gerenciar as propostas de ação até sua conclusão e registrar como se deu a comunicação das ações para a equipe de desenvolvimento. A Figura 5.12 ilustra a atividade sendo realizada na ferramenta para a prova de conceito descrita na seção 5.2. Após conclusão desta atividade finaliza a sessão de análise causal e encerra a aplicação de DPPI.

## Development Activity Improvement

Development Activity:	Functional Specification	Project:	SIGIC
Module:	MPTV	Date:	Feb 25, 2010
SEPG Members:			

Manage Actions to Conclusion					
Action	Estimated Due Date	Due Date	Observation	Status	Save
Study the domain and the pre-existing system	Feb 25, 2010	Feb 24, 2010	The existing system and process were presented again to	Done	<a href="#">Save Action</a>
Publish relevant examples of oversight related defects and presenting them to the functional specification team				Not Started	<a href="#">Save Action</a>
Modify the functional specification template by creating a separate session for the business rules, listing them all together and separate from the use case description	Feb 25, 2010			In Progress	<a href="#">Save Action</a>

Communicate Implemented Actions	
Communication Vehicle:	Oral Presentation
Communicator:	
Listeners:	
Date	

Finish

**Figura 5.12. Apoio Ferramental à Atividade de Melhoria da Atividade de Desenvolvimento.**

## 5.4 Considerações Finais

Neste capítulo a experiência de aplicar DPPI a um projeto de desenvolvimento de software Web real de larga escala e o apoio ferramental construído para facilitar sua aplicação foram descritos.

A experiência de aplicar DPPI indicou sua viabilidade de utilização, mas mostrou que sua aplicação manual pode envolver um esforço significativo. Desta forma, forneceu *insights* relevantes a respeito do apoio ferramental necessário para sua aplicação na indústria (Kalinowski *et al.*, 2010). Durante esta experiência a inferência diagnóstica da rede Bayesiana apoiou a identificação das causas principais eficientemente, motivando avaliações adicionais.

O apoio ferramental, por sua vez, apóia a aplicação sistemática de DPPI e fecha o ciclo automatizado de retroalimentação de casos de aprendizado proposto pela abordagem, permitindo a geração das probabilidades para cada uma das causas (refletindo a inferência diagnóstica Bayesiana) ser a causa principal dos erros

sistemáticos sendo analisados, sem requerer esforço adicional. Estas probabilidades podem ser utilizadas para a elaboração do diagrama de causa e efeito probabilístico (que também poderia ser gerado de forma automatizada). Entretanto, os benefícios de utilizar tais diagramas durante reuniões de análise causal ainda não foram avaliados objetivamente; embora a utilização tenha mostrado resultados promissores durante a experiência de aplicar DPPI descrita neste capítulo.

A investigação adicional através de um estudo experimental (conduzido em um problema real e com profissionais de desenvolvimento de software) dos benefícios da utilização de diagramas de causa e efeitos probabilísticos durante reuniões de análise causal para apoiar a identificação das causas é descrita no capítulo seguinte.

## **CAPÍTULO 6 - AVALIAÇÃO DA ABORDAGEM DE DPPI PARA A IDENTIFICAÇÃO DE CAUSAS DE DEFEITOS DE SOFTWARE**

*Neste capítulo é descrito o estudo experimental para avaliar se o uso da abordagem de DPPI, que faz uso de diagramas de causa e efeitos probabilísticos gerados a partir de sessões de análise causal aplicadas para a mesma atividade de desenvolvimento em módulos anteriores, traz benefícios para a identificação de causas de defeitos de software provenientes de inspeções.*

### **6.1 Introdução**

No campo da engenharia de software, a utilização de métodos experimentais pode trazer alguns benefícios, dentre os quais: acelerar o progresso através da rápida eliminação de abordagens não fundamentadas, suposições errôneas e modismos, e, ajudar a orientar a engenharia e a teoria para direções promissoras. É possível encontrar na literatura três estratégias experimentais: *survey*, estudo de caso e experimento. Uma descrição destas estratégias, pode ser encontrada em (Wohlin *et al.*, 2000a).

A estratégia experimental mais apropriada em uma situação concreta depende dos objetivos do estudo, das propriedades do processo de software usado durante a experimentação e dos resultados finais esperados (Travassos *et al.*, 2002).

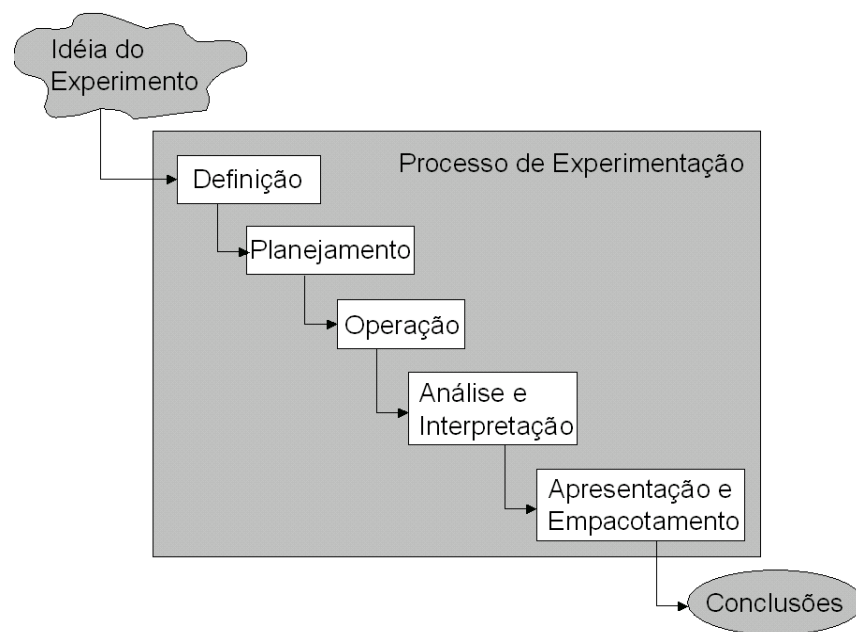
Neste capítulo um estudo experimental para avaliar se o uso da abordagem de DPPI traz benefícios para a identificação de causas de defeitos de software provenientes de inspeções de software é apresentado (Kalinowski *et al.*, 2011).

### **6.2 Estudo Experimental: Avaliando o uso da Abordagem de DPPI para a Identificação de Causas de Defeitos de Software.**

O objetivo deste estudo experimental é avaliar se a abordagem de DPPI, que faz uso de diagramas de causa e efeitos probabilísticos gerados a partir de sessões de

análise causal aplicadas para a mesma atividade de desenvolvimento em módulos anteriores, traz benefícios para a identificação de causas de defeitos de software provenientes de inspeções de software. O processo de experimentação seguido foi o descrito em (Wohlin *et al.*, 2000a) e está ilustrado na Figura 6.1. Uma descrição resumida deste processo, baseada em (Travassos *et al.*, 2002) se encontra a seguir.

A definição é a primeira atividade, onde o experimento é expresso em termos dos problemas e objetivos. A atividade de planejamento vem em seguida, onde o projeto do experimento é determinado, a instrumentação é considerada e os aspectos da validade do experimento são avaliados. A operação do experimento segue o planejamento. Neste momento os dados experimentais são coletados para serem analisados e avaliados na atividade de análise e interpretação. Finalmente, os resultados são apresentados e empacotados durante a atividade de apresentação e empacotamento.



**Figura 6.1. Processo de Experimentação. Adaptado de (Wohlin *et al.*, 2000a).**

A apresentação do estudo experimental segue o esboço para relatar experimentos sugerido em (Wohlin *et al.*, 2000a). Assim, o experimento será relatado em quatro seções: a definição do experimento, o planejamento do experimento, a operação do experimento propriamente dita e a análise dos dados e interpretação dos resultados. O pacote do experimento está disponível na COPPE/UFRJ e instrumentos deste podem ser encontrados no Anexo F.

### 6.2.1 Definição do Estudo Experimental

A fase de definição do estudo experimental descreve os objetivos de sua realização. No contexto do nosso estudo o objetivo é avaliar se a abordagem de DPPI, que faz uso de diagramas de causa e efeitos probabilísticos gerados a partir de sessões de análise causal anteriores, traz benefícios para a identificação de causas de defeitos de software provenientes de inspeções de software, quando comparada à realização desta atividade sem o uso de diagramas de causa e efeito probabilísticos. Daqui em diante, a abordagem que não faz uso do diagrama de causa e efeito probabilístico para apoiar a identificação das causas será referenciadas por abordagem *ad hoc*.

A estrutura a seguir, baseada no método GQM (Basili *et al.*, 1994), define o estudo experimental.

**“Analisar** o uso da abordagem de DPPI para a identificação de causas de defeitos de software provenientes de inspeções de software comparada à identificação *ad hoc* destas causas  
**com o propósito de** caracterizar  
**com respeito a** eficácia na identificação das causas dos defeitos, tempo gasto na identificação das causas e grau de satisfação dos participantes na realização da tarefa  
**do ponto de vista de** engenheiros de software  
**no contexto da** identificação de causas de defeitos de software provenientes de inspeções realizadas em especificações funcionais de um projeto de desenvolvimento de software para a web”.

Assim, pretende-se avaliar as seguintes questões gerais:

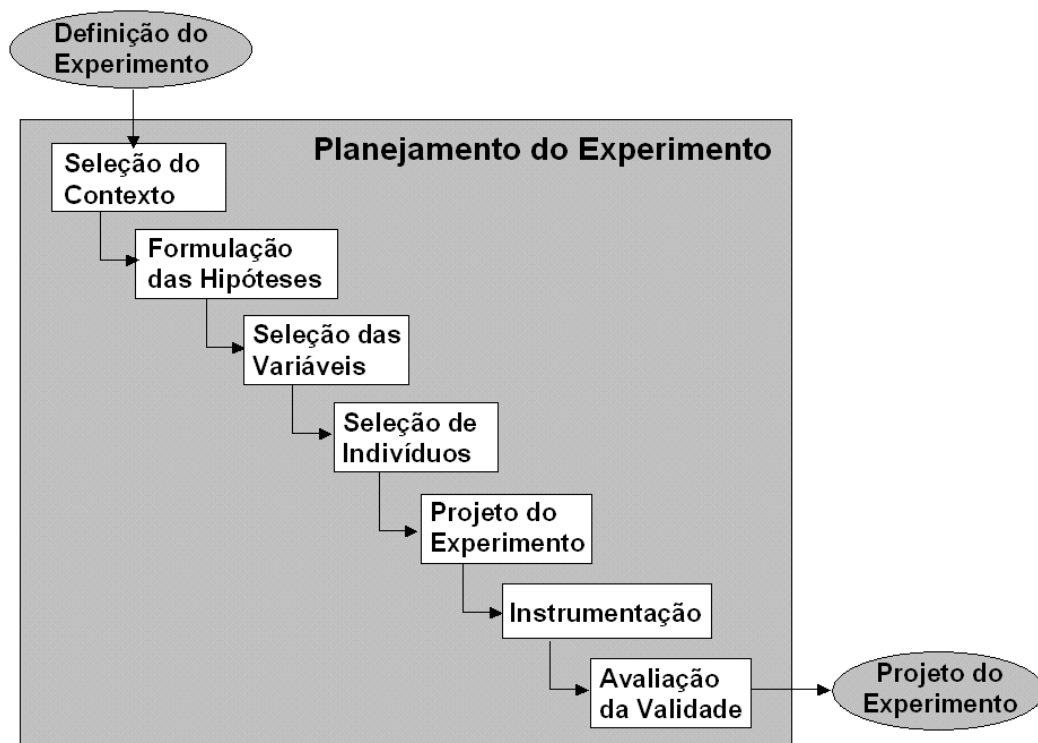
- Q1: A utilização da abordagem de DPPI implica em uma identificação mais eficaz das principais causas dos defeitos, quando comparada à identificação das causas de forma *ad hoc*?
- Q2: A utilização da abordagem de DPPI implica em um menor tempo gasto na identificação das causas, quando comparada à identificação das causas de forma *ad hoc*?
- Q3: A utilização da abordagem de DPPI implica em um grau de satisfação maior dos participantes na identificação das causas dos defeitos, quando comparada à identificação das causas de forma *ad hoc*?

Conforme será esclarecido em maiores detalhes na próxima seção, a caracterização do uso da abordagem de DPPI de acordo com a definição acima será

feita a partir da coleta e análise de dados quantitativos e qualitativos durante a operação do experimento.

### 6.2.2 Planejamento do Estudo Experimental

A fase de planejamento do experimento descreve como o experimento deve ser realizado. Esta descrição envolve: (1) a seleção do contexto, (2) a formulação das hipóteses, (3) a seleção das variáveis, (4) a seleção dos indivíduos, (5) o projeto do experimento, (6) a descrição da instrumentação e (7) a avaliação da validade do experimento. A Figura 6.2 ilustra as fases envolvidas no planejamento de experimentos.



**Figura 6.2. Planejamento do Experimento. Adaptado de (Wohlin et al., 2000a).**

A realização de cada uma das fases para o estudo experimental que visa caracterizar o uso da abordagem de DPPI para a identificação de causas de defeitos se encontra a seguir.

**Seleção do Contexto.** De acordo com (Wohlin et al., 2000a), o contexto pode ser caracterizado conforme quatro dimensões:

- processo: *on-line* / *off-line*;
- participantes: alunos / profissionais;
- realidade: problema real / modelado;

- generalidade: específico / geral.

Este estudo se propõe a ser executado de forma *off-line*, uma vez que o estudo se deu de forma retroativa e não em tempo real durante o projeto. Os participantes utilizados no estudo serão profissionais atuando em um problema real. Haverá um monitoramento contínuo sobre as atividades dos participantes. A generalidade do estudo é específica. A decisão de realizar o estudo experimental no contexto de um problema real foi eliminar possíveis fatores de confusão e viés relativo às causas dos defeitos.

**Formulação das Hipóteses.** Uma hipótese nula e três hipóteses alternativas foram elaboradas para o plano do experimento:

- **Hipótese nula ( $H_0$ ):** A utilização da abordagem de DPPI para a identificação das causas de defeitos de software não traz alterações quando comparada à identificação das causas dos defeitos de forma *ad hoc*.

$H_0 : (\Omega_1 = \Omega_2) \wedge (t_1 = t_2) \wedge (\mu_1 = \mu_2)$ , onde:

- $\Omega_1$  é a eficácia na identificação das principais causas de forma *ad hoc*.
  - $\Omega_2$  é a eficácia na identificação das principais causas utilizando a abordagem de DPPI.
  - $t_1$  é o tempo gasto na identificação das causas de forma *ad hoc*.
  - $t_2$  é o tempo gasto na identificação das causas utilizando a abordagem de DPPI.
  - $\mu_1$  é o grau de satisfação dos participantes na identificação das causas de forma *ad hoc*.
  - $\mu_2$  é o grau de satisfação dos participantes na identificação das causas utilizando a abordagem de DPPI.
- **Hipótese alternativa ( $H_1$ ):** A utilização da abordagem de DPPI implica em uma identificação mais eficaz das principais causas dos defeitos, quando comparada à identificação das causas de forma *ad hoc*.

$H_1 : \Omega_1 < \Omega_2$ , onde:

- $\Omega_1$  é a eficácia na identificação das principais causas de forma *ad hoc*.
- $\Omega_2$  é a eficácia na identificação das principais causas utilizando a abordagem de DPPI.



- **Hipótese alternativa (H<sub>2</sub>):** A utilização da abordagem de DPPI implica em um menor tempo gasto na identificação das causas dos defeitos, quando comparada à identificação das causas de forma *ad hoc*.

**H<sub>2</sub> :**  $t_1 > t_2$ , onde:

- $t_1$  é o tempo gasto na identificação das causas de forma *ad hoc*.
- $t_2$  é o tempo gasto na identificação das causas utilizando a abordagem de DPPI.

- **Hipótese alternativa (H<sub>3</sub>):** A utilização da abordagem de DPPI implica em um grau de satisfação maior dos participantes na identificação das causas dos defeitos, quando comparada à identificação das causas de forma *ad hoc*.

**H<sub>3</sub> :**  $\mu_1 < \mu_2$ , onde:

- $\mu_1$  é o grau de satisfação dos participantes na identificação das causas de forma *ad hoc*.
- $\mu_2$  é o grau de satisfação dos participantes na identificação das causas utilizando a abordagem de DPPI.

**Seleção de Variáveis.** O experimento utiliza as seguintes variáveis:

- Variável independente:
  - A abordagem utilizada para a identificação das causas dos defeitos.
- Variáveis dependentes:
  - A eficácia na identificação das principais causas para cada erro sistemático. O valor para esta variável representa o percentual das causas principais encontradas para cada um dos erros sistemáticos.
  - O tempo gasto na identificação das causas.
  - O grau de satisfação dos participantes na identificação das causas.

**Seleção de Indivíduos (Participantes).** Como o estudo experimental será conduzido no contexto de um projeto de software real e os profissionais envolvidos neste projeto, os participantes selecionados foram os que efetivamente atuariam em uma reunião de análise causal, de acordo com seus papéis no projeto.

Assim, foram selecionados os gerentes (representando o SEPG, já que o processo era definido para o projeto com a participação deles), os analistas de requisitos (os autores dos documentos em que os defeitos foram encontrados) e os

inspetores dos diferentes módulos do projeto de desenvolvimento escolhido. O projeto escolhido foi o SIGIC, o mesmo da prova de conceito. O escopo deste projeto é desenvolver um novo sistema de informação para gerenciar atividades de pesquisa e desenvolvimento da Fundação COPPETEC.

Um questionário deve ser entregue aos participantes para que possam ser caracterizados em termos de sua experiência prática com desenvolvimento de software e de sua formação acadêmica. A caracterização dos participantes possibilita o uso de agrupamento (*blocking*) (Wohlin *et al.*, 2000a) para eliminar fatores de confusão da análise dos resultados.

**Projeto do Experimento.** O projeto do estudo experimental teve que lidar com diversas restrições impostas pelo fato de ser realizado no contexto de um projeto de desenvolvimento de software real e com a equipe efetiva deste projeto.

Dado este contexto, o projeto utilizou um fator (a identificação de causas de defeitos de especificações funcionais), dois tratamentos (utilizar a abordagem de DPPI e utilizar uma abordagem *ad hoc*) e dois objetos de estudo (os defeitos de duas atividades de especificação funcional realizadas em dois módulos consecutivos do projeto).

Para instrumentar o estudo experimental, a atividade de preparação para análise causal de DPPI foi realizada antes de iniciar o experimento, plotando o gráfico de Pareto para identificar as principais categorias de defeitos e identificando os erros sistemáticos referentes a estes defeitos. Para cada um dos dois módulos cinco erros sistemáticos foram identificados.

A tarefa dos participantes envolvia identificar até três causas principais para cada um dos erros sistemáticos. Todos os participantes estavam envolvidos no projeto e fizeram parte das reuniões de inspeção em que o conjunto de defeitos foi discutido ao longo do projeto. Portanto, os participantes eram familiares com as especificações funcionais dos módulos (como deveriam ser para realizar a análise causal (Jones, 1985)). Por outro lado, a restrição de utilizar participantes familiares com as especificações funcionais dos módulos limitou o número de participantes possível para cinco.

Outra restrição em função de realizar o estudo no contexto de um projeto real de desenvolvimento foi que as causas corretas dos defeitos não eram conhecidas a priori. Esta questão foi tratada utilizando um dos cinco participantes possíveis para, considerando cada erro sistemático, baseado no conjunto de causas identificadas pelos outros quatro participantes, selecionar até três causas principais. O participante mais experiente, que foi também o moderador das inspeções destes módulos, foi

escolhido para este papel. Por ser o moderador das inspeções ele era extremamente familiar com o conjunto de defeitos, uma vez que durante as reuniões de inspeção ele era responsável por liderar as discussões destes defeitos que levou à sua categorização de acordo com as naturezas de defeitos descritas por Shull (1998).

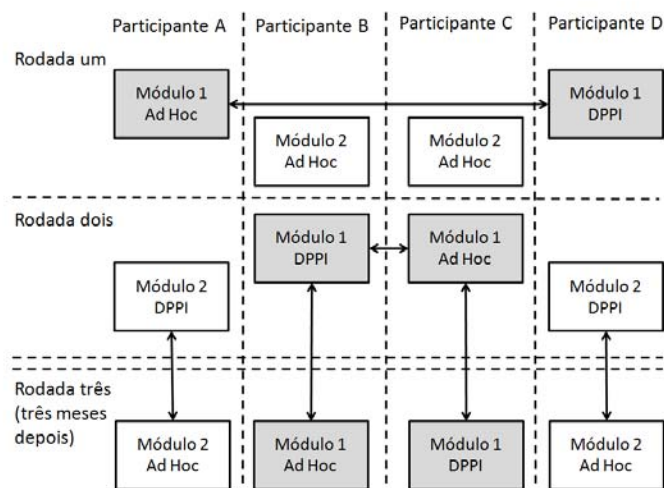
Ao selecionar as causas mais relevantes para cada erro sistemático este participante não sabia que o conjunto de causas que ele utilizou como ponto de partida foi montado juntando todas as causas citadas pelos outros quatro participantes. Durante o processo, ele tinha liberdade de escolher entre o conjunto de causas fornecidas ou até mesmo escolher outras causas. Em todos os casos ele selecionou causas do conjunto e novas não foram adicionadas. Adicionalmente, para evitar viés, suas escolhas foram revistas com o diretor responsável pelo projeto, que participou em reuniões de *status report* semanais, onde os principais problemas, riscos e o cronograma do projeto eram discutidos.

Desta forma, somente quatro participantes estavam disponíveis para efetivamente conduzir o estudo experimental e todos estes tinham caracterização bastante parecida (conforme detalhado mais adiante, Tabela 6.2). Para que fosse possível fazer observações relevantes utilizando apenas quatro participantes, possíveis arranjos destes participantes pelos tratamentos escolhidos (utilizar DPPI ou não) e os objetos do estudo (cada um dos dois módulos) foram estudados para reduzir esta ameaça à validade de conclusão.

O projeto resultante para a condução do estudo experimental encontra-se ilustrado na Figura 6.3. Para facilitar o entendimento desta Figura, os dois objetos do estudo foram representados em duas cores distintas (cinza e branco) e tratamentos empregados em cada um destes módulos foram representados junto aos próprios módulos. As linhas representam as diferentes rodadas do estudo e as colunas representam os participantes.

Como pode ser observado nesta Figura, cada um dos participantes realizou a tarefa de identificar as causas principais para os defeitos em três rodadas separadas do estudo experimental. As primeiras duas rodadas foram consecutivas, enquanto a terceira ocorreu propositalmente três meses após a realização das primeiras duas. Os cenários de observação encontram-se indicados na Figura 6.3 através de setas duplas e maiores detalhes sobre os mesmos encontram-se a seguir.

A primeira rodada foi utilizada para que os participantes se familiarizassem com a tarefa a ser realizada. Uma comparação interessante realizada nesta rodada foi entre os resultados dos participantes A e D, que utilizaram os tratamentos distintos no mesmo objeto (no caso o Módulo 1), já que ambos neste momento não estavam familiarizados com a tarefa a ser realizada.



**Figura 6.3. Projeto do Estudo Experimental. Três Rodadas, Envolvendo Quatro Participantes, Dois Tratamentos e Dois Objetos de Estudo.**

Na segunda rodada, por sua vez, os participantes já estavam cientes da tarefa a ser realizada e a realizaram em outro objeto (o módulo em que ainda não a haviam realizado). Nesta rodada, uma comparação interessante realizada foi entre os resultados dos participantes B e C, que haviam sido familiarizados à tarefa a realizando a mesma de maneira *ad hoc* em um objeto (Módulo 2) e agora estavam realizando a mesma utilizando tratamentos distintos no objeto Módulo 1.

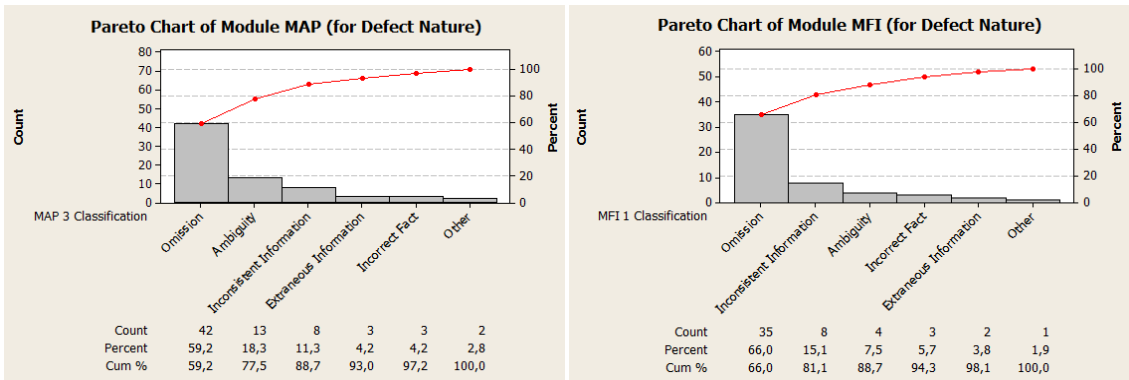
Desta forma, juntando as comparações das duas rodadas, de certa forma os fatores de confusão “treinamento” (a rodada 1 compara participantes sem treinamento e a rodada 2 participantes com treinamento) e “habilidade individual para identificar causas” (rodadas 1 e 2 comparam participantes diferentes) estariam isolados. Entretanto, tendo em vista o número reduzido de participantes, os resultados ainda seriam fortemente dependentes dos participantes e estas rodadas permitiriam apenas observações preliminares de tendências e uma compreensão inicial dos possíveis efeitos de se utilizar um tratamento ou outro sobre as variáveis dependentes.

Assim, o plano envolveu a realização de uma terceira rodada do experimento, três meses após as duas rodadas iniciais, visando comparar o efeito de variar o tratamento para um mesmo participante atuando sobre o mesmo objeto. O intuito era inverter o tratamento aplicado pelo participante na rodada 2, utilizando o mesmo objeto, e comparar os resultados deste mesmo participante nas rodadas 2 e 3, fazendo uso de um *crossed design* (Wohlin *et al.*, 2000a). O *crossed design* se manifesta entre as rodadas 2 e 3 para o módulo 1, onde um participante aplicou primeiro o tratamento *ad-hoc* no módulo enquanto o outro aplicou primeiro o

tratamento DPPI, desta forma é possível isolar o fator de confusão “momento da realização” na análise dos resultados obtidos para este módulo.

Ainda em relação ao fator “momento da realização”, o período de tempo de três meses entre uma rodada e outra era para que a memória de curto prazo não pudesse ser acionada pelos participantes para relacionar as principais causas de cada um dos cinco erros sistemáticos a serem analisados. Assim, antes do início da tarefa nesta rodada os participantes foram entrevistados para assegurar que não lembravam das causas dos cinco erros sistemáticos do módulo. Todos afirmaram não lembrar, mas mesmo que lembrassem, para a análise do módulo 1 este fator está isolado em função do *crossed design*, já que para este módulo na rodada 3 há um participante que migrou do tratamento DPPI para *ad hoc* e outro que migrou do tratamento *ad hoc* para DPPI. É importante ressaltar, entretanto, que, para a análise geral de ambos os módulos o fator da memória poderia eventualmente beneficiar o tratamento *ad hoc*, já que três participantes utilizaram este tratamento na rodada 3.

**Instrumentação.** Para a realização do experimento, a instrumentação foi preparada com os dados dos defeitos das inspeções de requisitos de dois módulos do projeto SIGIC. Foram escolhidos dois módulos de domínios distintos, chamados MAP (Módulo de Acompanhamento de Projetos) e MFI (Módulo Financeiro). Daqui em diante o módulo MAP será referenciado como Módulo 1 e o módulo MFI será referenciado como Módulo 2. Como a identificação das principais causas ocorre após a preparação para a análise causal, as principais categorias de defeitos foram identificadas previamente utilizando gráficos de Pareto (Figura 6.4) e os defeitos foram associados a erros sistemáticos (Tabela 6.1). Assim, os participantes já recebiam para cada módulo as listas de defeitos separadas por erro sistemático para então dar início à análise das causas principais.



**Figura 6.4. Gráficos de Pareto para os Módulos MAP e MFI. Principais Categorias de Defeitos.**

**Tabela 6.1. Erros Sistemáticos Identificados nos Módulos 1 (MAP) e 2 (MFI).**

<b>Erros Sistemáticos do Módulo 1 (MAP)</b>	<b>Erros Sistemáticos do Módulo 2 (MFI)</b>
1.1. Omissão de Campos e Informações.	1.1. Omissão de Campos e Informações.
1.2. Omissão de Regras de Negócio.	1.2. Omissão de Regras de Negócio.
1.3. Omissão de Detalhes de Navegação/Fluxo.	1.3. Omissão de Detalhes de Navegação/Fluxo.
1.4. Omissão de Definições e Referências.	1.4. Omissão de Definições e Referências.
1.5. Inserção de Informação Ambígua.	1.5. Inserção de Informação Inconsistente.

Além da lista de defeitos do módulo sendo analisado, o tratamento de DPPI recebe diagramas de causa-efeito probabilísticos elaborados conforme descrito no capítulo 4, a partir da inferência diagnóstica de uma rede Bayesiana alimentada com dados dos 197 defeitos dos módulos anteriores do SIGIC.

Assim, de forma resumida, o experimento conta com os seguintes instrumentos:

- Um formulário de consentimento (Apêndice E.1).
- Um questionário de caracterização dos participantes (Apêndice E.2).
- A descrição da tarefa a ser desempenhada pelos participantes, juntamente com as listas de defeitos associadas aos erros sistemáticos e, para o tratamento de DPPI, diagramas de causa e efeito probabilísticos para as categorias de defeito associadas aos erros sistemáticos sendo analisados (Apêndice E.3).
- Um questionário de acompanhamento, para possibilitar a análise qualitativa (veja Apêndice E.4).

**Avaliação da Validade.** Considerações a respeito da validade interna, externa, de construção e de conclusão do experimento se encontram descritas a seguir:

- **Validade interna.** Como se trata de um estudo envolvendo mais de um participante, onde cada grupo é submetido a tratamentos distintos, a maior ameaça de validade interna seria a relação dos resultados com a seleção dos participantes do estudo. A caracterização dos participantes e a terceira rodada do estudo foram utilizados para evitar esta ameaça e aumentar a validade interna. Na terceira rodada, o projeto do estudo fez uso de um *cross design* onde cada participante utiliza os dois tratamentos possíveis sobre um mesmo objeto.

- **Validade externa.** O estudo envolve profissionais e utiliza um problema real, a identificação de causas de defeitos de módulos do projeto SIGIC. Levando em consideração que este projeto segue um processo de desenvolvimento específico e que seus participantes possuem qualificações específicas, ele se propõe a ser válido apenas dentro deste contexto. Não é possível generalizar os resultados para outros contextos empresariais ou outros problemas reais, sendo necessário para isto elaborar novos estudos para ampliar a validade externa. Entretanto, os resultados do estudo podem servir como possíveis indícios de benefícios associados ao uso dos diagramas de causa e efeito probabilísticos para a identificação de causas de defeitos também em outros contextos, principalmente para projetos que compartilhem características com o projeto SIGIC, que utiliza um processo iterativo incremental, onde inspeções de requisitos são realizadas após a especificação funcional de cada um dos módulos.
- **Validade de construção.** Para obter validade de construção precisamos verificar duas coisas: que os tratamentos representem bem a construção da causa e que os resultados do experimento representem bem a construção do efeito. Como o experimento possui apenas um fator e dois tratamentos, a causa da variação nos resultados pode ser mapeada no uso ou não do diagrama de causa e efeito probabilístico. Os resultados, por sua vez, estão ligados diretamente ao efeito, uma vez que estes podem ser extraídos diretamente a partir dos valores das variáveis dependentes.
- **Validade de conclusão.** O propósito do estudo é a caracterização e a obtenção de resultados preliminares a respeito do uso dos diagramas de causa e efeito probabilísticos. O projeto do experimento e o pequeno número de participantes não permitem a aplicação de testes estatísticos (como *t-test* (Wohlin *et al.*, 2000a)). Assim, os resultados deverão ser tratados apenas como indícios preliminares de possíveis benefícios (ou não) do uso da abordagem DPPI para identificar causas de defeitos de software. Desta forma, a validade de conclusão está atrelada à replicação do estudo em outros contextos para viabilizar a aplicação de testes estatísticos e ainda aumentar a validade externa do estudo.

### 6.2.3 Operação

Durante a fase de operação do estudo experimental, inicialmente os participantes foram solicitados a preencher os formulários de consentimento e

caracterização. Um resumo da caracterização dos participantes do estudo, explicitando a experiência prática com desenvolvimento de software e o nível de formação encontra-se na Tabela 6.2.

**Tabela 6.2. Experiência Prática e Nível de Formação dos Participantes.**

Participante	Experiência Prática Desenvolvendo Software	Nível de Formação
A	6 anos em Projetos na Indústria.	Doutorando
B	Mais de 10 anos em Projetos na Indústria.	Doutorando
C	6 anos em Projetos na Indústria.	Doutorando
D	8 anos em Projetos na Indústria.	Doutorando

Posteriormente, para cada rodada prevista no projeto do estudo, eles foram solicitados a realizar a tarefa, que envolvia identificar até três causas principais para cada erro sistemático, realizando a leitura dos defeitos relacionados. Durante este processo eles foram monitorados e o esforço de execução da tarefa (em minutos) foi registrado. Para isto, os participantes do tratamento *ad hoc* receberam os cinco erros sistemáticos juntamente com a descrição dos defeitos relacionados e as categorias de causa sugeridas pelas diretrizes descritas no capítulo 3 (entrada, método, pessoas, ferramentas e organização). Participantes do tratamento DPPI também receberam os cinco erros sistemáticos com a descrição dos defeitos relacionados, mas ao invés das categorias de causa receberam um diagrama de causa e efeito probabilístico para os tipos de defeitos sendo analisados.

Conforme mencionado anteriormente, os diagramas de causa e efeito probabilísticos fornecidos foram elaborados a partir dos dados dos 197 defeitos dos quatro módulos anteriores do SIGIC. Para isto, cada um destes 197 defeitos foi associado retroativamente a uma causa (para facilitar esta tarefa um *brainstorm* sobre as possíveis causas foi realizado com os participantes do projeto), a rede Bayesiana foi construída e a inferência diagnóstica foi realizada para cada um dos tipos de defeito associados aos erros sistemáticos a serem analisados.

Por fim, os participantes foram solicitados a preencher um questionário de acompanhamento, permitindo obter dados qualitativos a respeito da satisfação na realização da tarefa, a percepção da complexidade da tarefa, a



confiança depositada nos resultados produzidos, uma descrição resumida de como a tarefa foi realizada e sugestões de melhoria.

#### 6.2.4 Análise e Interpretação dos Dados

Os resultados quantitativos da realização da tarefa em relação à eficácia na identificação das causas para cada um dos erros sistemáticos e ao tempo para a realização da tarefa encontram-se na Tabela 6.3.

Os resultados qualitativos da realização da tarefa, por outro lado, a respeito da satisfação na realização da tarefa, da percepção da complexidade da tarefa e da confiança depositada nos resultados produzidos encontram-se na Tabela 6.4.

**Tabela 6.3. Resultados Quantitativos do Estudo Experimental.**

Part	Rodada	Módulo	Tratamento	Eficácia (% de Causas Principais Identificado)						Tempo
				1.1	1.2	1.3	1.4	1.5	Média	
A	1	1	<i>ad hoc</i>	33,3%	0,0%	33,3%	66,7%	33,3%	33,3%	45
A	2	2	DPPI	66,7%	66,7%	66,7%	33,3%	66,7%	60,0%	20
A	3	2	<i>ad hoc</i>	66,7%	33,3%	33,3%	0,0%	33,3%	33,3%	27
B	1	2	<i>ad hoc</i>	66,7%	33,3%	0,0%	0,0%	0,0%	20%	95
B	2	1	DPPI	33,3%	33,3%	66,7%	33,3%	0,0%	33,3%	30
B	3	1	<i>ad hoc</i>	33,3%	33,3%	33,3%	0,0%	0,0%	20,0%	75
C	1	2	<i>ad hoc</i>	33,3%	0,0%	33,3%	66,7%	66,7%	40,0%	50
C	2	1	<i>ad hoc</i>	33,3%	33,3%	33,3%	0,0%	66,7%	33,3%	45
C	3	1	DPPI	33,3%	66,7%	66,7%	33,3%	33,3%	46,7%	30
D	1	1	DPPI	66,7%	66,7%	66,7%	33,3%	0,0%	46,7%	30
D	2	2	DPPI	66,7%	33,3%	66,7%	0,0%	33,3%	40,0%	20
D	3	2	<i>ad hoc</i>	0,0%	33,3%	33,3%	0,0%	33,3%	20,0%	25

**Tabela 6.4. Resultados Qualitativos do Estudo Experimental.**

Part	Rodada	Módulo	Tratamento	Satisfação na Realização da Tarefa	Percepção de Complexidade	Confiança nos Resultados (de 0 a 10)
A	1	1	<i>ad hoc</i>	Parcialmente Satisfeito	Muito Complexa	5
A	2	2	DPPI	Largamente Satisfeito	Complexa	7
A	3	2	<i>ad hoc</i>	Largamente Satisfeito	Simples	8
B	1	2	<i>ad hoc</i>	Largamente Satisfeito	Simples	8
B	2	1	DPPI	Largamente Satisfeito	Muito Simples	8
B	3	1	<i>ad hoc</i>	Largamente Satisfeito	Complexa	9
C	1	2	<i>ad hoc</i>	Totalmente Satisfeito	Simples	7
C	2	1	<i>ad hoc</i>	Totalmente Satisfeito	Simples	7
C	3	1	DPPI	Totalmente Satisfeito	Simples	8
D	1	1	DPPI	Largamente Satisfeito	Complexa	9
D	2	2	DPPI	Largamente Satisfeito	Simples	9
D	3	2	<i>ad hoc</i>	Totalmente Satisfeito	Muito Simples	8

A análise das hipóteses alternativas referentes à eficácia ( $H_1$ ) e ao tempo ( $H_2$ ) em função destes resultados, para os dois primeiros cenários de observação, se encontra disscorrida a seguir. Estes dois cenários comparam a eficiência e o tempo de participantes diferentes (embora com caracterização semelhante) utilizando tratamentos distintos. A hipótese  $H_3$  não foi incluída na análise destes cenários porque utiliza dados qualitativos (satisfação na realização da tarefa), perdendo o significado para comparação envolvendo participantes diferentes.

É importante ressaltar que os dois primeiros cenários de observação são apenas comparações simples para permitir uma melhor compreensão dos comportamentos gerais. A comparação efetiva entre os resultados dos dois tratamentos (envolvendo as três hipóteses) é mais bem retratada pelo terceiro cenário de observação.

- **Cenário de Observação 1:** Conforme o projeto do experimento, este cenário envolve comparar o resultado da rodada 1 do participante A com o resultado da rodada 1 do participante D, tendo em vista que eles atuaram sobre o mesmo objeto (módulo 1) utilizando tratamentos distintos.

- **Hipótese alternativa (H<sub>1</sub>):** A utilização da abordagem de DPPI implica em uma identificação mais eficaz das principais causas dos defeitos, quando comparada à identificação das causas de forma *ad hoc*.
  - Neste cenário é possível observar que a eficácia média do participante D (46,7%), que utilizou o tratamento DPPI, foi maior do que a eficácia do participante A (33,3%), que utilizou o tratamento *ad hoc*.
- **Hipótese alternativa (H<sub>2</sub>):** A utilização da abordagem de DPPI implica em um menor tempo gasto na identificação das causas dos defeitos, quando comparada à identificação das causas de forma *ad hoc*.
  - Neste cenário é possível observar que o tempo para realizar a tarefa do participante D (30 minutos), que utilizou o tratamento DPPI, foi menor do que o do participante A (45 minutos), que utilizou o tratamento *ad hoc*.
- **Cenário de Observação 2:** Conforme o projeto do experimento, este cenário envolve comparar o resultado da rodada 2 do participante B com o resultado da rodada 2 do participante C, tendo em vista que eles atuaram sobre o mesmo objeto (módulo 1) utilizando tratamentos distintos e já estando familiarizados com a tarefa a ser realizada.
  - **Hipótese alternativa (H<sub>1</sub>):** A utilização da abordagem de DPPI implica em uma identificação mais eficaz das principais causas dos defeitos, quando comparada à identificação das causas de forma *ad hoc*.
    - Neste cenário a eficácia média do participante B (33,3%), que utilizou o tratamento DPPI, foi igual à eficácia média do participante C, que utilizou o tratamento *ad hoc*.
  - **Hipótese alternativa (H<sub>2</sub>):** A utilização da abordagem de DPPI implica em um menor tempo gasto na identificação das causas dos defeitos, quando comparada à identificação das causas de forma *ad hoc*.
    - Neste cenário é possível observar que o tempo para realizar a tarefa do participante B (30 minutos), que utilizou o tratamento DPPI, foi menor do que o do participante C (45 minutos), que utilizou o tratamento *ad hoc*.

A análise dos resultados destes cenários de observação fornece indícios preliminares de que o uso da abordagem de DPPI possa implicar em uma maior eficácia e um menor tempo gasto na identificação de causas de defeitos,

principalmente em relação ao tempo (50% maior sem o uso de DPPI em ambos os cenários).

Ainda em relação às primeiras duas rodadas do estudo experimental, outra questão interessante que pode ser observada em função do arranjo do estudo é a comparação dos dois tratamentos aplicados ao módulo 2, onde com DPPI os participantes A e D tiveram eficácia 66,7% maior e esforço 72,4% menor, quando comparados com os participantes B e C, que utilizaram a abordagem *ad hoc*. Nota-se que, nesta análise, o tratamento *ad hoc* é prejudicado uma vez que ele foi aplicado na rodada 1 (com os participantes realizando a tarefa pela primeira vez), sugerindo um viés do fator de aprendizado. Entretanto, quando comparamos o caso oposto deste arranjo no módulo 1, com o participante D aplicando DPPI na rodada 1 e o participante C aplicando *ad hoc* na rodada 2, este efeito de aprendizado não pôde ser observado. De fato, o participante D, que aplicou DPPI no módulo 1 na rodada 1 obteve uma eficácia 40,0% maior e gastou 33,3% menos tempo na realização da tarefa, quando comparado ao participante C, que aplicou *ad hoc* no módulo 1 na rodada 2. Entretanto, em função do número reduzido de participantes nenhuma conclusão pode ser descrita.

O terceiro cenário permite uma análise mais elaborada desta percepção inicial de aumento na eficácia e redução do tempo. Este cenário envolve cada um dos quatro participantes aplicando os dois tratamentos ao mesmo objeto, comparando os resultados da rodada 3 com os da rodada 2. Desta forma, permite observar se o uso de DPPI implica em alteração na eficácia e no tempo de um mesmo participante. A análise das hipóteses em função dos resultados obtidos, tendo em vista este cenário de observação segue.

- **Cenário de Observação 3:** Conforme o projeto do experimento, este cenário envolve comparar, para cada participante, os resultados obtidos por este participante utilizando os dois tratamentos, entre as rodadas 2 e 3. Ou seja, a aplicação de dois tratamentos por um mesmo participante a um mesmo objeto é observada. Assim, como se trata de um mesmo participante, é possível observar se houve variação em função do uso do tratamento de DPPI, quando comparado ao uso tratamento *ad hoc*, e se essa variação foi positiva ou negativa.

Para facilitar a análise, a Tabela 6.5 mostra os resultados de cada um dos participantes, organizados por módulo, para os dois tratamentos. É possível notar na Tabela 6.3, que destes participantes apenas o participante C aplicou DPPI após aplicar *ad hoc*. Desta forma, o tratamento *ad hoc* poderia estar sendo favorecido pelo aprendizado e por uma eventual

memorização de resultados. A análise das hipóteses alternativas em função destes resultados segue.

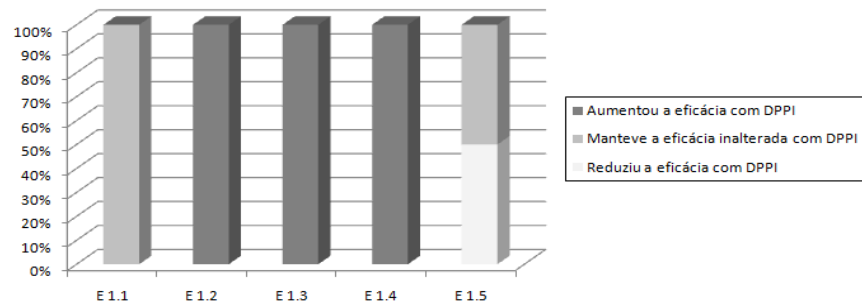
**Tabela 6.5. Resultados para cada um dos Participantes Aplicando os dois Tratamentos ao mesmo Módulo.**

Módulo	Participante	Tratamento	Eficácia (% de Causas Principais Identificado)						Tempo
			1.1	1.2	1.3	1.4	1.5	Média	
1	B	DPPI	33,3%	33,3%	66,7%	33,3%	0,0%	33,3%	30
1	B	<i>ad hoc</i>	33,3%	33,3%	33,3%	0,0%	0,0%	20,0%	75
1	C	DPPI	33,3%	66,7%	66,7%	33,3%	33,3%	46,7%	30
1	C	<i>ad hoc</i>	33,3%	33,3%	33,3%	0,0%	66,7%	33,3%	45
2	A	DPPI	66,7%	66,7%	66,7%	33,3%	66,7%	60,0%	20
2	A	<i>ad hoc</i>	66,7%	33,3%	33,3%	0,0%	33,3%	33,3%	27
2	D	DPPI	66,7%	33,3%	66,7%	0,0%	33,3%	40,0%	20
2	D	<i>ad hoc</i>	0,0%	33,3%	33,3%	0,0%	33,3%	20,0%	25

- **Hipótese alternativa ( $H_1$ ):** A utilização da abordagem de DPPI implica em uma identificação mais eficaz das principais causas dos defeitos, quando comparada à identificação das causas de forma *ad hoc*.
  - Todos os participantes obtiveram eficácia média maior aplicando o tratamento DPPI, quando comparada à eficácia da aplicação do tratamento *ad hoc*. Como haviam dois participantes para cada módulo e os módulos representavam domínios distintos isto reforça a observação inicial e a hipótese alternativa  $H_1$ .

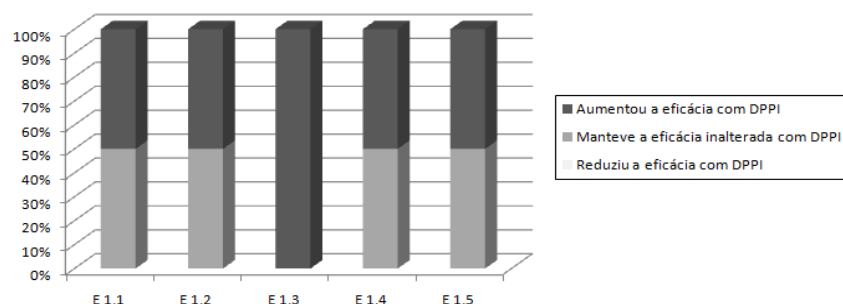
Para o módulo 1, os participantes B e C com a abordagem de DPPI obtiveram uma eficácia média 66,7% e 40% maior, respectivamente. A Figura 6.5 mostra a variação da eficácia dos participantes entre o tratamento *ad hoc* e o tratamento DPPI para o módulo 1, considerando cada um dos erros sistemáticos isoladamente. Nesta Figura é possível observar que o diagrama de causa e efeito probabilístico parece ter contribuído com a análise das causas de cada um dos erros sistemáticos do módulo 1. O único erro sistemático que apresentou redução da eficácia para um dos participantes foi o E 1.5 (Inserção de Informação Ambígua), entretanto a redução ocorreu para um participante na análise do último erro

sistemático na terceira rodada do estudo e, portanto poderia também estar associada a outros fatores, como cansaço.



**Figura 6.5. Variação da Eficácia dos Participantes por Erro Sistemático Analisado no Módulo 1.**

Para o módulo 2, os participantes A e D com a abordagem de DPPI obtiveram uma eficácia média 80,0% e 100% maior, respectivamente. A Figura 6.6 mostra a variação da eficácia dos participantes entre o tratamento *ad hoc* e o tratamento DPPI para o módulo 2, considerando cada um dos erros sistemáticos isoladamente. Nesta Figura é possível observar que o diagrama de causa e efeito probabilístico parece ter contribuído com a análise das causas de cada um dos erros sistemáticos do módulo 2. De fato, para todos os erros sistemáticos analisados neste módulo houve aumento da eficácia com o uso da abordagem de DPPI.



**Figura 6.6. Variação da Eficácia dos Participantes por Erro Sistemático Analisado no Módulo 2.**

- **Hipótese alternativa (H<sub>2</sub>):** A utilização da abordagem de DPPI implica em um menor tempo gasto na identificação das causas dos defeitos, quando comparada à identificação das causas de forma *ad hoc*.
  - Todos os participantes gastaram menos tempo na realização da tarefa aplicando o tratamento DPPI, quando comparado ao tempo gasto no tratamento *ad hoc*. Como haviam dois participantes para cada módulo e os módulos representavam

domínios distintos isto reforça a observação inicial e a hipótese alternativa  $H_2$ .

Analisando os módulos separadamente, para o módulo 1, os participantes B e C com a abordagem de DPPI obtiveram uma redução de tempo de 60,0% e 33,3%, respectivamente. Para o módulo 2, os participantes A e D, por sua vez, com a abordagem de DPPI obtiveram uma redução de tempo de 25,9% e 20,0%, respectivamente.

- **Hipótese alternativa ( $H_3$ ):** A utilização da abordagem de DPPI implica em um grau de satisfação maior dos participantes na identificação das causas dos defeitos, quando comparada à identificação das causas de forma *ad hoc*.
  - Com base na Tabela 6.4, que contém os resultados qualitativos, nada pôde ser observado em relação a esta hipótese. De fato, todos os participantes mantiveram ou aumentaram sua satisfação na realização da tarefa entre as rodadas consecutivas do estudo experimental, independente do tratamento adotado.

A percepção da complexidade, por sua vez, reduziu entre as diferentes rodadas, também independentemente do tratamento adotado, exceto para o participante B, que registrou uma complexidade consistentemente maior para o tratamento *ad hoc*.

A confiança nos resultados obtidos apresentou tendência de aumento entre as rodadas, exceto para o participante D, que registrou uma confiança menor nos resultados obtidos utilizando o tratamento *ad hoc* (na última rodada).

Assim, de forma geral, os três cenários de observação, especialmente o terceiro fornecem indícios preliminares refutando a hipótese nula e mostrando um comportamento consistente de aumento da eficácia na identificação das causas e redução do tempo gasto na realização da tarefa com o uso da abordagem de DPPI. De fato, no terceiro cenário todos os participantes apresentaram aumento da eficácia e redução do tempo para o tratamento que fazia uso da abordagem de DPPI.

Adicionalmente, a leitura das respostas do questionário de acompanhamento mostra que os participantes consideraram o diagrama de causa e efeito probabilístico útil para apoiar a identificação das causas. De fato, no tratamento de DPPI todos os

participantes fizeram uso do diagrama de causa e efeito probabilístico fornecido para apoiar a tarefa. O participante A mencionou ter observado o gráfico e complementado esta observação com seu conhecimento tácito a respeito do módulo e a leitura efetiva dos defeitos. O participante B, por sua vez, mencionou ter tentado utilizar o diagrama como oráculo, tentando avaliar através da leitura dos defeitos se as causas se “encaixavam” para aquele conjunto de defeitos, iniciando a análise pelas causas de maior probabilidade, mas não ficando restrito às causas apresentadas no diagrama. Este participante mencionou ainda que, “a grande maioria das causas saiu pelo diagrama”. O participante C, por sua vez mencionou durante a descrição dos procedimentos adotados para identificar as causas que, além da leitura dos defeitos, “Os diagramas fornecidos foram de grande ajuda no sentido de reconhecer um conjunto importante de causas comumente associadas ao tipo de defeito sendo analisado”. O participante D apenas mencionou ter lido os defeitos e ter utilizado os diagramas, sem fornecer maiores detalhes sobre como este uso se deu.

Por fim, uma comparação adicional que pode ser feita é a das probabilidades contidas nos diagramas de causa e efeito probabilístico, elaborado com base em dados de defeitos de módulos anteriores, com as efetivas causas principais consideradas como gabarito para cada um dos módulos. Conforme mencionado anteriormente, o gabarito foi obtido utilizando um quinto participante como oráculo, selecionando entre o conjunto de causas apontadas pelos quatro outros participantes as principais (fornecendo uma justificativa para suas escolhas) e validando estas escolhas com o diretor responsável pelo projeto.

O gabarito das causas principais para cada um dos módulos se encontra nas Tabelas 6.6 e 6.7 (para facilitar o mapeamento os termos foram mantidos em inglês, que foi a língua utilizada para alimentar a rede Bayesiana e que reflete também nos diagramas de causa e efeito probabilístico). As causas apresentam apenas o conjunto de causas principais, não havendo uma ordem de relevância entre elas. Os diagramas de causa e efeito probabilístico utilizados se encontram no Anexo F3.3. O diagrama referente a omissões foi utilizado em ambos os módulos para os erros sistemáticos 1.1 a 1.4. O diagrama referente a ambigüidades foi utilizado para o erro sistemático 1.5 (inserção de informação ambígua) do módulo 1 (MAP) e o diagrama referente a informações inconsistentes foi utilizado para o erro sistemático 1.5 (inserção de informação inconsistente) do módulo 2 (MFI).

Nesta comparação é possível verificar que, para cada erro sistemático, o diagrama de causa e efeito probabilístico correspondente continha probabilidades para as causas do gabarito, mas que nem sempre estas probabilidades eram as maiores. Por exemplo, para o erro sistemático 1.5 do módulo 1, o diagrama de causa e efeito



probabilístico do Anexo F3.3 apontou como a causa de maior probabilidade a falta de conhecimento de boas práticas da engenharia de requisitos, entretanto esta causa não foi uma das principais na análise causal deste módulo, de acordo com o gabarito. É natural que isto ocorra, uma vez que o diagrama é elaborado com base em dados de módulos anteriores.

É importante reforçar que o diagrama visa apoiar a identificação das causas e não fornecer o gabarito de forma automatizada e que ele foi alimentado apenas com os dados dos quatro módulos anteriores (197 defeitos). Assim, de certa forma, o diagrama chamou a atenção dos participantes para as causas apontadas pelo gabarito (todas tinham probabilidade diagnóstica maior que zero). Mesmo as causas que possuem probabilidade zero (que são as que ocorreram para outros tipos de defeito, diferente do tipo sendo analisado) são listadas no diagrama. O que o diagrama faz é destacar as causas de probabilidade maior, de acordo com a realidade da organização para projetos/módulos similares ao que foi executado.

**Tabela 6.6. Gabarito de Causas principais para os Erros Sistemáticos do Módulo 1 (MAP).**

<b>Erro Sistemático</b>	<b>Causa 1</b>	<b>Causa 2</b>	<b>Causa 3</b>
1.1 Omissão de Campos e Informações	P.1 - Not Formalizing Tacit Knowledge	P.2 - Lack of Domain Knowledge	P.4 – Oversight
1.2 Omissão de Regras de Negócio	P.2 - Lack of Domain Knowledge	P.3 - Lack of Good Requirement Practices	P.4 – Oversight
1.3 Omissão de Detalhes de Navegação/Fluxo	P.1 - Not Formalizing Tacit Knowledge	P.3 - Lack of Good Requirement Practices	P.4 – Oversight
1.4 Omissão de Definições e Referências	M.4 -Traceability not Institutionalized	P.4 - Oversight	T.2 - Limited tool used for Traceability
1.5 Inserção de Informação Ambígua	I.1 - Size and Complexity of the Problem Domain	I.2 - Different terms with the Same Meaning	I.4 - Bias of the Existing System

**Tabela 6.7. Gabarito de Causas principais para os Erros Sistemáticos do Módulo 2 (MFI).**

<b>Erro Sistemático</b>	<b>Causa 1</b>	<b>Causa 2</b>	<b>Causa 3</b>
1.1 Omissão de Campos e Informações	I.1 - Size and Complexity of the Problem Domain	P.2 - Lack of Domain Knowledge	P.3 - Lack of Good Requirement Practices
1.2 Omissão de Regras de Negócio	I.1 - Size and Complexity of the Problem Domain	P.2 - Lack of Domain Knowledge	P.3 - Lack of Good Requirement Practices
1.3 Omissão de Detalhes de Navegação/Fluxo	P.1 - Not Formalizing Tacit Knowledge	P.3 - Lack of Good Requirement Practices	P.4 – Oversight
1.4 Omissão de Definições e Referências	I.1 - Size and Complexity of the Problem Domain	P.4 - Oversight	T.2 - Limited tool used for Traceability
1.5 Inserção de Informação Inconsistente	I.1 - Size and Complexity of the Problem Domain	I.2 - Different terms with the Same Meaning	P.3 - Lack of Good Requirement Practices

Embora os resultados gerais indiquem uma melhoria significativa para o tratamento de DPPI, é importante ressaltar que o projeto do estudo experimental não permite inferências sobre a validade externa do estudo, uma vez que ele foi realizado no contexto real de um projeto específico de desenvolvimento de software. De fato, em função do número limitado de participantes também não há validade de conclusão, uma vez que testes estatisticamente significativos não puderam ser aplicados. Entretanto, o projeto do estudo e seu arranjo permitiram fornecer diversos argumentos que servem como indícios preliminares de benefícios associados ao uso dos diagramas de causa e efeito probabilísticos de DPPI para apoiar a identificação de causas de defeitos em reuniões de análise causal.

### **6.3 Considerações Finais**

Neste capítulo o estudo experimental para avaliar se o uso da abordagem de DPPI, que faz uso de diagramas de causa e efeitos probabilísticos gerados a partir de sessões de análise causal anteriores, traz benefícios para a identificação de causas de defeitos de software provenientes de inspeções foi descrito.

O projeto do estudo envolveu diversos arranjos entre dois tratamentos (aplicar DPPI ou não), dois objetos (módulos MAP e MFI do projeto SIGIC) e quatro participantes, organizados em três rodadas separadas do estudo. Estas rodadas permitiram analisar as hipóteses do estudo com base em três cenários distintos de observação.

Os resultados gerais indicam uma melhoria significativa para o tratamento de DPPI, em relação à eficácia na identificação das principais causas dos defeitos e ao tempo gasto na realização da tarefa. No último cenário de observação, em particular, que envolveu a comparação de cada um dos participantes aplicando os dois tratamentos no mesmo objeto, todos os participantes apresentaram maior eficácia na identificação das causas (pelo menos 40% maior) e um menor tempo gasto (pelo menos 20% menor) para o tratamento de DPPI.

Em função da especificidade do estudo, o projeto do estudo experimental não permite inferências sobre a validade externa do estudo, uma vez que ele foi realizado no contexto real de um projeto particular de desenvolvimento de software. Em função do número limitado de participantes também não há validade de conclusão, uma vez que testes estatisticamente significativos não puderam ser aplicados.

Assim, a validade de conclusão está atrelada à repetição do estudo em outros contextos para viabilizar a aplicação de testes estatísticos e ainda aumentar a validade

externa. Além de aplicar o estudo no contexto de outros projetos, ele pode ainda ser realizado para outras atividades de desenvolvimento, já que o uso de DPPI não se restringe à atividade de especificação funcional.

Entretanto, o projeto do estudo e seu arranjo experimental permitiram fornecer argumentos consistentes e que servem como indícios preliminares concretos de benefícios associados ao uso dos diagramas de causa e efeito probabilísticos de DPPI para apoiar a identificação de causas de defeitos em reuniões de análise causal.

## CAPÍTULO 7 - CONSIDERAÇÕES FINAIS

*Neste capítulo são apresentadas as considerações finais sobre a pesquisa realizada, incluindo os resultados obtidos, as contribuições, as limitações e os próximos passos a serem realizados com a continuidade da pesquisa.*

### 7.1 Considerações Finais

O objetivo central de muitas pesquisas, nas diferentes ciências, é a elucidação de relações de causa e efeito entre variáveis ou eventos (Pearl, 2000). Considerando a análise causal de defeitos de software, para organizações de software esta elucidação implicaria em ter, de acordo com as características do projeto, o conhecimento sobre quais causas normalmente resultam em que tipo de defeitos.

Na engenharia de software, a análise causal de defeitos tem se mostrado uma prática capaz de melhorar processos e reduzir a taxa de introdução de defeitos em diferentes contextos organizacionais, como o da IBM (Mays *et al.*, 1990), da Computer Science Corporation (Dangerfield *et al.*, 1992), da HP (Grady, 1996) e da InfoSys (Jalote e Agrawal, 2005), entre outros. Entretanto, de acordo com a revisão sistemática realizada no contexto desta tese (executada em 2006, 2007, 2009 e 2010), as abordagens de análise causal de defeitos encontradas descrevem atividades a ser realizadas, mas fornecem pouco detalhamento sobre as tarefas que devem ser realizadas no contexto destas atividades e sobre as técnicas que podem apoiar a realização eficiente destas tarefas.

Além do pouco detalhamento das abordagens encontradas, foi possível observar que nenhuma das abordagens buscava integrar mecanismos de aprendizado das relações causa e efeito às reuniões de análise causal. Isto significa que, em todas as abordagens analisadas, o conhecimento a respeito de relações causa e efeito, obtido durante cada reunião de análise causal, era somente utilizado para iniciar ações para melhorar o processo de desenvolvimento e posteriormente não aproveitado.

Neste contexto, a abordagem DPPI foi criada (Kalinowski *et al.*, 2008b) e refinada (Kalinowski *et al.*, 2010). DPPI representa uma abordagem para realizar, medir e controlar a análise causal de defeitos provenientes de inspeções de software de acordo com as melhores práticas identificadas em diretrizes elaboradas a partir de resultados da revisão sistemática (Kalinowski *et al.*, 2008a).

Adicionalmente, de acordo com os resultados da revisão sistemática, DPPI representa, no momento da apresentação desta tese, a única abordagem que integra

mecanismos de aprendizado de relações de causa e efeito (fazendo uso de redes Bayesianas) nas reuniões de análise causal, tratando explicitamente um ciclo de retroalimentação antes não considerado. Para facilitar o uso do conhecimento contido nas redes Bayesianas em reuniões de análise causal, uma representação gráfica, denominada Diagrama de Causa e Efeito Probabilístico (Kalinowski *et al.*, 2008b) foi criada.

A experiência de aplicar DPPI manualmente a um projeto Web de larga escala indicou sua viabilidade e permitiu obter uma percepção mais apurada sobre o apoio ferramental requerido para facilitar sua utilização (Kalinowski *et al.*, 2010). Além disto, durante esta experiência particular, o diagrama de causa e efeito probabilístico se mostrou útil para apoiar a identificação das causas principais, motivando investigações adicionais.

Assim, requisitos para o apoio ferramental puderam ser definidos e o mesmo pôde ser construído no contexto de um projeto final de graduação (Paes, 2010). Este apoio ferramental automatiza o ciclo de retroalimentação proposto por DPPI. Isto é feito estabelecendo e mantendo os casos de aprendizado (relações entre causas e tipos de defeitos) da rede Bayesiana, com base nos próprios resultados de cada reunião de análise causal, sem requerer esforço adicional, conforme definido em DPPI. A inferência Bayesiana para o tipo de defeito sendo analisado durante uma reunião de análise causal também foi automatizada. Assim, os diagramas de causa e efeito probabilísticos podem ser facilmente elaborados.

Por fim, um estudo experimental foi conduzido para avaliar se o uso de DPPI, com diagramas de causa e efeitos probabilísticos gerados a partir de sessões de análise causal anteriores, traz benefícios para a identificação de causas de defeitos de software provenientes de inspeções. O projeto do estudo envolveu diversos arranjos entre dois tratamentos (aplicar DPPI ou não), dois objetos (os módulos MAP e MFI do projeto SIGIC) e quatro participantes, organizados em três rodadas separadas do estudo. Os resultados gerais deste estudo fornecem indícios de uma melhoria significativa para o tratamento de DPPI, tanto em relação à eficácia na identificação das principais causas dos defeitos quanto ao tempo gasto na realização da tarefa.

Com base na experiência pessoal na pesquisa referente a análise causal de defeitos de software, envolvendo, entre outros, a concepção e a avaliação da abordagem DPPI, e na vivência junto a organizações desenvolvedoras de software interessadas na melhoria de seus processos, acreditamos que a abordagem resultante desta tese é útil e traz ganhos para aquelas organizações que efetivamente desejam fazer do aprendizado com seus erros um instrumento para aumentar sua competitividade e qualidade.

O restante deste capítulo apresenta os resultados obtidos, as limitações e os trabalhos futuros desta tese.

## 7.2 Resultados Obtidos

Analisando os quatro objetivos específicos definidos para este trabalho e descritos na seção 1.5, os seguintes resultados foram obtidos:

- **O1: Estabelecer um mecanismo sistemático e dinâmico para a identificação e constante atualização de um conjunto de boas práticas para a realização eficiente das tarefas associadas à análise causal de defeitos de software.** Um protocolo de revisão sistemática foi elaborado e executado em quatro anos diferentes (2006, 2007, 2009 e 2010), permitindo sua re-execução futura. A execução do protocolo nos diferentes anos permitiu elaborar tabelas que resumem conhecimento a respeito de análise causal de defeitos (Anexo C). Adicionalmente, diretrizes foram elaboradas com base neste conhecimento. Estes resultados foram descritos nos capítulos 2 e 3.
- **O2: Organizar este conjunto de práticas no contexto de uma abordagem de análise causal de defeitos de software. Esta abordagem deve prover um mecanismo para manter e utilizar o conhecimento gerado a respeito de relacionamentos causais ao longo das diferentes sessões de análise causal.** A abordagem DPPI foi criada (Kalinowski *et al.*, 2008b) e refinada (Kalinowski *et al.*, 2010), organizando as boas práticas identificadas durante a revisão sistemática e explicitadas nas diretrizes. A contribuição principal de DPPI é integrar mecanismos de aprendizado de relações de causa e efeito nas reuniões de análise causal, fazendo uso de redes Bayesianas e diagramas de causa e efeito probabilísticos. Este resultado foi descrito no capítulo 4.
- **O3: Avaliar a viabilidade de aplicação da abordagem resultante em projetos de software reais e prover apoio ferramental para facilitar sua utilização.** Uma prova de conceito foi realizada, em que a aplicação de DPPI a um projeto de desenvolvimento Web real de larga escala se mostrou viável (Kalinowski *et al.*, 2010). Com base nesta experiência, requisitos para apoio ferramental foram definidos e posteriormente o mesmo foi construído, no contexto de um projeto final de graduação. Estes resultados foram descritos no capítulo 5.

- **O4: Avaliar os benefícios que esta abordagem traz em relação a diferentes aspectos, como: (i) eficácia na identificação das principais causas, (ii) esforço na identificação das principais causas e (iii) satisfação dos usuários ao realizar análise causal.** Um estudo experimental foi conduzido para avaliar se o uso de DPPI, com diagramas de causa e efeitos probabilísticos gerados a partir de sessões de análise causal anteriores, traz benefícios para a identificação de causas de defeitos de software provenientes de inspeções (Kalinowski et al., 2011). Os resultados gerais deste estudo fornecem indícios de uma melhoria significativa para o tratamento de DPPI, tanto em relação à eficácia na identificação das principais causas dos defeitos quanto ao tempo gasto na tarefa. O estudo e seus resultados foram descritos no capítulo 6.

## **7.3 Contribuições**

As principais contribuições desta pesquisa podem ser organizadas em duas categorias, descritas nas subseções a seguir.

### **7.3.1 Diretrizes de Apoio à Organizações que Desejam Implementar Análise Causal de Defeitos**

Estas diretrizes são compostas por (1) respostas baseadas em evidência para dúvidas que podem ser enfrentadas por profissionais ao implementar o processo de análise causal de defeitos de software e (2) pelas tabelas de conhecimentos a respeito do processo de análise causal de defeitos e suas atividades. Adicionalmente, o uso dessas diretrizes para apoiar a realização de análise causal de defeitos de acordo com as práticas específicas da área de processo CAR do CMMI foi discutido. Estas diretrizes podem ser utilizadas por organizações desenvolvedoras de software como complemento a outras fontes, como o Guia de Implementação do MPS (SOFTEX, 2009).

### **7.3.2 Abordagem para Prevenção de Defeitos Provenientes de Inspeções para Apoiar a Melhoria dos Processos de Engenharia do Software**

A abordagem DPPI foi criada e refinada (Kalinowski et al., 2010), organizando as boas práticas identificadas durante a revisão sistemática e explicitadas nas diretrizes. DPPI é a primeira abordagem de análise causal de defeitos que integra

mecanismos de aprendizado de relações de causa e efeito na realização das reuniões de análise causal.

A viabilidade de aplicação da abordagem foi avaliada através de uma prova de conceito, em que a aplicação de DPPI a um projeto de desenvolvimento Web real de larga escala se mostrou viável (Kalinowski *et al.*, 2010). Apoio ferramental para facilitar o uso da abordagem DPPI foi definido e construído no contexto de um projeto final de graduação (Paes, 2010). Além disto, um estudo experimental foi conduzido em três rodadas, fornecendo indícios de que a abordagem de DPPI (com o uso do conhecimento das relações causais a respeito de defeitos de software) resulta no aumento na eficácia e na redução do esforço durante a identificação das principais causas dos defeitos (Kalinowski *et al.*, 2011).

## 7.4 Limitações

As limitações desta tese estão relacionadas, principalmente, aos seguintes itens:

- **Escopo da Abordagem DPPI.**
  - O escopo de DPPI se limita à análise de defeitos provenientes de inspeções de software. De fato, inspeções fornecem informações a respeito de defeitos cedo no processo de desenvolvimento e assim permitem a aplicação de DPPI imediatamente após a atividade de desenvolvimento. Isto permite que DPPI seja aplicada em processos iterativos incrementais, por exemplo. Acreditamos que no caso de falhas provenientes de testes a abordagem poderia ser seguida da mesma forma, bastando antes analisar as falhas para encontrar os defeitos (faltas) correspondentes e utilizar os dados dos defeitos e não das falhas. Entretanto, nenhuma investigação foi realizada neste sentido.
  - O escopo de DPPI não envolve a avaliação da similaridade entre projetos. A avaliação da similaridade entre projetos é extremamente complexa e é estudada em outras áreas da engenharia de software, como a gerência quantitativa de processos (Florac e Carleton, 1999). De fato, o conhecimento contido nas redes Bayesianas foi avaliado para diferentes iterações de um mesmo projeto e acreditamos que ele seja útil principalmente para projetos similares. Desta forma, organizações que possuem projetos de diferentes domínios, utilizando



diferentes processos, diferentes tecnologias e equipes com experiência heterogênea devem separar os seus projetos em grupos de projetos similares e instanciar DPPI separadamente para cada um destes grupos. Estes critérios de separação dos projetos em grupos para análise não foram tratados nesta tese.

- **Avaliações da Abordagem DPPI.**

- Embora as avaliações tenham fornecido indícios interessantes, elas não possuem validade de conclusão ou validade externa, em função do pequeno número de participantes e do contexto específico em que foram realizadas. Entretanto, esta limitação pode ser difícil de contornar, uma vez que os participantes precisam ter tido participação efetiva no projeto (já que se trata de análise causal). Repetições das avaliações em outros contextos podem ajudar a ampliar a validade de conclusão e a validade externa.
- Todas as avaliações de DPPI foram realizadas para a atividade de engenharia do software referente ao levantamento e especificação funcional. Entretanto, DPPI se propõe a ser aplicada também para melhorar outras atividades de engenharia do software, como a especificação da solução técnica e a codificação, entre outras.

## **7.5 Trabalhos Futuros**

Como trabalhos futuros, a pesquisa apresentada nesta tese pode ser continuada, considerando, por exemplo, os seguintes direcionamentos:

- Avaliar a possibilidade de ampliar o escopo de DPPI para defeitos provenientes de outras atividades de garantia e controle da qualidade.
- Investigar uma forma apropriada para caracterizar a similaridade entre projetos. Desta forma seria possível saber de antemão para quais grupos de projeto DPPI deve ser instanciada separadamente. Adicionalmente, tendo esta caracterização, seria possível utilizar uma rede Bayesiana que faça uso de lógica de primeira ordem (Paes *et al.*, 2006), incluindo na rede a caracterização do projeto. Desta forma, a rede poderia fazer a inferência para determinado tipo de projeto e apenas uma instância de DPPI seria necessária para tratar todos os projetos da organização.

- Realizar avaliações adicionais, replicando as já realizadas em outros contextos ou ainda para outras atividades de engenharia do software, como a elaboração da solução técnica ou a codificação.
- Estender DPPI para que a abordagem seja capaz de lidar, além de defeitos de software, também com outros problemas que possam vir a ocorrer durante o desenvolvimento de software.

## REFERÊNCIAS BIBLIOGRÁFICAS

- Abbott, J., "Life of Napoleon Bonaparte", Kessinger Publishing, ISBN 1-417-97063-4, 2005.
- Al-Shehab, A. J., Hughes, R. T., Winstanley, G., "Facilitating Organisational Learning through Causal Mapping Techniques in IS/IT Project Risk Management", In: Lecture Notes in Computer Science, 3782 NAI, 145 - 154, 2005.
- Babbie, E., "The Practice of Social Research", Wadsworth Publishing, Belmont, CA, ISBN: 978-0495598411, 1986.
- Basili, V.R., Caldera, C., Rombach, D., "Goal Question Metric Paradigm", Encyclopedia of Software Engineering (Marciniak J. editor), vol. 1, John Wiley & Sons, 528-532, 1994.
- Basili, V.R., "Software Development: A Paradigm for the Future", In: Proc. of Compsac. New York: IEEE Computer Society Press, pp. 471-185, 1989.
- Basili, V.R., Perricone, T., "Software Errors and Complexity: An Empirical Investigation", In: Communications of the ACM, vol. 27, 1984.
- Beizer, B., "Software Testing Techniques", van Nostrand Reinhold Co., New York, NY, USA, 2<sup>nd</sup> edition, ISBN: 978-1850328803, 1990.
- Bhandari, I., "Attribute Focusing: Machine-Assisted Knowledge Discovery Applied to Software Production Process Control", In: Proc. of the Workshop on Knowledge Discovery in Databases, AAAI Tech. Rep. Series, Rep. No. WF-93-02, 1993.
- Bhandari, I., Halliday, M., Tarver, E., Brown, D., Chaar, J., Chillarege, R., "A Case Study of Software Process Improvement During Development", In: IEEE Transactions on Software Engineering, 19 (12), 1157 – 1170, 1993.
- Bhandari, I., Roth, N., "Post-process Feedback with and without Attribute Focusing: a Comparative Evaluation", In: Proc. of ICSE '93: 15th International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 89-98, 1993.
- Biolchini, J., Mian, P.G., Natali, A.C.C., Travassos, G.H., "Systematic Review in Software Engineering", In: Relatório Técnico COPPE/UFRJ – disponível em [www.cos.ufrj.br](http://www.cos.ufrj.br), 2005.

- Blackburn, S., "Oxford Dictionary of Philosophy", Oxford University Press, 2a Edição, ISBN: 978-0199541430, 2008.
- Briand, L., Freimut, B., Vollei, F., "Assessing the Cost-Effectiveness of Inspections by Combining Project Data and Expert Opinion", In: Proc. of the 11th International Symposium on Software Reliability Engineering, pp. 124-135, 2000.
- Boehm, B., "A View of 20th and 21st Century Software Engineering", In: Proc. of ICSE '06: 28th International Conference on Software Engineering, ACM Press, New York, NY, USA, pp. 12-29, 2006.
- Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., Khalil, M., "Lessons from applying the systematic literature review process within the software engineering domain", In: Journal of Systems and Software, Vol. 80 (4), pp. 571-583, 2007.
- Bush, M.W., "Getting Started on Metrics - JPL Productivity and Quality", In: Proc. of ICSE '90: 12th International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 133-142, 1990.
- Card, D.N., "Defect Analysis: Basic Techniques for Management and Learning", In: Advances in Computers, vol. 65, chapter 7, pp. 259-295, 2005.
- Card, D.N., "Learning from our Mistakes with Defect Causal Analysis", In: IEEE Software, 15(1), 56 – 63, 1998.
- Card, D.N., "Defect Causal Analysis Drives Down Error Rates", In: IEEE Software 1/1993, Volume 10, Issue 4, 7/1993, p.98-99, 1993.
- Chang, C., Chu, C., "Software defect prediction using intertransaction association rule mining", In: International Journal of Software Engineering and Knowledge Engineering, 19, 747-764, 2009.
- Chang, C.; Chu, C., Yeh, Y., "Integrating in-process software defect prediction with association mining to discover defect pattern", In: Journal on Information and Software Technology, 51( 2), 375 – 384, 2009.
- Chang, C., Chu, C., "Improvement of causal analysis using multivariate statistical process control", In: Software Quality Journal, 16, 377-409, 2008.
- Chang, C., Chu, C., "Defect Prevention in Software Processes: An Action-Based Approach", In: The Journal of Systems and Software, Vol. 80, issue 4, April 2007, pp. 559-570, 2007.

- Chernak, Y., "A statistical Approach to the Inspection Checklist Formal Synthesis and Improvement", In: IEEE Transactions on Software Engineering, 22(12), 866-874, 1996.
- Chillarege, R., Bhandari, I., Chaar, J., Halliday, M., Moebus, D., Ray, B., Wong, M.Y., "Ortogonal Defect Classification – A Concept for In-Process Measurement", In: IEEE Transactions on Software Engineering, vol. 18, pp. 943-956, 1992.
- Cobb, R.H., Mills, H.D., "Engineering Software under Statistical Quality Control", In: IEEE Software, Vol. 7, No. 6, pp. 44-53, 1990.
- Collofello, J., Gosalla, B., "Application of Causal Analysis to the Software Modification Process", In: Software Practice and Experience, 23(10), pp. 1095–1105, 1993.
- Dalal, S. R., Horgan, J. R., Kettenring, J. R., "Reliable Software and Communication: Software Quality, Reliability, and Safety", In: Proc. of ICSE'93: 15th International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 425-435, 1993.
- Damele, G., Bazzana, G., Andreis, F., Aquilio, S., "Process Improvement through Root Cause Analysis", In: Proc. of the Third International Conference on Achieving Quality in Software, pp. 35-47, 1996.
- Damm, L., Lundberg, L., Wohlin, C., A model for software rework reduction through a combination of anomaly metrics, Journal of Systems and Software, 81, 1968-1982, 2008.
- Damm, L., Lundberg, L., "Company-wide Implementation of Metrics for Early Software Fault Detection", In: Proc. of ICSE'07: 29th International Conference on Software Engineering, Minneapolis, 2007.
- Damm, L., Lundberg, L., Wohlin, C., "Faults-slip-Through – A Concept for Measuring the Efficiency of the Test Process", In: Software Process: Improvement and Practice, Wiley Interscience, no.11, pp. 47-59, 2006.
- Dangerfield, O., Ambardekar, P., Paluzzi, P., Card, D., Giblin, D., "Defect Causal Analysis: A Report from the Field", In: Proc. of the International Conference on Software Quality, American Society for Quality Control, 1992.
- DeMillo, R.A, Mathur, A.P., "A grammar based fault classification scheme and its application to the classification of the errors of TEX". In: Technical Report, Software Engineering Research Center and Department of Computer Science, Purdue University, 1995.

- Eckes, G., "The Six Sigma Revolution: How General Electric and Others Turned Process Into Profits", John Wiley and Sons, 2000.
- Eisenstadt, M., "My hairiest bug war stories", In: Communications of the ACM, 40 (4):30-37, 1997.
- Endres, A., "An Analysis of Errors and Their Causes in Systems Programs", In: IEEE Transactions on Software Engineering, SE-1, 2, June 1975, pp. 140-149, 1975.
- Fagan, M.E., "Design and Code Inspection to Reduce Errors in Program Development", IBM Systems Journal, 15 (3), 182-211, 1976.
- Fairley, R. E., "Managing by the Numbers: a tutorial on quantitative measurement and control of software projects", In: Proc. of ICSE'99: 21st International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 677- 678, 1999.
- Fenton, N., Neil, M., Marsh, W., Hearty, P., Marquez, D., Krause, P., Mishra, R., "Predicting software defects in varying development lifecycles using Bayesian nets", In: Information and Software Technology, 49, pp. 32-43, 2007.
- Fenton, N., "A Critique of Software Defect Prediction Models", In: IEEE Transactions on Software Engineering, 25(5), 675-689, 1999.
- Florac, A.W., Carleton A.D., "Measuring the Software Process: Statistical Process Control for Software Process Improvement", Pearson Education, 1999.
- Freimut, B., "Developing and Using Defect Classification Schemes", In: Technical Report of the Fraunhofer Institute for Experimental Software Engineering, IESE 072.01/E, Kaiserslautern, 2001.
- Goncalves, F., Bezerra, C., Belchior, A., Coelho, C., Pires, C., "Implementing Causal Analysis and Resolution in Software Development Projects: The MiniDMAIC Approach", In: Proc. of the 19th Australian Conference on Software Engineering, pp. 112-119, 2008.
- Grady, R. B., "Software Failure Analysis for High-Return Process Improvement Decisions", In: Hewlett-Packard Journal, 47 (4), 15 - 24, 1996.
- Grady, R. B., "Practical Software Metrics for Project Management and Process Improvement", Prentice Hall, Upper Saddle River, NJ, 1992.
- Gras, J.J., "End-to-End Defect Modeling", In: IEEE Software, 21(5), 98-100, 2004.

- Gray, J., "Why do computers stop and what can be done about it?", In: Proc. of the Symposium on Reliability in Distributed Software and Database Systems (SRDS-5), pages 3-12, IEEE CS Press, 1986.
- Hamill, M., Goseva-Popstojanova, K. "Common Trends in Software Fault and Failure Data", In: IEEE Transactions on Software Engineering, 35, 484 -496, 2009.
- Han, T., Li, B., Xu, L., "A universal fault diagnostic expert system based on Bayesian Network", In: Proc. of the International Conference on Computer Science and Software Engineering, CSSE 2008, 260 – 263, 2008.
- Hayes, J.H., Raphael, I., Holbrook, E.A., Pruett, D.M., "A case history of International Space Station requirement faults", In: Proc. of the 11th IEEE International Conference on Engineering of Complex Computer Systems, Stanford University, California, 2006.
- Hearty, P., Fenton, N.E., Neil, M., and Cates, P., "Automated population of causal models for improved software risk assessment", In: Proc. of the International Conference on Automated Software Engineering, Long Beach, USA, 2005.
- Hong, G., Xie, M., Shanmugan, P., "A Statistical Method for Controlling Software Defect Detection Process", In: Computers and Industrial Engineering 37 (1-2), pp. 137–140, 1999.
- Humphrey, W.S., "Characterizing the Software Process: a Maturity Framework", In: IEEE Software, Vol. 5, No. 2, pp.73-79, 1988.
- Ishikawa, K., "Guide to Quality Control", Asian Productivity Organization, Tokyo, 1976 (revisado em 1982).
- IEEE, "IEEE Standard Classification for Software Anomalies". IEEE Std 1044-2009, C1 -15, 2010.
- Inhelder, B., Piaget, J., "The growth of logical thinking from childhood to adolescence", New York: Basic Books Meltzoff, 1958.
- ISO/IEC, "ISO/IEC 12207:2008 – Systems and software engineering – Software life cycle processes", ISO/IEC 12207:2008, 2008a.
- ISO/IEC, "ISO/IEC 15504 – Information Technology – Process Assessment – Part 7 – Assessment of Organizational Maturity", ISO/IEC 15504, 2008b.
- JabRef, "JabRef Reference Manager", disponível em <http://jabref.sourceforge.net/>, acesso em 12 de Fevereiro de 2011, 2011.

- Jacobs, J.C., van Moll, J.H., Krause, P.J., Kusters, R.J., Trienekens, J.J.M., "Effects of Virtual Development on Product Quality: Exploring Defect Causes", In: Proc. of the 11th Annual International Workshop on Software Technology and Engineering Practice (STEP'04), 2004.
- Jacobs, J.C., van Moll, J.H., Krause, P.J., Kusters, R.J., Trienekens, J.J.M., Brombacher, A., "Exploring Defect Causes in Products Developed by Virtual Teams", In: Journal on Information and Software Technology, 47( 6), 399 – 410, 2005.
- Jalote, P., Munshi, R., Probsting, T., "The When-Who-How analysis of defects for improving the quality control process", In: Journal of Systems and Software, 80, 584-589, 2007.
- Jalote, P., Agrawal, N., "Using Defect Analysis Feedback for Improving Quality and Productivity in Iterative Software Development", In: Proc. of the 3rd International Conference on Information and Communication Technology, ICICT, pp. 701–713, Cairo, 2005.
- Jalote, P., Saxena, A., "Optimum control limits for employing statistical process control in software process", In: IEEE Transactions on Software Engineering, 28, 1126-1134, 2002.
- Jalote, P., "CMM in Practice – Processes for Executing Software Projects at Infosys", SEI Series on Software Engineering, Addison Wesley, 2000.
- Jantti, M., Toroi, T., Eerola, A., "Difficulties in establishing a defect management process: A case study", In: Lecture Notes in Computer Science 4034 NCS, pp. 142-150, 2006.
- Jones, C.L., "A process-integrated approach to defect prevention", In: IBM Systems Journal, 24(2), pp. 150-67, 1985.
- Kalinowski, M., Mendes, E., Travassos G.H., "Automating and Evaluating Probabilistic Cause-Effect Diagrams to Improve Defect Causal Analysis", In: Proc. of the 12th Int. Conf. on Product Focused Software Development and Process Improvement (PROFES 2011), LNCS, Bari, Italy, 2011.
- Kalinowski, M., Travassos G.H., Mendes, E., Card, D.N., "Applying DPPI: A Defect Causal Analysis Approach Using Bayesian Networks", In: Proc. of the 11th Int. Conf. on Product Focused Software Development and Process Improvement (PROFES 2010), LNCS 6156, pp. 92–106, Limerick, Ireland, 2010.



- Kalinowski, M., Travassos, G.H., "Towards a Defect Causal Analysis Approach for Software Process Improvement and Organizational Learning", In: Proc. of the Workshop de Teses e Dissertações em Qualidade de Software (WTDQS) do VII Simpósio Brasileiro de Qualidade de Software, Florianópolis, 2008.
- Kalinowski, M., Travassos, G.H., Card, D.N., "Guidance for Efficiently Implementing Defect Causal Analysis", In: Proc. of the VII Simpósio Brasileiro de Qualidade de Software (SBQS), Florianópolis, 2008a.
- Kalinowski, M., Travassos, G. H., Card, D. N., "Towards a Defect Prevention Based Process Improvement Approach", In: Proc. of the 34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 199-206, Parma, Italy, 2008b.
- Kalinowski, M., Spínola, R.O., Dias-Neto, A.C., Bott, A., Travassos, G.H, "Inspeções de Requisitos de Software em Desenvolvimento Incremental: Uma Experiência Prática", In: Proc. of the VI Simpósio Brasileiro de Qualidade de Software (SBQS), Porto de Galinhas, 2007.
- Kalinowski, M., Travassos, G.H., "ISPIS: From Conception Towards Industry Readiness", In: Proc. of the XXVI International Conference of the Chilean Computer Science Society (SCCC), Iquique, Chile, 2007.
- Kalinowski, M., Travassos, G.H., "A Computational Framework for Supporting Software Inspections", In: Proc. of the 19th IEEE International Conference on Automated Software Engineering (ASE), Linz, Austria, 2004.
- Keene, J., "Managing Software Reliability and Support Costs", In: Proc. of the Leesburg Workshop on Reliability and Maintainability Computer-Aided Engineering in Concurrent Engineering, pp. 201-205, 1991.
- Kelsey, R.B., "Integrating a Defect Typology With Containment Metrics", In: ACM SIGSOFT Software Engineering Notes, 22(2), 64-67, 1997.
- Kitchenham, B.A., Charters, S., "Guidelines for Performing Systematic Literature Reviews in Software Engineering", In: Tech Report EBSE-2007 (version 2.3), Keele University, UK, 2007.
- Kitchenham, B.A., "Procedures for Performing Systematic Reviews", In: Joint Technical Report Keele University and National ICT Australia Ltd., 2004.
- Kitchenham, B.A., Dybä, T., Jorgensen, M., "Evidence-Based Software Engineering". In: Proc. of the International Conference on Software Engineering, ICSE 2004, pp. 273-281, 2004.

- Knuth, D.E., "The errors of TEX", In: Journal of Software Practice and Experience, 19 (7), pp. 607-685, 1989.
- Krause, P., Freimut, B., Suryan, W., "New Directions in Measurement for Software Quality Control", In: Proc. of the 10th International Workshop on Software Technology and Engineering Practice, STEP 2002, pp. 129-143, 2002.
- Lederer, A.L., Prasad, J., "A Causal Model for Software Cost Estimating Error", In: IEEE Transactions on Software Engineering, 24 (2), 137-148, 1998.
- Leszak, M., "Software defect analysis of a multi-release telecommunications system", In: Proc. of the 6th International Conference on Product Focused Software Development and Process Improvement (PROFES 2005), LNCS 3547, pp. 98-114, Oulu, Finland, 2005.
- Leszak, M., Perry, D. E., Stoll, D., "Classification and Evaluation of Defects in a Project Retrospective", In: Journal of Systems and Software, 61( 3), 173 - 187, 2002.
- Leszak, M., Perry, D., Stoll, D., "A Case Study in Root Cause Defect Analysis", In: Proc. of ICSE'00: International Conference on Software Engineering, pp. 428-437, 2000.
- Leveson, N.G., Harvey, R.R., "Software Fault Tree Analysis", In: Journal of Systems and Software, vol. 3, no.2, pp. 173-182, 1983.
- Li, Z., Tan, L., Wang, X., Lu, S., Zhou, Y., Zhai, C., "Have things changed now? An empirical study of bug characteristics in modern open source software", In: Proc. of the 1st Workshop on Architectural and System Support for Improving Software Dependability, pp. 25-33, 2006.
- Linger, R. C., "Cleanroom Software Engineering for Zero-Defect Software", In: Proc. of ICSE'93: 15th International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 2-13, 1993.
- Mafra, S.N., Barcelos, R.F., Travassos, G.H., "Aplicando uma Metodologia Baseada em Evidência na Definição de Novas Tecnologias de Software", In: Proc. of the XX Simpósio Brasileiro de Engenharia de Software (SBES), Florianópolis, 2006.
- Mays, R.G., "Applications of Defect Prevention in Software Development", In: IEEE Journal on Selected Areas in Communications, Vol.8, No.2, February 1990.
- Mays, R.G., Jones, C.L., Holloway, G.J., Studinski, D.P., "Experiences with Defect Prevention", In: IBM Systems Journal, 29(1), pp. 4-32, 1990.

- McDonald, M., Musson, R., Smith, R., "The Practical Guide to Defect Prevention – Techniques to Meet the Demand for More Reliable Software", Microsoft Best Practices Series, Microsoft Press, ISBN 978-0-7356-2253-1, 2008.
- Mellor, R., "The Roman Historians", London: Routledge, ISBN: 978-0-4151-1774-6, 1999.
- MiniTab, "MiniTab – Software Quality Improvement", disponível em <http://www.minitab.com/>, ultimo acesso em 12/02/2011, 2011.
- van Moll, J., Jacobs, J., Freimut, B., Trienekens, J., "The Importance of Life Cycle Modeling to Defect Detection and Prevention", In: Proc. of the 10th International Workshop on Software Technology and Engineering Practice, 2002. STEP 2002', pp. 144-155, 2002.
- Montgomery, D.C., "Introduction to Statistical Quality Control", John Wiley & Sons, Inc., ISBN: 978-0-470-16992-6, 1991.
- Montoni, M., Kalinowski, M., Lupo, P., Abrantes, J.F., Ferreira, A.I.F., Rocha, A. R, "Uma Metodologia para Desenvolvimento de Modelos de Desempenho de Processos para Gerência Quantitativa de Projetos de Software", In: Proc. of the VI Simpósio Brasileiro de Qualidade de Software (SBQS), Porto de Galinhas, 2007.
- Nakajo, T., Kume, H., "A Case History Analysis of Software Error Cause-Effect Relationships", In: IEEE Transactions on Software Engineering, 17(8), 830-838, 1991.
- Nakajo, T., "Foolproofing and Quality Feedback: Keys of Process-Based Management", In: Proc. of the Fifteenth Annual International Computer Software and Applications Conference, COMPSAC'91, IEEE Computer Society Press, 390-391, 1991.
- Nakamura, T., Hochstein, L., Basili, V.R., "Identifying Domain-Specific Defect Classes Using Inspections and Change History", In: Proc. of the International Symposium on Empirical Software Engineering (ISESE), Sept. 21-22, Rio de Janeiro, Brazil. p. 346-355, 2006.
- Nakashima, T., Oyama, M., Hisada, H., Ishii, N., "Analysis of software bug causes and its prevention", In: Journal on Information and Software Technology, 41, pp 1059-1068, 1999.
- Netica, "Netica Bayesian Network Software from Norsys", disponível em <http://www.norsys.com/>, ultimo acesso em 12/02/2011, 2011.

- Oshana, R., Coyle, F.P., "Implementing Cleanroom Software Engineering into a Mature CMM-Based Software Organization", In: Proc. of the 19th International Conference on Software Engineering, ICSE'97, ACM Press, New York, NY, USA, pp. 572-573, 1997.
- Paes, A., Revoredo, K, Zaverucha, G., "PFORTE: Revising Probabilistic FOL Theories", In: Proc. of the IBERAMIA-SBIA, Ribeirão Preto, Brasil, pp. 441-450, 2006.
- Paes, L.M.J.A., "Apoio Computacional a uma Abordagem para Análise Causal de Defeitos que Utiliza Redes Bayesianas", Projeto Final do Curso de Engenharia da Computação da UFRJ, 2010.
- Pan, K., Kim, S., Whitehead, J., "Toward an understanding of bug fix patterns", In: Journal on Empirical Software Engineering, 14, 286-315, 2009.
- Paul, R. A., Bastani, F.; Yen, I., Challagulla, V. U., "Defect-based Reliability Analysis for Mission-Critical Software", In: Proc. of the IEEE Computer Society's International Computer Software and Applications Conference, pp. 439 - 444, 2000.
- Pearl, J., "Causality: Reasoning, Models and Inference", Cambridge University Press, ISBN: 978-0-521-77362-1, 2000.
- Pearl, J., "Bayesian Networks: A model of self-activated memory for evidential reasoning". In: Proc. of the Cognitive Science Society, pp. 329-334. Greenwich, CT: Ablex, 1985.
- Pfleeger, S.L., Atlee, J.M., "Software Engineering: Theory and Practice", 4a edição, Prentice Hall, ISBN: 978-0136061694, 2009.
- Philips, R.T., "An approach to software causal analysis and defect extinction", In: Proc. of the IEEE Globecom Conference, pp. 412-416, 1986.
- Ploski, J., Rohr, M., Schwenkenberg, P., Hasselbring, W., "Research issues in software fault categorization", In: SIGSOFT Software Engineering Notes 32(6), 2007.
- Poppendieck, M., Poppendieck, T., "Lean Software Development: An Agile Toolkit", Addison Wesley Professional, ISBN: 0-321-43738-1, 2007.
- Pratt, W. M., "Experiences in the Application of Customer-Based Metrics in Improving Software Service Quality", In: Proc. of the International Conference on Communications, pp. 1459 - 1462, 1991.

- Robinson, B., Francis, P., Ekdahl, F., "A defect-driven process for software quality improvement", In: Proc. of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), ACM, New York, NY, USA, pp. 333-335, 2008.
- Robitaille, D., "Root Cause Analysis – Basic Tools and Techniques", Paton Press, ISBN: 1-932828-02-8, 2004.
- Rombach, D., Endres, A., "A Handbook of Software and Systems Engineering – Empirical Observations, Laws and Theories", The Fraunhofer IESE Series on Software Engineering, Pearson Addison Wesley, ISBN: , 0-321-15420-7, 2003.
- Rosen, C., "PLUNGE DA: A Case Study", In: SIGSOFT Software Engineering Notes 22(2), pp. 82-83, 1997.
- Russel, S., Norvig, P., "Inteligência Artificial", Tradução da Segund Edição, Editora Campus, ISBN: 85-352-1177-2, 2004.
- SEI, "CMMI for Development - Version 1.2". CMU/SEI-2006-TR-008. CMMI Product Team, August 2006, disponível em [www.sei.cmu.edu/cmmi](http://www.sei.cmu.edu/cmmi), 2006a.
- SEI, CMMI para Desenvolvimento – Versão 1.2. CMU/SEI-2006-TR-008. Equipe do Produto CMMI. Agosto de 2006, disponível em [www.sei.cmu.edu/cmmi](http://www.sei.cmu.edu/cmmi), 2006b.
- Shenvi, A. A., "Defect prevention with orthogonal defect classification", In: Proc. of the 2nd India Software Engineering Conference, ISEC, 83 – 87, 2009.
- Shull, F., "Developing Techniques for Using Software Documents: A Series of Empirical Studies", Ph.D. thesis, University of Maryland, College Park, 1998.
- Shull, F., Carver, J., Travassos, G.H., "An Empirical Methodology for Introducing Software Processes", In: Proc. of the European Software Engineering Conference, Vienna, Austria, pp. 288-296, 2001.
- SOFTEX, "Modelo MPS – Melhoria de Processo do Software Brasileiro, Guia Geral do MPS, Guia de Aquisição do MPS, Guia de Implementação do MPS, e Guia de Avaliação MPS". Sociedade Softex, Brasil. Disponível em <http://www.softex.br/mpsbr> , 2009.
- Spínola, R.O., Kalinowski, M., Travassos, G. H., "Uma Infra-Estrutura para Integração de Ferramentas CASE", In: Proc. of the XVIII Simpósio Brasileiro em Engenharia de Software (SBES), Brasília, 2004.

- Stanfill, C., Waltz, D., "Towards memory-based reasoning", In: Communications of the ACM, Vol.29, No.12, pp.1213-1228, 1986.
- Tian, J., "Software Quality Engineering – Testing, Quality Assurance and Quantifiable Improvement", IEEE Press, Wiley Interscience, ISBN 0-471-71345-7, 2005.
- Travassos, G.H., Gurov, D., Amaral, E.A.G.G., 2002, "Introdução à Engenharia de Software Experimental", In: Relatório Técnico COPPE/UFRJ – disponível em [www.cos.ufrj.br](http://www.cos.ufrj.br), 2002.
- Travassos, G.H., Shull, F., Carver, J., "Working with UML: A Software Design Process Based on Inspections for the Unified Modeling Language", In: Advances in Computers, vol. 54, Academic Press, 2001.
- Troster, J., Henshaw, J., Buss, E., "Filtering for Quality", In: Proc. of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research, (CASCON 93), IBM Press, pp. 429-449, 1993.
- Vantine, W., Benfield, K., Pritts, D., Ballard, K., "Evaluating and Incorporating New Age Software Technology for Identifying Systemic Root Causes", In: Proc. of the Joint ESA-NASA Space Flight Safety Conference, ESTEC, ESA SP-486, Noordwijk (NL), pp. 369-376, 2002.
- Walia, G., Carver, J., Philip, T., "Requirements Error Abstraction and Classification: An Empirical Study", In: Proc. of the 2006 International Symposium on Empirical Software Engineering (ISESE), Sept. 21-22, Rio de Janeiro, Brazil. pp. 336-345, 2006.
- Wang, Q., Jiang, N., Gou, L., Liu, X., Li, M., Wang, Y., "BSR: A Statistic-based Approach for Establishing and Refining Software Process Performance Baseline", In: Proc. of ICSE'06: 28th International Conference on Software Engineering, ACM Press, New York, NY, USA, pp. 585-594., 2006.
- WEKA, "WEKA: Open Source Machine Learning Software in Java", disponível em <http://www.cs.waikato.ac.nz/ml/weka/>, último acesso em 15/02/2011, 2011.
- Wheeler, D., Chambers, D.S., "Understanding Statistical Process Control", SPC Press, ISBN: 978-0945320135, 1992.
- Wigglesworth, J., "Surveys as a Method for Improving the Development Process", In: Proc. of the 1993 conference of the Centre for Advanced Studies on Collaborative research, IBM Press, pp. 337-355, 1993.

- Wilde, O., "Lady Windermere's Fan", Published in The Importance of Being Earnest and Other Plays. London: Penguin, ISBN 0-14-048209-1, 1940.
- Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Wesslén, A., "Experimentation in Software Engineering – An Introduction", Kluwer Academic Publishers, ISBN 0-7923-8682-5, 2000a.
- Wohlin, C., Host, M., Ohlsson, M., "Understanding the Sources of Software Defects: A Filtering Approach", In: Proc. of the 8th International Workshop on Program Comprehension, IWPC 2000', pp. 9-17, 2000b.
- Yu, W. D., "Software fault prevention approach in coding and root cause analysis", In: Bell Labs Technical Journal, Vol.3 (2), pp.3-21, 1998.

# ANEXO A – LISTAGEM DE ARTIGOS DA EXECUÇÃO DO PROTOCOLO DA REVISÃO SISTEMÁTICA

*Este anexo apresenta as listagens de artigos obtidos nas diferentes execuções do protocolo da revisão sistemática.*

## A1 –Artigos Obtidos em Agosto de 2006 das Máquinas de Busca

A Tabela A.1 contém a listagem dos 159 artigos retornados pela aplicação do protocolo nas máquinas de busca na execução de Agosto de 2006. A coluna de status indica o resultado do processo de filtragem. Os valores possíveis para o status são: ETA (eliminado por título e abstract), EC (eliminado por conteúdo) e I (incluído). Os artigos incluídos encontram-se destacados em cinza.

**Tabela A.1. Artigos obtidos em Agosto de 2006 das Máquinas de Busca.**

#	Artigo	Status
1	Abu Bakar, N.; Mohamed, A. & Ismail, M. (2004), Software development of a voltage sag analysis tool for power quality study, Proceedings of the National Power and Energy Conference, pp. 36-40, 2004.	ETA
2	Abu Bakar, N.; Mohamed, A.; Ismail, M. & Hamzah, N., A voltage sag analysis software tool for determine areas of vulnerability, in 'TENCON 2004. 2004 IEEE Region 10 Conference', pp. 299-302 Vol. 3, 2004.	ETA
3	Al-Shehab, A. J., Hughes, R. T., Winstanley, G., Facilitating Organisational Learning through Causal Mapping Techniques in IS/IT Project Risk Management, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3782 NAI, 145 - 154, 2005.	I
4	Armour, P. G., Software: Hard Data, Communications of the ACM, Vol. 49 (9), pp. 15 -17, 2006.	EC
5	Armour, P. G. (2005), 'Project portfolios: organizational management of risk', Commun. ACM 48(3), 17--20.	ETA
6	Avritzer, A. & Larson, B. (1993), 'Load testing software using deterministic state testing', , 82 - 88.	ETA
7	Barney, D.; Haley, D. & Nikandros, G. (2001), Calculating train braking distance, in 'SCS '01: Proceedings of the Sixth Australian workshop on Safety critical systems and software', Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 23--29.	ETA



8	Ben-Asher, Y.; Eytani, Y.; Farchi, E. & Ur, S. (2006), Producing scheduling that causes concurrent programs to fail, in 'PADTAD '06: Proceeding of the 2006 workshop on Parallel and distributed systems: testing and debugging', ACM Press, New York, NY, USA, pp. 37--40.	ETA
9	Berenbach, B. (2006), Impact of organizational structure on distributed requirements engineering processes: lessons learned, in 'GSD '06: Proceedings of the 2006 international workshop on Global software development for the practitioner', ACM Press, New York, NY, USA, pp. 15--19.	ETA
10	Bertolino, A. (2004), 'The (Im)maturity level of software testing', SIGSOFT Softw. Eng. Notes 29(5), 1--4.	ETA
11	Bhandari, I., Halliday, M., Tarver, E., Brown, D., Chaar, J., Chillarege, R., A Case Study of Software Process Improvement During Development, IEEE Transactions on Software Engineering, 19 (12), 1157 – 1170, 1993.	I
12	Bhandari, I., Roth, N., Post-process Feedback with and without Attribute Focusing: a Comparative Evaluation, in 'ICSE '93: Proceedings of the 15th International Conference on Software Engineering', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 89-98, 1993.	I
13	Biffl, S., Grossmann, W., Evaluating the accuracy of defect estimation models based on inspection data from two inspection cycles, in 'ICSE '01: Proceedings of the 23rd International Conference on Software Engineering', IEEE Computer Society, Washington, DC, USA, pp. 145-154, 2001.	EC
14	Blanqui, F. (1997), 'A document-centered approach for an open CASE environment framework connected with the world-wide web', SIGSOFT Softw. Eng. Notes 22(2), 58--63.	ETA
15	Boehm, B., A view of 20th and 21st century software engineering, in 'ICSE '06: Proceeding of the 28th international conference on Software engineering', ACM Press, New York, NY, USA, pp. 12-29, 2006.	I
16	Brazendale, J. (1995), 'IEC 1508: functional safety: safety-related systems', Proceedings of the IEEE International Software Engineering Standards Symposium, 8 - 17.	ETA
17	Brykczynski, B. & Wheeler, D. A., An annotated bibliography on software inspections, SIGSOFT Software Engineering Notes, Vol. 18 (1), pp. 81-88, 1993.	EC
18	Brykczynski, B., The software inspection process-applying the principles of Deming and Crosby, Information and Systems Engineering, Vol. 1 (1), pp. 23-37, 1995.	EC
19	Bush, M., Getting started on metrics - JPL productivity and quality, in 'ICSE '90: Proceedings of the 12th international conference on Software engineering', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 133-142, 1990.	I
20	Buss, E. & Henshaw, J. (1992), Experiences in program understanding, in 'CASCON '92: Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research', IBM Press, , pp. 157--189.	ETA
21	Buss, E. & Henshaw, J. (1991), A software reverse engineering experience, in 'CASCON '91: Proceedings of the 1991 conference of the Centre for Advanced Studies on Collaborative research', IBM Press, , pp. 55--73.	ETA

22	Card, D., Learning from our mistakes with defect causal analysis, IEEE Software, 15(1), pp. 56-63, 1998.	I
23	Carroll, E. R. (2005), Estimating software based on use case points, in 'OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications', ACM Press, New York, NY, USA, pp. 257--265.	ETA
24	Cerone, A.; Lindsay, P. & Connelly, S. (2005), Formal analysis of human-computer interaction using model-checking, in 'Software Engineering and Formal Methods, 2005. SEFM 2005. Third IEEE International Conference on', pp. 352--361.	ETA
25	Chaar, J. K.; Halliday, M. J.; Bhandari, I. S. & Chillarege, R. (1993), 'In-process evaluation for software inspection and test', IEEE Transactions on Software Engineering 19( 11), 1055 - 1070.	ETA
26	Chaki, S.; Groce, A. & Strichman, O. (2004), 'Explaining abstract counterexamples', Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 73 - 82.	ETA
27	Changfoot, M.; Fines, T.; Lawson, R.; Lecic, N.; Morenz, P.; Mortson, D.; Patel, P.; Roberts, S.; Sahib, S. & Sing, R. (1993), How one product team met the quality challenge, in 'CASCON '93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research', IBM Press, , pp. 371--399.	ETA
28	Cheng, K.; Wang, C.; Lee, J.; Chou, Y.; Huang, C. & Wu, C. (2003), FAME: A Fault-Pattern Based Memory Failure Analysis Framework, in 'ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design', IEEE Computer Society, Washington, DC, USA, pp. 595.	ETA
29	Chernak, Y., A statistical Approach to the Inspection Checklist Formal Synthesis and Improvement, IEEE Transactions on Software Engineering, 22(12), 866-874, 1996.	I
30	Chi, D.; Lin, H. & Kuo, W. (1989), Software reliability and redundancy optimization, in 'Reliability and Maintainability Symposium, 1989. Proceedings., Annual', pp. 41--45.	ETA
31	Chmiel, R. & Loui, M. C. (2004), Debugging: from novice to expert, in 'SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education', ACM Press, New York, NY, USA, pp. 17--21.	ETA
32	Ciolkowski, M., Laitenberger, O., Rombach, D., Shull, F., Perry, D., Software inspections, reviews & walkthroughs, in 'ICSE '02: Proceedings of the 24th International Conference on Software Engineering', ACM Press, New York, NY, USA, pp. 641-642, 2002.	EC
33	Collofello, J., Gosalla, B., Application of Causal Analysis to the Software Modification Process, Software Practice and Experience, 23(10), pp. 1095--1105, 1993.	I
34	Cook, J. (2001), 'Supporting rapid prototyping through frequent and reliable deployment of evolving components', Proceedings of the International Workshop on Rapid System Prototyping, 194 - 199.	ETA
35	Cortellessa, V.; Goseva-Popstojanova, K.; Appukkutty, K.; Guedem, A.; Hassan, A.; Elnaggar, R.; Abdelmoez, W. & Ammar, H. (2005), 'Model-based performance risk analysis', Software Engineering, IEEE Transactions on 31(1), 3--20.	ETA

36	Cronqvist, M. (2004), 'Troubleshooting a large erlang system', Proceedings of the ACM SIGPLAN 2004 Erlang Workshop, 11 - 15.	ETA
37	D, K.; Y, L.; N, V.; P, H.; Y, T.; C, C. & J, G. (), 'Object state testing and fault analysis for reliable software systems', Proceedings .	ETA
38	Dadzie, J. (2005), 'Understanding software patching', Queue 3(2), 24--30.	ETA
39	Dalal, S. R., Horgan, J. R., Kettenring, J. R., Reliable Software and Communication: Software Quality, Reliability, and Safety, in 'ICSE '93: Proceedings of the 15th International Conference on Software Engineering', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 425-435, 1993.	I
40	Damele, G., Bazzana, G., Andreis, F., Aquilio, S., Process Improvement through Root Cause Analysis, in 'Proceedings of Third International Conference on Achieving Quality in Software', pp. 35--47, 1996.	I
41	De Lemos, R., Analysing failure behaviours in component interaction, Journal of Systems and Software, 71, (1-2), pp. 97-115, 2004.	EC
42	Deb, S.; Pattipati, K. R.; Raghavan, V.; Shakeri, M. & Shrestha, R. (1994), 'Multi-signal flow graphs: a novel approach for system testability analysis and fault diagnosis', AUTOTESTCON (Proceedings), 361 - 373.	ETA
43	Doernhoefer, M. (1999), 'Surfing the Net for Software Engineering Notes', SIGSOFT Softw. Eng. Notes 24(3), 15--24.	ETA
44	Ebert, C. (1997), Experiences with criticality predictions in software development, in 'ESEC '97/FSE-5: Proceedings of the 6th European conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering', Springer-Verlag New York, Inc., New York, NY, USA, pp. 278--293.	ETA
45	Eick, S. G.; Loader, C. R.; Long, M. D.; Votta, L. G. & Wiel, S. V. (1992), Estimating software fault content before coding, in 'ICSE '92: Proceedings of the 14th international conference on Software engineering', ACM Press, New York, NY, USA, pp. 59--65.	ETA
46	Elbaum, S. & Munson, J. (2000), 'Investigating software failures with a software black box', IEEE Aerospace Conference Proceedings 4, 547 - 566.	ETA
47	Esaki, K., Yamada, S., Takahashi, M., Hihara, K., A quality engineering approach to human factors affecting software reliability in design process, Electronics and Communications in Japan, Part III: Fundamental Electronic Science (English translation of Denshi Tsushin Gakkai Ronbunshi) 85 (3), pp. 33-42, 2002.	EC
48	Fairley, R. E., Managing by the Numbers: a tutorial on quantitative measurement and control of software projects, in 'Proceedings of the 21st International Conference on Software Engineering, ICSE 99', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 677- 678, 1999.	I
49	Fetzer, C. & Xiao, Z. (2003), 'HEALERS: A Toolkit for Enhancing the Robustness and Security of Existing Applications', Proceedings of the International Conference on Dependable Systems and Networks, 317 - 322.	ETA
50	FM, H.; RA, P. & WE, S. (), 'Hardware/software FMECA', 1983 Proceedings Annual Reliability and Maintainability Symposium.	ETA
51	Galli, M.; Lanza, M.; Nierstrasz, O. & Wuyts, R. (2004), 'Ordering broken unit tests for focused debugging', IEEE International Conference on Software Maintenance, ICSM, 114 - 123.	ETA

52	Gittens, M.; Lutfiyya, H. & Bauer, M. (2004), 'Continuous evolutionary one-step-ahead testing', SIGSOFT Softw. Eng. Notes 29(5), 1--4.	ETA
53	Goldberg, A.; Wang, T. C. & Zimmerman, D. (1994), Applications of feasible path analysis to program testing, in 'ISSTA '94: Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis', ACM Press, New York, NY, USA, pp. 80--94.	ETA
54	Goodall, S. (1994), 'Analog/mixed signal fault diagnosis algorithm and tool review', AUTOTESTCON (Proceedings), 351 - 360.	ETA
55	Grady, R. B., Software Failure Analysis for High-Return Process Improvement Decisions, Hewlett-Packard Journal, 47 (4), 15 - 24, 1996.	I
56	Gras, J.J., End-to-End Defect Modeling, IEEE Software, 21(5), 98-100, 2004.	I
57	Gutzman, H. W. & Kellerman, E. (1982), Uses of APL in a manufacturing environment, in 'APL '82: Proceedings of the international conference on APL', ACM Press, New York, NY, USA, pp. 117--120.	ETA
58	Harper, R. & O'Loughlin, M. J. (1987), Manufacturing process analysis—tools and applications, in 'WSC '87: Proceedings of the 19th conference on Winter simulation', ACM Press, New York, NY, USA, pp. 731--737.	ETA
59	Harrell, J. (2006), 'Using SPI to identify root causes', Circuits Assembly 17 (6).	ETA
60	Hassan, A. E., Holt, R. C., Mockus, A., Report on MSR 2004: International workshop on mining software repositories, SIGSOFT Softw. Eng. Notes 30(1), pp. 4, 2005.	EC
61	Hentenryck, P. V. (2000), 'Constraint programming', SIGSOFT Softw. Eng. Notes 25(1), 89--90.	ETA
62	Hong, G., Xie, M., Shanmugan, P., A Statistical Method for Controlling Software Defect Detection Process, Computers and Industrial Engineering 37 (1-2), pp. 137-140, 1999.	I
63	Howden, W. E. & Shi, G. M. (1996), Linear and structural event sequence analysis, in 'ISSTA '96: Proceedings of the 1996 ACM SIGSOFT international symposium on Software testing and analysis', ACM Press, New York, NY, USA, pp. 98--106.	ETA
64	Howe, A. E. & Fuegi, A. D. (1994), 'Methods for finding influences on program failure', Proceedings of the International Conference on Tools with Artificial Intelligence, 764 - 767.	ETA
65	Howell, C. (1995), 'Surfing the Net for Software Engineering Notes', SIGSOFT Softw. Eng. Notes 20(3), 2--7.	ETA
66	Howles, T. (2003), 'Fostering the growth of a software quality culture', SIGCSE Bull. 35(2), 45--47.	ETA
67	Humphrey, W. S.; Kitson, D. H. & Gale, J. (1991), A comparison of U.S. and Japanese software process maturity, in 'ICSE '91: Proceedings of the 13th international conference on Software engineering', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 38--49.	ETA
68	Humphrey, W. S.; Kitson, D. H. & Kasse, T. C. (1989), The state of software engineering practice, in 'ICSE '89: Proceedings of the 11th international conference on Software engineering', ACM Press, New York, NY, USA, pp. 277--285.	ETA
69	J, G. & A, W. (), 'Application of fault tree analysis for software safety requirements development', Real Time Systems '95.	ETA

70	Jacobs, J.C., van Moll, J.H., Krause, P.J., Kusters, R.J., Trienekens, J.J.M., Brombacher, A., Exploring Defect Causes in Products Developed by Virtual Teams, Journal on Information and Software Technology, 47( 6), 399 – 410, 2005.	I
71	Jacobs, J.C., van Moll, J.H., Krause, P.J., Kusters, R.J., Trienekens, J.J.M., Effects of Virtual Development on Product Quality: Exploring Defect Causes, Proceedings of the 11th Annual International Workshop on Software Technology and Engineering Practice (STEP'04), 2004.	I
72	Jalote, P., Agrawal, N., “Using Defect Analysis Feedback for Improving Quality and Productivity in Iterative Software Development”, (Invited paper, 3rd International Conference on Information and Communication Technology, ICICT, 2005.), pp. 701 – 713, Cairo, 2005.	I
73	Jantti, M., Toroi, T., Eerola, A., Difficulties in establishing a defect management process: A case study, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 4034 NCS, pp. 142-150, 2006.	I
74	Johnson, C. & Telford, A. (1996), 'Extending the application of formal methods to analyse human error and system failure during accident investigations', Software Engineering Journal 11(6), 355--365.	ETA
75	Jones, C.L., A process-integrated approach to defect prevention, IBM Systems Journal, 24(2), 150-67, 1985.	I
76	Kajko-Mattsson, M.; Forssander, S. & Olsson, U. (2001), Corrective maintenance maturity model (CM3): maintainer's education and training, in 'ICSE '01: Proceedings of the 23rd International Conference on Software Engineering', IEEE Computer Society, Washington, DC, USA, pp. 610--619.	ETA
77	Kalyanpur, A.; Parsia, B.; Sirin, E. & Cuenca-Grau, B. (2006), 'Repairing unsatisfiable concepts in OWL ontologies', Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 4011 NCS, 170 - 184.	ETA
78	Kamoun, F. (2005), 'Toward best maintenance practices in communications network management', Int. J. Netw. Manag. 15(5), 321--334.	ETA
79	Keene, J., Managing Software Reliability and Support Costs, in 'Proceedings of the Leesburg Workshop on Reliability and Maintainability Computer-Aided Engineering in Concurrent Engineering', pp. 201-205, 1991.	I
80	Kelsey, R.B., Bad fixes, change specifications, and linguistic constraints on problem diagnosis, ACM SIGSOFT Software Engineering Notes, 21(2), 74-78, 1996.	EC
81	Kelsey, R.B., Integrating a Defect Typology with Containment Metrics, ACM SIGSOFT Software Engineering Notes, 22(2), 64-67, 1997.	I
82	Kelsey, R. B. (1996), 'Book Review: An ISO 9000 Approach To Building Quality Software', SIGSOFT Softw. Eng. Notes 21(4), 97--98.	ETA
83	Kelsey, R.B., (1996), 'Book Review: A Quantitative Approach to Software Management: The ami Handbook', SIGSOFT Softw. Eng. Notes 21(4), 98--99.	ETA
84	Kitchenham, B. A. & Pfleeger, S. L. (2002), 'Principles of survey research: part 3: constructing a survey instrument', SIGSOFT Softw. Eng. Notes 27(2), 20--24.	ETA



85	Knight, J. C. (2005), 'Focusing software education on engineering', SIGSOFT Softw. Eng. Notes 30(2), 3--5.	ETA
86	Kocan, F. & Saab, D. G. (1999), Dynamic fault diagnosis on reconfigurable hardware, in 'DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation', ACM Press, New York, NY, USA, pp. 691--696.	ETA
87	Krause, P., Freimut, B., Suryan, W., New Directions in Measurement for Software Quality Control, in 'Proceedings of the 10 <sup>th</sup> International Workshop on Software Technology and Engineering Practice, STEP 2002', pp. 129-143, 2002.	I
88	Kuhn, D.; Wallace, D. & Gallo, J. (2004), 'Software fault interactions and implications for software testing', Software Engineering, IEEE Transactions on 30(6), 418--421.	ETA
89	Lane, J. A., Zubrow, D., Integrating measurement with improvement: an action-oriented approach: experience report, in 'ICSE '97: Proceedings of the 19th international conference on Software engineering', ACM Press, New York, NY, USA, pp. 380-389, 1997.	EC
90	Lederer, A.L., Prasad, J., A Causal Model for Software Cost Estimating Error, IEEE Transactions on Software Engineering, 24 (2), pp. 137-148, 1998.	I
91	Lee, J. & Cha, S. (2005), 'Fault tree construction of hybrid system requirements using qualitative formal method', Reliability Engineering and System Safety 87(1), 121-31.	ETA
92	Leitch, R. & Stefanini, A. (1989), 'Task dependent tools for intelligent automation', Artificial Intelligence in Engineering 4( 3), 126 - 143.	ETA
93	Leszak, M., Perry, D., Stoll, D., A Case Study in Root Cause Defect Analysis, in 'International Conference on Software Engineering, ICSE 2000', pp. 428-437, 2000.	I
94	Leszak, M., Perry, D. E., Stoll, D., Classification and evaluation of defects in a project retrospective, Journal of Systems and Software, 61( 3), 173 - 187, 2002.	I
95	Linger, R. C., Cleanroom Software Engineering for Zero-Defect Software, in 'ICSE '93: Proceedings of the 15th International Conference on Software Engineering', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 2-13, 1993.	I
96	Loopik, A. (1998), 'ROSALIND-An analog debug environment feasibility study and system proposal', Hewlett Packard Lab.	ETA
97	Lucas Martinez, F. J. & Toval Alvarez, A. (2005), 'A precise approach for the analysis of the UML models consistency', Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 3770 NCS, 74 - 84.	ETA
98	Lutz, R. R. & Mikulski, I. C. (2003), Requirements discovery during the testing of safety-critical software, in 'ICSE '03: Proceedings of the 25th International Conference on Software Engineering', IEEE Computer Society, Washington, DC, USA, pp. 578--583.	ETA
99	Lyu, M. (1991), 'PANEL: research and development issues in software reliability engineering', SIGSOFT Softw. Eng. Notes 16(2), 23--30.	ETA
100	Maly, W. (1987), Realistic fault modeling for VLSI testing, in 'DAC '87: Proceedings of the 24th ACM/IEEE conference on Design automation', ACM Press, New York, NY, USA, pp. 173--180.	ETA

101	Mathiassen, L. & Sørensen, C. (1994), Managing CASE introduction: beyond software process maturity, in 'SIGCPR '94: Proceedings of the 1994 computer personnel research conference on Reinventing IS : managing information technology in changing organizations', ACM Press, New York, NY, USA, pp. 242--251.	ETA
102	Matras, J. R. (1994), 'Requirements for abnormal conditions and events (ACES) analysis', IEEE Nuclear Science Symposium & Medical Imaging Conference 3, 1079 - 1082.	ETA
103	Mays, R.G., Applications of Defect Prevention in Software Development, IEEE Journal on Selected Areas in Communications, Vol.8, No.2, pp. 164-168, February 1990.	I
104	Mays, R.G., Jones, C.L., Holloway, G.J., Studinski, D.P., Experiences with Defect Prevention, IBM Systems Journal, 29(1), pp. 4-32, 1990.	I
105	Müller, H. A.; Tilley, S. R. & Wong, K. (1993), Understanding software systems using reverse engineering technology perspectives from the Rigi project, in 'CASCON '93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research', IBM Press, , pp. 217--226.	ETA
106	Miller, M. J.; Pulgar-Vidal, F. & Ferrin, D. M. (2002), Making simulation relevant in business: achieving higher levels of CMMI maturity using simulation, in 'WSC '02: Proceedings of the 34th conference on Winter simulation', Winter Simulation Conference, , pp. 1473--1478.	ETA
107	Mogilensky, J. (1991), Process maturity as a guide to phased Ada adoption, in 'WADAS '91: Proceedings of the eighth annual Washington Ada symposium & summer SIGAda meeting on Ada', ACM Press, New York, NY, USA, pp. 16--23.	ETA
108	Mohri, Y. & Kikuno, T. (1991), Fault analysis based on fault reporting in JSP software development, in 'Computer Software and Applications Conference, 1991. COMPSAC '91., Proceedings of the Fifteenth Annual International', pp. 591--596.	ETA
109	Molina, P.; Moraes, R.; Baggio, J. & Tognon, E. (2004), 'Continuous wave Doppler Methods to dialysis access monitoring', Annual International Conference of the IEEE Engineering in Medicine and Biology - Proceedings 26 I, 2352 - 2355.	ETA
110	van Moll, J., Jacobs, J., Freimut, B., Trienekens, J., The Importance of Life Cycle Modeling to Defect Detection and Prevention, in '10 <sup>th</sup> International Workshop on Software Technology and Engineering Practice, 2002. STEP 2002', pp. 144-155, 2002.	I
111	Nakajo, T., Kume, H., A Case History Analysis of Software Error Cause-Effect Relationships, IEEE Transactions on Software Engineering, 17(8), 830-838, 1991.	I
112	Nakajo, T., Foolproofing and Quality Feedback: Keys of Process-Based Management, Proceedings of the Fifteenth Annual International Computer Software and Applications Conference, COMPSAC'91, IEEE Computer Society Press, 390-391, 1991.	I
113	Nyst, C. T. L. M. & van der Schaaf, T. W. (2005), The use of human factors and risk analysis in anesthesia in the Netherlands, in 'EACE '05: Proceedings of the 2005 annual conference on European association of cognitive ergonomics', University of Athens, , pp. 205--211.	ETA

114	O'Connor, R. & Coleman, G. (2002), Strategies for personal process improvement a comparison, in 'SAC '02: Proceedings of the 2002 ACM symposium on Applied computing', ACM Press, New York, NY, USA, pp. 1036--1040.	ETA
115	Oshana, R., Coyle, F.P., Implementing Cleanroom Software Engineering into a Mature CMM-Based Software Organization, Proceedings of the 19th International Conference on Software Engineering, ICSE'97, ACM Press, New York, NY, USA, pp. 572-573, 1997.	I
116	Paradkar, A.; Tai, K. & Vouk, M. (1996), 'Automatic test-generation for predicates', IEEE Transactions on Reliability 45( 4), 515 - 530.	ETA
117	Paul, R. A., Bastani, F.; Yen, I., Challagulla, V. U., Defect-based Reliability Analysis for Mission-Critical Software, Proceedings of the IEEE Computer Society's International Computer Software and Applications Conference, pp. 439 - 444, 2000.	I
118	Philips, R.T., An approach to software causal analysis and defect extinction, in 'Proceedings of the IEEE Globecom Conference, pp. 412-416, 1986.	I
119	Prasad, V. (2005), Parallel processors and an approach to the development of inference engine, in 'Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference on', pp. 300--305.	ETA
120	Pratt, W. M., Experiences in the Application of Customer-Based Metrics in Improving Software Service Quality, Conference Record of the International Conference on Communications, pp. 1459 – 1462, 1991.	I
121	Reinach, S. & Viale, A. (2006), 'Application of a human error framework to conduct train accident/incident investigations', Accident Analysis and Prevention 38( 2), 396 - 406.	ETA
122	Rosen, C., PLUNGE DA: A Case Study, ACM SIGSOFT Software Engineering Notes, 22(2), pp. 82-83, 1997.	I
123	Rzevski, G., Identification of factors which cause software failure, 'Proceedings Annual Reliability and Maintainability Symposium', 1982.	EC
124	Saab, K.; Hamida, N. B. & Kaminska, B. (2000), Closing the gap between analog and digital, in 'DAC '00: Proceedings of the 37th conference on Design automation', ACM Press, New York, NY, USA, pp. 774--779.	ETA
125	Safford, I. E. L. (1993), 'Improving software development through the proper implementation of a standard software process; case study and lessons learned', , 382 - 387.	ETA
126	Salsbury, T. & Diamond, R. (2000), 'Implementation and testing of a fault detection software tool for improving control system performance in a large commercial building', Proceedings ACEEE Summer Study on Energy Efficiency in Buildings 7, 7 - 147.	ETA
127	Schneider, A.; Diener, K.; Ivask, E.; Raik, J.; Ubar, R.; Miklos, P. (2002), Internet-Based Collaborative Test Generation with MOSCITO, in 'DATE '02: Proceedings of the conference on Design, automation and test in Europe', IEEE Computer Society, Washington, DC, USA, pp. 221	ETA
128	Shaw, D.; Al-Khalili, D. & Rozon, C. (2002), 'Fault security analysis of CMOS VLSI circuits using defect-injectable VHDL models', Integration, the VLSI Journal 32( 1-2), 77 - 97.	ETA



129	Shaw, D.; Al-Khalili, D. & Rozon, C. (2001), Accurate CMOS bridge fault modeling with neural network-based VHDL saboteurs, in 'ICCAD '01: Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design', IEEE Press, Piscataway, NJ, USA, pp. 531--536.	ETA
130	Sherriff, M. (2005), Utilizing verification and validation certificates to estimate software defect density, in 'ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering', ACM Press, New York, NY, USA, pp. 381--384.	ETA
131	Silva, N.; Vital, J., Franca, J., Automatic non-linearity signature analysis of data converters, IEE Conference Publication (393), pp. 116-123, 1994.	EC
132	Soloway, E.; Lampert, R.; Letovsky, S.; Littman, D. & Pinto, J. (1988), 'Designing documentation to compensate for delocalized plans', Commun. ACM 31(11), 1259--1267.	ETA
133	Spiegel, G. & Stroele, A. P. (1995), A unified approach to the extraction of realistic multiple bridging and break faults, in 'EURO-DAC '95/EURO-VHDL '95: Proceedings of the conference on European design automation', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 184--189.	ETA
134	Stalhane, T., Root cause analysis and gap analysis - a tale of two methods, in: 'Proceedings of the 11 <sup>th</sup> European Conference on Software Process Improvement, EuroSPI 2004, Trondheim, Norway, 2004.	EC
135	Strunge, J. J. (1977), Fault modeling in a hierarchical simulator, in 'DAC '77: Proceedings of the 14th conference on Design automation', IEEE Press, Piscataway, NJ, USA, pp. 118--127.	ETA
136	Sycara, K. & Miyashita, K. (1994), 'Learning from failure experiences in case-based schedule repair', Proceedings of the Hawaii International Conference on System Sciences 3, 122 - 131.	ETA
137	T, C.; J, S.; K, M.; J, C.; B, K. & J, M. (), 'Human error in the software generation process', Technology and Assessment of Safety Critical Systems.	ETA
138	Thomas, M. (2004), Engineering judgement, in 'SCS '04: Proceedings of the 9th Australian workshop on Safety critical systems and software', Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 43--47.	ETA
139	Tian, J., Early measurement and improvement of software reliability, ACM SIGSOFT Software Engineering Notes, 25 (1), pp. 89, 2000.	EC
140	Troster, J., Henshaw, J., Buss, E., Filtering for Quality, in 'Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research, CASCON 93', IBM Press, pp. 429-449, 1993.	I
141	Tu, F.; Azam, M. S.; Shlapak, Y.; Pattipati, K.; Karanam, R. & Amin, S. (2001), 'Integrated reliability analysis, diagnostics and prognostics for critical power systems', AUTOTESTCON (Proceedings), 416 - 440.	ETA
142	Valentin, R.; Cunningham, J.; Osterman, M.; Dasgupta, A.; Pecht, M. G. & Tsagos, D. (2002), Weapon and communication systems: virtual life assessment of electronic hardware used in the Advanced Amphibious Assault Vehicle (AAAV), in 'WSC '02: Proceedings of the 34th conference on Winter simulation', Winter Simulation Conference, , pp. 948--953.	ETA

143	Vantine, W., Benfield, K., Pritts, D., Ballard, K., Evaluating and Incorporating New Age Software Technology for Identifying Systemic Root Causes, in: 'Proceedings of the Joint ESA-NASA Space Flight Safety Conference, ESTEC', ESA SP-486, Noordwijk (NL), pp. 369 - 376, 2002.	I
144	Venkatesh, M.; Leo, R. J.; Kuzawinski, K. M.; Diamond, L. P. & Verville, J. C. (1995), Supporting teamwork at Xerox, in 'SIGCPR '95: Proceedings of the 1995 ACM SIGCPR conference on Supporting teams, groups, and learning inside and outside the IS function reinventing IS', ACM Press, New York, NY, USA, pp. 194-201.	ETA
145	Visconti, M. & Cook, C. (1993), 'Software system documentation process maturity model', , 352 - 357.	ETA
146	Voas, J.; Charron, F.; McGraw, G.; Miller, K. & Friedman, M. (1997), 'Predicting how badly 'good' software can behave', IEEE Software 14(4), 73 - 83.	ETA
147	Vouk, M. A. & Tai, K. C. (1993), Some issues in multi-phase software reliability modeling, in 'CASCON '93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research', IBM Press, , pp. 513--523.	ETA
148	Wang, Q., Jiang, N., Gou, L., Liu, X., Li, M., Wang, Y., BSR: A Statistic-based Approach for Establishing and Refining Software Process Performance Baseline, in 'ICSE '06: Proceeding of the 28th International Conference on Software Engineering', ACM Press, New York, USA, pp. 585-594, 2006.	I
149	Watson, I., Abdullah, S., Developing case-based reasoning systems: a case study in diagnosing building defects, in 'Case Based Reasoning: Prospects for Applications, IEE Colloquium on', pp. 11-13, 1994.	EC
150	Weber-Wulff, D. (2000), 'Object-oriented software engineering: conquering complex and changing systems', SIGSOFT' Softw. Eng. Notes 25(3), 65--66.	ETA
151	Weiss, D. M.; Bennett, D.; Payseur, J. Y.; Tendick, P. & Zhang, P. (2002), Goal-oriented software assessment, in 'ICSE '02: Proceedings of the 24th International Conference on Software Engineering', ACM Press, New York, NY, USA, pp. 221-231.	ETA
152	Wigglesworth, J., Surveys as a Method for Improving the Development Process, in 'CASCON '93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research', IBM Press, pp. 337-355, 1993.	I
153	Wohlin, C., Host, M., Ohlsson, M., Understanding the Sources of Software Defects: A Filtering Approach, in '8th International Workshop on Program Comprehension, 2000, IWPC 2000', pp. 9-17, 2000.	I
154	Y, I.; H, I.; K, O.; H, I. & R, K. (), 'A practical study on software quality management for sub-contracted products', Software Quality Concern for People.	ETA
155	Y, P.; D, P. & C, G. (), 'Automating the failure modes and effects analysis of safety critical systems', Proceedings .	ETA
156	Yu, W. D., Software fault prevention approach in coding and root cause analysis, Bell Labs Technical Journal 3(2), 3 – 21, 1998.	EC
157	Zeller, A. (2001), 'Automated debugging: Are we close?', Computer 34 (11), 26 - 31.	ETA

158	Zhang, L.; Zhang, Y. & Yang, X. (2005), 'Design of cylindricity error virtual measuring instrument and application of openGL', Dongbei Daxue Xuebao/Journal of Northeastern University 26( 5), 478 - 480.	ETA
159	Crawford, D. (1998), 'Readers “thinking objectively”', Commun. ACM 41(4), 22--26.	ETA

## A2 – Artigos Adicionais Obtidos em Setembro de 2007 das Máquinas de Busca

Segue a listagem dos 15 artigos adicionais retornados pela reaplicação do protocolo nas máquinas de busca em Setembro de 2007. A coluna de status indica o resultado do processo de filtragem. Os valores possíveis para o status são: ETA (eliminado por título e abstract), EC (eliminado por conteúdo) e I (incluído). Os artigos incluídos encontram-se destacados em cinza.

**Tabela A.2. Artigos Adicionais Obtidos em Setembro de 2007 das Máquinas de Busca.**

#	Artigo	Status
1	Bauer, A.; Leucker, M. & Schallhart, C. (2006), Model-based runtime analysis of distributed reactive systems, in , pp. 243 - 252.	ETA
2	Chang, C., Chu, C., Defect Prevention in Software Processes: An Action-Based Approach, The Journal of Systems and Software, Vol. 80, issue 4, April 2007, pp. 559-570, 2007.	I
3	Damm, L., Lundberg, L., Company-wide Implementation of Metrics for Early Software Fault Detection, Proceedings of the 29 <sup>th</sup> International Conference on Software Engineering (ICSE'07), Minneapolis, 2007.	I
4	Davey, C. & Friedman, J. (2007), Software Systems Engineering with Model-Based Design, in 'SEAS '07: Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems', IEEE Computer Society, Washington, DC, USA, pp. 7.	ETA
5	Gokhale, S.S., Crigler, J.R., Farr, W.H., Wallace, D.R., System availability analysis considering hardware/software failure severities, Proceedings 29th Annual IEEE / NASA Software Engineering Workshop (SEW-29 2005), Maryland, USA, 2005.	ETA
6	Halverson, C. A.; Ellis, J. B.; Danis, C. & Kellogg, W. A. (2006), Designing task visualizations to support the coordination of work in software development, in 'CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work', ACM Press, New York, NY, USA, pp. 39--48.	ETA
7	Hayes, J.H., Raphael, I., Holbrook, E.A., Pruett, D.M., A case history of International Space Station requirement faults, Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems, Stanford University, California, 2006.	I
8	Jetley, R., Zhang, Y., Iyer, S. P., Using abstraction-driven slicing for postmortem analysis of software, in 14th IEEE International Conference on Program Comprehension, pp. 107 – 116, 2006.	EC
9	Krena, B.; Letko, Z.; Tzoref, R.; Ur, S. & Vojnar, T. (2007), Healing data races on-the-fly, in 'PADTAD '07: Proceedings of the 2007 ACM workshop on Parallel and distributed systems: testing and debugging', ACM Press, New York, NY, USA, pp. 54--64.	ETA

10	Li, Z., Tan, L., Wang, X., Lu, S., Zhou, Y., Zhai, C., Have things changed now? An empirical study of bug characteristics in modern open source software, Proceedings of ASID'06: 1st Workshop on Architectural and System Support for Improving Software Dependability, pp. 25-33, 2006.	I
11	Mahanti, P. & Banerjee, S. (2006), Automated testing in software engineering: Using ant colony and self-regulated swarms, in , pp. 443 - 448.	ETA
12	Nakamura, T., Hochstein, L., Basili, V.R., Identifying Domain-Specific Defect Classes Using Inspections and Change History, Proceedings of the 2006 International Symposium on Empirical Software Engineering (ISESE), Sept. 21-22, Rio de Janeiro, Brazil. p. 346-355, 2006.	I
13	Tzoref, R.; Ur, S. & Yom-Tov, E. (2007), Instrumenting where it hurts: an automatic concurrent debugging technique, in 'ISSTA '07: Proceedings of the 2007 international symposium on Software testing and analysis', ACM Press, New York, NY, USA, pp. 27--38.	ETA
14	Vilbergsdóttir, S. G.; Hvannberg, E. T. & Law, E. L. (2006), Classification of usability problems (CUP) scheme: augmentation and exploitation, in 'NordiCHI '06: Proceedings of the 4th Nordic conference on Human-computer interaction', ACM Press, New York, NY, USA, pp. 281--290.	ETA
15	Walia, G., Carver, J., Philip, T., Requirements Error Abstraction and Classification: An Empirical Study, Proceedings of the 2006 International Symposium on Empirical Software Engineering (ISESE), Sept. 21-22, Rio de Janeiro, Brazil. p. 336-345, 2006.	I

### A3 – Artigos Adicionais Obtidos em Janeiro de 2009 das Máquinas de Busca

Segue a listagem dos 24 artigos adicionais retornados pela reaplicação do protocolo nas máquinas de busca em Janeiro de 2009. A coluna de status indica o resultado do processo de filtragem. Os valores possíveis para o status são: ETA (eliminado por título e abstract), EC (eliminado por conteúdo) e I (incluído). Os artigos incluídos encontram-se destacados em cinza.

**Tabela A.3. Artigos Adicionais Obtidos em Janeiro de 2009 das Máquinas de Busca.**

#	Artigo	Status
1	Bressoud, T. C. (2008), 'Session scribe notes for Twenty-First ACM Symposium on Operating Systems Principles', SIGOPS Oper. Syst. Rev. 42(3), 136--151.	ETA
2	Chang, C., Chu, C., Yeh, Y., Integrating in-process software defect prediction with association mining to discover defect pattern, Journal on Information and Software Technology 51( 2), 375 – 384, 2009.	I
3	Chen, D.; Johansson, R.; Lonn, H.; Papadopoulos, Y.; Sandberg, A.; Torner, F. & Torngren, M. (2008),Modelling support for design of safety-critical automotive embedded systems, in , pp. 72 - 85.	ETA
4	Colvin, R., Grunske, L., Winter, K., Timed Behavior Trees for Failure Mode and Effects Analysis of time-critical systems, Journal of Systems and Software 81( 12), pp. 2163-2182, 2008.	EC
5	Cotroneo, D.; Pietrantuono, R.; Mariani, L. & Pastore, F. (2007), Investigation of failure causes in workload-driven reliability testing, in , pp. 78 - 85.	ETA
6	Denger, C.; Trapp, M. & Liggesmeyer, P. (2008),SafeSpection - A systematic customization approach for software hazard identification, in , pp. 44 - 57.	ETA
7	Ganesh, V., Challenges in adopting traditional processmodels in a package implementation scenario, in 'ISEC '08: Proceedings of the 1st conference on India software engineering conference', ACM, New York, NY, USA, pp. 139-140, 2008.	EC
8	Goncalves, F., Bezerra, C., Belchior, A., Coelho, C., Pires, C., Implementing Causal Analysis and Resolution in Software Development Projects: The MiniDMAIC Approach, 19th Australian Conference on Software Engineering, 112-119, 2008.	I
9	Jagadeesh Chandra Bose, R. & Suresh, U. (2008), 'Root Cause Analysis Using Sequence Alignment and Latent Semantic Indexing', Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on, 367-376.	ETA

10	Kalinowski, M., Travassos, G.H., Card, D.N., "Towards a Defect Prevention Based Process Improvement Approach", 34th EuroMicro Conference on Software Engineering and Advanced Applications (SEAA), Parma, Itália, 2008.	I
11	Kojima, T., Hasegawa, T., Misumi, M., Nakamura, T., Risk analysis of software process measurements, SOFTWARE QUALITY JOURNAL 16(3), 361-376, 2008.	EC
12	Li, Y.; Xu, S.; Han, C.; Yu, T. & Xing, Z. (2008), 'Application of quantitative fault tree analysis to software development for space camera', Guangxue Jingmi Gongcheng/Optics and Precision Engineering, 16(11), 2180 - 2186.	ETA
13	McCaffery, F.; Pikkariainen, M. & Richardson, I. (2008), Ahaa --agile, hybrid assessment method for automotive, safety critical smes, in 'ICSE '08: Proceedings of the 30th international conference on Software engineering', ACM, New York, NY, USA, pp. 551--560.	ETA
14	Neumann, P. G. (2007), 'Risks to the public', SIGSOFT' Softw. Eng. Notes 32(6), 20--27.	ETA
15	Nikandros, G. & Tombs, D. (2007), Measuring railway signals passed at danger, in 'SCS '07: Proceedings of the twelfth Australian workshop on Safety critical systems and software and safety-related programmable systems', Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 41--46.	ETA
16	Ozarin, N., Lessons Learned on Five Large-Scale System Developments, Instrumentation & Measurement Magazine, IEEE 11(1), 18-23, 2008.	EC
17	Ozarin, N., Lessons Learned on Five Large-Scale System Developments, 2007 1st Annual IEEE Systems Conference, 1-7, 2007.	EC
18	Pang, K. & Ali, S. (2007), Retrospective analysis for mining the causes in manufacturing processes, in , pp. 4052657 - .	ETA
19	Ploski, J., Rohr, M., Schwenkenberg, P., Hasselbring, W., Research issues in software fault categorization, SIGSOFT' Softw. Eng. Notes 32(6), 2007.	I
20	Qureshi, Z. H. (2007), A review of accident modelling approaches for complex socio-technical systems, in 'SCS '07: Proceedings of the twelfth Australian workshop on Safety critical systems and software and safety-related programmable systems', Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 47--59.	ETA
21	Robinson, B., Francis, P., Ekdahl, F., A defect-driven process for software quality improvement, in 'ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement', pp. 333-335, Kaiserslautern, Germany, 2008.	I
22	Suetsugu, R.; Yuen, S. & Agusa, K. (2007), A synchronization flow analysis of concurrent objects in AIBO OPEN-R programs based on communicating processes, in , pp. 366 - 373.	ETA
23	Van De Weerd, I.; Brinkkemper, S. & Versendaal, J. (2007), Concepts for incremental method evolution: Empirical exploration and validation in requirements management, in , pp. 469 - 484.	ETA
24	Zhu, Q., Goal trees and fault trees for root cause analysis, International Conference on Software Maintenance, pp. 436 - 439, 2008.	EC



## A4 – Artigos Adicionais Obtidos em Julho de 2010 das Máquinas de Busca

Segue a listagem dos 26 artigos adicionais retornados pela reaplicação do protocolo nas máquinas de busca em Julho de 2010. A coluna de status indica o resultado do processo de filtragem. Os valores possíveis para o status são: ETA (eliminado por título e abstract), EC (eliminado por conteúdo) e I (incluído). Os artigos incluídos encontram-se destacados em cinza.

**Tabela A.4. Artigos Adicionais Obtidos em Julho de 2010 das Máquinas de Busca.**

#	Artigo	Status
1	Alshathry, O.; Janicke, H.; Zedan, H. & Alhussein, A (2009). Quantitative Quality Assurance Approach. Proceedings International Conference on New Trends in Information and Service Science, NISS 2009, 405-408.	EC
2	Bernardo, C.; Fernandes, D.; Dias, L.; Montini, D.; da Silva, D. & da Cunha, A (2009). Using GQM for Testing Design Patterns in Real-Time and Embedded Systems on a Software Production Line. Information Technology: New Generations, ITNG '09. Sixth International Conference on, 1397-1404.	ETA
3	Chandra, S. & Khan, R. A (2009). Software security metric identification framework (SSM), ICAC3 '09: Proceedings of the International Conference on Advances in Computing, Communication and Control ACM, 2009, 725-731.	ETA
4	Chang, C., Chu, C., Software defect prediction using intertransaction association rule mining, International Journal of Software Engineering and Knowledge Engineering, 19, 747-764, 2009.	I
5	Chen, Y. & Zhang, H (2009). Research and Design of High-Voltage Electronic Power Equipment Monitor System Based on Wireless Communication Technology, Power and Energy Engineering Conference, 2009. APPEEC 2009. Asia-Pacific, 1-4.	ETA
6	Cusick, J. J. & Ma, G (2010). Creating an ITIL inspired Incident Management approach: Roots, response, and results. Network Operations and Management Symposium Workshops (NOMS Wksp), 2010 IEEE/IFIP, 142 -148.	ETA
7	Fernandes, D.; Cardoso, F.; Montini, D.; Dias, L.; de Souza Pereira Moreira, G. & da Cunha, A (2009). Final Inspection for Design Pattern Homologation Using a Real Time Embedded Software in a Production Line. Information Technology: New Generations, ITNG '09. Sixth International Conference on, 1428 -1435.	ETA
8	Gomes, A.; Vivacqua, A. & Garcia, A (2009). Troubleshooting collaborative ontology design. Computer Supported Cooperative Work in Design, 2009. CSCWD 2009. 13th International Conference on, 149 -154.	ETA



9	Hamill, M., Goseva-Popstojanova, K. Common Trends in Software Fault and Failure Data, IEEE Transactions on Software Engineering, 35, 484 - 496, 2009.	I
10	Han, T., Li, B., Xu, L., A universal fault diagnostic expert system based on Bayesian Network, Proceedings Int. Conf. on Computer Science and Software Engineering, CSSE 2008, 260 – 263, 2008.	I
11	Hanchate, D., Sayyad, S., Shinde, S, Defect classification as problem classification for quality control in the software project management by DTL, 2nd International Conference on Computer Engineering and Technology (ICCET), V7-623 -V7-627, 2010.	I
12	Hong, Z. & Binbin, L (2009). Integraed Analysis of Software FMEA and FTA. Information Technology and Computer Science, ITCS 2009, International Conference on, 184 -187.	EC
13	IEEE, IEEE Standard Classification for Software Anomalies. <i>IEEE Std 1044-2009</i> , C1-15, 2010.	I
14	Ihara, A.; Ohira, M. & Matsumoto, K (2009). An analysis method for improving a bug modification process in open source software development. IWPSE-Evol '09: Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops, <i>ACM</i> , 135-144.	ETA
15	Jiang, L. & Su, Z (2008). Profile-guided program simplification for effective testing and analysis. Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 48-58.	ETA
16	Kalinowski, M., Travassos G.H., Mendes, E., Card, D.N., Applying DPPI: A Defect Causal Analysis Approach Using Bayesian Networks, 11th International Conference on Product Focused Software Development and Process Improvement (PROFES 2010), LNCS 6156, pp. 92–106, Limerick, Ireland, 2010.	I
17	Lai, Z.; Cheung, S. C. & Chan, W. K (2010). Detecting atomic-set serializability violations in multithreaded programs through active randomized testing. ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, 235-244.	ETA
18	Li, S.; Zhao, D. & Zhang, X (2009). Study on the uncertain problems in power grid fault diagnosis. Proceedings - International Conference on Computational Intelligence and Software Engineering, CiSE 2009.	ETA
19	Liu, X.; Dong, D. & Luo, Y (2009). Fault diagnosis of train-ground wireless communication unit based on fuzzy neural network. 4th IEEE Conference on Industrial Electronics and Applications, ICIEA 2009, 348-352.	ETA
20	Lu, T. & Przytula, K (2010). Discovery of root causes of system failures by means of analysis of repair records. IEEE Aerospace Conference, 1-8.	EC
21	Mariani, L.; Pastore, F. & Pezze, M (2009). A toolset for automated failure analysis. Proceedings – International Conference on Software Engineering (ICSE), 563-566.	EC
22	Rao, R., An Analysis of Missed Structure Field Handling Bugs, Sixth International Conference on Software Engineering Research, Management and Applications, SERA '08., 41-48, 2008.	I
23	Shenvi, A. A., Defect prevention with orthogonal defect classification, Proceedings of the 2nd India Software Engineering Conference, ISEC, 83 – 87, 2009.	I

24	Yang, Z.; Wang, C.; Gupta, A. & Ivanvčić, F (2009). Model checking sequential software programs via mixed symbolic analysis. ACM Trans. Des. Autom. Electron. Syst., 14, 1-26.	ETA
25	Yanyan, Z. & Renzuo, X (2008). A Human Factors fault tree analysis method for software engineering. Industrial Engineering and Engineering Management, IEEM 2008. IEEE International Conference on, 1971 - 1975.	EC
26	Zhao, F. & Qiao, S (2009). Radius selection algorithms for sphere decoding. ACM International Conference Proceeding Series, 169-174.	ETA

## A5 – Artigos Obtidos em Julho de 2010 da Máquina de Busca SCOPUS

Segue a listagem dos 262 artigos retornados pela aplicação do protocolo na máquina de busca SCOPUS, em Julho de 2010. Devido ao grande número de artigos, a tabela foi sutilmente reestruturada para simplificar a visualização das informações. Como muitos dos artigos já haviam sido analisados anteriormente, ao invés da coluna de status, apenas os artigos incluídos são destacados em cinza.

**Tabela A.5. Artigos Obtidos em Julho de 2010 da Máquina de Busca SCOPUS.**

Autores	Título	Journal/Proceedings	Ano
Abdel-Hamid, M. S.	On the portability of quantitative software estimation models	Information and Management	1987
Abreu, Z. P. V. G. A.	Spectrum-based multiple fault localization	ASE2009 - 24th IEEE/ACM Int. Conf. on Automated Software Engineering	2009
Abreu, Z. P. G. R. v. G. A.	A practical evaluation of spectrum-based fault localization	Journal of Systems and Software	2009
Acharya, K. V.	Mining health models for performance monitoring of services	ASE2009 - 24th IEEE/ACM Int. Conf. on Automated Software Engineering	2009
Alshathry, J. H. Z. H. A. A.	Quantitative Quality Assurance Approach	Int. Conf. on New Trends in Information and Service Science	2009
Angeli, C.	Online expert system for fault diagnosis in hydraulic systems	Expert Systems	1999
Aranda, V. G.	The secret life of bugs: Going past the errors and omissions in software repositories	Int. Conf. on Software Engineering	2009
Arekapudi, X. F. P. J. H. I.	ATPG for timing errors in globally asynchronous locally synchronous systems	Journal of Circuits, Systems and Computers	2003
Arnold, A. D. D. S. B. L. G. M. B. S. M.	Stack trace analysis for large scale debugging	21st Int. Parallel and Distributed Processing Symp., IPDPS	2007
Baik, C. S. H. C. G. J. P. C. Y. H.	$\Omega$ line problem in optimistic log-based rollback recovery protocol	IEICE Transactions on Information and Systems	2004
Balakrishnan, G. M.	PED: Proof-guided error diagnosis by triangulation of program error causes	6th IEEE Int. Conf. on Software Engineering and Formal Methods, SEFM 2008	2008
Ball, T. <i>et al.</i>	Thorough static analysis of device drivers	2006 EuroSys Conf.	2006
Bartoli, A.	Implementing a replicated service with group communication	Journal of Systems Architecture	2004
Bassin, B. S. S. P.	Metrics to evaluate vendor-developed software based on test case execution results	IBM Systems Journal	2002
Batanov, C. Z.	An object-oriented expert system for fault diagnosis in the ethylene distillation process	Computers in Industry	1995
Battikha, M.	Scheduling bridge and highway inspection/test activities with QUALITIME	of the 9th Int. Conf. on Applications of Advanced Technology in Transportation	2006
Bauer, L. M. S. C.	Model-based runtime analysis of distributed reactive systems	Australian Software Engineering Conf., ASWEC	2006
Berk, F. K. W. N. B. R. H. C.	Test station configuration and health management	AUTOTESTCON	2002
Bernardo, F. D. D. L. M. D. D. S. D. D. C. A.	Using GQM for testing design patterns in real-time and embedded systems on a software production line	ITNG 2009 - 6th Int. Conf. on Information Technology: New Generations	2009
Bertolino, S. L.	Using testability measures for dependability assessment	Int. Conf. on Software Engineering	1995
Beyer, H. T. M. R. R. A.	Path invariants	ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)	2007
Bhalla, S.	Independent dependency tracking in a mobile adhoc computing environment	Int. Conf. on Communication System Software and Middleware	2006
Bhandari, H. M. T. E. B. D. C. J. C. R.	Case study of software process improvement during development	IEEE Transactions on Software Engineering	1993
Boddu, X. L.	Incorporating modular imperfect coverage into dynamic hierarchical systems analysis	DASC 2007: 3rd IEEE Int. Symp. Dep., Autonomic and Secure Computing	2007
Boddu, G. L. M. S. C. B.	RETNA: From requirements to testing in a natural way	IEEE Int. Conf. on Requirements Engineering	2004
Bond, N. N. K. S.	Tracking bad apples: Reporting the origin of null	Conf. on Object-Oriented Prog. Systems,	2007

G. S. M. K.	and undefined value errors	Languages, and Applications (OOPSLA)	
de Borst, R.	Computation of post-bifurcation and post-failure behavior of strain-softening solids	Computers and Structures	1987
Bose, S. U.	Root cause analysis using sequence alignment and latent semantic indexing	Australian Software Engineering Conf., ASWEC	2008
Bratthall, J. M.	Can you trust a single data source exploratory software engineering case study?	Empirical Software Engineering	2002
Brown, T. I. P. R.	Early stage failure modeling and analysis applied to a wave energy converter	ASME Int. Computers and Information in Engineering Conf.	2009
Burrell, I. D. b.	An expert system for the analysis of faults in an electricity supply network: Problems and achievements	Computers in Industry	1998
Byers, S. N.	A cause-based approach to preventing software vulnerabilities	ARES 2008 - 3rd Int. Conf. on Availability, Security, and Reliability	2008
Byers, S. N.	Design of a process for software security	Second Int. Conf. on Availability, Reliability and Security, ARES 2007	2007
Calori, S. T. Z. S.	Robustness analysis using fmea and bbn: Case study for a web-based application	Webist 2007 - 3rd Int. Conf. on Web Information Systems and Technologies	2007
Card, D. N.	Learning from our mistakes with defect causal analysis	IEEE Software	1998
Carley, D. J. R. J. T. M.	Toward an interoperable dynamic network analysis toolkit	Decision Support Systems	2007
Carlisle, J.	Ethical considerations of the software-dependent organization	Journal of Systems and Software	1999
Chaar, H. M. J. B. I. S. C. R.	In-process evaluation for software inspection and test	IEEE Transactions on Software Engineering	1993
Chaki, G. A. S. O.	Explaining abstract counterexamples	ACM SIGSOFT Symp. on the Foundations of Software Engineering	2004
Challagulla, B. F. Y. I. P. R.	Empirical assessment of machine learning based software defect prediction techniques	Int. Journal on Artificial Intelligence Tools	2008
Chang, C. C.	Software defect prediction using intertransaction association rule mining	Int. Journal of Software Engineering and Knowledge Engineering	2009
Chang, C. C.	Improvement of causal analysis using multivariate statistical process control	Software Quality Journal	2008
Chang, C. C.	Defect prevention in software processes: An action-based approach	Journal of Systems and Software	2007
Chang, C. C. Y. Y.	Integrating in-process software defect prediction with association mining to discover defect pattern	Information and Software Technology	2009
Chang, M. L. P. M.	Self-healing strategies for component integration faults	ASE2008 the 23rd IEEE/ACM Int. Conf. Automated Software Engineering	2008
Chatzoglou, M. L.	A Rule-Based Approach to Developing Software Development Prediction Models	Automated Software Engineering	1998
Chaudhuri, G. S. L. R.	Continuity analysis of programs	ACM SIGPLAN Notices	2010
Chen, X. J. K. Z. I. R. W. K.	Modeling and evaluating the security threats of transient errors in firewall software	Performance Evaluation	2004
Chen, U. R. R. G. E. S. V. R. J.	Can fault-exposure-potential estimates improve the fault detection abilities of test suites?	Software Testing Verification and Reliability	2002
Chen, M. G.	System safety requirements as control structures	Int. Computer Software and Applications Conf.	2009
Chen, M. G. b.	Formalizing safety requirements using controlling automata	2nd Int. Conf. on Dependability, DEPEND 2009	2009
Chernak, Y. b.	A statistical approach to the inspection checklist formal synthesis and improvement	IEEE Transactions on Software Engineering	1996
Chiu, C.	Applying means-end chain theory to eliciting system requirements and understanding users perceptual orientations	Information and Management	2005
Chmiel, L. M.	Debugging: From novice to expert	SIGCSE Bulletin	2004
Cholewa, W.	Real-time diagnostic expert systems	Computer Assisted Mechanics and Engineering Sciences	2002
Cleve, Z. A.	Locating causes of program failures	Int. Conf. on Software Engineering	2005
Cohen, B. B. A. C. S. E. P. V. c.	A cognitive blueprint of collaboration in context: Distributed cognition in the psychiatric emergency department	Artificial Intelligence in Medicine	2006
Collofello, G. B.	Application of causal analysis to the software modification process	Software - Practice and Experience	1993
Colvin, G. L. W. K.	Timed Behavior Trees for Failure Mode and Effects Analysis of time-critical systems	Journal of Systems and Software	2008
Conti, A. K. G. J. S. J. C. J. A. M. O. H. L. C.	Countering security information overload through alert and packet visualization	IEEE Computer Graphics and Applications	2006
Cook, J.	Supporting rapid prototyping through frequent and reliable deployment of evolving components	Int. Workshop on Rapid System Prototyping	2001
Cotroneo, P. R. M. L. P. F.	Investigation of failure causes in workload-driven reliability testing	SOQUA'07: Fourth Int. Workshop on Software Quality Assurance	2007

Cowart, S. M. W. M. R. A. M. D. H. Y. L. X.	Revealing a signaling role of phytosphingosine-1-phosphate in yeast	Molecular Systems Biology	2010
Dalcher, D.	Forensic ECBS: The way forward	Int. Symp. and Workshop on Engineering of Computer Based Systems	2001
Damm, L. L.	Company-wide implementation of metrics for early software fault detection	Int. Conf. on Software Engineering	2007
Damm, L. L.	Results from introducing component-level test automation and Test-Driven Development	Journal of Systems and Software	2006
Damm, L. L. W. C.	Faults-slip-through - A concept for measuring the efficiency of the test process	Software Process Improvement and Practice	2006
Damm, L. L. W. C.	A model for software rework reduction through a combination of anomaly metrics	Journal of Systems and Software	2008
De Almeida Jr., C. J. J. B. B. P. S.	Best practices in code inspection for safety-critical software	IEEE Software	2003
De Lemos, R.	Analysing failure behaviours in component interaction	Journal of Systems and Software	2004
De Leon, A. J. b.	Hidden implementation dependencies in high assurance and critical computing systems	IEEE Transactions on Software Engineering	2006
Di Fatta, L. S. S. E.	Discriminative pattern mining in software fault detection	Third Int. Workshop on Software Quality Assurance, SOQUA 2006	2006
Eaton, M. A. b. c. d.	An empirical approach to evaluating web application compliance across diverse client platform configurations	Int. Journal of Web Engineering and Technology	2007
Efkemann, H. T.	Specification of Conditions for Error Diagnostics	Electronic Notes in Theoretical Computer Science	2008
Eick, L. C. R. L. M. V. L. G. V. W. S.	Estimating software fault content before coding	Int. Conf. on Software Engineering	1992
El Emam, M. N. e. f.	An instrument for measuring the success of the requirements engineering process in information systems development	Empirical Software Engineering	1996
Elbaum, M. J. C.	Software Black Box: An alternative mechanism for failure analysis	Int. Symp. on Software Reliability Engineering, ISSRE	2000
Fernandes, C. F. M. D. D. L. M. G. D. C. A.	Final inspection for design pattern homologation using a real time embedded software in a production line	ITNG 2009 - 6th Int. Conf. on Information Technology: New Generations	2009
Fetzer, X. Z.	HEALERS: A Toolkit for Enhancing the Robustness and Security of Existing Applications	Int. Conf. on Dependable Systems and Networks	2003
Fisteus, F. L. K. C.	Applying model checking to BPEL4WS business collaborations	ACM Symp. on Applied Computing	2005
Flanagan, F. S.	Atomizer: A dynamic atomicity checker for multithreaded programs	Conf. Record of the Annual ACM Symp. on Principles of Programming Languages	2004
Forbes, L. D. L. E.	Predicting reliability investment to achieve given reliability improvement	Annual Reliability and Maintainability Symp.	2009
Fraser, W. F.	Using LTL rewriting to improve the performance of model-checker based test-case generation	3rd Int. Workshop Advances in Model Based Testing, AMOST 2007	2007
Fu, R. B.	Exception-chain analysis: Revealing exception handling architecture in Java server applications	Int. Conf. on Software Engineering	2007
Gao, K. G. K. P.	Approaches to building self healing systems using dependency analysis	IEEE Symp. Record on Network Operations and Management Symp.	2004
Ghosh, K. J.	Bytecode fault injection for Java software	Journal of Systems and Software	2008
Gittens, K. Y. G. D.	The vital few versus the trivial many: Examining the Pareto principle for software	Int. Computer Software and Applications Conf.	2005
Gladyshev, P. A.	Finite state machine approach to digital event reconstruction	Digital Investigation	2004
Gonçalves, B. C. b. B. A. C. C. P. C.	Implementing causal analysis and resolution in software development projects: The MiniDMAIC approach	Australian Software Engineering Conf., ASWEC	2008
Grady, R.	Software failure analysis for high-return process improvement decisions	Hewlett-Packard Journal	1996
Green, D. I. A.	How to design, verify and validate emergency shutdown systems	ISA Transactions	1995
Gregoriades, S. A.	Scenario-based assessment of nonfunctional requirements	IEEE Transactions on Software Engineering	2005
Grimstad, J. M.	A framework for the analysis of software cost estimation accuracy	ISCE'06 - 5th ACM-IEEE Int. Symp. on Empirical Software Engineering	2006
Groce, K. D.	Making the most of BMC counterexamples	Electronic Notes in Theoretical Computer Science	2005
Grottke, L. L. V. K. T. K. b.	Analysis of software aging in a Web server	IEEE Transactions on Reliability	2006
Grunske, C. R. W. K.	Probabilistic model-checking support for FMEA	4th Int. Conf. on the Quantitative Evaluation of Systems, QEST 2007	2007
Gummadi, K. N. M. T. G. R.	Declarative failure recovery for sensor networks	ACM Int. Conf. Proceeding Series	2007
Gupta, L. J. C. R. R.	Change profiles of a reused class framework vs. two	Information and Software Technology	2010

H. L. E.	of its applications		
Gupta, L. J. C. R. R. H. L. E.	A case study comparing defect profiles of a reused framework and of applications reusing it	Empirical Software Engineering	2009
Hamill, G. K.	Common trends in software fault and failure data	IEEE Transactions on Software Engineering	2009
Han, L. B. X. L.	A universal fault diagnostic expert system based on Bayesian Network	Int. Conf. on Computer Science and Software Engineering, CSSE 2008	2008
Hansen, G. J.	Cheap and small counterexamples	6th IEEE Int. Conf. on Software Engineering and Formal Methods, SEFM 2008	2008
Hao, Z. L. P. Y. M. H. S. J.	On similarity-awareness in testing-based fault localization	Automated Software Engineering	2008
Hariri, T. C.	Bridging fault diagnostic tool based on $\Delta_{IDDQ}$ probabilistic signatures, circuit layout parasitics and logic errors	IET Computers and Digital Techniques	2007
Hayes, I. R. C. H. E. P. D.	A case history of Int. space station requirement faults	IEEE Int. Conf. on Engineering of Complex Computer Systems, ICECCS	2006
Hegde, D. T. R. P.	Implementation of hierarchical conformance testing platform for LAPB	IETE Journal of Research	2000
Hofbaur, W. F.	A causal analysis method for concurrent hybrid automata	National Conf. on Artificial Intelligence	2006
Howden, W.	Testability, failure rates, detectability, trustability and reliability	COMPASS - Annual Conf. on Computer Assurance	1994
Huang, L. C. e. C. N. c. f.	Fuzzy decision tree approach for embedding risk assessment information into software cost estimation model	Journal of Information Science and Engineering	2006
Hunt, J. b.	Model-based software diagnosis	Applied Artificial Intelligence	1998
Huotari, L. K. e. N. M. f.	Improving graphical information system model use with elision and connecting lines	ACM Transactions on Computer-Human Interaction	2004
Ihara, O. M. M. K.	An analysis method for improving a bug modification process in open source software development	Int. Workshop on Principles of Software Evolution (IWPSE)	2009
Ikram, R. S.	Requirements change management process models: An evaluation	IASTED Int. Conf. on Software Engineering, SE 2007	2007
Jacobs, V. M. J. b. K. P. K. R. T. J. B. A.	Exploring defect causes in products developed by virtual teams	Information and Software Technology	2005
Jalote, M. R. P. T.	The When-Who-How analysis of defects for improving the quality control process	Journal of Systems and Software	2007
Jalote, S. A.	Optimum control limits for employing statistical process control in software process	IEEE Transactions on Software Engineering	2002
Janssem, A. R. V. G. A.	Zoltar: A toolset for automatic fault localization	ASE2009 - 24th IEEE/ACM Int. Conf. on Automated Software Engineering	2009
Jia, C. J. J. X. L. C.	Design and analysis of an efficient and reliable atomic multicast protocol	Computer Communications	1998
Jiang, S. Z.	Profile-guided program simplification for effective testing and analysis	ACM SIGSOFT Symp. on the Foundations of Software Engineering	2008
Jiang, Z. K. Z. X. T. L. W. W.	Research on product quality design into supply chain	2009 Chinese Control and Decision Conf., CCDC 2009	2009
Johnson, T. A.	Extending the application of formal methods to analyse human error and system failure during accident investigations	Software Engineering Journal	1996
Jones, J.	Fault localization using visualization of test information	Int. Conf. on Software Engineering	2004
Jones, H. M.	Empirical evaluation of the tarantula automatic fault-localization technique	20th IEEE/ACM Int. Conf. on Automated Software Engineering, ASE 2005	2005
Jovalekic, S.	Graphical test planning for distributed discrete real-time systems	10th IASTED Int. Conf. on Software Engineering and Applications, SEA 2006	2006
Jung, H.	Evaluating the ordering of the SPICE capability levels: An empirical study	Information and Software Technology	2005
Jäntti, M. A. P. N. K. T.	Improving the problem management process from knowledge management perspective	Lecture Notes in Computer Science	2007
Kajko-Mattsson, M.	Problem management maturity within corrective maintenance	Journal of Software Maintenance and Evolution	2002
Kalinowski, M. T. G. C. D.	Towards a defect prevention based process improvement approach	34th EUROMICRO Conf. on Soft. Eng. and Advanced Applications, SEAA 2008	2008
Kalinowski, M. T. G. M. E. C. D.	Applying DPPI: A Defect Causal Analysis Approach Using Bayesian Networks	11th Int. Conf. on Product Focused Software Development and Process Improvement (PROFES 2010), LNCS 6156	2010
Keenan, H. H. d. e. f. K. D. S. R. g.	Usability problem taxonomy: A framework for classification and analysis	Empirical Software Engineering	1999
Khanna, C. M. V. P. B. S. C. M. V. P. c. c.	Automated rule-based diagnosis through a distributed monitor system	IEEE Transactions on Dependable and Secure Computing	2007
Killian, A. J. B. R. J. R. V. A.	Mace: Language support for building distributed systems	ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)	2007
Kim, A. Y. F. S. H.	Anomaly-based fault detection in Pervasive	5th Int. Conf. on Pervasive Services, ICPS	2008



S. Y. M.	Computing System		
Kim, K. B.	Probabilistic Schedulability Analysis of Harmonic Multi-Task Systems with Dual-Modular Temporal Redundancy	Real-Time Systems	2004
Kim, Z. T. W. J. E. Z. A.	Predicting faults from cached history	Int. Conf. on Software Engineering	2007
Kitchenham, D. T. d. J. M.	Evidence-based software engineering	Int. Conf. on Software Engineering	2004
Klās, E. F. V. L. R. S. T. G. J.	A framework for the balanced optimization of quality assurance strategies focusing on small and medium sized enterprises	Conf. EUROMICRO	2009
Ko, A.	Debugging by asking questions about program output	Int. Conf. on Software Engineering	2006
Ko, M. B.	Designing the Whyline: A debugging interface for asking questions about program behavior	Conf. on Human Factors in Computing Systems	2004
Kodavade, A. S.	Object oriented concurrent fault diagnostic system for 8051 based microcontroller system	4th IEEE Workshop on Intelligent Data Acq. and Adv. Comp. Systems, IDAACS	2007
Kojima, H. T. M. M. N. T.	Risk analysis of software process measurements	Software Quality Journal	2008
Koochakzadeh, G. V. M. F.	Test redundancy measurement based on coverage information: Evaluations and lessons learned	2nd Int. Conf. on Software Testing, Verification, and Validation, ICST 2009	2009
Kumar, U. S.	Component-ontological representation of function for reasoning about devices	Artificial Intelligence in Engineering	1998
Kumar, K. V. V. M.	On the complexity of error explanation	Lecture Notes in Computer Science	2005
Kuo, C. T. L. H. C. W.	Enhancing adaptive random testing in high dimensional input domains	ACM Symp. on Applied Computing	2007
Laitenberger, B. T. S. T.	An industrial case study to examine a non-traditional inspection implementation for requirements specifications	Empirical Software Engineering	2002
Lau, Y. Y. d.	An extended fault class hierarchy for specification-based testing	ACM Transactions on Software Engineering and Methodology	2005
Lederer, P. J.	A causal model for software cost estimating error	IEEE Transactions on Software Engineering	1998
Lee, L. W. C. H. K. Y.	Specification and analysis of timing requirements for real-time systems in the CBD approach	Real-Time Systems	2007
Lee, C. K. K. J. V. G.	A methodology for variable structure system specification: Formalism, framework, and its application to ATM-based network system	ETRI Journal	1996
Leino, M. T. S. J.	Generating error traces from verification-condition counterexamples	Science of Computer Programming	2005
Leitch, S. A.	Task dependent tools for intelligent automation	Artificial Intelligence in Engineering	1989
Lerner, F. M. G. D. C. C.	Searching for type-error messages	ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)	2007
Leszak, M.	Software defect analysis of a multi-release telecommunications system	Lecture Notes in Computer Science	2005
Leszak, M. M.	Successful global development of a large-scale embedded telecommunications product	Int. Conf. on Global Software Engineering, ICGSE 2007	2007
Leszak, P. D. S. D.	Classification and evaluation of defects in a project retrospective	Journal of Systems and Software	2002
Leszak, P. D. E. S. D.	Case study in root cause defect analysis	Int. Conf. on Software Engineering	2000
Lí, Z. D. Z. X.	Study on the uncertain problems in power grid fault diagnosis	2009 Int. Conf. on Computational Intelligence and Software Engineering, CiSE 2009	2009
Lí, T. L. W. X. L. S. Z. Y. Z. C.	Have things changed now?: An empirical study of bug characteristics in modern open source software	ASID'06: 1st Workshop on Architectural and System Support for Improving Software Dependability	2006
Lí, G. M. b. M. S. M. N. M. A. G. D. C. E.	Analysis of pervasive multiple-component defects in a large software system	IEEE Int. Conf. on Software Maintenance, ICSM	2009
Lin, H. M.	A GDSS for ranking a firm's core capability strategies	Journal of Computer Information Systems	2007
Lin, H. Y. d. B. J.	Realising B2B e-commerce benefits: The link with IT maturity, evaluation practices, and B2BEC adoption readiness	European Journal of Information Systems	2007
Lin, P. G. T. H. L. K.	An investigation into business-to-business electronic commerce organizations	Journal of Research and Practice in Information Technology	2008
Liu, H. J.	Failure proximity: A fault localization-based approach	ACM SIGSOFT Symp. on the Foundations of Software Engineering	2006
Liu, Z. X. H. J.	A systematic study of failure proximity	IEEE Transactions on Software Engineering	2008
Liu, D. D. L. Y.	Fault diagnosis of train-ground wireless communication unit based on fuzzy neural network	2009 4th IEEE Conf. on Industrial Electronics and Applications, ICIEA 2009	2009
Lucero, D.	Influencing software competencies across the DoD acquisition workforce	CrossTalk	2010
Lutz, M. I.	Empirical analysis of safety-critical anomalies during operations	IEEE Transactions on Software Engineering	2004

Lutz, M. I.	Requirements discovery during the testing of safety-critical software	Int. Conf. on Software Engineering	2003
Ma, T. J.	Web error classification and analysis for reliability improvement	Journal of Systems and Software	2007
Manevich, S. M. A. S. D. M. Y. Z.	PSE: Explaining program failures via postmortem static analysis	ACM SIGSOFT Symp. on the Foundations of Software Engineering	2004
Mariani, P. F. P. M. b. b.	A toolset for automated failure analysis	Int. Conf. on Software Engineering	2009
Matsumura, M. S. M. A. M. K.	Analyzing factors of defect correction effort in a multi-vendor information system development	Journal of Computer Information Systems	2008
Mays, R. G.	Applications of defect prevention in software development	IEEE Journal on Selected Areas in Communications	1990
Mazmanishvili, R. O.	Construction example of regional phase maps of error probability for chosen satellite communication network configuration	Engineering Simulation	1998
Menkhaus, F. U. W. J.	Analysis and verification of the interaction model in software design	IEEE Int. Conf. on Engineering of Complex Computer Systems, ICECCS	2005
Messer, A. a <i>et al.</i>	Susceptibility of modern systems and software to soft errors	HP Laboratories Technical Report	2001
Millstein, T.	Practical predicate dispatch	ACM SIGPLAN Notices	2004
Min, D. X.	Alert verification based on attack classification in collaborative intrusion detection	SNPD: 8th ACIS Int. Conf. on Soft. Eng., Art. Int., Net., and Par./Dist. Comp.	2007
Misherghi, S. Z.	HDD: Hierarchical Delta Debugging	Int. Conf. on Software Engineering	2006
Morgenshtern, R. T. D. D.	Factors affecting duration and effort estimation errors in software development projects	Information and Software Technology	2007
Moses, J.	Bayesian probability distributions for assessing measurement of subjective software attributes	Information and Software Technology	2000
Munson, N. A. S. J. d.	Software faults: A quantifiable definition	Advances in Engineering Software	2006
Murrell, P. R.	A survey of tools for the validation and verification of knowledge-based systems: 1985-1995	Decision Support Systems	1997
Nakashima, O. M. H. H. I. N.	Analysis of software bug causes and its prevention	Information and Software Technology	1999
Neukom, H.	Ubisco and CDC: Analysis of a failure	IEEE Annals of the History of Computing	2009
Nieves, S. A. c.	Human and organizational error as a basis for process reengineering: With applications to systems integration planning and marketing	IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans.	1998
Nurmuliani, Z. D. F. S.	Analysis of requirements volatility during software development life cycle	Australian Software Engineering Conf., ASWEC	2004
Ohsugi, M. A. K. N. B. M. T. M. K. T. M. K.	Is this cost estimate reliable? - the relationship between homogeneity of analogues and estimation reliability	1st Int. Symp. on Empirical Software Engineering and Measurement, ESEM 2007	2007
Ojiako, J. E. G. D.	A qualitative re-construction of project measurement criteria	Industrial Management and Data Systems	2008
Oral, B. A.	Defect prediction for embedded software	22nd Int. Symp. on Computer and Information Sciences, ISCIS	2007
Orozco, V. S. K. A.	Multiple description coding for voice over IP using sinusoidal speech coding	ICASSP, IEEE Int. Conf. on Acoustics, Speech and Signal Processing -	2006
Ortmeier, S. G.	Formal Fault Tree Analysis - Practical Experiences	Electronic Notes in Theoretical Computer Science	2007
Pan, K. S. W. J. E.	Toward an understanding of bug fix patterns	Empirical Software Engineering	2009
Pang, A. S.	Retrospective analysis for mining the causes in manufacturing processes	CIMCA 2006: Comp. Intelligence for Modelling, Control and Automation.	2007
Pappu, W. A.	Successful Failures: Who is to Blame?	Towards a Vision for Information Technology in Civil Engineering	2003
Paradkar, W. I. T. H. V. R.	Automatic test-generation for predicates	IEEE Transactions on Reliability	1996
Pattabiraman, N. N. K. Z. I. R.	SymPLIFIED: Symbolic program-level fault injection and error detection framework	Int. Conf. on Dependable Systems and Networks	2008
Paul, B. F. Y. I. C. V. U.	Defect-based reliability analysis for mission-critical software	IEEE Computer Society's Int. Computer Software and Applications Conf.	2000
Phomasakha Na Sakolnakorn, M. P.	Decision tree-based model for automatic assignment of IT service desk outsourcing in banking business	9th ACIS Int. Conf. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2008	2008
Podgurski, L. D. F. P. M. W. M. S. J. W. B.	Automated support for classifying software failure reports	Int. Conf. on Software Engineering	2003
Popov, S. L. M. J. K. S.	Estimating bounds on the reliability of diverse systems	IEEE Transactions on Software Engineering	2003
Prabhakaran, A. A. A. R.	Model-based failure analysis of journaling file systems	Int. Conf. on Dependable Systems and Networks	2005
Prakash, S. M. c. c.	Low-cost checkpointing and failure recovery in mobile computing systems	IEEE Transactions on Parallel and Distributed Systems	1996
Prakash, B. R.	Causality and the spatial-temporal ordering in	Mobile Networks and Applications	2004



	mobile systems		
Pu, Z. Y.	Localizing program errors via slicing and reasoning	IEEE Int. Symp. on High Assurance Systems Engineering	2008
Pérez-Miñana, G. J.	Improving fault prediction using Bayesian networks for the development of embedded software applications	Software Testing Verification and Reliability	2006
Ramzan, I. N.	Requirement change management process models: Activities, artifacts and roles	10th IEEE Int. Multitopic Conf. 2006, INMIC	2006
Rantanen, P. B. W. D. M. K.	Five-dimensional taxonomy to relate human errors and technological interventions in a human factors literature database	Journal of the American Society for Information Science and Technology	2006
Rao D., R.	An analysis of missed structure field handling bugs	6th ACIS Int. Conf. on Soft. Eng. Research, Management and Applications, SERA 2008	2008
Rinard, C. C. D. D. R. D. L. T.	A dynamic technique for eliminating buffer overflow vulnerabilities (and other memory errors)	Annual Computer Security Applications Conf., ACSAC	2004
Robey, S. L. V. L.	Perceptions of conflict and success in information systems development projects	Journal of Management Information Systems	1993
Robinson, F. P. E. F.	A defect-driven process for software quality improvement	ESEM'08: 2008 ACM-IEEE Int. Symp. on Empirical Software Engineering and Measurement	2008
Rukšėnas, C. P. B. A.	Modelling and analysing cognitive causes of security breaches	Innovations in Systems and Software Engineering	2008
Rungta, M. E.	An improved distance heuristic function for directed software model checking	Formal Methods in Computer Aided Design, FMCAD 2006	2006
Sarala, V. S.	Algorithms for defect detection in object oriented programs	Information Technology Journal	2006
Sarangi, G. B. T. J.	CADRE: Cycle-accurate deterministic replay for hardware debugging	Int. Conf. on Dependable Systems and Networks	2006
Sato, T. H. H. Y. T. Y.	Failure analysis of open faults by using detecting/un-detecting information on tests	Asian Test Symp.	2004
Segal, J. A. L. D. C. B.	Using electrical bitmap results from embedded memory to enhance yield	IEEE Design and Test of Computers	2001
Sharma, K. D. K. P.	Predicting uncertain behavior of industrial system using FM-A practical case	Applied Soft Computing Journal	2008
Sharma, K. D. K. P.	FM - A pragmatic tool to model, analyse and predict complex behaviour of industrial systems	Engineering Computations (Swansea, Wales)	2007
Shaw, A. D. R. C.	Fault security analysis of CMOS VLSI circuits using defect-injectable VHDL models	Integration, the VLSI Journal	2002
Shen, S. C. L. C. L. X.	Reference-driven performance anomaly identification	SIGMETRICS/Performance'09 - 11th Int. Joint Conf. on Measurement and Modeling of Computer Systems	2009
Shenvi, A.	Defect prevention with orthogonal defect classification	2nd India Software Engineering Conf., ISEC 2009	2009
Shukla, S. P. C. D.	A framework for statistical testing of software components	Int. Journal of Software Engineering and Knowledge Engineering	2007
Shunshan, J. P. E. L.	Root cause analysis and proactive problem prediction for self-healing	2007 Int. Conf. on Convergence Information Technology, ICCIT 2007	2007
Sing, S. B.	Improving TCP performance: Identifying corruption based packet loss	ICON 2007 - 2007 15th IEEE Int. Conf. on Networks	2007
Singh, M. P. R. T. D. P.	Using queries for distributed monitoring and forensics	2006 EuroSys Conf.	2006
Siponen, M.	Critical analysis of different approaches to minimizing user-related faults in information systems security: Implications for research and practice	Information Management and Computer Security	2000
Smith, J.	Certification of on-line learning neural networks	IASTED Int. Conf. on Artificial Intelligence and Soft Computing	2003
Soden, H. C. L.	IC diagnosis: Industry issues	IEEE Int. Test Conf. (TC)	1997
Sorin, H. M. W. D.	Dynamic Verification of End-to-End Multiprocessor Invariants	Int. Conf. on Dependable Systems and Networks	2003
Springett, M.	Evaluating cause and effect in user experience	Digital Creativity	2009
Sterritt, R.	Autonomic networks: Engineering the self-healing property	Engineering Applications of Artificial Intelligence	2004
Su, A. M. F. J.	AutoBash: Improving configuration management with operating system causality analysis	Operating Systems Review (ACM)	2007
Subramanian, E. L. V. R. T. W. M. R.	Fault mitigation in safety-critical software systems	IEEE Symp. on Computer-Based Medical Systems	1996
Sutcliffe, G. A.	Automating scenario analysis of human and system reliability	IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans	2007
Sycara, M. K.	Learning from failure experiences in case-based schedule repair	Hawaii Int. Conf. on System Sciences	1994
Taylor, B. P. K. M. Z. D. L. K. T. V.	Towards declarative monitoring of declarative service compositions	Int. Conf. on Data Engineering	2007
Thomas, D. A. B. V.	An analysis of errors in a reuse-oriented	Journal of Systems and Software	1997

c.	development environment		
Tian, R. S. L. Z.	Evaluating web software reliability based on workload and failure data extracted from server logs	IEEE Transactions on Software Engineering	2004
Tourwé, B. J. K. A. G. K.	Induced intentional software views	Computer Languages, Systems and Structures	2004
Tu, A. M. S. Y. P. K. K. R. A. S.	Integrated reliability analysis, diagnostics and prognostics for critical power systems	AUTOTESTCON	2001
Tucek, L. S. H. C. X. S. Z. Y.	Triage: Diagnosing production run failures at the user's site	Operating Systems Review (ACM)	2007
Veiga, P. P. F. P.	Complete distributed garbage collection using DGC-consistent cuts and.NET AOP-support	IET Software	2007
Vilbergssdóttir, H. E. L. E.	Classification of Usability Problems (CUP) scheme: Augmentation and exploitation	ACM Int. Conf. Proceeding Series	2006
Vinter, L. M. K. D.	Applying software metrics to formal specifications: A cognitive approach	Int. Software Metrics Symp.,	1998
Visconti, C. C.	Software system documentation process maturity model	1993 ACM Computer Science Conf.	1993
Voas, J.	Certifying software for high-assurance environments	IEEE Software	1999
Walia, C. J. P. T.	Requirement error abstraction and classification: An empirical study	ISCE'06 - 5th ACM-IEEE Int. Symp. on Empirical Software Engineering	2006
Wang, C. A.	A novel hash-based approach for mining frequent itemsets over data streams requiring less memory space	Data Mining and Knowledge Discovery	2009
Wang, V. C. D. J. C. Y. W. H. Y. C. Z. Z.	Strider: A black-box, state-based approach to change and configuration management and support	Science of Computer Programming	2004
Weimer, N. G.	Finding and preventing run-time error handling mistakes	ACM SIGPLAN Notices	2004
Wong, H. J. S. M. Z. W. Z. D.	Applying design metrics to predict fault-proneness: A case study on a large-scale software system	Software - Practice and Experience	2000
Yadla, H. J. D. A.	Tracing requirements to defect reports: An application of information retrieval techniques	Innovations in Systems and Software Engineering	2005
Yang, L. W. S. C.	A cross-layer approach to packet size adaptation for improved utilization in mobile wireless networks	Int. Journal of Innovative Computing, Information and Control	2009
Yang, W. C. G. A. I. F.	Model checking sequential software programs via mixed symbolic analysis	ACM Transactions on Design Automation of Electronic Systems	2009
Yih, T. J. d.	Developing and checking prescriptive specifications for safety improvement	Microprocessors and Microsystems	1998
Yuan, M. H. X. W. T. L. Z. Y. P. S.	SherLog: Error diagnosis by connecting clues from run-time logs	ACM SIGPLAN Notices	2010
Zhang, Z. H. G. S. H. Q.	Quality modelling for web-based information systems	IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems	2001
Zhao, Q. S.	Radius selection algorithms for sphere decoding	ACM Int. Conf. Proceeding Series	2009
Zhu, Q.	Goal trees and fault trees for root cause analysis	IEEE Int. Conf. on Software Maintenance, ICSM	2008

## ANEXO B – INFORMAÇÕES EXTRAÍDAS DOS ARTIGOS RECUPERADOS

*Este anexo apresenta as informações extraídas dos artigos selecionados como parte da revisão sistemática a respeito de análise causal de defeitos de software.*

### B1 – Informações Extraídas dos Artigos Recuperados na Revisão de Agosto de 2006

<b>Título do Artigo</b>
An Analysis of Errors and Their Causes in Systems Programs
<b>Referência Completa</b>
Endres, A., "An Analysis of Errors and Their Causes in Systems Programs", IEEE Transactions on Software Engineering, SE-1, 2, June 1975, pp. 140-149, 1975.
<b>Fonte</b>
Não retornado (Está na base da IEEE e atende à string de busca do protocolo).
<b>Esquema de Classificação de Defeitos</b>
<p>O esquema envolve o tipo de defeito, dados de identificação, descrição do defeito e dados da correção. O artigo menciona ainda que os seguintes dados adicionais deveriam ser determinados sobre os defeitos através de sua análise: localização, momento de inserção, responsável, tipo, causa, como prevenir e como detectar.</p> <p>Foram criados três grupos (A, B e C) de tipos de defeitos para defeitos de codificação:</p> <ul style="list-style-type: none"><li>• Os defeitos do grupo A são referentes ao mal entendimento do problema ou à escolha inadequada de um algoritmo para resolvê-lo.</li><li>• Os defeitos do grupo B são referentes a implementações incorretas ou incompletas.</li><li>• Os defeitos do grupo C são referentes a descuidos e que não deveriam estar presentes (como não aderência a padrões, mensagens escritas erradamente, entre outros).</li></ul> <p>Cada um dos grupos possui então um refinamento dos tipos de defeitos (em menor granularidade) que podem ocorrer nos mesmos.</p>
<b>Esquema de Classificação de Causas de Defeitos</b>
As causas (tecnológicas ou organizacionais) são organizadas nos mesmos três

<p>grupos (A, B e C) que os defeitos:</p> <ul style="list-style-type: none"> <li>• Causas do grupo A são referentes a defeitos de mal entendimento do problema ou à escolha inadequada de um algoritmo para resolvê-lo.</li> <li>• Causas do grupo B são referentes a defeitos de implementações incorretas ou incompletas.</li> <li>• Causas do grupo C são referentes a defeitos de descuido e que não deveriam estar presentes (como não aderência a padrões, mensagens escritas erradamente, entre outros).</li> </ul> <p>Cada grupo possui então um refinamento dos tipos de causas (em menor granularidade).</p>
<b>Identificação do Processo ou Abordagem</b>
Análise de Defeitos e suas Causas.
<b>Descrição do Processo ou Abordagem</b>
<p>Na experiência cada defeito de codificação foi analisado de modo que as seguintes perguntas pudessem ser respondidas sobre o mesmo:</p> <ul style="list-style-type: none"> <li>• Onde (módulo) o defeito foi inserido?</li> <li>• Quando (fase de desenvolvimento) o defeito foi inserido?</li> <li>• Quem inseriu o defeito?</li> <li>• O que foi feito de maneira errada?</li> <li>• Porque o foi feito de maneira errada?</li> <li>• O que poderia ter sido feito para prevenir esse defeito?</li> <li>• Por qual procedimento o defeito poderia ter sido detectado?</li> </ul>
<b>Identificação de Conhecimento</b>
<p>Seguem algumas observações referentes à experiência da IBM relatada no artigo:</p> <ol style="list-style-type: none"> <li>1. 78% dos defeitos estavam concentrados em 21% dos módulos.</li> <li>2. 85% dos defeitos puderam ser corrigidos modificando um único módulo.</li> <li>3. Aproximadamente metade de todos os erros eram referentes ao mal entendimento do problema e entre 60-70% dos defeitos são relacionados a requisitos ou projeto.</li> <li>4. Este tipo de abordagem pode contribuir para um catálogo de fatores de defeitos (causas dos defeitos ou maneiras de prevenir) de codificação.</li> </ol>
<b>Descrição de Conhecimento</b>
Obtido da experiência na IBM de Boeblingen, referente à análise de defeitos de uma nova liberação ( <i>release</i> 28) do sistema operacional DOS/VS.
<b>Tipo de Estudo</b>
Relato de Experiência.

<b>Título do Artigo</b>
A process-integrated approach to defect prevention.
<b>Referência Completa</b>
Jones, C.L., A process-integrated approach to defect prevention, IBM Systems Journal, 24(2), 150-67, 1985.
<b>Fonte</b>
INSPEC.
<b>Esquema de Classificação de Defeitos</b>
Não tem um esquema, apenas menciona que a fase em que o defeito foi inserido deve ser determinada (no artigo ela é determinada como parte da reunião de análise causal).
<b>Esquema de Classificação de Causas de Defeitos</b>
O artigo cita quatro categorias de causas de defeitos, que de acordo com o mesmo poderiam ser utilizadas para qualquer defeito: Descuido, educação, comunicação e transcrição.
<b>Identificação do Processo ou Abordagem</b>
Abordagem integrada ao processo para prevenção de defeitos (depois referenciada como processo tradicional de prevenção de defeitos).
<b>Descrição do Processo ou Abordagem</b>
<p>A abordagem divide uma atividade de um processo em subatividades de ‘entrada’, ‘tarefa’, ‘validação’ (inspeções para verificar se todo o trabalho foi realizado de forma apropriada) e ‘saída’. Onde a ‘tarefa’ e a ‘validação’ podem ocorrer parcialmente em paralelo.</p> <p>A atividade é então modificada de modo a englobar uma metodologia de análise causal e resolução da seguinte forma:</p> <ul style="list-style-type: none"> <li>• Sub-atividade de entrada. É adicionada uma reunião de início. Nesta reunião os objetivos de equipe devem ser estabelecidos.</li> <li>• Sub-atividade de validação. Melhorada por incluir uma descrição do defeito na base de dados e, durante o retrabalho adicionar uma descrição da solução, além de uma análise causal preliminar, feita pelo próprio autor (indicado pelo artigo como a pessoa mais apropriada para realizar esta tarefa). Podem ser incluídos ainda a categoria do defeito, a atividade em que o mesmo foi introduzido e sugestões de prevenção.</li> <li>• Sub-atividade de saída. Nesta atividade é introduzida uma reunião de análise causal. Esta reunião compreende: <ul style="list-style-type: none"> <li>○ Analisar defeitos (determinando origem, categoria da causa, causa, como prevenir e ação corretiva).</li> <li>○ Avaliar os resultados em relação aos objetivos de equipe.</li> <li>○ Avaliar a atividade do processo para que melhorias de processo possam ser discutidas.</li> </ul> </li> </ul>

<p>Adicionalmente o artigo menciona que existem dados não provenientes de inspeções que eventualmente podem ser analisados: dados de testes funcionais, dados de problemas em campo, dados sobre mudanças de projeto e sugestões da equipe de garantia da qualidade.</p> <p>A abordagem sugere que uma equipe de ação (<i>action team</i>) responda pelas sugestões de melhoria. Esta equipe deve possuir um gerente responsável por priorizar tarefas e assegurar que o trabalho é realizado e que tenha visibilidade, disseminando <i>feedback</i>. O artigo menciona ainda a importância de uma ferramenta para lidar tanto com dados de defeitos como dados das ações, uma vez que ambos estão logicamente conectados.</p>
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. A aplicação da metodologia resulta em uma menor injeção de defeitos em cada atividade do desenvolvimento, reduzindo o número de defeitos encontrados nos testes em até 50%.</li> <li>2. A aplicação da metodologia resulta em melhorias nas áreas de comunicação e na percepção da qualidade.</li> <li>3. Análise causal deve enfatizar prevenção de defeitos e não detecção de defeitos.</li> <li>4. O custo de implementar a metodologia de prevenção de defeitos não é alto.</li> <li>5. Sem coletar e analisar estatisticamente os dados de defeitos e suas tendências falhamos em utilizar a informação mais valiosa que temos para ajustar o processo de desenvolvimento.</li> </ol>
<b>Descrição de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. e 2. Resultados preliminares da aplicação da metodologia na IBM em duas áreas de produtos.</li> <li>3. Apenas uma definição para a metodologia apresentada no artigo.</li> <li>4. O artigo apresenta um planejamento completo dos requerimentos de tempo e de recursos para implementar a metodologia.</li> <li>5. Afirmação final do artigo.</li> </ol>
<b>Tipo de Estudo</b>
Relato de experiência (na IBM).

<b>Título do Artigo</b>
An approach to software causal analysis and defect extinction
<b>Referência Completa</b>
Philips, R.T., An approach to software causal analysis and defect extinction, in 'Proceedings of the IEEE Globecom Conference, pp. 412-416, 1986.
<b>Fonte</b>
INSPEC, EI Compendex
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Análise causal de defeitos de software, baseado no conceito de ciclos de qualidade, congruente com a abordagem de (Jones, 1985).
<b>Descrição do Processo ou Abordagem</b>
O artigo descreve uma versão embrionária do processo de análise causal de defeitos de software aplicado na IBM para tratar tanto defeitos encontrados em campo quanto defeitos encontrados nas atividades de testes.
<b>Identificação de Conhecimento</b>
1. Análise causal de defeitos de software resultou em 50% de redução na taxa de defeitos encontrados no ciclo de vida.
<b>Descrição de Conhecimento</b>
Conhecimento obtido na IBM ao comparar uma equipe de desenvolvimento onde ações foram implementadas pela equipe de qualidade (após a análise causal) com um grupo de controle seguindo as técnicas convencionais.
<b>Tipo de Estudo</b>
Relato de Experiência (na IBM).

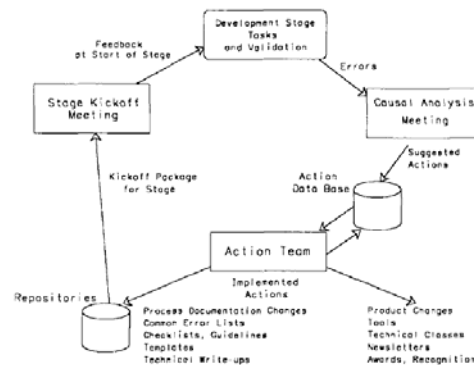
<b>Título do Artigo</b>
Getting Started on Metrics - JPL Productivity and Quality.
<b>Referência Completa</b>
Bush, M.W., Getting Started on Metrics - JPL Productivity and Quality, in 'ICSE '90: Proceedings of the 12th International Conference on Software Engineering', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 133-142, 1990.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
A abordagem descrita no artigo não é de análise causal, mas sim para reconstruir quinze anos de história de software no <i>Jet Propulsion Laboratory</i> , obtendo dados (incluindo dados sobre defeitos) de projetos passados.
<b>Descrição do Processo ou Abordagem</b>
Não se aplica.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Dados crus de incidência de defeitos (número de defeitos por subsistema) permitem uma percepção de qualidade, mas não provêem meios para melhoria. De acordo com os dados dos últimos quinze anos sobre qualidade (defeitos por linha de código), o Jet Propulsion Laboratory, sem as métricas, não conseguiu se manter entre as organizações mais eficientes. Em contraste, no período de 1982 a 1985 a IBM (em software produzido para a NASA) mostrou uma redução no número de defeitos por mil linhas de código de 2,25 para 0,08. Entre os fatores para tal redução estaria o uso de inspeções para a detecção de defeitos cedo no processo de desenvolvimento.</li> <li>2. Sugestão do autor sobre medição: iniciar de forma simples e o quanto antes.</li> </ol>
<b>Descrição de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Experiência proveniente da coleção de dados referentes a aproximadamente cinco milhões de linhas de código, esforço de trinta mil homens-mês e mais de um terço de um bilhão de dólares de valor no <i>Jet Propulsion Laboratory</i>.</li> </ol>
<b>Tipo de Estudo</b>
Relato de experiência (no <i>Jet Propulsion Laboratory/NASA</i> )



<b>Título do Artigo</b>
Applications of Defect Prevention in Software Development.
<b>Referência Completa</b>
Mays, R.G., Applications of Defect Prevention in Software Development, IEEE Journal on Selected Areas in Communications, Vol.8, No.2, February 1990.
<b>Fonte</b>
IEEE, INSPEC, EI COMPENDEX.
<b>Esquema de Classificação de Defeitos</b>
Não tem um esquema, apenas menciona que a fase em que o defeito foi inserido deve ser determinada (no artigo ela é determinada como parte da reunião de análise causal).
<b>Esquema de Classificação de Causas de Defeitos</b>
O artigo cita como as quatro categorias básicas de causas de defeitos: descuido, educação, comunicação e transcrição.
<b>Identificação do Processo ou Abordagem</b>
Processo tradicional de prevenção de defeitos.
<b>Descrição do Processo ou Abordagem</b>
<p>Processo composto por 4 atividades:</p> <ul style="list-style-type: none"> <li>• Reuniões de Análise Causal: Reuniões de desenvolvedores, ocorrendo ao final de cada fase. Deve ocorrer dentro de 2 horas. Normalmente possui de 4 a 7 participantes. As seguintes questões são respondidas para cada defeito: <ul style="list-style-type: none"> <li>○ O que é o defeito?</li> <li>○ Qual a categoria da causa do defeito?</li> <li>○ Qual a causa do defeito?</li> <li>○ Em que fase o defeito foi inserido?</li> <li>○ Que tipo de ação preveniria este tipo de defeito de re-ocorrer?</li> </ul> </li> <li>• Implementação das Ações (<i>Action Team</i>): Desenvolvedores que trabalham tempo parcial para assegurar que as ações sugeridas são implementadas. Cada membro da equipe tem responsabilidades específicas: <ul style="list-style-type: none"> <li>○ Definição do processo e documentação;</li> <li>○ Requisitos de ferramentas e implementações;</li> <li>○ Educação;</li> <li>○ Representantes de projeto, desenvolvimento e testes para lidar com ações que requeiram expertise nestas áreas;</li> <li>○ Um gerente para lidar com comunicações e negociações com outros gerentes.</li> </ul> </li> <li>• Reuniões de início de fase: Ocorre antes do início de uma fase, serve para fornecer feedback aos desenvolvedores. Normalmente levam de 1 a 2 horas. É liderada pelo líder técnico da equipe. O pacote de kickoff contém as últimas atualizações do processo do Action Team e inclui: <ul style="list-style-type: none"> <li>○ Descrição do processo para a fase;</li> <li>○ Exemplos de saídas que devem ser produzidas;</li> <li>○ Métodos de validação que serão utilizados;</li> </ul> </li> </ul>

- Uma lista de defeitos comum para a fase;
- Atribuições de tarefas e o cronograma a ser seguido.
- Coleção de dados e acompanhamento: Base de dados de ações e ferramentas de apoio para acompanhar ações sugeridas na medida em que são tratadas pelo Action Team. Uma ações normalmente passa por:
  - Atribuição de tarefa;
  - Investigação;
  - Implementação;
  - Fechamento.

Este processo está ilustrado na figura abaixo (extraída do artigo).



### Identificação de Conhecimento

1. O processo de prevenção de defeitos pode ser aplicado nas diferentes fases de desenvolvimento.
2. O processo de prevenção de defeitos apresenta ganhos e baixo custo.

### Descrição de Conhecimento

1. O artigo mostra que o processo pode ser aplicado nas diferentes fases de desenvolvimento, como projeto, codificação, testes. Isto é feito ao relatar a aplicação do processo em diferentes setores da IBM, em atividades de projeto, codificação, testes, entre outros.
2. Relata redução de defeitos na ordem de 50% na IBM e custo abaixo de um homem-ano para uma organização com 150-200 desenvolvedores.

### Tipo de Estudo

Relato de Experiência (na IBM).

<b>Título do Artigo</b>
Experiences with Defect Prevention.
<b>Referência Completa</b>
Mays, R.G., Jones, C.L., Holloway, G.J., Studinski, D.P., Experiences with Defect Prevention, IBM Systems Journal, 29(1), pp. 4-32, 1990.
<b>Fonte</b>
INSPEC.
<b>Esquema de Classificação de Defeitos</b>
Não tem um esquema, apenas menciona que a fase em que o defeito foi inserido deve ser determinada (no artigo ela é determinada como parte da reunião de análise causal).
<b>Esquema de Classificação de Causas de Defeitos</b>
O artigo cita como as quatro categorias básicas de causas de defeitos: descuido, educação, comunicação e transcrição. O artigo menciona que, além destas quatro, a categoria 'falha no processo' tem sido utilizada em algumas organizações, mas na opinião dos autores as falhas no processo só devem ser consideradas depois que as causas humanas foram identificadas, idealmente quando as ações preventivas forem propostas.
<b>Identificação do Processo ou Abordagem</b>
Processo tradicional de prevenção de defeitos.
<b>Descrição do Processo ou Abordagem</b>
O mesmo processo descrito em (Jones, 1985) e (Mays, 1990). Cita o uso de diagramas de causa e efeito (Ishikawa, 1976) e de gráficos de Pareto na manufatura, mas não os utiliza efetivamente para software.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Com o processo de prevenção de defeitos uma redução de defeitos em mais de 50% pode ser alcançada a um custo de menos de meio por cento dos recursos da área de produto, em um período de 3 anos.</li> <li>2. Há quatro elementos chave no processo de prevenção de defeitos: (1) análise causal sistemática, (2) equipe de ação gerenciada, (3) reuniões de início de fase e (4) ferramentas para coleção de dados e acompanhamento das ações.</li> </ol>
<b>Descrição de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Experiência obtida de mais de vinte e cinco organizações de sete laboratórios de desenvolvimento da IBM. Adicionalmente um produto foi estudado em detalhes, utilizando dados de oito liberações sem prevenção de defeitos, introduzida nas próximas duas liberações. Este estudo revelou uma redução de 54% nos defeitos, com significância estatística (<math>p &lt; 0,05</math>) de acordo com o teste estatístico <i>t-test</i>. Neste estudo pôde ser observada ainda uma redução de 60% no número de defeitos revelados após a liberação para os clientes.</li> <li>2. Experiência obtida de mais de vinte e cinco organizações de sete laboratórios de desenvolvimento da IBM.</li> </ol>

<b>Tipo de Estudo</b>
Relato de Experiência (na IBM).

<b>Título do Artigo</b>
Managing Software Reliability and Support Costs.
<b>Referência Completa</b>
Keene, J., Managing Software Reliability and Support Costs, in 'Proceedings of the Leesburg Workshop on Reliability and Maintainability Computer-Aided Engineering in Concurrent Engineering', pp. 201-205, 1991.
<b>Fonte</b>
IEEE, INSPEC.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Não tem.
<b>Descrição do Processo ou Abordagem</b>
Não se aplica.
<b>Identificação de Conhecimento</b>
1. O processo de prevenção de defeitos de Mays (1990) é uma iniciativa que ajuda em assegurar que os requisitos de um software estão corretos. Como consequência tem alto impacto nos custos de suporte de software em campo.
<b>Descrição de Conhecimento</b>
Afirmção do artigo, não há nenhuma fundamentação experimental para a afirmação.
<b>Tipo de Estudo</b>
Teórico.

<b>Título do Artigo</b>
A Case History Analysis of Software Error Cause-Effect Relationships.
<b>Referência Completa</b>
Nakajo, T. & Kume, H. (1991), A Case History Analysis of Software Error Cause-Effect Relationships, IEEE Transactions on Software Engineering, 17(8), 830-838, 1991.
<b>Fonte</b>
IEEE, INSPEC, Web of Science.
<b>Esquema de Classificação de Defeitos</b>
As categorias de defeitos utilizadas no artigo são: defeitos internos do módulo, defeitos de interface do módulo e defeitos de funcionalidade do módulo.
<b>Esquema de Classificação de Causas de Defeitos</b>
<p>O artigo divide causas associadas a erros humanos nas seguintes categorias:</p> <ul style="list-style-type: none"> <li>• Erros de comunicação dentro de uma equipe de desenvolvimento.</li> <li>• Erros de comunicação entre diferentes equipes de desenvolvimento.</li> <li>• Erros de entendimento dos requisitos.</li> <li>• Erros ao desenvolver requisitos.</li> </ul>
<b>Identificação do Processo ou Abordagem</b>
Abordagem para identificar relacionamentos de causa efeito de defeitos de software.
<b>Descrição do Processo ou Abordagem</b>
<p>Foram analisados aproximadamente 700 defeitos em 4 produtos de softwares comerciais de controle e mensuração, visando obter seus relacionamentos de causa efeito. A abordagem utilizada envolveu os seguintes passos:</p> <ul style="list-style-type: none"> <li>• Definir pontos de observação ao longo do caminho que leva uma causa a um efeito de um defeito de software. No exemplo os pontos de observação definidos foram: falha no programa, erro de origem humana e falha no processo.</li> <li>• Obter os dados dos pontos de observação para cada defeito utilizando análise da árvore de defeito (<i>Fault Tree Analysis</i>) (Leveson e Harvey, 1983).</li> <li>• Categorizar os dados de cada ponto de observação, obtendo o relacionamento entre estes através de uma tabela de cruzamento de índices. No exemplo foram utilizadas tabelas cruzando defeitos de programa com erros de origem humana e erros de origem humana com falhas no processo.</li> </ul>
<b>Identificação de Conhecimento</b>
<p>A aplicação da abordagem resultou em quatro mecanismos de ocorrência de defeitos.</p> <ol style="list-style-type: none"> <li>1. Falha de processo (definições de interface inapropriadas + falta de métodos para registrar definições de interfaces) → Erro de origem humana (falta de entendimento das especificações de interface de software) → Defeito de Programa (defeitos de interface do módulo).</li> </ol>

<p>2. Falha de processo (definições de interface inapropriadas + falta de métodos de comunicação entre engenheiros de software e de hardware) → Erro de origem humana (falta de entendimento das especificações de interface de hardware) → Defeito de Programa (defeitos de interface do módulo).</p> <p>3. Falha de processo (correspondência complicada entre requisitos e maneiras de realizá-los + falta de método sistemático para descrever funções externas do sistema) → Erro de origem humana (falta de entendimento das funções do sistema) → Defeito de Programa (defeitos de funcionalidade do módulo).</p> <p>4. Falha de processo (correspondência complicada entre requisitos e maneiras de realizá-los + falta de métodos descrevendo funções dos módulos e os relacionamentos entre o módulo e funções externas do sistema) → Erro de origem humana (falta de entendimento das funções do módulo) → Defeito de Programa (defeitos de funcionalidade do módulo).</p>
<b>Descrição de Conhecimento</b>
<p>Obtido a partir da aplicação da abordagem a 670 defeitos de software ocorrendo durante o desenvolvimento de 4 pacotes de equipamentos de controle de mensuração na Yokogawa-Hewlett-Packard. As conclusões foram obtidas a partir de pacotes de software específicos, e são limitadas a estes. Entretanto os autores acreditam que possam ser generalizadas e aplicadas ao desenvolvimento de outros produtos de software.</p>
<b>Tipo de Estudo</b>
<p>Relato de experiência (na Yokogawa-Hewlett-Packard)</p>

<b>Título do Artigo</b>
Foolproofing and Quality Feedback: Keys of Process-Based Management.
<b>Referência Completa</b>
Nakajo, T., Foolproofing and Quality Feedback: Keys of Process-Based Management, Proceedings of the Fifteenth Annual International Computer Software and Applications Conference, COMPSAC'91, IEEE Computer Society Press, 390-391, 1991.
<b>Fonte</b>
INSPEC.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
O artigo cita três métodos de engenharia de software para análise de causa e efeito visando melhoria de processos: análise baseada em métricas, análise da árvore de defeito ( <i>fault tree analysis</i> ) e análise baseada em casos ( <i>case-based analysis</i> ).
<b>Descrição do Processo ou Abordagem</b>
<p>Análise baseada em métricas nada mais é do que um método convencional para analisar de forma empírica as relações de causa e efeito (Basili e Perricone, 1984), também conhecido por <i>goal-oriented analysis</i>, tendo sua base no paradigma <i>goal question metric</i> (Basili, 1989). Normalmente utiliza coeficientes de correlação entre métricas de taxas de defeitos e métricas de características do processo. Análise da árvore de defeito (<i>fault tree analysis</i>) (Leveson e Harvey, 1983), por sua vez, utiliza um diagrama de árvore para representar os relacionamentos de causa e efeito de cada defeito e ações corretivas.</p> <p>Por fim, análise baseada em casos (<i>case-based reasoning</i>) nada mais é do que a abordagem descrita em (Nakajo e Kume, 1991), artigo analisado como parte desta revisão sistemática. De forma resumida a análise baseada em casos segue FTA, facilitando a extração de relações causa e efeito ao restringir a estrutura da árvore lógica.</p>
<b>Identificação de Conhecimento</b>
Não tem.
<b>Descrição de Conhecimento</b>
Não tem.
<b>Tipo de Estudo</b>
Teórico.



<b>Título do Artigo</b>
Experiences in the Application of Customer-Based Metrics in Improving Software Service Quality.
<b>Referência Completa</b>
Pratt, W. M., Experiences in the Application of Customer-Based Metrics in Improving Software Service Quality, Conference Record of the International Conference on Communications, pp. 1459 - 1462, 1991.
<b>Fonte</b>
EI Compendex.
<b>Esquema de Classificação de Defeitos</b>
Não tem. Mas o autor menciona que o mecanismo de captura de defeitos deve prover informações de tempo real, consistentes e precisas sobre o defeito. Exemplos destas informações citadas no artigo são: descrição do defeito, quando e aonde ocorreu, quando e aonde foi identificado e como se manifestou.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Abordagem para assegurar qualidade de serviço utilizando medidas baseadas nos clientes. Utiliza análise causal de defeitos seguindo o processo tradicional de prevenção de defeitos.
<b>Descrição do Processo ou Abordagem</b>
<p>A abordagem envolve os seguintes passos:</p> <ul style="list-style-type: none"> <li>• Entender o que guia a percepção de qualidade de serviço por parte do cliente.</li> <li>• Desenvolver medidas que refletem os guias de percepção de qualidade.</li> <li>• Estabelecer um mecanismo formal de captura de defeitos.</li> <li>• Desenvolver um processo de análise causal de defeitos.</li> <li>• Alinhar as métricas internas com as métricas do cliente.</li> </ul> <p>O passo 'Desenvolver um processo de análise causal de defeitos', de acordo com os autores, pode ser implementado utilizando o processo de prevenção de defeitos de Mays <i>et al</i> (1990).</p>
<b>Identificação de Conhecimento</b>
1. A aplicação da abordagem, ligando a captura de defeitos à análise causal e esforços de melhoria, se mostrou valiosa. O volume de melhorias de processo e de ações corretivas aumentou e está resultando em maior qualidade do software entregue ao cliente.
<b>Descrição de Conhecimento</b>

Experiência relatada sobre o uso da abordagem na <i>Northern Telecom</i> para a entrega de uma aplicação de software para clientes do escritório central de telefonia digital.
<b>Tipo de Estudo</b>
Relato de experiência ( <i>na Northern Telecom</i> ).

<b>Título do Artigo</b>
A Case Study of Software Process Improvement During Development.
<b>Referência Completa</b>
Bhandari, I., Halliday, M., Tarver, E., Brown, D., Chaar, J., Chillarege, R., A Case Study of Software Process Improvement During Development, IEEE Transactions on Software Engineering, 19 (12), 1157 – 1170, 1993.
<b>Fonte</b>
EI Compendex.
<b>Esquema de Classificação de Defeitos</b>
<p>ODC (Chillarege <i>et al.</i>, 1992), com atributos para relacionar um defeito a um conjunto de atividades do processo, conforme já sugerido no próprio ODC. Atributos do ODC: tipo de defeito, omitido/incorreto, gatilho (<i>trigger</i>, como foi detectado), origem (nova função ou correção) e impacto. Atributos para relacionar às atividades do processo: fase de detecção, fase de introdução e componente (localização).</p> <p>Os tipos são os definidos no ODC: interface, funcionalidade, build/package/merge, atribuição, documentação, verificação, algoritmo e sincronização/serialização.</p>
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Abordagem para melhoria de processos baseada em defeitos, utilizando <i>attribute focusing</i> .
<b>Descrição do Processo ou Abordagem</b>
<p>A abordagem envolve duas atividades:</p> <ul style="list-style-type: none"> <li>• Classificação de defeitos, de acordo com o esquema de classificação descrito acima.</li> <li>• Análise de defeitos e <i>feedback</i>. Nesta atividade é utilizado <i>attribute focusing</i> (AF) (Bhandari, 1993), processando os dados automaticamente para produzir um conjunto de gráficos que então são interpretados pela equipe do projeto. AF faz uso de funções de interesse (<i>interestingness functions</i>) e de funções de filtragem. As funções de interesse ordenam os valores dos atributos para refletir seu interesse potencial para análise. Heurísticas comumente utilizadas para as funções de interesse são: medidas de magnitude, associação, correlação e entropia de informação. As funções de filtragem filtram os resultados da função de interesse e determinam quais gráficos devem ser analisados. Em seguida, em sessões de <i>feedback</i> envolvendo pessoas do projeto os seguintes elementos são analisados para os gráficos automaticamente produzidos: causa, implicação, ação corretiva e justificativa da causa.</li> </ul>
<b>Identificação de Conhecimento</b>

<ol style="list-style-type: none"> <li>1. A interpretação de gráficos de AF ajuda a identificar e corrigir problemas, bem como validar a efetividade de ações corretivas.</li> <li>2. O custo da classificação de defeitos é menor do que 4 pessoas minuto por defeito (Chillarege <i>et al.</i>, 1992).</li> <li>3. O custo da aplicação da abordagem não é maior do que 25% do tempo de uma pessoa (considerando um membro da equipe para assegurar que os dados são coletados e para agendar as sessões de feedback, a classificação dos defeitos e as sessões de feedback).</li> <li>4. Comparando com análise causal de defeitos de software, considerando cada defeito isoladamente: <ol style="list-style-type: none"> <li>a. Análise causal é uma abordagem custosa e assim limitada a um subconjunto de defeitos. Desta forma, problemas que se manifestam através de tendências na população de defeitos podem não ser corrigidos.</li> <li>b. Defeitos que podem ser descobertos apenas através de sua descrição textual não serão encontrados por AF.</li> <li>c. Quando a função de interesse não puder ser explicada, defeitos podem ser analisados por sua descrição textual aplicando análise causal.</li> </ol> </li> <li>5. Comparando abordagens de análise orientada a objetivos (goal-oriented approaches), baseados no goal question metric paradigm (GQM) (Basili, 1989) e na captura e acompanhamento de dados sobre defeitos: <ol style="list-style-type: none"> <li>a. Análise orientada a objetivos requer dados históricos.</li> <li>b. Muitas vezes as funções interesse do método não podem substituir objetivos obtidos através da experiência de indivíduos.</li> </ol> </li> <li>6. A abordagem pode ser combinada de forma sinérgica com análise causal de defeitos e análise orientada a objetivos.</li> </ol>	
<p><b>Descrição de Conhecimento</b></p> <p>Conhecimento obtido a partir de resultados de um estudo de caso envolvendo uma equipe de 25 pessoas desenvolvendo um sistema operacional. Neste estudo de caso foi utilizado um processo modificado de desenvolvimento, incorporando técnicas do estado da arte e ferramentas avançadas. O estudo de caso fez uso de um projeto piloto (primeiros 5KLOCs) e além do método proposto utilizou análise causal de defeitos individuais e análise orientada a objetivos.</p>	
<p><b>Tipo de Estudo</b></p>	
<p>Estudo de caso (na IBM).</p>	

<b>Título do Artigo</b>
Post-process Feedback with and without Attribute Focusing: a Comparative Evaluation.
<b>Referência Completa</b>
Bhandari, I., Roth, N., Post-process Feedback with and without Attribute Focusing: a Comparative Evaluation, in 'ICSE '93: Proceedings of the 15th International Conference on Software Engineering', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 89-98, 1993.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
<p>Um questionário múltipla-escolha foi preenchido para cada defeito a ser analisado. O questionário continha as seguintes perguntas:</p> <ol style="list-style-type: none"> <li>1. Em qual componente o defeito foi encontrado? Opções de resposta: Os componentes do sistema.</li> <li>2. Como o defeito foi introduzido? Opções de resposta: (a) Nova liberação do produto, (b) Campo, (c) Desenvolvimento, (d) Não sei.</li> <li>3. Onde o defeito deveria ter sido detectado? Opções de resposta: as fases do desenvolvimento.</li> <li>4. Porque o defeito não foi detectado? Opções de resposta: (a) Configuração de hardware inapropriada, (b) Combinação de produtos de software, (c) Baixa ou nenhuma exploração de funções, (d) Não foi estressado, (e) Problema não foi reconhecido ou não pode ser diagnosticado, (f) Problema intermitente, (g) Outro motivo.</li> <li>5. Qual seria a melhor forma de encontrar um novo problema como este? Opções de resposta: (a) Inspeções/Revisões, (b) Casos de teste, (c) Testes de estresse, (d) Testes envolvendo outros produtos ou componentes, (e) Outra forma.</li> <li>6. Durante o processo de desenvolvimento, é provável que casos de teste de regressão fossem escritos para expor o problema? Opções de resposta: (a) Sim, (b) Provavelmente não.</li> </ol>
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Análise post-mortem de defeitos de software utilizando <i>Attribute Focusing</i> .

Descrição do Processo ou Abordagem
Após a classificação dos defeitos de acordo com o questionário múltipla-escolha descrito acima, ocorre uma análise de defeitos, conforme descrito em (Bhandari <i>et al.</i> , 1993), artigo analisado como parte desta revisão sistemática.
Identificação de Conhecimento
<ol style="list-style-type: none"> <li>1. <i>Attribute focusing</i> é uma abordagem útil para análise de dados de defeitos durante o desenvolvimento, por permitir uma rápida análise de defeitos.</li> <li>2. <i>Attribute focusing</i> mostrou melhorar as práticas atuais de análise de dados de defeitos post-mortem para projetos complexos de software, em termos de: <ol style="list-style-type: none"> <li>a. Completude. Medido a partir do número de recomendações.</li> <li>b. Redução no tempo de análise.</li> <li>c. Eficiência das recomendações, demonstrada pela habilidade de sugerir melhores recomendações e prover um melhor entendimento dos dados.</li> </ol> </li> <li>3. <i>Attribute focusing</i> deveria ser adicionada ao conjunto de técnicas consideradas úteis para análise de dados e correção de processos.</li> </ol>
Descrição de Conhecimento
<ol style="list-style-type: none"> <li>1. Observação do estudo de caso apresentado em (Bhandari <i>et al.</i>, 1993), artigo analisado como parte desta revisão sistemática.</li> <li>2. Observação de um estudo de caso descrito no próprio artigo, no qual análise post-mortem com e sem <i>attribute focusing</i> são comparadas. Neste estudo de caso cinco equipes analisaram dados de defeitos de software e fizeram recomendações para melhorar a qualidade de um grande produto de sistema operacional. Subseqüentemente os mesmos dados foram re-analisados utilizando <i>attribute focusing</i>. A nova análise sugeriu as mesmas recomendações da análise feita pelas cinco equipes, levando menos tempo. Com <i>attribute focusing</i> duas das equipes foram capazes de sugerir recomendações adicionais, quatro equipes consideraram que foi possível obter um melhor entendimento dos dados sendo analisados e um problema organizacional, comum aos cinco projetos, pode ser facilmente identificado.</li> <li>3. Sugestão dos autores com base nos resultados do estudo de caso.</li> </ol>
Tipo de Estudo
Estudo de Caso (na IBM).

<b>Título do Artigo</b>
Defect Causal Analysis Drives Down Error Rates
<b>Referência Completa</b>
Card, D., Defect Causal Analysis Drives Down Error Rates, IEEE Software 1/1993, Volume 10, Issue 4, 7/1993, p.98-99, 1993.
<b>Fonte</b>
Não retornado (Artigo de controle, está na base da IEEE e atende à string de busca do protocolo).
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
A CSC utiliza o mesmo esquema de classificação de defeitos sugerido por Kaoru Ishikawa (Ishikawa, 1976), contendo as seguintes quatro categorias: ferramentas, entradas, pessoas e métodos.
<b>Identificação do Processo ou Abordagem</b>
Processo de análise causal de defeitos (compatível com o processo de prevenção de defeitos de Mays (1990)).
<b>Descrição do Processo ou Abordagem</b>
De acordo com o artigo, na análise causal de defeitos, os defeitos encontrados durante a atividade de testes são relatados aos produtores, que se encontram periodicamente (normalmente ao final de uma fase ou de um <i>build</i> ) para analisar as causas e propor ações para prevenir que eles ocorram novamente no futuro. Os produtores então passam as propostas de ação a uma equipe separada para implementação.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Uma premissa para a análise causal de defeitos é que os produtores (desenvolvedores e mantenedores), por possuírem mais intimidade com o processo e o produto, são os melhores para determinar como defeitos foram introduzidos no produto e como alterar o processo para prevenir que os mesmos sejam introduzidos novamente.</li> <li>2. Análise causal de defeitos é uma tecnologia que ajuda a aumentar a maturidade das organizações, por facilitar: <ol style="list-style-type: none"> <li>a. a percepção da qualidade;</li> <li>b. o comprometimento com o processo;</li> <li>c. o reconhecimento pela necessidade de medição de qualidade.</li> </ol> </li> <li>3. A equipe de análise causal pode cair em diversas armadilhas se o processo não for introduzido de forma cuidadosa, entre elas: <ol style="list-style-type: none"> <li>a. tentar consertar processos alheios (como por exemplo o processo de testes);</li> <li>b. defender seus próprios processos;</li> <li>c. perder o foco (o processo deve ser documentado e monitorado);</li> <li>d. demorar para implementar as ações (as ações devem ser gerenciadas, se as ações</li> </ol> </li> </ol>

<p>não são tomadas as propostas são inúteis).</p> <p>4. Na CSC com um investimento de 1,5% do orçamento do projeto foi possível obter uma redução nas taxas de defeitos em mais de 50% ao longo de dois anos. Estes números são citados no artigo e estão publicados em (Dangerfield <i>et al.</i>, 1992).</p>
<b>Descrição de Conhecimento</b>
Obtido da experiência da institucionalização do processo na Computer Sciences Corporation (CSC)
<b>Tipo de Estudo</b>
Relato de Experiência (na Computer Sciences Corporation).



<b>Título do Artigo</b>
Application of Causal Analysis to the Software Modification Process.
<b>Referência Completa</b>
Collofello, J., Gosalla, B., Application of Causal Analysis to the Software Modification Process, Software Practice and Experience, 23(10), pp. 1095–1105, 1993.
<b>Fonte</b>
INSPEC, EI Compendex, Web of Science.
<b>Esquema de Classificação de Defeitos</b>
<p>O artigo cita que as seguintes informações devem ser obtidas para cada defeito: Tipo de defeito, fase em que o defeito foi encontrado, fase em que o defeito foi inserido, causa do defeito, soluções para prevenir que o defeito re-ocorra no futuro. Além disto os defeitos são priorizados de acordo com três categorias de severidade: críticos, relevantes e pouco relevantes.</p> <p>Os tipos de defeitos utilizados no artigo são específicos para defeitos de manutenção de software. Eles foram determinados baseado na experiência dos autores analisando centenas de defeitos de manutenção. Os tipos são: defeitos de projeto, incompatibilidade de interface, sincronização incorreta de código proveniente de projetos paralelos, remendo (<i>patch</i>) incorreto de objetos e exaustão de recursos do sistema.</p>
<b>Esquema de Classificação de Causas de Defeitos</b>
Novamente o artigo cria seu próprio esquema de classificação, específico para causas de defeitos referentes a manutenção de software. O esquema envolve as seguintes categorias: conhecimento do sistema / experiência, comunicação, impactos de software, métodos e padrões, implantação, ferramentas de apoio, erro de origem humana.
<b>Identificação do Processo ou Abordagem</b>
Processo tradicional de prevenção de defeitos (Jones, 1985).
<b>Descrição do Processo ou Abordagem</b>
<p>Processo tradicional, aplicando a seguinte abordagem na atividade de análise causal:</p> <ol style="list-style-type: none"> <li>1. Obter relatos de problemas.</li> <li>2. Priorizar problemas.</li> <li>3. Categorizar defeitos.</li> <li>4. Analisar causas de defeitos.</li> <li>5. Desenvolver recomendações.</li> </ol>
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Aproximadamente 80% dos defeitos gerados durante manutenção de software, detectados através de testes de integração são causados por conhecimento e/ou experiência insuficiente, problemas de comunicação ou falha em considerar todos os impactos da mudança.</li> </ol>

2. A aplicação de análise causal ao processo de manutenção de software contribui para uma redução de defeitos na organização.
<b>Descrição de Conhecimento</b>
Obtido através da aplicação do processo de prevenção de defeitos na criação de uma nova versão de um grande (aprox. 2,9 milhões de linhas de código) sistema de telefonia. O estudo se limitou a analisar defeitos (mais de 200) encontrados durante testes de integração. Portanto os resultados são limitados aos tipos de defeito normalmente encontrados por meio de testes de integração.
<b>Tipo de Estudo</b>
Estudo de caso (na <i>AG Communication Systems Corporation</i> ).

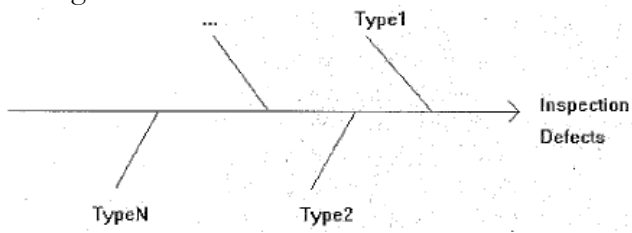
<b>Título do Artigo</b>
Reliable Software and Communication: Software Quality, Reliability, and Safety.
<b>Referência Completa</b>
Dalal, S. R., Horgan, J. R., Kettenring, J. R., Reliable Software and Communication: Software Quality, Reliability, and Safety, in 'ICSE '93: Proceedings of the 15th International Conference on Software Engineering', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 425-435, 1993.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Não tem, apresenta uma abordagem para melhoria do processo de software, utiliza análise de dados de defeitos.
<b>Descrição do Processo ou Abordagem</b>
O artigo analisa o processo de desenvolvimento de software e sugere oportunidades para sua melhoria, utilizando uma combinação de controle estatístico de processos com outras técnicas de controle. De acordo com os autores a análise causal de defeitos tem um papel central e crucial para melhorar o processo de desenvolvimento de software.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. A análise causal de defeitos tem um papel central e crucial para melhorar o processo de desenvolvimento de software.</li> <li>2. Abordagens estatísticas podem ser empregadas para controle de processos de desenvolvimento de software (HUMPHREY, 1988) (COBB e MILLS, 1990).</li> <li>3. O número de defeitos não é informação suficiente para apoiar a controlar e melhorar o processo de software.</li> <li>4. O diagrama de causa e efeito tem sido considerado uma ferramenta útil para análise causal de defeitos por diversas organizações de software.</li> <li>5. Outras formas de estabelecer e aumentar a maturidade de processos de software não devem ser negligenciadas. O modelo de maturidade do Software Engineering Institute (SEI) é um exemplo de arcabouço conceitual para melhoria de processos.</li> <li>6. Técnicas de controle estatístico de processos podem trazer ganhos significativos ao processo de desenvolvimento de software.</li> </ol>
<b>Descrição de Conhecimento</b>
Conhecimento, em sua maioria, representando opiniões pessoais dos autores, não havendo nenhum tipo de avaliação experimental.

Tipo de Estudo
Teórico.

<b>Título do Artigo</b>
Cleanroom Software Engineering for Zero-Defect Software.
<b>Referência Completa</b>
Linger, R. C., Cleanroom Software Engineering for Zero-Defect Software, in 'ICSE '93: Proceedings of the 15th International Conference on Software Engineering', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 2-13, 1993.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
<i>Cleanroom</i> engloba verificações de corretude, mas não possui um processo de análise causal. A abordagem sugere utilizar testes estatísticos de uso, que possibilitam a priorização de defeitos que contribuem mais para o aumento do tempo médio até falha ( <i>mean time to failure</i> ).
<b>Descrição do Processo ou Abordagem</b>
Os testes estatísticos de uso ( <i>statistical usage testing</i> ) visam testar o software da maneira como será utilizado por seus usuários. Utilizando testes estatísticos de uso é natural que os defeitos de maior taxa de incidência sejam encontrados antes e que assim possam ser removidos, tendo um grande efeito sobre a qualidade do produto.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. A aplicação da abordagem <i>cleanroom</i> resulta em software com poucos defeitos.</li> <li>2. Uma pequena fração de erros (menos de 3%) é responsável por aproximadamente dois terços das falhas de software reportadas (COBB e MILLS, 1990).</li> <li>3. Testes estatísticos de uso são mais de 20 vezes mais efetivos para aumentar o tempo médio até falhas do que testes de cobertura (COBB e MILLS, 1990).</li> </ol>
<b>Descrição de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Dados de 15 projetos (na IBM, NASA e em universidades) utilizando <i>cleanroom</i>, onde o número de defeitos por KLOC variou entre 0 e 5,1, apresentando uma média de 3,3.</li> </ol> <p>2 e 3. Referência citada no artigo.</p>
<b>Tipo de Estudo</b>
Relato de Experiência (apresentando <i>Cleanroom</i> e o resultado de seu uso em diversos projetos).

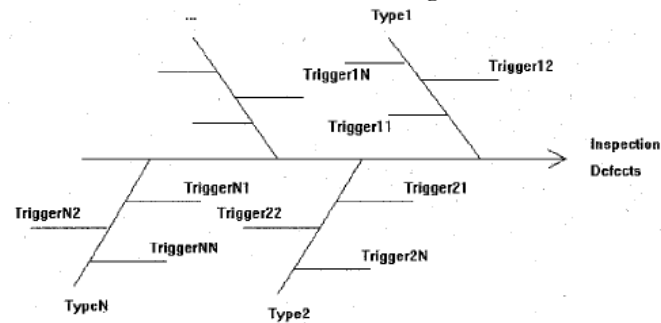
<b>Título do Artigo</b>
Filtering for Quality.
<b>Referência Completa</b>
Troster, J., Henshaw, J., Buss, E., Filtering for Quality, in 'Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research, CASCON 93', IBM Press, pp. 429-449, 1993.
<b>Fonte</b>
ACM, INSPEC.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Filtragem de defeitos.
<b>Descrição do Processo ou Abordagem</b>
<p>O processo de filtragem de defeitos é uma extensão do processo tradicional de prevenção de defeitos de Mays <i>et al</i> (1990) que utiliza engenharia reversa para analisar defeitos de código de software.</p> <p>Filtragem de defeitos envolve a automação de ações corretivas, utilizando um modelo de linguagem e um banco de dados de regras (os filtros) para inspecionar código fonte. Para a aplicação dos filtros uma árvore abstrata de sintaxe (<i>abstract syntax tree</i>) é gerada através da engenharia reversa de código.</p>
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. O processo tradicional de prevenção de defeitos, por focar em defeitos individuais muitas vezes não captura defeitos sistemáticos.</li> <li>2. Filtragem de defeitos é uma abordagem eficiente para prevenir defeitos de código.</li> </ol>
<b>Descrição de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Afirmção do artigo com base na definição do processo.</li> <li>2. A abordagem foi implementada em uma unidade da IBM e no momento da escrita do artigo estava sendo incorporada como parte do processo padrão de desenvolvimento desta unidade.</li> </ol>
<b>Tipo de Estudo</b>
Relato de experiência (no laboratório de Toronto da IBM).

<b>Título do Artigo</b>
Surveys as a Method for Improving the Development Process.
<b>Referência Completa</b>
Wigglesworth, J., Surveys as a Method for Improving the Development Process, in 'CASCON '93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research', IBM Press, pp. 337-355, 1993.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Utilização de <i>surveys</i> para melhoria de processos de software.
<b>Descrição do Processo ou Abordagem</b>
<p>A abordagem sugere a utilização de um <i>survey</i> genérico, GENPHASE, após as diferentes fases de desenvolvimento. Os resultados deste <i>survey</i>, contendo dados quantitativos e qualitativos, são então utilizados para obter defeitos de processo que podem servir de entrada para reuniões de análise causal.</p> <p>Os autores consideram que a abordagem se enquadra no processo tradicional de prevenção de defeitos de Mays <i>et al</i> (1990), podendo ser utilizada como catalisador de defeitos de processo. Assim o foco do processo tradicional, que era somente em defeitos nos produtos de software, seria complementado com defeitos de processo encontrados a partir da análise dos resultados do <i>survey</i>.</p>
<b>Identificação de Conhecimento</b>
1. Os custos de implementar e manter um programa de survey como o da abordagem descrita no artigo é baixo e válido, uma vez que resultados quantitativos podem ser utilizados para identificar tendências em projetos e entre projetos, e resultados qualitativos podem ser utilizados como entrada em um programa de análise causal.
<b>Descrição de Conhecimento</b>
Conhecimento obtido com base na experiência da utilização piloto da abordagem em quatro projetos na IBM.
<b>Tipo de Estudo</b>
Estudo de Caso (na IBM).

<b>Título do artigo</b>
A statistical Approach to the Inspection Checklist Formal Synthesis and Improvement.
<b>Referência Completa</b>
Chernak, Y., A statistical Approach to the Inspection Checklist Formal Synthesis and Improvement, IEEE Transactions on Software Engineering, 22(12), 866-874, 1996.
<b>Fonte</b>
IEEE, INSPEC.
<b>Esquema de Classificação de Defeitos</b>
ODC (orthogonal defect classification) (Chillarege <i>et al.</i> , 1992). Os tipos de defeito são determinados de acordo com a organização.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Abordagem baseada em análise causal para produzir e melhorar <i>checklists</i> .
<b>Descrição do Processo ou Abordagem</b>
<p>De acordo com a abordagem <i>checklists</i> devem possuir dois componentes: o atributo “Where to look” e o atributo “How to detect”. Tendo definida esta estrutura para <i>checklists</i>, o artigo mostra como ODC (<i>orthogonal defect classification</i>) da IBM pode ser utilizado para modelar defeitos de inspeções e como os itens do <i>checklist</i> podem ser identificados através de uma análise de regressão de dados estatísticos sobre defeitos. O tipo de defeito da abordagem ODC foi mapeado para o componente “Where to look” e o gatilho do defeito (<i>defect trigger</i>) foi mapeado para o componente “How to detect”.</p> <p>O método para identificar os itens do <i>checklist</i> envolve os cinco seguintes passos:</p> <ol style="list-style-type: none"> <li>1. Selecionar dados. Os dados devem ser referentes ao tipo de inspeção para o qual o <i>checklist</i> será produzido.</li> <li>2. Fazer suposição referente aos tipos de defeito encontrados no tipo de inspeção em questão. Para isto é sugerido o uso de um diagrama de Ishikawa, conforme ilustrado a seguir.</li> </ol>  <ol style="list-style-type: none"> <li>3. Avaliar as suposições estatisticamente. Para isto é sugerido utilizar análise (e gráfico) de Pareto, assegurando que os tipos de defeito efetivamente responsáveis pela maioria de defeitos estejam sendo utilizados.</li> </ol>



4. Fazer suposição sobre gatilhos dos defeitos. Novamente um diagrama de Ishikawa é sugerido, resultando em um modelo de causa-efeito de tipos de defeitos e gatilhos correspondentes, conforme ilustrado a seguir.



5. Avaliar as suposições estatisticamente. Novamente é sugerido utilizar análise (e gráfico) de Pareto, assegurando que os gatilhos identificados são os que revelam o tipo de defeito com maior frequência.

#### **Identificação de Conhecimento**

1. A técnica é aplicável e permite aprimorar *checklists* de acordo com os mais recentes dados de projeto.

#### **Descrição de Conhecimento**

Conhecimento obtido a partir de resultados de um estudo de caso envolvendo um projeto real de um aplicativo financeiro de larga escala.

#### **Tipo de Estudo**

Estudo de caso.

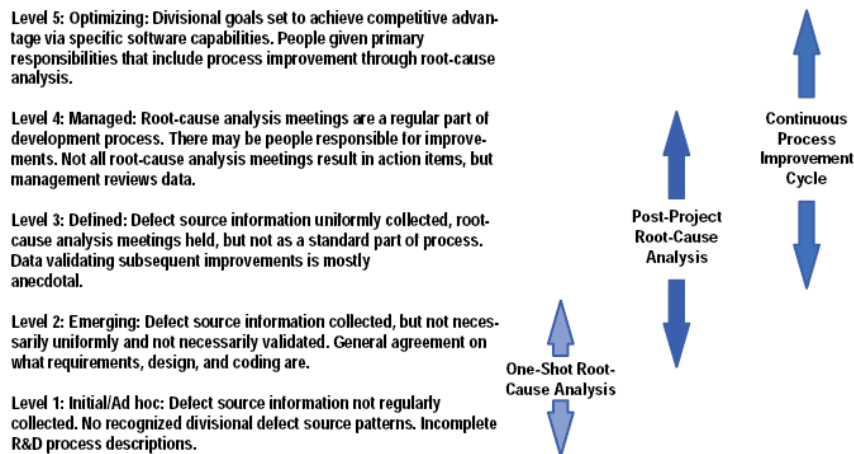
<b>Título do Artigo</b>
Process Improvement Through Root Cause Analysis.
<b>Referência Completa</b>
Damele, G., Bazzana, G., Andreis, F., Aquilio, S., Process Improvement through Root Cause Analysis, in 'Proceedings of Third International Conference on Achieving Quality in Software', pp. 35-47, 1996.
<b>Fonte</b>
INSPEC.
<b>Esquema de Classificação de Defeitos</b>
Não explicitado.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não explicitado.
<b>Identificação do Processo ou Abordagem</b>
Implantação de análise causal de falhas de software e de atrasos de cronograma na Italtel SIT BUCT Línea UT. Similar ao processo de prevenção de defeitos (Jones, 1985).
<b>Descrição do Processo ou Abordagem</b>
Abordagem congruente com a apresentada em (Card, 1993), envolvendo a identificação de causas para a introdução de defeitos e para a não detecção de defeitos, utilizando diagramas de Ishikawa.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Do ponto de vista técnico a análise causal de defeitos contribuiu para um melhor entendimento das práticas de desenvolvimento e para a adoção de diversas ações corretivas.</li> <li>2. Do ponto de vista metodológico, a análise causal de defeitos se mostrou poderosa em destacar oportunidades de melhoria.</li> </ol>
<b>Descrição de Conhecimento</b>
Baseado em dados quantitativos coletados na Italtel SIT BUCT Línea UT, coletados seguindo uma abordagem PDCA (Plan-Do-Check-Act).
<b>Tipo de Estudo</b>
Relato de Experiência (Italtel SIT BUCT Línea UT).

<b>Título do Artigo</b>
Software Failure Analysis for High-Return Process Improvement Decisions.
<b>Referência Completa</b>
Grady, R. B., Software Failure Analysis for High-Return Process Improvement Decisions, Hewlett-Packard Journal, 47 (4), 15 - 24, 1996.
<b>Fonte</b>
INSPEC, EI Compendex.
<b>Esquema de Classificação de Defeitos</b>
<p>Modelo de classificação próprio da Hewlett-Packard (Grady, 1992). Registrando para cada defeito sua origem, seu tipo e o seu modo, de acordo com a figura a seguir, extraída do artigo.</p> <pre> graph TD     subgraph Origin [Origin (Where?)]         S[Specifications/Requirements]         D[Design]         C[Code]         ES[Environmental Support]         Doc[Documentation]         O[Other]     end      subgraph Type [Type (What?)]         RS[Requirements or Specifications Functionality Other]         HI[Hardware Interface Software Interface User Interface Functional Description Other]         PC[Process (Interprocess) Communications Data Definition Module Design Logic Description Error Checking Standards Other]         L[Logic Computation Data Handling Module Interface/Implementation Standards Other]         TS[Test Software Test Hardware Development Tools Integration Software Other]     end      subgraph Mode [Mode (Why?)]         M[Missing]         U[Unclear]         W[Wrong]         Ch[Changed]         BW[Better Way]     end      S --- RS     S --- F[Functionality]     S --- O1[Other]     D --- HI     D --- SI[Software Interface]     D --- UI[User Interface]     D --- FD[Functional Description]     D --- O2[Other]     C --- PC     C --- DD[Data Definition]     C --- MD[Module Design]     C --- LD[Logic Description]     C --- EC[Error Checking]     C --- S1[Standards]     C --- O3[Other]     ES --- L     ES --- C1[Computation]     ES --- DH[Data Handling]     ES --- MII[Module Interface/Implementation]     ES --- S2[Standards]     ES --- O4[Other]     Doc --- TS     Doc --- TH[Test Hardware]     Doc --- DT[Development Tools]     Doc --- IS[Integration Software]     Doc --- O5[Other]     O --- TS     O --- TH     O --- DT     O --- IS     O --- O6[Other]      RS --- M     RS --- U     RS --- W     RS --- Ch     RS --- BW     HI --- M     HI --- U     HI --- W     HI --- Ch     HI --- BW     PC --- M     PC --- U     PC --- W     PC --- Ch     PC --- BW     L --- M     L --- U     L --- W     L --- Ch     L --- BW     TS --- M     TS --- U     TS --- W     TS --- Ch     TS --- BW </pre>
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem, sugere que causas identificadas sejam agrupadas em áreas relacionadas, criando assim um diagrama de afinidades (affinity diagram). Os diagramas de afinidades formam então a base para os diagramas de causa e efeito (Ishikawa, 1976).
<b>Identificação do Processo ou Abordagem</b>
São descritas três abordagens: análise causal avulsa ( <i>one-shot root cause analysis</i> ), análise causal após projeto ( <i>post-project root cause analysis</i> ) e ciclo de melhoria contínua de processos ( <i>continuous process improvement cycle</i> ).
<b>Descrição do Processo ou Abordagem</b>
<ul style="list-style-type: none"> <li>Análise causal avulsa. Envolve uma reunião de análise causal (o artigo identifica passos para tal reunião) e a recomendação de melhorias a serem entregues à gerência para que possam ser atribuídas.</li> </ul>

- Análise causal após projeto. Ocorre da mesma forma que a análise causal avulsa, mas já tendo coletado dados de análise causal de defeitos anteriormente. Organizações que já medem seus dados de defeitos e tem um entendimento de seus padrões de defeitos atuam sobre seus dados de forma mais eficiente.
- Ciclo de melhoria contínua de processos. Envolve o uso de análise causal como instrumento para a melhoria contínua de processos. As análises causais podem ser iniciadas ao fim de atividades de desenvolvimento, após uma série de inspeções ou num tempo período de tempo arbitrário.

O autor menciona um aumento na maturidade das abordagens de análise causal e

cria um modelo de maturidade para análise causal, conforme ilustrado na figura a seguir.



Como técnica específica para identificar causas sugere o uso de diagramas de causa e efeito (Ishikawa, 1976). As amostras de defeitos são selecionadas informalmente.

Identificação de Conhecimento
<ol style="list-style-type: none"> <li>1. Análise causal de defeitos traz retornos quase imediatos, particularmente em visualizar o progresso.</li> <li>2. Não é difícil institucionalizar o uso de um modelo de defeitos e do processo de análise causal.</li> <li>3. O uso de análise causal de defeitos é um meio de obter um balanceamento ideal entre reagir a defeitos e prevenir a ocorrência de defeitos no futuro.</li> <li>4. Migrar para níveis de maturidade de análise causal de defeitos mais altos (veja sessão anterior), ajuda a: <ol style="list-style-type: none"> <li>a. Aumento da probabilidade de sucesso ao implementar mudanças no processo.</li> <li>b. Identificação acelerada das melhores práticas.</li> <li>c. Aumento dos retornos potenciais.</li> </ol> </li> </ol>
Descrição de Conhecimento
<p>Experiência obtida através do uso de análise causal de defeitos de software em divisões da Hewlett-Packard. Uma divisão em que defeitos de interface eram muito frequentes conseguiu através da análise causal reduzir o número de defeitos de interface de 20% para 5% do total de defeitos do software.</p>
Tipo de Estudo
<p>Relato de experiência.</p>

Título do Artigo																																				
Integrating a Defect Typology With Containment Metrics.																																				
Referência Completa																																				
Kelsey, R.B., Integrating a Defect Typology With Containment Metrics, ACM SIGSOFT Software Engineering Notes, 22(2), 64-67, 1997.																																				
Fonte																																				
ACM, INSPEC.																																				
Esquema de Classificação de Defeitos																																				
<p>Sugere registrar além do tipo de defeito o momento em que eles foram encontrados, permitindo derivar métricas de contenção de fase (phase containment metrics).</p> <p>Em relação aos tipos de defeitos, uma tipologia de defeitos (<i>defect typology</i>), derivada da tipologia normalmente utilizada em <i>checklists</i> de inspeção (não referenciada no artigo). A nova tipologia é genérica de modo que possa ser aplicada a todas as fases de desenvolvimento. A tipologia segue.</p> <p><b>Injection Phase Types:</b></p> <table><tr><td>CRQ</td><td>customer requirements were inadequately or incorrectly specified</td></tr><tr><td>FRQ</td><td>functional requirements were inadequately or incorrectly specified</td></tr><tr><td>DRQ</td><td>design requirements were inadequately or incorrectly specified</td></tr><tr><td>CMD</td><td>linguistic or algorithmic defects were introduced during coding or module development</td></tr><tr><td>TS</td><td>test specifications &amp; plans were inadequate or incorrect</td></tr><tr><td>COM</td><td>the human or document interfaces between phases of cycle were inadequate</td></tr></table> <p><b>Operational Context Types:</b></p> <table><tr><td>OP</td><td>a defect occurs in a user-initiated operation</td></tr><tr><td>FN</td><td>a defect occurs in a function (discrete code path) supporting an operation</td></tr><tr><td>UI</td><td>a defect occurs in the user interface</td></tr><tr><td>IN</td><td>a defect occurs in the installation of the product</td></tr><tr><td>UE</td><td>a defect is dependent upon the user environment</td></tr><tr><td>SC</td><td>the product does not scale to some specific environment</td></tr><tr><td>SCM</td><td>a defect occurs due to systems incompatibility</td></tr></table> <p><b>Location Types:</b></p> <table><tr><td>IN</td><td>a defect is located in internal code interfaces</td></tr><tr><td>EN</td><td>a defect is located in external code interfaces</td></tr><tr><td>EH</td><td>a defect is located in error/exception/recovery handling logic</td></tr><tr><td>RM</td><td>a defect is located in resource (internal or external) use</td></tr><tr><td>DI</td><td>a defect is located in data/structure initialization or use</td></tr></table> <p>De acordo com o autor a tipologia categoriza tanto sintomas de um defeito como suas causas.</p>	CRQ	customer requirements were inadequately or incorrectly specified	FRQ	functional requirements were inadequately or incorrectly specified	DRQ	design requirements were inadequately or incorrectly specified	CMD	linguistic or algorithmic defects were introduced during coding or module development	TS	test specifications & plans were inadequate or incorrect	COM	the human or document interfaces between phases of cycle were inadequate	OP	a defect occurs in a user-initiated operation	FN	a defect occurs in a function (discrete code path) supporting an operation	UI	a defect occurs in the user interface	IN	a defect occurs in the installation of the product	UE	a defect is dependent upon the user environment	SC	the product does not scale to some specific environment	SCM	a defect occurs due to systems incompatibility	IN	a defect is located in internal code interfaces	EN	a defect is located in external code interfaces	EH	a defect is located in error/exception/recovery handling logic	RM	a defect is located in resource (internal or external) use	DI	a defect is located in data/structure initialization or use
CRQ	customer requirements were inadequately or incorrectly specified																																			
FRQ	functional requirements were inadequately or incorrectly specified																																			
DRQ	design requirements were inadequately or incorrectly specified																																			
CMD	linguistic or algorithmic defects were introduced during coding or module development																																			
TS	test specifications & plans were inadequate or incorrect																																			
COM	the human or document interfaces between phases of cycle were inadequate																																			
OP	a defect occurs in a user-initiated operation																																			
FN	a defect occurs in a function (discrete code path) supporting an operation																																			
UI	a defect occurs in the user interface																																			
IN	a defect occurs in the installation of the product																																			
UE	a defect is dependent upon the user environment																																			
SC	the product does not scale to some specific environment																																			
SCM	a defect occurs due to systems incompatibility																																			
IN	a defect is located in internal code interfaces																																			
EN	a defect is located in external code interfaces																																			
EH	a defect is located in error/exception/recovery handling logic																																			
RM	a defect is located in resource (internal or external) use																																			
DI	a defect is located in data/structure initialization or use																																			
Esquema de Classificação de Causas de Defeitos																																				
Não tem.																																				
Identificação do Processo ou Abordagem																																				
Integração da tipologia de defeitos com métricas de contenção de fase para apoiar análise causal de defeitos.																																				
Descrição do Processo ou Abordagem																																				
A abordagem envolve integrar a tipologia com métricas de contenção das fases de																																				

desenvolvimento, obtendo métricas como as descritas na tabela a seguir.

Metric	Compared With	Goal
Number of defects found during the current cycle {per type} injected previously	Number of defects found during the current cycle {per type} injected this cycle	Determine effectiveness of current vs. previous containment procedures, establish comparative injection rates
Number of defects found during the current cycle {per type} injected this cycle	Number of defects found after release {per type}	Determine effectiveness of current containment procedures, especially for configuration dependent errors.
Number of defects found by code inspection {per type}	Number of defects found by internal test facilities {per type}	Determine effectiveness of inspections; refocus if necessary
Number of defects found by internal test facilities {per type}	Number of defects found by external customers {per type}	Determine suitability and coverage of test types and test code
Number of defects found by external test customers {per type}		Determine effectiveness of all containment processes; watch for incidence rates in scaling or environmental areas.
Number of defects found after release {per type}.	Number of defects found during the current cycle {per type} injected previously, and Number of defects found during the current cycle {per type} injected this cycle	Determine effectiveness of current containment procedures (especially for SC and SCM defects); establish drift or inadequacy of requirements (CRQ, FRQ, OP, FN, IN, EH)

De acordo com o autor esta combinação torna possível determinar aonde um defeito ocorreu, porque foi introduzido e quando foi detectado. A sugestão final do autor é que, seja qual for a tipologia e qual for o conjunto de métricas de contenção o objetivo deve sempre ser apoiar a análise causal de defeitos de ambos, produto e processo.

#### Identificação de Conhecimento

1. A combinação de uma taxonomia com métricas de contenção apoia em determinar aonde um defeito ocorreu, porque foi introduzido e quando foi detectado.

#### Descrição de Conhecimento

Argumentação do autor, discorrida através de um conjunto de exemplos de análise causal utilizando a combinação.

#### Tipo de Estudo

Teórico.

<b>Título do Artigo</b>
Implementing Cleanroom Software Engineering into a Mature CMM-Based Software Organization.
<b>Referência Completa</b>
Oshana, R., Coyle, F.P., Implementing Cleanroom Software Engineering into a Mature CMM-Based Software Organization, Proceedings of the 19th International Conference on Software Engineering, ICSE'97, ACM Press, New York, NY, USA, pp. 572-573, 1997.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Não tem.
<b>Descrição do Processo ou Abordagem</b>
Não tem.
<b>Identificação de Conhecimento</b>
1. Cleanroom se encaixou naturalmente na área de análise causal do CMM, uma vez que engloba mecanismos de <i>feedback</i> a partir de resultados dos testes estatísticos de usabilidade (e da métrica <i>mean time to failure</i> ).
<b>Descrição de Conhecimento</b>
Relato de experiência da introdução da abordagem <i>cleanroom</i> em uma organização de software de alta maturidade CMM (que possui processo de análise causal definido).
<b>Tipo de Estudo</b>
Relato de experiência (na <i>Texas Instruments</i> ).

<b>Título do Artigo</b>
Plunge DA: A Case Study
<b>Referência Completa</b>
Rosen, C., PLUNGE DA: A Case Study, SIGSOFT Softw. Eng. Notes 22(2), pp. 82-83, 1997.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Reunião informal.
<b>Descrição do Processo ou Abordagem</b>
Reunião informal.
<b>Identificação de Conhecimento</b>
Nenhum. O artigo trata de um estudo de caso referente a uma reunião fictícia de análise causal a respeito de quedas no sistema originadas pelo <i>bug</i> do milênio.
<b>Descrição de Conhecimento</b>
Não se aplica.
<b>Tipo de Estudo</b>
Teórico. (Estudo de caso fictício).



<b>Título do Artigo</b>
Learning from our Mistakes with Defect Causal Analysis.
<b>Referência Completa</b>
Card, D.N., Learning from our Mistakes with Defect Causal Analysis, IEEE Software, 15(1), 56 – 63, 1998.
<b>Fonte</b>
IEEE, INSPEC, EI Compendex.
<b>Esquema de Classificação de Defeitos</b>
O artigo menciona três dimensões úteis para classificar defeitos: momento de introdução, momento de detecção e tipo de defeito. Cita como tipos de defeitos típicos os presentes no esquema ODC: interface, computacional, lógico, de inicialização, estrutura de dados e documentação. Menciona que os tipos de defeito devem ser ajustados para atender às necessidades da organização.
<b>Esquema de Classificação de Causas de Defeitos</b>
São citadas as quatro categorias de causas descritas em (Ishikawa, 1976): (1) métodos, (2) ferramentas e ambiente, (3) pessoas e (4) entradas e requisitos. Novamente as categorias devem ser ajustadas para atender às necessidades da organização.
<b>Identificação do Processo ou Abordagem</b>
Processo de análise causal de defeitos (compatível com o processo de prevenção de defeitos de Mays (1990)).
<b>Descrição do Processo ou Abordagem</b>
<p>Visão simplificada do processo tradicional de prevenção de defeitos descrito em (Mays, 1990), refletindo ajustes para sua introdução na <i>Computer Sciences Corporation</i> (CSC). Possui foco somente nas atividades de análise causal e da equipe de ação.</p> <p>Sugere o uso de tabelas ou gráficos de Pareto para identificar defeitos sistemáticos e (se necessário) de diagramas de Ishikawa para identificar as causas associadas aos defeitos sistemáticos. Como resultado da reunião de análise causal sugere um relatório contendo as seguintes informações: identificação da reunião, descrição dos defeitos sistemáticos, lista de defeitos associados a cada defeito sistemático e ações propostas.</p>
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Análise causal de defeitos difere de outros métodos de avaliação de processos por apontar ações específicas ao invés de sugerir amplas áreas que requerem a atenção da equipe de melhoria de processos.</li> <li>2. De acordo com o artigo três princípios guiam as abordagens de análise causal de defeitos para a melhoria da qualidade: <ol style="list-style-type: none"> <li>a. Reduzir defeitos para aumentar a qualidade;</li> </ol> </li> </ol>

<ul style="list-style-type: none"> <li>b. Aplicar expertise local;</li> <li>c. Focar em erros sistemáticos.</li> </ul> <p>3. Os pré-requisitos para se aplicar análise causal de defeitos são:</p> <ul style="list-style-type: none"> <li>a. Defeitos existem;</li> <li>b. Defeitos foram documentados;</li> <li>c. Existe vontade de prevenir a ocorrência dos defeitos em projetos futuros;</li> <li>d. Um processo de software básico precisa estar definido para prover o arcabouço para as ações que devem ser implementadas.</li> </ul> <p>4. A análise de mais de 20 defeitos para análise em uma única sessão não é recomendada. O moderador deve selecionar a amostra de defeitos a serem analisados antes da reunião.</p> <p>5. Apoio da gerência e respeito pelas habilidades técnicas dos desenvolvedores de software são importantes para o sucesso da implementação de análise causal de defeitos.</p> <p>6. Taxas de ocorrência de defeitos são reduzidas em média em 50%.</p> <p>7. O custo da implementação de análise causal de defeitos varia de 0,5% a 1,5% do investimento do projeto.</p> <p>8. Embora análise causal de defeitos seja exigida apenas nos níveis mais altos de maturidade dos modelos de qualidade, ela pode ajudar a melhorar os processos de organizações bem antes que estes níveis sejam atingidos.</p>
<b>Descrição de Conhecimento</b>
<p>1 a 5. Afirmações do artigo baseadas na estrutura do processo.</p> <p>6. Variação das taxas observada na implementação de análise causal de defeitos na IBM e na CSC. Adicionalmente, na CSC defeitos severos desapareceram do projeto em que análise causal foi implementada e o projeto recebeu prêmio de seu cliente pela qualidade do software entregue.</p> <p>7. Custos observados respectivamente na implementação de análise causal de defeitos na IBM e na CSC. Na IBM as equipes dos projetos onde análise causal de defeitos foi introduzida contavam com centenas de participantes e na CSC com dezenas.</p> <p>8. Afirmção do artigo.</p>
<b>Tipo de Estudo</b>
Relato de Experiência (na <i>Computer Sciences Corporation</i> ).

<b>Título do Artigo</b>
A Causal Model for Software Cost Estimating Error.
<b>Referência Completa</b>
Lederer, A.L., Prasad, J., A Causal Model for Software Cost Estimating Error, IEEE Transactions on Software Engineering, 24 (2), 137-148, 1998.
<b>Fonte</b>
IEEE, INSPEC, EI Compendex, Web of Science.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Metodologia para elaboração de um mapa causal para estimar o erro em estimativas de custo de desenvolvimento de software.
<b>Descrição do Processo ou Abordagem</b>
A abordagem envolveu a elaboração de um mapa causal hipotético inicial refinado posteriormente com base em dados empíricos e interpretação estatística para obter um modelo melhorado. A partir dos fatores do mapa causal inicial um questionário foi elaborado e respostas para o mesmo foram obtidas de um conjunto de 112 gerentes e analistas de sistemas. O modelo foi então estimado utilizando o sistema EQS ( <i>Equations Modeling System</i> ) e o teste estatístico Satorra-Bentler. Foram acrescentados ao modelo original os fatores e arestas representando relações estatisticamente relevantes.
<b>Identificação de Conhecimento</b>
Embora a abordagem de construção dos mapas causais possa ser relevante para a pesquisa em questão, isto não se aplica aos resultados do artigo, sobre estimativa de custos no desenvolvimento de software.
<b>Descrição de Conhecimento</b>
Não se aplica.
<b>Tipo de Estudo</b>
Estudo de caso.

<b>Título do Artigo</b>
Managing by the Numbers: a tutorial on quantitative measurement and control of software projects.
<b>Referência Completa</b>
Fairley, R. E., Managing by the Numbers: a tutorial on quantitative measurement and control of software projects, <i>in</i> 'Proceedings of the 21st International Conference on Software Engineering, ICSE 99', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 677- 678, 1999.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
O artigo apenas menciona que informações como a natureza do defeito, o momento de inserção, o momento e método de detecção e a quantidade de retrabalho envolvidos apóiam análise causal, provendo a base para melhoria de processos.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem. O artigo menciona apenas que ações corretivas para desvios devem ser documentadas em itens de ação.
<b>Identificação do Processo ou Abordagem</b>
Não tem abordagem própria de análise causal de defeitos. Trata-se do resumo de um tutorial que apresenta uma visão geral de mecanismos para gerência quantitativa de projetos de software.
<b>Descrição do Processo ou Abordagem</b>
Não se aplica.
<b>Identificação de Conhecimento</b>
Não tem.
<b>Descrição de Conhecimento</b>
Não se aplica.
<b>Tipo de Estudo</b>
Teórico.

<b>Título do artigo</b>
A Statistical Method for Controlling Software Defect Detection Process.
<b>Referência Completa</b>
Hong, G., Xie, M., Shanmugan, P., A Statistical Method for Controlling Software Defect Detection Process, Computers and Industrial Engineering 37( 1-2), 137 – 140, 1999.
<b>Fonte</b>
EI Compendex, Web of Science.
<b>Esquema de Classificação de Defeitos</b>
Utiliza, mas não tem no artigo nem é referenciado. Apenas cita como exemplos de categorias erros de documentação, violação de padrões e projeto mal codificado, juntos responsáveis por 65% dos defeitos em dados de 32 inspeções no <i>Motorola Singapore Software Center</i> .
<b>Esquema de Classificação de Causas de Defeitos</b>
Utiliza, mas não tem no artigo. Apenas cita como exemplos descuido, falta de conhecimento do domínio e projeto não suficientemente detalhado. Representando respectivamente 33%, 22% e 16% das causas nos dados de 32 inspeções no <i>Motorola Singapore Software Center</i> .
<b>Identificação do Processo ou Abordagem</b>
Abordagem que junta controle estatístico de processos com análise causal.
<b>Descrição do Processo ou Abordagem</b>
Em uma análise de dados provenientes de 32 inspeções no <i>Motorola Singapore Software Center</i> o artigo mostra a aplicação da abordagem da seguinte forma: <ul style="list-style-type: none"> <li>• Utilizando o gráfico U, projetado para tratar de defeitos detectados por unidade inspecionada ou testada, veja Montgomery (1991). Assim, o controle estatístico foi feito pela densidade de defeitos considerando os limites do gráfico U.</li> <li>• Nas inspeções em que a densidade de defeitos se encontrava fora dos limites do gráfico U um gráfico de Pareto foi utilizado para identificar as categorias de defeitos mais comuns.</li> <li>• Para cada categoria de defeitos uma análise causal foi conduzida para encontrar as causas mais comuns dos defeitos, visando evitar sua ocorrência no futuro.</li> </ul>
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Mesmo em um único ciclo de produção de software, muitos eventos são numerosos e repetitivos, oferecendo oportunidades para melhoria de processos com ganhos imediatos para o ciclo em questão.</li> <li>2. Os pontos chave para o uso de técnicas de gerência quantitativa de processos de software são comprometimento e persistência.</li> </ol>
<b>Descrição de Conhecimento</b>

Observações no <i>Motorola Singapore Software Center</i> .
<b>Tipo de Estudo</b>
Estudo de caso (utilizando dados reais de 32 inspeções de software no <i>Motorola Singapore Software Center</i> ).

Título do Artigo																													
A Case Study in Root Cause Defect Analysis.																													
Referência Completa																													
Leszak, M., Perry, D., Stoll, D., A Case Study in Root Cause Defect Analysis, in 'International Conference on Software Engineering, ICSE 2000', pp. 428-437, 2000.																													
Fonte																													
ACM, IEEE, INSPEC, EI Compendex.																													
Esquema de Classificação de Defeitos																													
<p>As seguintes informações são armazenadas para cada defeito: atividade de detecção, localização do defeito, causas do defeito, classificação do defeito e análise de barreira (como detectar mais cedo ou prevenir a ocorrência do defeito).</p> <p>A classificação de defeitos é própria, dividindo os defeitos em três classes: implementação, interface e externos. Cada uma destas classes é então subdividida nos tipos de defeitos ilustrados na tabela a seguir.</p> <table> <tr> <th>Implementation</th><th>Interface</th><th>External</th></tr> <tr> <td>1. Data design/usage</td><td>9. Data design/usage</td><td>16. Development environment</td></tr> <tr> <td>2. Resource allocation/usage</td><td>10. Functionality design/usage</td><td>17. Test environment (tools/infrastructure)</td></tr> <tr> <td>3. Exception handling</td><td>11. Communication protocol</td><td>18. Test environment (test cases/suites)</td></tr> <tr> <td>4. Algorithm</td><td>12. Process coordination</td><td>19. Concurrent work (other releases)</td></tr> <tr> <td>5. Functionality</td><td>13. Unexpected interactions</td><td>20. Previous (inherited from previous release)</td></tr> <tr> <td>6. Performance</td><td>14. Change coordination</td><td>21. Other</td></tr> <tr> <td>7. Language pitfalls</td><td>15. Other</td><td></td></tr> <tr> <td>8. Other</td><td></td><td></td></tr> </table> <p>A classificação conta ainda com: (1) a natureza do defeito: incorreto, incompleto e outra; (2) a redefinição da severidade do defeito (se necessário) e (3) o esforço para reproduzir, investigar e corrigir o defeito.</p> <p>Para o esforço uma escala uniforme foi utilizada: zero, menos de um dia, um a cinco dias, cinco a vinte dias e mais de vinte dias.</p>			Implementation	Interface	External	1. Data design/usage	9. Data design/usage	16. Development environment	2. Resource allocation/usage	10. Functionality design/usage	17. Test environment (tools/infrastructure)	3. Exception handling	11. Communication protocol	18. Test environment (test cases/suites)	4. Algorithm	12. Process coordination	19. Concurrent work (other releases)	5. Functionality	13. Unexpected interactions	20. Previous (inherited from previous release)	6. Performance	14. Change coordination	21. Other	7. Language pitfalls	15. Other		8. Other		
Implementation	Interface	External																											
1. Data design/usage	9. Data design/usage	16. Development environment																											
2. Resource allocation/usage	10. Functionality design/usage	17. Test environment (tools/infrastructure)																											
3. Exception handling	11. Communication protocol	18. Test environment (test cases/suites)																											
4. Algorithm	12. Process coordination	19. Concurrent work (other releases)																											
5. Functionality	13. Unexpected interactions	20. Previous (inherited from previous release)																											
6. Performance	14. Change coordination	21. Other																											
7. Language pitfalls	15. Other																												
8. Other																													
Esquema de Classificação de Causas de Defeitos																													
<p>As causas são classificadas de acordo com quatro classes, formando um espaço de causas de quatro dimensões:</p> <ul style="list-style-type: none"> <li>• Causas de atividade. São as atividades ou documentos em que o defeito foi introduzido. Além da atividade em si as causas de atividade podem ser qualificadas pela natureza da causa como incorreta, incompleta, ambígua, modificada, não atendendo às necessidades do usuário e não aplicável.</li> <li>• Causas relacionadas a fatores humanos. São elas: troca da coordenação, falta de conhecimento do domínio, falta de conhecimento do sistema, falta de conhecimento sobre ferramentas, falta de conhecimento sobre o processo, erro individual, introduzido na remoção de outro defeito, problema de comunicação, falta de percepção sobre a necessidade de documentação e não aplicável.</li> <li>• Causas de projeto. São elas: pressão de tempo, erro da gerência, causado por outro</li> </ul>																													

<p>produto e não aplicável.</p> <ul style="list-style-type: none"> <li>• Causas de revisão. São elas: revisão incompleta (ou não realizada), não houve preparação suficiente, participação não adequada e não aplicável.</li> <li>• Outras causas.</li> </ul>
<p><b>Identificação do Processo ou Abordagem</b></p>
<p>Metodologia própria baseada no CMM e similar à descrita por Endres (1975). O processo seguido, de acordo com os autores, é ainda congruente com o processo genérico apresentado em (Card, 1998).</p>
<p><b>Descrição do Processo ou Abordagem</b></p>
<p>A abordagem utilizada no estudo de caso envolveu quatro passos:</p> <ul style="list-style-type: none"> <li>• Seleção dos problemas mais relevantes. O objetivo deste passo foi identificar um critério de seleção que filtrasse os problemas que juntos possuem uma parte significativa de esforço de retrabalho (normalmente os que são encontrados tarde no processo de desenvolvimento). Também foram analisados os problemas reportados após a entrega do produto para verificar se as características destes problemas diferem das dos demais.</li> <li>• Priorização de medidas de compensação (<i>countermeasures</i>). Um <i>brainstorm</i> sobre propostas de medidas de compensação foi realizado. Em seguida estas foram priorizadas. Para a priorização a equipe utilizou as seguintes entradas: <ul style="list-style-type: none"> <li>o Potencial de economia da medida. Esforço máximo que pode ser economizado ao evitar os problemas desta classe de causa.</li> <li>o Eficiência da medida. Percentual estimado de problemas desta classe de causa que serão influenciados pela medida.</li> <li>o Custo da medida. Normalizado através de uma escala de 0.2 (custo maior do que 600.000,00 dólares) a 1 (custo menor do que 6.000,00 dólares).</li> </ul> <p>O produto destas entradas resultou então em um ranking inicial, a partir desta a priorização final foi obtida de forma informal pela equipe.</p> </li> <li>• Definição das ações de melhoria. Um workshop de dois dias foi conduzido para derivar as ações de melhoria para as medidas de compensação priorizadas no passo anterior.</li> <li>• Implantação das ações de melhoria. As ações foram aprovadas e implementadas. É importante ressaltar que a análise causal no estudo de caso em questão foi aplicada somente como atividade <i>post-mortem</i> e que os autores pretendem fazer com que a abordagem seja aplicada durante os projetos de desenvolvimento.</li> </ul>
<p><b>Identificação de Conhecimento</b></p>
<p>Nos resultados do estudo de caso observou-se que:</p> <ol style="list-style-type: none"> <li>1. Os custos de correção de defeitos não cresceram exponencialmente, tendendo mais a um crescimento linear.</li> <li>2. A maioria dos defeitos não foi originada nas fases iniciais do desenvolvimento.</li> <li>3. Dentro de um mesmo projeto, os diferentes subsistemas mostraram caracterizações de defeitos bastante distintas.</li> <li>4. Há uma significativa influência de fatores humanos na introdução de defeitos.</li> <li>5. Análise causal de defeitos possui esforço baixo e tolerável em relação aos seus aparentes benefícios.</li> </ol>

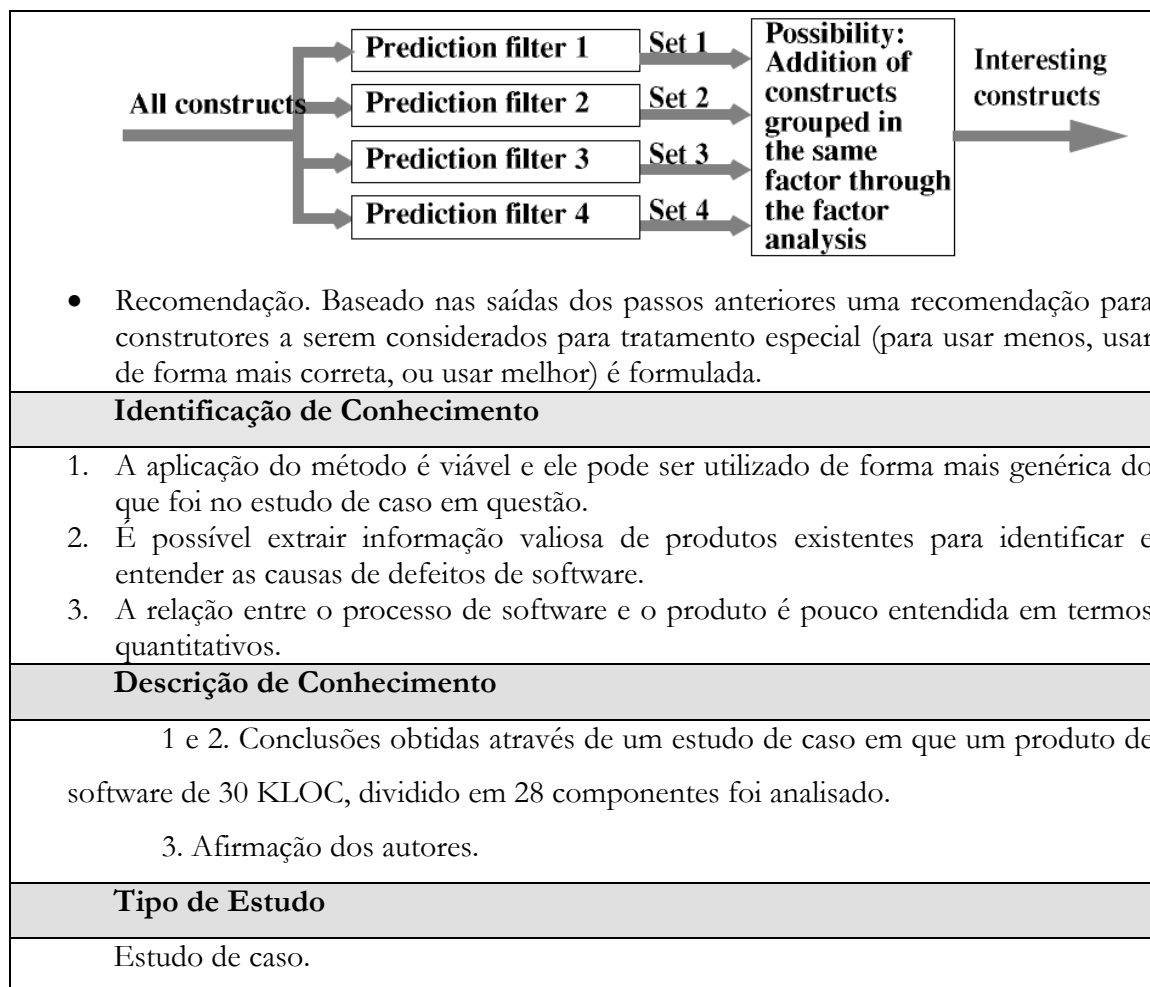


<b>Descrição de Conhecimento</b>
No estudo de caso uma análise causal de defeitos foi conduzida ao final de um projeto real de desenvolvimento de um produto na Lucent Technologies. O projeto foi considerado crítico e envolveu 180 pessoas durante 19 meses.
<b>Tipo de Estudo</b>
Estudo de caso.

<b>Título do Artigo</b>
Defect-based Reliability Analysis for Mission-Critical Software.
<b>Referência Completa</b>
Paul, R. A., Bastani, F.; Yen, I., Challagulla, V. U., Defect-based Reliability Analysis for Mission-Critical Software, Proceedings of the IEEE Computer Society's International Computer Software and Applications Conference, pp. 439 - 444, 2000.
<b>Fonte</b>
EI Compendex.
<b>Esquema de Classificação de Defeitos</b>
Não tem. O artigo cita ODC ( <i>orthogonal defect classification</i> ) (Chillarege <i>et al</i> , 1992), mas a abordagem apresentada pode ser aplicada somente a um tipo de defeito por vez.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Não é uma abordagem de análise causal de defeitos. A abordagem é para determinar a confiabilidade de um software em relação a um tipo específico de defeito.
<b>Descrição do Processo ou Abordagem</b>
<p>A abordagem considera um tipo específico de defeito. Dados de projetos de software são coletados e mapeados para atributos de interesse pré-definidos para aquele tipo de defeito. O projeto em questão é então comparado aos dados de projetos coletados que melhor se adequam aos do próprio projeto, utilizando regras de partição obtidos dos dados existentes. Em seguida uma análise de regressão é realizada nos dados a serem comparados para prever o número de defeitos daquele tipo remanescentes no software.</p> <p>A abordagem para identificar os dados a serem utilizados para a análise de regressão é similar ao método <i>memory based reasoning</i> (Stanfill e Waltz, 1986).</p>
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. A abordagem utiliza hipóteses geradas pela modelagem do tipo de defeito para identificar as partições, assim conhecimento sobre os fatores que interferem na ocorrência de diferentes tipos de defeitos precisa ser adquirido de forma independente da abordagem. Conhecendo estes fatores, valores precisos de estimativa podem ser obtidos.</li> </ol>
<b>Descrição de Conhecimento</b>
Adquirido pela própria definição da abordagem e pela experiência de aplicar a mesma para estimar a confiabilidade de 11 sistemas reais de larga escala em relação a defeitos do tipo 'bug do milênio' ( <i>Y2K defects</i> ), em função dos seguintes fatores: domínio de

aplicação, nível de interação externa, idade do sistema, qualidade intrínseca do sistema e testabilidade do sistema.
<b>Tipo de Estudo</b>
Relato de experiência.

<b>Título do Artigo</b>
Understanding the Sources of Software Defects: A Filtering Approach.
<b>Referência Completa</b>
Wohlin, C., Host, M., Ohlsson, M., Understanding the Sources of Software Defects: A Filtering Approach, in '8th International Workshop on Program Comprehension, 2000, IWPC 2000', pp. 9-17, 2000.
<b>Fonte</b>
IEEE, INSPEC.
<b>Esquema de Classificação de Defeitos</b>
Não tem. A relação das causas é apenas com a contagem de defeitos por componente.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem. As causas são apenas os construtores mais utilizados durante o projeto e a codificação.
<b>Identificação do Processo ou Abordagem</b>
<ul style="list-style-type: none"> <li>• A abordagem apresentada no artigo identifica causas de defeitos a partir da aplicação de filtros (utilizando análise de correlação) relacionando a presença das causas com a ocorrência dos defeitos.</li> </ul>
<b>Descrição do Processo ou Abordagem</b>
<p>A abordagem envolve os seguintes passos:</p> <ul style="list-style-type: none"> <li>• Coleção de Dados. No exemplo do artigo os dados coletados foram número de defeitos, medida do tamanho, número de ocorrências de cada um dos construtores nos documentos de projeto e código (as possíveis causas). A coleção dos dados idealmente deve ocorrer durante o próprio desenvolvimento.</li> <li>• Entendimento dos Dados. Análise de como as medidas se relacionam, de forma a entender, por exemplo, quais causas estão relacionadas quais fatores (<i>factor analysis</i>).</li> <li>• Aplicação dos Filtros. A partir da aplicação dos filtros chega se a subconjuntos de construtores que possuem maior influência nos defeitos revelados. A abordagem é flexível em relação a quais filtros utilizar. Os seguintes filtros foram utilizados no estudo de caso: <ul style="list-style-type: none"> <li>○ Filtro1: Identifica o construtor que mais contribui com o número de defeitos. <math>\max(\text{corr}(\#defects, \#occurrences))</math></li> <li>○ Filtro2: Identifica o construtores cuja ocorrência relativa contribui mais o número relativo de defeitos. <math>\max(\text{corr}(\#defects/size, \#occurrences/size))</math></li> <li>○ Filtro3: Identifica os construtores que contribuem mais para o número de defeitos do que contribuem para o tamanho. <math>\text{corr}(\#defects, \#occurrences) &gt; \text{corr}(size, \#occurrences)</math>.</li> <li>○ Filtro4: Identifica os construtores que contribuem mais do que o tamanho no número de defeitos. <math>\text{corr}(\#defects, \#occurrences) &gt; \text{corr}(\#defects, size)</math>.</li> </ul> <p>A aplicação dos filtros está ilustrada na figura abaixo. Além dos construtores (possíveis causas) obtidos através da aplicação dos filtros, construtores relacionados com os mesmos fatores podem ser considerados interessantes.</p> </li> </ul>



<b>Título do Artigo</b>
New Directions in Measurement for Software Quality Control.
<b>Referência Completa</b>
Krause, P., Freimut, B., Suryan, W., New Directions in Measurement for Software Quality Control, in 'Proceedings of the 10 <sup>th</sup> International Workshop on Software Technology and Engineering Practice, STEP 2002', pp. 129-143, 2002.
<b>Fonte</b>
IEEE, INSPEC.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Abordagem para controle da qualidade utilizando modelos causais baseados em redes bayesianas.
<b>Descrição do Processo ou Abordagem</b>
<p>A abordagem pode ser resumida da seguinte forma:</p> <ul style="list-style-type: none"> <li>• Derivar objetivos de qualidade dos objetivos de negócio.</li> <li>• Utilizar redes bayesianas para implementar modelos causais e utilizar estas redes para controlar a entrega dos objetivos de qualidade.</li> <li>• Como há uma limitação nos relacionamentos de causa e efeito que se pode esperar encontrar através de dados estatísticos, julgamento de especialistas é utilizado para complementar as informações (ou aumentar as evidências) das redes bayesianas.</li> <li>• A ferramenta AID (Assess, Improve, Decide), baseada na máquina de inferência Hugin (veja <a href="http://www.hugin.com">http://www.hugin.com</a>), foi construída para facilitar a aplicação da abordagem.</li> </ul>
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. A abordagem de manufatura para controle da qualidade é inadequada no contexto de produtos de software. Em software o processo de controlar o desenvolvimento é uma questão de controlar processos de criatividade e projeto por humanos. Na manufatura tradicional de produtos, por outro lado, o controle é normalmente da replicação de um projeto estabelecido.</li> <li>2. O uso de redes bayesianas é um modo eficiente de controlar a entrega de objetivos de qualidade no desenvolvimento de software.</li> <li>3. Elicitação cuidadosa de julgamento de especialistas pode ser uma fonte de dados confiável e valorosa.</li> </ol>
<b>Descrição de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Argumentação dos autores. Não há nenhuma fundamentação experimental para a</li> </ol>

<p>afirmação.</p> <ol style="list-style-type: none"> <li>Um estudo de caso foi conduzido com o apoio do <i>Philips Software Centre</i> (PSC) de Bangalore, Índia. Dados foram coletados de 28 projetos de três divisões de negócio da Philips. Os resultados, embora não estatisticamente significativos devido à grande quantidade de informações que não puderam ser obtidas, foram bastante positivos, apoiando a corretude do modelo e a aplicabilidade da abordagem.</li> <li>O artigo apresenta uma abordagem para obtenção de conhecimento de especialistas utilizando simulações de Monte-Carlo. Esta abordagem tem sido utilizada e foi avaliada em um estudo de caso conduzido em uma unidade de negócio da Siemens AG, na Alemanha (Briand <i>et al.</i>, 2000).</li> </ol>
<b>Tipo de Estudo</b>
Estudo de Caso (com dados do Philips Software Centre).

<b>Título do Artigo</b>
Classification and evaluation of defects in a project retrospective.
<b>Referência Completa</b>
Leszak, M., Perry, D. E., Stoll, D., Classification and evaluation of defects in a project retrospective, Journal of Systems and Software, 61( 3), 173 - 187, 2002.
<b>Fonte</b>
INSPEC, EI Compendex.
<b>Esquema de Classificação de Defeitos</b>
O mesmo utilizado em (Leszak <i>et al.</i> , 2000), artigo analisado como parte desta revisão sistemática.
<b>Esquema de Classificação de Causas de Defeitos</b>
O mesmo utilizado em (Leszak <i>et al.</i> , 2000), artigo analisado como parte desta revisão sistemática.
<b>Identificação do Processo ou Abordagem</b>
O mesmo utilizado em (Leszak <i>et al.</i> , 2000), artigo analisado como parte desta revisão sistemática.
<b>Descrição do Processo ou Abordagem</b>
<p>O mesmo utilizado em (Leszak <i>et al.</i>, 2000), artigo analisado como parte desta revisão sistemática.</p> <p>O artigo faz ainda uma breve descrição sobre a utilização da abordagem durante o desenvolvimento em um projeto de médio porte, fazendo uso de um esquema de classificação mais detalhado, incorporando detalhes do ODC (<i>orthogonal defect classification</i>) (Chillarege <i>et al.</i>, 1992). De acordo com os autores, a utilização neste projeto falhou em obter resultados de análise significativos pelos seguintes motivos:</p> <ul style="list-style-type: none"> <li>• O esquema de classificação de defeitos era complexo e preenchido diretamente pelos desenvolvedores. Muitos registros de defeitos se mostraram inconsistentes e as informações corretas não puderam ser recuperadas facilmente.</li> <li>• O número baixo de defeitos e o complexo esquema de classificação resultaram em um número muito baixo de defeitos para cada valor de atributo do esquema.</li> <li>• A equipe de projeto considerou que o esquema de classificação complexo resultou em um grande número de relatórios de medição para ser estudado, não compatível com a agenda do projeto.</li> </ul> <p>Além do estudo de caso sobre análise causal de defeitos já apresentado em (Leszak <i>et al.</i>, 2000), dois outros estudos são descritos no artigo. Respectivamente sobre conformidade de processos e sobre a relação de tamanho e complexidade.</p>
<b>Identificação de Conhecimento</b>



<ol style="list-style-type: none"> <li>1. A tendência de defeitos é difícil de investigar quando o esquema de classificação é complicado e a amostra de defeitos é pequena.</li> <li>2. Os autores planejam para a próxima liberação do produto: <ol style="list-style-type: none"> <li>a. Realizar o preenchimento dos atributos dos defeitos para análise causal por membros de uma pequena equipe, dedicada a análise causal.</li> <li>b. Realizar a análise após cada entrega para a equipe de testes (CARD, 1998).</li> <li>c. Selecionar os defeitos a serem analisados de forma aleatória.</li> <li>d. Submeter os resultados da análise a validação por pares.</li> </ol> </li> </ol>
<b>Descrição de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Com base nas lições aprendidas da utilização de análise causal de defeitos de software durante o processo de desenvolvimento na Lucent Technologies.</li> </ol>
<b>Tipo de Estudo</b>
Estudo de caso.

<b>Título do Artigo</b>
The Importance of Life Cycle Modeling to Defect Detection and Prevention.
<b>Referência Completa</b>
van Moll, J., Jacobs, J., Freimut, B., Trienekens, J., The Importance of Life Cycle Modeling to Defect Detection and Prevention, in '10 <sup>th</sup> International Workshop on Software Technology and Engineering Practice, 2002. STEP 2002', pp. 144-155, 2002.
<b>Fonte</b>
IEEE, INSPEC
<b>Esquema de Classificação de Defeitos</b>
O artigo apenas comenta que diversos esquemas de classificação de defeitos podem ser utilizados, dependendo do propósito da análise de dados. Assim, diferentes atributos tais como localização, fase de introdução, sintoma e causa podem ser utilizados (FREIMUT, 2001). Ele cita ainda um esquema específico, o Orthogonal Defect Classification (ODC) (Chillarege <i>et al.</i> , 1992).
<b>Esquema de Classificação de Causas de Defeitos</b>
O artigo apenas cita os esquemas de classificação de causas apresentados em (LESZAK <i>et al.</i> , 2000) e (MAYS <i>et al.</i> , 1990), artigos analisados neste relatório como parte da revisão sistemática.
<b>Identificação do Processo ou Abordagem</b>
O artigo cita os processo de análise causal de defeitos (CARD, 1998) e prevenção de defeitos (MAYS <i>et al.</i> , 1990), artigos analisados neste relatório como parte da revisão sistemática.
<b>Descrição do Processo ou Abordagem</b>
O processo seguido não é próprio do artigo. Entretanto técnicas específicas são citadas para identificar defeitos de forma sistemática e encontrar as causas dos defeitos. Para identificar defeitos que ocorrem de forma sistemática o artigo menciona que o esquema de classificação de defeitos pode ser utilizado para aplicar uma análise de Pareto. Para chegar até a causa do defeito, por sua vez, o artigo menciona que um <i>brainstorm</i> pode ser conduzido, apoiado por um diagrama de espinha de peixe ( <i>fishbone</i> ).
<b>Identificação de Conhecimento</b>
1. A detecção e a prevenção de defeitos se beneficiam de um ciclo de vida de desenvolvimento bem definido.
<b>Descrição de Conhecimento</b>

<p>O artigo argumenta que os processos de teste e de inspeção se beneficiam de um ciclo de vida de desenvolvimento bem definido por facilitar a obtenção de medidas para a detecção e prevenção. Entretanto não há nenhuma fundamentação experimental para a afirmação.</p>
<p><b>Tipo de Estudo</b></p>
<p>Teórico.</p>

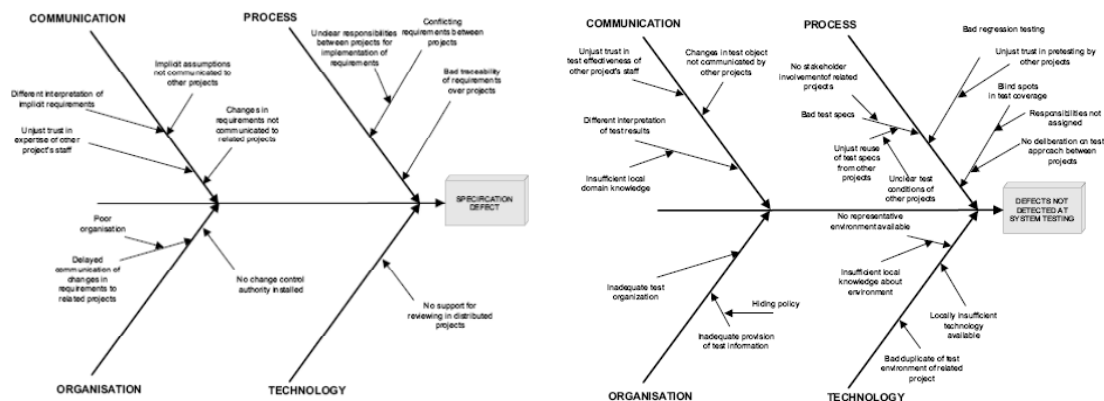
<b>Título do Artigo</b>
Evaluating and Incorporating New Age Software Technology for Identifying Systemic Root Causes.
<b>Referência Completa</b>
Vantine, W., Benfield, K., Pritts, D., Ballard, K., Evaluating and Incorporating New Age Software Technology for Identifying Systemic Root Causes, in: 'Proceedings of the Joint ESA-NASA Space Flight Safety Conference, ESTEC', ESA SP-486, Noordwijk (NL), pp. 369 - 376, 2002.
<b>Fonte</b>
EI Compendex.
<b>7.5.1 Esquema de Classificação de Defeitos</b>
Não tem.
<b>7.5.2 Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Apoio a análise causal pela ferramenta Reason.
<b>Descrição do Processo ou Abordagem</b>
<p>A abordagem envolve a descrição do problema e a adição de causas. A ferramenta então realiza validações lógicas derivando perguntas com respostas 'sim/não' com base nos dados informados. Após as validações lógicas os dados são classificados pelo usuário como causa, dados não corrigíveis ou dados insuficientes. Os dados classificados causas podem então ser apresentados à gerência. A comunicação dos dados é feita tornando os acessíveis pela Internet.</p> <p>Pelas informações apresentadas no artigo a ferramenta não lida com categorias de problemas nem categorias de causas, não aproveitando eficientemente o conhecimento obtido a partir de análises causais para análises futuras. Além disto não apóia o trabalho em equipe para identificação das causas.</p>
<b>Identificação de Conhecimento</b>
1. No artigo critérios de avaliação foram estabelecidos, sendo os quatro principais: (1) corretude técnica dos dados, (2) facilidade de uso, (3) análise de dados e (4) armazenamento e comunicação de dados. A ferramenta Reason foi considerada a mais adequada aos critérios.
<b>Descrição de Conhecimento</b>

Os resultados da avaliação não são apresentados no artigo e nenhuma ferramenta que não seja Reason é citada, mostrando pouca validade da avaliação. Além do mais existe o viés de que Reason foi a ferramenta escolhida pela empresa do autor do artigo para clientes publico alvo evento em que o artigo foi apresentado.
<b>Tipo de Estudo</b>
Teórico.

<b>Título do artigo</b>
End-to-End Defect Modeling.
<b>Referência Completa</b>
Gras, J.J., End-to-End Defect Modeling, IEEE Software, 21(5), 98-100, 2004.
<b>Fonte</b>
INSPEC.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Modelo dinâmico baseado em redes bayesianas e relações de causa e efeito de defeitos de software.
<b>Descrição do Processo ou Abordagem</b>
Durante a modelagem os fatores a serem considerados são elicitados, assim como seu relacionamento causa-efeito. As relações são então quantificadas com base na opinião de especialistas e dados. As previsões resultantes dos modelos são então utilizadas para refinar os planos do projeto.
<b>Identificação de Conhecimento</b>
1. Redes bayesianas são adequadas para modelar conhecimento sobre relações de causa e efeito.
<b>Descrição de Conhecimento</b>
Observação do artigo tendo em vista propriedades das redes bayesianas. Abordagem foi utilizada na Motorola Labs.
<b>Tipo de Estudo</b>
Relato de experiência (na Motorola Labs).

<b>Título do Artigo</b>
Effects of Virtual Development on Product Quality: Exploring Defect Causes.
<b>Referência Completa</b>
Jacobs, J.C., van Moll, J.H., Krause, P.J., Kusters, R.J., Trienekens, J.J.M., Effects of Virtual Development on Product Quality: Exploring Defect Causes, Proceedings of the 11th Annual International Workshop on Software Technology and Engineering Practice (STEP'04), 2004.
<b>Fonte</b>
INSPEC.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Associa causas a categorias de risco em desenvolvimento com equipes geograficamente distribuídas (comunicação, processo, organização e tecnologia).
<b>Identificação do Processo ou Abordagem</b>
Análise causal de defeitos a partir de conhecimento de especialistas.
<b>Descrição do Processo ou Abordagem</b>
<p>O objetivo da abordagem era encontrar as causas de injeção e detecção tardia (ou não detecção) de defeitos de software em desenvolvimento com equipes geograficamente distribuídas. Entretanto, dados reais de defeitos de software de empresas geograficamente distribuídas não puderam ser obtidos. Assim, a abordagem utilizada no artigo difere da análise causal de defeitos (Card, 1998) e do processo de prevenção de defeitos (Mays <i>et al.</i>, 1990) por fazer um inventário de defeitos ao invés de utilizar amostras de defeitos detectados em projetos reais. Os defeitos e as causas associadas eram provenientes da experiência acumulada dos participantes.</p> <p>Foram escolhidos seis especialistas em desenvolvimento com equipes geograficamente distribuídas. Seis reuniões de análise causal foram conduzidas, com diferentes combinações dos seis especialistas. Durante as reuniões, diagramas de causa e efeito (Ishikawa, 1976) foram utilizados, associando as causas a categorias de risco em desenvolvimento com equipes geograficamente distribuídas (comunicação, processo, organização e tecnologia). Só eram listadas causas relacionadas com desenvolvimento geograficamente distribuído. Os diagramas resultantes, respectivamente sobre injeção de</p>

defeitos e detecção tardia (ou não detecção), seguem.



### Identificação de Conhecimento

1. Esquemas de classificação como o esquema HP (da *hewlett packard*), o padrão IEEE 1044 para anomalias de software ou o ODC (orthogonal defect classification) são raramente utilizados em desenvolvimento com equipes geograficamente distribuídas.
2. Os diagramas de causa e efeito (Ishikawa, 1976) foram considerados uma ferramenta útil para a análise causal baseada no conhecimento de especialistas.
3. As causas para injeção de defeitos foram encontradas principalmente na fase de especificação dos requisitos do software.
4. As causas para a detecção tardia (ou não detecção) foram encontradas principalmente na fase de testes de integração.
5. Desenvolvimento geograficamente distribuído aumenta a probabilidade de injeção de defeitos, apresentando novas causas para defeitos e uma maior probabilidade de ocorrência para algumas causas existentes.

### Descrição de Conhecimento

1. Argumentação dos autores.  
2 a 5. Observações da aplicação da abordagem para análise causal de defeitos de software em desenvolvimento geograficamente distribuído.

### Tipo de Estudo

Estudo de Caso.



<b>Título do Artigo</b>
Facilitating Organizational Learning through Causal Mapping Techniques in IS/IT Project Risk Management
<b>Referência Completa</b>
Al-Shehab, A. J., Hughes, R. T., Winstanley, G., Facilitating Organisational Learning through Causal Mapping Techniques in IS/IT Project Risk Management, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3782 NAI, 145 - 154, 2005.
<b>Fonte</b>
EI Compendex.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Uso de mapas causais para avaliação de projetos passados visando apoiar a gerência de riscos de novos projetos.
<b>Descrição do Processo ou Abordagem</b>
O artigo descreve estudos de caso envolvendo a construção de mapas causais a partir de um dado cenário. A abordagem envolve a construção de mapas causas individualmente e depois o exercício de juntar os diferentes conceitos apresentados. Foi conduzido tanto na academia quanto na indústria (utilizando gerentes e desenvolvedores). Em ambos os casos os mapas causais resultantes utilizavam fatores bastante diferentes.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. A construção de mapas causais permite mostrar as relações entre diferentes riscos (causas), sendo uma boa representação gráfica para discussão.</li> <li>2. Problemas ontológicos permanecem, como a percepção diferente dos participantes sobre os fatores que levam a um determinado cenário. Participantes têm diferentes visões sobre o que aconteceu em um projeto (tanto observando descrições textuais quanto considerando projetos reais).</li> <li>3. Gerentes e desenvolvedores tendem a ter visões bastante diferentes sobre o que deu errado dentro de um projeto.</li> </ol>
<b>Descrição de Conhecimento</b>
Observações resultantes da condução de dois estudos de caso realizados na academia e na indústria. Os estudos envolveram respectivamente 15 estudantes de mestrado e 13 profissionais da indústria (5 gerentes e 8 desenvolvedores) construindo

mapas causais individualmente e depois discutindo os mesmos em grupo para chegar a um consenso.

<b>Tipo de Estudo</b>
Estudo de caso.

<b>Título do Artigo</b>
Defect Analysis: Basic Techniques for Management and Learning.
<b>Referência Completa</b>
Card, D., “Defect Analysis: Basic Techniques for Management and Learning”, <i>Advances in Computers</i> , vol. 65, chapter 7, pp. 259-295, 2005.
<b>Fonte</b>
Não indexado nas bases pesquisadas (Artigo de controle).
<b>Esquema de Classificação de Defeitos</b>
<p>O artigo sugere três dimensões para classificar defeitos:</p> <ul style="list-style-type: none"> <li>• Momento de introdução;</li> <li>• Momento de detecção;</li> <li>• Tipo de defeito.</li> </ul> <p>Os tipos de defeito comumente utilizados são: interface, computacional, lógico, inicialização e estrutura de dados. Segundo o autor ODC (Chillarege <i>et al.</i>, 1992), que considera as dimensões acima citadas, oferece um esquema de classificação compreensível para análise causal de defeitos. Entretanto, o esquema de classificação de defeitos deve ser adaptado de maneira a apoiar as necessidades de análise da organização que está implementando análise causal de defeitos.</p>
<b>Esquema de Classificação de Causas de Defeitos</b>
<p>O artigo recita as quatro categorias de causa descritas por Ishikawa (1976): métodos, ferramentas/ambiente, pessoas e entradas (requisitos).</p> <p>Entretanto, o esquema de classificação das causas de defeitos deve ser adaptado de maneira a apoiar as necessidades de análise da organização que está implementando análise causal de defeitos.</p>
<b>Identificação do Processo ou Abordagem</b>
Processo análogo ao processo de análise causal de defeitos de software apresentado em (Card, 1998), com detalhamento e experiências adicionais.
<b>Descrição do Processo ou Abordagem</b>
<p>O artigo apresenta o mesmo processo descrito em (Card, 1998), acrescentando um detalhamento adicional nos procedimentos para análise causal de defeitos. Análise causal é descrita em seis passos:</p> <ul style="list-style-type: none"> <li>• Selecionar amostra de defeitos. Se a análise causal é iniciada por uma anomalia na distribuição de defeitos ou por uma situação fora de controle de um gráfico de controle então são os defeitos referentes a esta anomalia que devem ser analisados. Se a análise causal é para melhoria contínua deve considerar uma amostra representativa da população de defeitos. De acordo com o artigo, dificilmente uma</li> </ul>

<p>equipe consegue tratar mais de 20 defeitos em uma reunião de 2 horas.</p> <ul style="list-style-type: none"> <li>• Classificar e agrupar defeitos. É sugerido o uso de um esquema de classificação como ODC (Chillarege <i>et al.</i>, 1992) e o uso de gráficos de Pareto ou tabelas de índices cruzados para o agrupamento.</li> <li>• Encontrar os erros sistemáticos. Um erro sistemático é um erro que resulta em defeitos similares se repetindo em diferentes ocasiões.</li> <li>• Determinar as causas principais. Muitas causas podem contribuir para um erro sistemático, o foco deve ser em encontrar as principais. Nesta atividade é sugerido o uso de diagramas de Ishikawa (Ishikawa, 1976).</li> <li>• Desenvolver propostas de ação. Uma vez que a causa principal de um defeito sistemático foi encontrada, itens de ação devem ser desenvolvidos para promover a prevenção (ou detecção mais cedo, em casos que a prevenção não é possível) dos erros sistemáticos. Normalmente o número de ações é pequeno. As ações devem ser tão pontuais e específicas quanto possível, facilitando o acompanhamento objetivo do estado de sua implantação.</li> <li>• Documentar os resultados da reunião de análise causal. Registros dos resultados da reunião de análise causal precisam ser mantidos para assegurar que as ações serão implementadas. De acordo com o artigo, para o sucesso de um programa de análise causal é essencial que uma equipe de ação seja formada. Os resultados da reunião de análise causal devem ser fornecidos para esta equipe, que tem entre suas responsabilidades encontros regulares a respeito das ações propostas.</li> </ul>
<p><b>Identificação de Conhecimento</b></p>
<ol style="list-style-type: none"> <li>1. Para gráficos de controle a respeito de defeitos o gráfico “U” costuma ser adequado. Ele se aplica a dados com distribuição de Poisson, com uma área de oportunidade variada (por exemplo, tamanho de um documento) para que o evento ocorra (por exemplo, detecção de um defeito).</li> <li>2. O artigo cita três condições que de acordo com Babbie (1986) precisam ser satisfeitas para que se tenha um relacionamento causal:             <ol style="list-style-type: none"> <li>a. Deve haver uma correlação entre a hipotética causa e o seu efeito;</li> <li>b. A causa precisa preceder o efeito;</li> <li>c. O mecanismo ligando a causa ao defeito precisa ser identificado.</li> </ol> </li> <li>3. De acordo com o artigo três princípios guiam as abordagens de análise causal de defeitos para a melhoria da qualidade:             <ol style="list-style-type: none"> <li>a. Reduzir defeitos para aumentar a qualidade;</li> <li>b. Aplicar expertise local;</li> <li>c. Focar em erros sistemáticos.</li> </ol> </li> <li>4. Os pré-requisitos para se aplicar análise causal de defeitos são:             <ol style="list-style-type: none"> <li>a. Defeitos existem;</li> <li>b. Defeitos foram documentados;</li> <li>c. Existe vontade de prevenir a ocorrência dos defeitos em projetos futuros;</li> <li>d. Um processo de software básico precisa estar definido para prover o arcabouço para as ações que devem ser implementadas.</li> </ol> </li> <li>5. Diversas experiências mostram um declínio significativo na taxa de defeitos tanto para projetos envolvendo centenas de pessoas (Mays <i>et al.</i>, 1990) (Leszak <i>et al.</i>, 2002) quanto para projetos menores (Dangerfield <i>et al.</i>, 1992) (Yu, 1998).</li> <li>6. Reduzir fontes sistemáticas de defeitos normalmente também provê benefícios de redução de custos e tempo de desenvolvimento.</li> <li>7. A distribuição dos tipos de defeitos pode ser um indicador de curto prazo mais sensível para mudanças de processos do que a taxa de defeitos em si.</li> <li>8. Entre os benefícios da aplicação de análise causal de defeitos se encontram os</li> </ol>

<p>seguintes:</p> <ul style="list-style-type: none"> <li>a. Percepção da qualidade;</li> <li>b. Comprometimento com o processo;</li> <li>c. Entendimento das medidas de qualidade.</li> </ul> <p>9. O custo de um programa de análise causal de defeitos, incluindo tanto a análise causal quanto a implementação das ações varia de 0,5% (Mays <i>et al.</i>, 1990) a 1,5% (Dangerfield <i>et al.</i>, 1992) do orçamento do projeto de software.</p>	
<b>Descrição de Conhecimento</b>	
Compilação de conhecimentos obtidos de diversas experiências da indústria.	
<b>Tipo de Estudo</b>	
Teórico, baseado em experiências da indústria.	

<b>Título do Artigo</b>
Exploring Defect Causes in Products Developed by Virtual Teams.
<b>Referência Completa</b>
Jacobs, J.C., van Moll, J.H., Krause, P.J., Kusters, R.J., Trienekens, J.J.M., Brombacher, A., Exploring Defect Causes in Products Developed by Virtual Teams, Journal on Information and Software Technology, 47( 6), 399 – 410, 2005.
<b>Fonte</b>
EI Compendex, Web of Science.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Utiliza a mesma categorização da descrita em (Jacobs <i>et al.</i> , 2004). Associa causas a categorias de risco em desenvolvimento com equipes geograficamente distribuídas (comunicação, processo, organização e tecnologia).
<b>Identificação do Processo ou Abordagem</b>
Utiliza a mesma abordagem da descrita em (Jacobs <i>et al.</i> , 2004). Análise causal de defeitos a partir de conhecimento de especialistas.
<b>Descrição do Processo ou Abordagem</b>
Abordagem apresentada em (Jacobs <i>et al.</i> , 2004). Artigo analisado como parte desta revisão sistemática.
<b>Identificação de Conhecimento</b>
Conhecimento identificado em (Jacobs <i>et al.</i> , 2004). Artigo analisado como parte desta revisão sistemática.
<b>Descrição de Conhecimento</b>
Conhecimento descrito em (Jacobs <i>et al.</i> , 2004). Artigo analisado como parte desta revisão sistemática.
<b>Tipo de Estudo</b>
Estudo de Caso.

<b>Título do Artigo</b>
Using Defect Analysis Feedback for Improving Quality and Productivity in Iterative Software Development.
<b>Referência Completa</b>
Jalote, P., Agrawal, N., “Using Defect Analysis Feedback for Improving Quality and Productivity in Iterative Software Development”, (Invited paper, 3rd International Conference on Information and Communication Technology, ICICT, 2005.), pp. 701 – 713, Cairo, 2005.
<b>Fonte</b>
IEEE.
<b>Esquema de Classificação de Defeitos</b>
Menciona que organizações comumente registram além do tipo de defeito a fase de detecção e de inserção. Sugere que os tipos de defeito sejam adequados para a realidade da organização. Na experiência na InfoSys os seguintes tipos foram utilizados: Lógica, Padrões, Código Redundante, Interface Gráfica e Arquitetura.
<b>Esquema de Classificação de Causas de Defeitos</b>
<p>Menciona que na manufatura as categorias costumam ser pessoas, máquinas, métodos, materiais, medição e ambiente (5M e 1E) (Robitaille, 2004).</p> <p>Entretanto, sugere as seguintes categorias de causa para software: processo, pessoas e tecnologia. O artigo recita que estes são os fatores mais impactantes na qualidade e produtividade de software de acordo com (Jalote, 2000) e (SEI, 1995).</p>
<b>Identificação do Processo ou Abordagem</b>
Análogo ao processo de análise causal de defeitos de software apresentado em (Card, 1998).
<b>Descrição do Processo ou Abordagem</b>
<p>Na experiência da InfoSys relatada no artigo análise causal de defeitos foi aplicada em um projeto de desenvolvimento com ciclo de vida incremental.</p> <p>A abordagem se resume em:</p> <ul style="list-style-type: none"> <li>• Categorizar os defeitos conforme descrito no esquema de classificação acima.</li> <li>• A seguir um gráfico de Pareto é utilizado para identificar os tipos de defeito que deveriam ser analisados.</li> <li>• Posteriormente, na reunião de análise causal sessões de brainstorming são conduzidas para encontrar tanto a causa raiz quanto as propostas de ação. Na InfoSys nestas reuniões o uso de diagramas de Ishikawa é comum. Mas as causas e as propostas de ação no final da reunião são registradas de forma tabular.</li> <li>• Após a reunião as propostas de ação são integradas no cronograma do projeto e</li> </ul>

então implementadas e monitoradas como qualquer outra tarefa do projeto.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Na experiência da InfoSys, relatada no artigo, foi possível: <ol style="list-style-type: none"> <li>a. Reduzir a taxa de defeitos de 0,33 defeitos por homem-hora para 0,1 entre as primeiras duas iterações do projeto.</li> <li>b. Reduzir o esforço de retrabalho de 15% para menos de 5% entre as primeiras duas iterações do projeto e para menos de 3% para a terceira iteração.</li> </ol> </li> <li>2. Os autores sugerem que o retorno da análise causal de defeitos em um projeto incremental é maior para as primeiras iterações. Nas iterações seguintes as principais oportunidades de melhoria podem já ter sido implementadas.</li> </ol>
<b>Descrição de Conhecimento</b>
Dados provenientes do relato de experiência da InfoSys.
<b>Tipo de Estudo</b>
Relato de Experiência.



<b>Título do Artigo</b>
A View of 20th and 21st Century Software Engineering.
<b>Referência Completa</b>
Boehm, B., A View of 20th and 21st Century Software Engineering, in 'ICSE '06: Proceeding of the 28th International Conference on Software Engineering', ACM Press, New York, NY, USA, pp. 12-29, 2006.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Não tem.
<b>Descrição do Processo ou Abordagem</b>
Não se aplica.
<b>Identificação de Conhecimento</b>
1. Prevenção de defeitos é considerada no artigo como um princípio atemporal da engenharia de software, marco da década de 70.
<b>Descrição de Conhecimento</b>
Conhecimento identificado pelo próprio autor. Não é fornecida fundamentação experimental para a afirmação.
<b>Tipo de Estudo</b>
Teórico.

<b>Título do Artigo</b>
Difficulties in Establishing a Defect Management Process: A Case Study.
<b>Referência Completa</b>
Jantti, M., Toroi, T., Eerola, A., Difficulties in establishing a defect management process: A case study, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 4034 NCS, pp. 142-150, 2006.
<b>Fonte</b>
INSPEC, EI Compendex.
<b>Esquema de Classificação de Defeitos</b>
Não tem. Apresenta um estudo de caso onde questionou empresas se coletavam os seguintes dados a respeito de seus defeitos: fase de detecção, atividade que originou o defeito, tipo de defeito, causa do defeito, tempo e custo para corrigir o defeito.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Processo de Gerenciamento de Defeitos.
<b>Descrição do Processo ou Abordagem</b>
Segundo os autores o gerenciamento de defeitos envolve a prevenção de defeitos, detecção de defeitos e sua correção, análise causal de defeitos e a melhoria do processo.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Entre as lições aprendidas do estudo de caso há uma referente a análise causal de defeitos de software. <ol style="list-style-type: none"> <li>a. No longo prazo a gerência de defeitos deve ser mais pró-ativa do que reativa. Focar em defeitos antes que eles ocorram é mais útil do que corrigir um grande número de erros repetitivos.</li> </ol> </li> </ol>
<b>Descrição de Conhecimento</b>
Lição aprendida de um estudo de caso sobre gerenciamento de defeitos conduzido com quatro organizações.
<b>Tipo de Estudo</b>
Estudo de Caso.

<b>Título do Artigo</b>
BSR: A Statistic-based Approach for Establishing and Refining Software Process Performance Baseline
<b>Referência Completa</b>
Wang, Q., Jiang, N., Gou, L., Liu, X., Li, M., Wang, Y., BSR: A Statistic-based Approach for Establishing and Refining Software Process Performance Baseline, in 'TCSE '06: Proceeding of the 28th International Conference on Software Engineering', ACM Press, New York, NY, USA, pp. 585-594., 2006.
<b>Fonte</b>
ACM
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Usa análise causal de defeitos para estabilizar processos.
<b>Descrição do Processo ou Abordagem</b>
Análise causal é utilizada na metodologia BSR como instrumento para estabilizar processos, removendo causas atribuíveis que tornam o processo instável. Não define abordagem para tal, mas sugere os métodos estatísticos de análise de pareto e diagramas de espinha de peixe ( <i>fishbone</i> ).
<b>Identificação de Conhecimento</b>
1. O artigo apresenta a metodologia BSR, que pode ser utilizada para estabelecer <i>baselines</i> de performance de processos e têm se mostrado efetiva para gerência estatística de processos. A abordagem pode ser utilizada tanto por organizações com altos níveis de maturidade quanto por organizações com níveis mais baixo. Tendo como único pré-requisito ter os processos escolhidos definidos e assegurar que as características de qualidade sendo observadas sejam efetivamente medidas.
<b>Descrição de Conhecimento</b>
A abordagem têm sido utilizada em mais de 50 organizações de software. O artigo detalha a aplicação de BSR em três organizações para estabelecer <i>baselines</i> de performance de processos.
<b>Tipo de Estudo</b>
Relato de experiência (em 3 organizações).

## **B2 – Informações Extraídas dos Artigos Recuperados na Revisão de Setembro de 2007**

<b>Título do Artigo</b>
A Case History of International Space Station Requirement Faults
<b>Referência Completa</b>
Hayes, J.H., Raphael, I., Holbrook, E.A., Pruett, D.M., A case history of International Space Station requirement faults, Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems, Stanford University, California, 2006.
<b>Fonte</b>
INSPEC.
<b>Esquema de Classificação de Defeitos</b>
A taxonomia para a classificação de defeitos se encontra descrita na tabela abaixo, extraída do artigo.

Major Fault	Sub-Faults
1. Requirements	Originate in Requirements phase; found in Requirements Specifications
.1 Incompleteness	.1.1 Incomplete Decomposition .1.2 Incomplete Requirement Description
.2 Omitted/Missing	.2.1 Omitted Requirement .2.2 Missing External Constants .2.3 Missing Description of Initial System State
.3 Incorrect	.3.1 Incorrect External Constants .3.2 Incorrect Input or Output Descriptions .3.3 Incorrect Description of Initial System State .3.4 Incorrect Assignment of Resources
.4 Ambiguous	.4.1 Improper Translation .4.2 Lack of Clarity
.5 Infeasible	.-----
.6 Inconsistent	.6.1 External Conflicts .6.2 Internal Conflicts
.7 Over-specification	.-----
.8 Not Traceable	.-----
.9 [reserved for the future]	.-----
.10 Non-Verifiable	.-----
.11 Misplaced	.-----
.12 Intentional Deviation	.-----
.13 Redundant	.-----

A taxonomia é parecida com a definida por Shull (1998) e, segundo os autores, muitos artigos continham estes mesmos tipos de defeito e poucos artigos descreviam ‘novos’ tipos de defeito.

#### Esquema de Classificação de Causas de Defeitos

Examinando 25 relatórios de problemas foram encontrados três tipos de causas comuns: processo inadequado, falta de entendimento dos requisitos e erro humano. Estas causas comuns foram mapeadas por engenheiros experientes aos tipos de defeitos conforme a figura abaixo, extraída do artigo.

<b>Identificação do Processo ou Abordagem</b>		
Análise baseada em defeitos (Fault-Based Analysis - FBA).		
<b>Descrição do Processo ou Abordagem</b>		
<p>A abordagem envolve as seguintes atividades:</p> <ul style="list-style-type: none"> <li>• Selecionar um item de configuração.</li> <li>• Categorizar os defeitos de requisitos do item de configuração de acordo com a taxonomia.</li> <li>• Determinar a frequência de defeitos para o item de configuração.</li> <li>• Identificar as categorias de defeitos cruciais para o item de configuração.</li> <li>• Repetir os passos acima para todos os itens de configuração.</li> </ul> <p>A exposição aos defeitos pode então ser calculada multiplicando para um determinado tipo de defeito o fator de tolerância pela probabilidade de ocorrência.</p> <p>Para identificar as causas comuns, a árvore de causas comuns é utilizada, embora considerada pelos autores ainda incompleta e pouco detalhada (A mesma não é aplicada no estudo de caso descrito no artigo, que se limita à classificação dos defeitos).</p>		
<b>Identificação de Conhecimento</b>		
1. Distribuição de defeitos de acordo com a taxonomia na ISS (International Space Station) da NASA. Limitado a este escopo e não muito relevante para este estudo secundário.		
<b>Descrição de Conhecimento</b>		
Adquirido através de um estudo de caso com dados de projetos reais na ISS (International Space Station) da NASA. Defeitos de três itens de configuração foram analisados por dois especialistas e classificados de acordo com a taxonomia. A primeira liberação dos itens de configuração continha 430, 339 e 339 requisitos, respectivamente.		
<b>Tipo de Estudo</b>		
Estudo de caso (na estação ISS da NASA).		

<b>Título do Artigo</b>		
Have Things Changed Now? An Empirical Study of Bug Characteristics in Modern Open Source Software.		
<b>Referência Completa</b>		
Li, Z., Tan, L., Wang, X., Lu, S., Zhou, Y., Zhai, C., Have things changed now? An empirical study of bug characteristics in modern open source software, Proceedings of ASID'06: 1st Workshop on Architectural and System Support for Improving Software Dependability, pp. 25-33, 2006.		
<b>Fonte</b>		
ACM, EI Compendex.		
<b>Esquema de Classificação de Defeitos</b>		
Classifica falhas de acordo com suas causas no código (os defeitos): memória, concorrência e semântica. As falhas de memória e semânticas por sua vez são subdivididas em subcategorias de acordo com a figura abaixo, extraída do artigo.		
<b>Memory Bug</b>	Memory Leak	Failures to release unused memory.
	Uninitialized Memory Read	Read memory data before it is initialized.
	Dangling Pointer	Pointers still keep freed memory addresses.
	NULL Pointer Dereference	Dereference of a null pointer.
	Overflow	Illegal access beyond the buffer boundary.
	Double Free	One memory location is freed twice.
<b>Semantic Bug</b>	Missing Features	A feature is supposed to be but is not implemented.
	Missing Cases	A case in a functionality is not implemented.
	Corner Cases	Some boundary cases are considered incorrectly or ignored.
	Wrong Control Flow	The control flow is incorrectly implemented.
	Exception Handling	Do not have proper exception handling.
	Processing	Processing such as evaluation of expressions and equations is incorrect.
	Typo	Typographical mistakes.
	Other Wrong Functionality Implementation	Any other semantic bug that does not meet the design requirement.
<b>Esquema de Classificação de Causas de Defeitos</b>		
Não tem.		
<b>Identificação do Processo ou Abordagem</b>		
Técnicas de classificação de textos com base em linguagem natural para classificar falhas de projetos open-source.		
<b>Descrição do Processo ou Abordagem</b>		
Foram utilizadas técnicas de classificação de textos com base em linguagem natural para classificar automaticamente 29.000 (vinte e nove mil) falhas de projetos open source. A ‘root cause analysis’ do artigo se limita a classificar as falhas de acordo com o tipo de defeito.		
<b>Identificação de Conhecimento</b>		
1. A maior parte dos defeitos ocasionando as falhas (mais de 80%) são semânticos. A maior parte dos erros semânticos por sua vez são causados por implementação de funcionalidades de modo a não atender aos requisitos.		

<b>Descrição de Conhecimento</b>
Resultado da análise das 29.000 falhas.
<b>Tipo de Estudo</b>
Estudo empírico sobre bases de projetos open source.



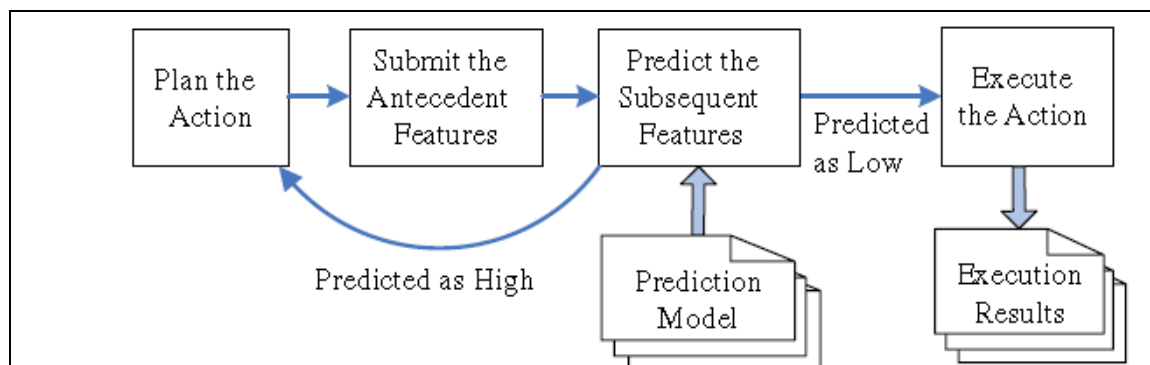
<b>Título do Artigo</b>
Identifying Domain-Specific Defect Classes Using Inspections and Change History
<b>Referência Completa</b>
Nakamura, T., Hochstein, L., Basili, V.R., Identifying Domain-Specific Defect Classes Using Inspections and Change History, Proceedings of the 2006 International Symposium on Empirical Software Engineering (ISESE), Sept. 21-22, Rio de Janeiro, Brazil. p. 346-355, 2006.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
<p>Sugerem um <i>template</i> contendo as seguintes informações (para defeitos encontrados em código fonte, sem documentação): problema, localização, defeito, falha, tempo para encontrar e corrigir, como foi corrigido e descrição com outras informações de contexto.</p> <p>Em relação à classificação dos defeitos, sugerem uma abordagem que primeiramente agrupa defeitos por informações de contexto similares e depois define uma categoria para estes grupamentos, abstraindo as características comuns.</p>
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Metodologia para analisar defeitos com base em leitura de código e análise do histórico de mudanças deste código.
<b>Descrição do Processo ou Abordagem</b>
<p>Nesta abordagem a ‘análise de defeitos’ serve para construir um esquema de classificação de defeitos de codificação e um perfil de frequência e severidade dos diferentes tipos de defeitos, em situações de documentação inexistente (tendo somente o código e o histórico de mudanças).</p> <p>Assim, no contexto desta pesquisa se limita a servir como preparação antes de conduzir uma ‘análise causal de defeitos’ da atividade de codificação.</p>
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. O uso da abordagem para a construção de esquemas de classificação para defeitos de codificação onde documentação pode levar a categorias bem definidos.</li> <li>2. Pessoas tem diferentes opiniões sobre que defeitos são importantes.</li> <li>3. Exemplos concretos de defeitos ajudam a entender as categorias de defeitos.</li> <li>4. A análise de defeitos seguindo a metodologia exige muito esforço por parte dos analistas e é complicada no início. Iniciar a análise com códigos mais simples é recomendado.</li> </ol>

<b>Descrição de Conhecimento</b>
Conhecimento obtido ao aplicar a abordagem em um estudo de caso com alunos de pós-graduação que aprenderam programação para computação de alto desempenho ( <i>high performance computing</i> – HPC) e resolveram problemas HPC em sala de aula.
<b>Tipo de Estudo</b>
Estudo de caso.

<b>Título do Artigo</b>
Requirement Error Abstraction and Classification: An Empirical Study
<b>Referência Completa</b>
Walia, G., Carver, J., Philip, T., Requirements Error Abstraction and Classification: An Empirical Study, Proceedings of the 2006 International Symposium on Empirical Software Engineering (ISESE), Sept. 21-22, Rio de Janeiro, Brazil. p. 336-345, 2006.
<b>Fonte</b>
ACM, EI Compendex.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
<p>Tem uma taxonomia para classificação de erros cometidos em documentos de requisitos (Requirements Error Classification Taxonomy - RET). Estes erros são as causas dos defeitos, mas não é fornecida uma taxonomia para classificar causas desses erros. A classificação dos erros para documentos de requisitos se encontra descrita na figura abaixo, extraída do artigo.</p> <pre> graph TD     RE[Requirement Errors] --&gt; PE[People Errors]     RE --&gt; P[Process Errors]     PE --&gt; DE[Documentation Errors]     P --&gt; DE </pre> <p>De acordo com os autores o esquema foi elaborado com base na literatura, em taxonomias de classificação de defeitos e dados históricos de projetos anteriores.</p>
<b>Identificação do Processo ou Abordagem</b>
Processo de Abstração de Erros (Error Abstraction Process – EAP).
<b>Descrição do Processo ou Abordagem</b>
O processo de abstração de defeitos consiste em analisar a natureza dos defeitos e

<p>determinar os erros responsáveis por tais defeitos.</p> <p>A abordagem utilizada no artigo envolve realizar a abstração de erros sobre listas de defeitos (de inspeções de documentos de requisitos) e depois tentar classificar os erros de acordo com a taxonomia descrita acima.</p>
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. O uso do processo de abstração de erros (EAP) melhora a eficiência e efetividade, tanto para equipes de inspeção quanto para inspetores individuais, permitindo que mais defeitos sejam encontrados nos documentos de requisitos. Após aplicar EAP e RET as equipes foram capazes de encontrar respectivamente um adicional de 75% e 156% de defeitos.</li> <li>2. A RET é de fácil uso e pode ser útil para melhorar a qualidade de software.</li> <li>3. Os tipos de erros na RET são válidos e provêem uma boa cobertura do espaço total de erros de requisitos.</li> <li>4. A contribuição de pesquisas de cognição humana, psicologia e outros campos ajuda a localizar mais defeitos. Cerca de 20 a 25% dos erros puderam ser classificados como erros de cognição humana.</li> <li>5. A performance individual durante EAP depende de várias variáveis independentes.</li> <li>6. A abordagem EAP pode ser utilizada para apoiar a identificação de erros sistemáticos em defeitos de software.</li> </ol>
<b>Descrição de Conhecimento</b>
<p>Resultados obtidos de um estudo experimental conduzido com 16 alunos de graduação, divididos em dois grupos. Eles realizaram inspeções para encontrar defeitos, depois aplicaram abstração de erros (EAP) na lista de defeitos, classificaram os erros com base na taxonomia RET e então retomaram o documento de requisitos para encontrar mais defeitos com base no aprendizado de aplicar EAP e RET.</p>
<b>Tipo de Estudo</b>
<p>Estudo Experimental.</p>

<b>Título do Artigo</b>
Defect Prevention in Software Processes: An Action-Based Approach
<b>Referência Completa</b>
Chang, C., Chu, C., "Defect Prevention in Software Processes: An Action-Based Approach", The Journal of Systems and Software, Vol. 80, issue 4, April 2007, pp. 559-570, 2007.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
Defeitos são classificados de acordo com as seguintes características: id, descrição, ação que gerou o defeito, ação onde o defeito foi detectado, ação utilizada para remover o defeito, severidade do defeito e o módulo que causou o defeito (localização). Não há uma taxonomia para tipos de defeitos definida.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não há. As ações da WBS (work breakdown structure) são consideradas as causas dos defeitos.
<b>Identificação do Processo ou Abordagem</b>
ABDP – Action-based Defect Prevention.
<b>Descrição do Processo ou Abordagem</b>
<p>A abordagem ABDP pode ser aplicada no processo de desenvolvimento de software para detectar ações que podem causar muitos defeitos. Ao detectar estas ações, ações corretivas podem ser tomadas para prevenir a ocorrência de defeitos.</p> <p>Nesta abordagem as ações (derivadas da WBS) possuem características antecessoras, que podem ser informadas no início da ação e características subsequentes, que não podem ser informadas no início da ação. As características subsequentes são estimadas a partir das características antecessoras utilizando um modelo de desempenho estatístico construído a partir de dados das ações já realizadas (no mesmo projeto). A Figura a seguir demonstra como se dá a execução de uma ação utilizando a abordagem ABDP.</p>



A técnica FSS (*Feature Subset Selection*) é utilizada para filtrar características não necessárias dos dados. Em seguida a técnica *under-sampling* é aplicada para resolver o problema da raridade de ações que geram defeitos.

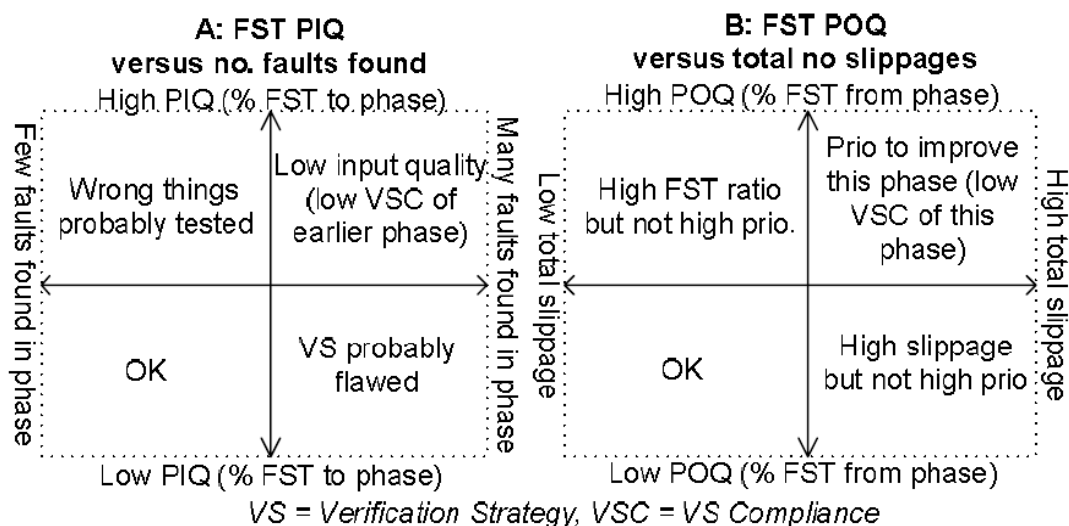
O modelo de desempenho do ABDP é montado como uma árvore de decisão (*classification tree*) construída com base no algoritmo C4.5, que consegue lidar tanto com dados discretos quanto contínuos. Técnicas de árvores de decisão são comumente utilizadas em abordagens de mineração de dados.

Identificação de Conhecimento
<ol style="list-style-type: none"> <li>1. ABDP pode ser utilizada para prever ações que geram um grande número de defeitos. No contexto de ABDP, FSS com <i>under-sampling</i> pode ser utilizado para gerar resultados aceitáveis para prever as ações que geram um grande número de defeitos.</li> <li>2. O algoritmo C4.5 utilizado para montar a árvore de decisão da abordagem ABDP tem sido utilizada em diversas áreas de pesquisa e produz bons resultados de previsão (Lim <i>et al.</i>, 2000).</li> </ol>
Descrição de Conhecimento
<ol style="list-style-type: none"> <li>1. Observações dos resultados do estudo experimental conduzido no artigo.</li> <li>2. Conforme referência (Lim <i>et al.</i>, 2000).</li> </ol>
Tipo de Estudo
<p>Estudo experimental envolvendo 66 iterações de uso da abordagem para avaliar sua eficiência (no projeto AMS-COMFT, contendo 7 módulos, 22 tarefas e 682 ações). O projeto AMS-COMFT trata de um sistema de gerenciamento de atendimentos para o ministério de finanças de Taiwan.</p>

<b>Título do Artigo</b>
Company-wide Implementation of Metrics for Early Software Fault Detection
<b>Referência Completa</b>
Damm, L., Lundberg, L., Company-wide Implementation of Metrics for Early Software Fault Detection, Proceedings of the 29 <sup>th</sup> International Conference on Software Engineering (ICSE'07), Minneapolis, 2007.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
Para cada defeito deve ser registrada a fase na qual deveria ter sido encontrado e a fase na qual foi encontrado, permitindo montar uma matriz.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Medição do FST ( <i>Faults-slip-Through</i> )
<b>Descrição do Processo ou Abordagem</b>
<p>A medida FST, inicialmente descrita em (Damm <i>et al.</i>, 2006), visa assegurar que defeitos sejam encontrados na atividade certa do desenvolvimento de software.</p> <p>O artigo menciona problemas nas métricas normalmente utilizadas como as seguintes duas: (1) a razão do número de defeitos encontrados em uma determinada na fase pelo total de número de defeitos encontrados a partir desta fase e (2) a quantidade de defeitos inseridos em uma atividade e removidos em atividades posteriores. Segundo os autores a primeira tem o problema que não encontrar defeitos nas fases posteriores não é necessariamente bom. A segunda, por sua vez, não indica quão boa a organização é em detectar defeitos rápido, uma vez que a métrica normalmente apresenta valores bem altos.</p> <p>Assim, com base na experiência da Ericsson, duas métricas FST (<i>Faults-slip-Through</i>) foram definidas e implementadas em diversas de suas unidades organizacionais. A descrição destas métricas, extraída do artigo, segue abaixo:</p> <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> <math display="block">PIQ (\% \text{ FST to phase X}) = \frac{SF^1}{PF^2} (1)</math> <p><sup>1</sup> SF (Should have Found) = No. of faults found in phase X that slipped from earlier phases.  <sup>2</sup> PF (Phase Found) = Total no. of faults found in phase X.</p> </div> <div style="text-align: center;"> <math display="block">POQ (\% \text{ FST from phase X}) = \frac{TS^1}{TF^2} (2)</math> <p><sup>1</sup> TS (Total Slippage) = Total no. of faults slipping from phase X (no matter when found).  <sup>2</sup> TF (Total Found) = Total no. of faults found in all phases.</p> </div> </div> <p>A primeira, PIQ (<i>Phase Input Quality</i>), permite prever o percentual de defeitos encontrados em uma atividade que deveriam ter sido encontrados antes. Valores altos nesta métrica podem indicar problemas. Maiores informações sobre estes problemas para</p>

análise causal podem ser obtidas a partir da segunda métrica, POQ (*Phase Output Quality*), que permite identificar em quais atividades concentrar as ações de melhoria.

Segundo os autores, estas métricas devem vir acompanhadas do número total de defeitos, onde o seguinte gráfico de quatro quadrantes de classificação pode ser utilizado para facilitar a análise.



#### Identificação de Conhecimento

O relato de experiência destaca as seguintes lições aprendidas sobre melhoria de processos com base em medição:

1. Comprometimento da gerência com melhoria é muito importante.
2. Alocação de recursos inapropriada é uma razão comum para pular verificação nas atividades iniciais de desenvolvimento.
3. Gerentes devem solicitar e acompanhar as ações de melhoria e não apenas dados de medição.
4. Apoio ferramental é um fator chave para obter sucesso com melhoria de processos baseada em medição.
5. O registro de defeitos deve ser verificado para assegurar que os relatórios são preenchidos corretamente.
6. Medição FST pode ser utilizada para quantificar os benefícios de adicionar atividades de melhoria da qualidade no início do desenvolvimento.

#### Descrição de Conhecimento

Obtido a partir da experiência de implementar medição FST em nove unidades organizacionais da Ericsson, cada uma empregando ao todo em torno de 200 pessoas.

#### Tipo de Estudo

Relato de Experiência (na Ericsson da Suécia).



## B3 – Informações Extraídas dos Artigos Recuperados na Revisão de Janeiro de 2009

<b>Título do Artigo</b>
Research issues in software fault categorization
<b>Referência Completa</b>
Ploski, J., Rohr, M., Schwenkenberg, P., Hasselbring, W., Research issues in software fault categorization, SIGSOFT Softw. Eng. Notes 32(6), 2007.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
O artigo cita vários esquemas de classificação de defeitos (faltas) e deficiências e ambigüidades nestes esquemas. Os esquemas são os de Knuth (Knuth, 1989), de Beizer (Beizer, 1990), o de Gray (Gray, 1986), ODC (Orthogonal Defect Classification) (Chillarege <i>et al.</i> , 1992), o de Eisenstadt (Eisenstadt, 1997), e o de DeMillo e Mathur (DeMillo e Mathur, 1995). Ele destaca ODC como um esquema de classificação pouco ambíguo e de fácil utilização, mas destaca que diferentes autores tem proposto diferentes esquemas e que poucos argumentos são fornecidos para justificar suas escolhas particulares.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Não tem. O foco do artigo é em esquemas de classificação de defeitos. O artigo trata especificamente de faltas (defeitos nos artefatos construídos ao longo do processo de desenvolvimento), que é o mesmo foco deste trabalho, por este motivo foi incluído na revisão.
<b>Descrição do Processo ou Abordagem</b>
Não se aplica.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. ODC é um esquema de classificação pouco ambíguo e de fácil utilização.</li> <li>2. Diferentes autores tem proposto diferentes esquemas de classificação fornecendo poucos argumentos para suas escolhas particulares.</li> </ol>
<b>Descrição de Conhecimento</b>

Obtido através de uma revisão bibliográfica e da análise dos esquemas de classificação encontrados por parte dos autores.
<b>Tipo de Estudo</b>
Revisão bibliográfica.

<b>Título do Artigo</b>
Implementing Causal Analysis and Resolution in Software Development Projects: The MiniDMAIC Approach
<b>Referência Completa</b>
Goncalves, F., Bezerra, C., Belchior, A., Coelho, C., Pires, C., Implementing Causal Analysis and Resolution in Software Development Projects: The MiniDMAIC Approach, 19th Australian Conference on Software Engineering, pp. 112-119, 2008.
<b>Fonte</b>
IEEE, INSPEC, EI Compendex.
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
MiniDMAIC.
<b>Descrição do Processo ou Abordagem</b>
<p>A abordagem MiniDMAIC simplifica a abordagem DMAIC do six sigma definindo passos simplificados para cada uma das suas atividades (<i>Define, Measure, Analyze, Improve</i> e <i>Control</i>). Nenhuma informação específica sobre como selecionar as amostras ou tratar defeitos de software é fornecida. Na atividade de determinação das causas sugere o uso do diagrama de causa e efeito, <i>brainstorming</i> e o método dos cinco porquês (<i>five whys method</i>).</p> <p>O artigo fornece ainda um mapeamento do MiniDMAIC para os objetivos específicos do processo de análise de causas de problemas e resolução do CMMI.</p>
<b>Identificação de Conhecimento</b>
1. MiniDMAIC facilita a implementação do processo de análise de causas de problemas e resolução do CMMI, pode ajudar organizações a atingir maiores níveis de maturidade, aumentar a satisfação de seus usuários e estabilizar e melhorar processos.
<b>Descrição de Conhecimento</b>
1. Suposições dos autores. Nenhum tipo de avaliação foi realizado.
<b>Tipo de Estudo</b>
Estudo Teórico.

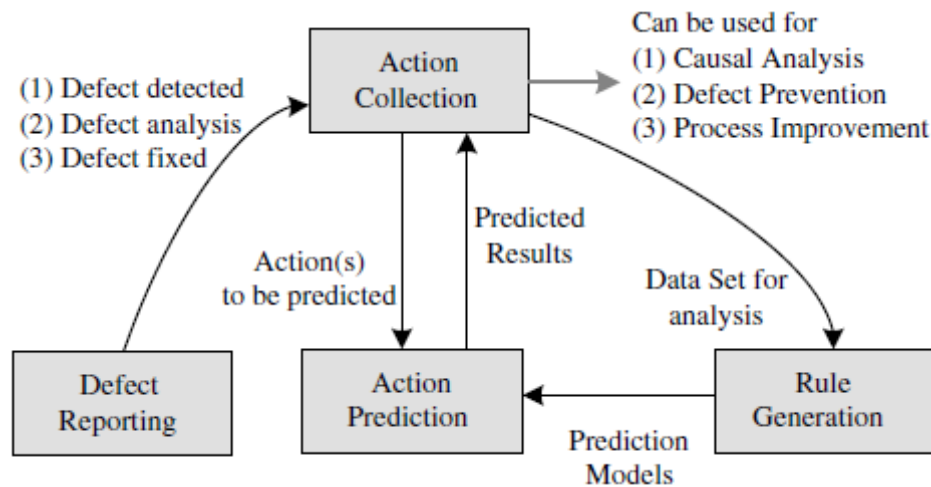
<b>Título do Artigo</b>
Towards a Defect Prevention Based Process Improvement Approach
<b>Referência Completa</b>
Kalinowski, M., Travassos, G. H., Card, D. N., Towards a Defect Prevention Based Process Improvement Approach, 34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 199-206, Parma, Italy, 2008.
<b>Fonte</b>
IEEE, INSPEC.
<b>Esquema de Classificação de Defeitos</b>
O artigo sugere o esquema de classificação de defeitos detalhado nas diretrizes para implementação de análise causal de defeitos, que podem ser encontradas em (Kalinowski <i>et al.</i> , 2008a) e no capítulo 3.
<b>Esquema de Classificação de Causas de Defeitos</b>
O artigo sugere o esquema de classificação de causas de defeitos detalhado nas diretrizes para implementação de análise causal de defeitos, que podem ser encontradas em (Kalinowski <i>et al.</i> , 2008a) e no capítulo 3.
<b>Identificação do Processo ou Abordagem</b>
DBPI – Defect prevention Based Process Improvement.
<b>Descrição do Processo ou Abordagem</b>
Versão inicial da abordagem descrita em maiores detalhes no capítulo 4.
<b>Identificação de Conhecimento</b>
1. O artigo é baseado em estudos secundários que envolvem muitas das informações extraídas neste anexo. Um apanhado do conhecimento extraído dos artigos se encontra no capítulo 2.
<b>Descrição de Conhecimento</b>
1. Obtido através de estudos secundários.
<b>Tipo de Estudo</b>
Estudo teórico envolvendo estudos secundários.

<b>Título do Artigo</b>
A defect-driven process for software quality improvement
<b>Referência Completa</b>
Robinson, B., Francis, P., Ekdahl, F., A defect-driven process for software quality improvement, in 'ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement', ACM, New York, NY, USA, pp. 333-335, 2008.
<b>Fonte</b>
ACM.
<b>Esquema de Classificação de Defeitos</b>
Adaptado do esquema de classificação de Beizer (Beizer, 1990). As oito categorias do esquema de Beizer foram aproveitadas: (i) Defeitos de requisitos, (ii) Funcionalidade implementada, (iii) Defeitos estruturais, (iv) Dados, (v) Implementação, (vi) Integração, (vii) Arquitetura do Software e (viii) Instalação e Testes. Além destas categoriais algumas subcategorias específicas foram acrescentadas por serem consideradas relevantes pela organização, tais como interface gráfica e defeitos de instalação.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem, apenas associa tipos de defeitos às atividades responsáveis por sua detecção.
<b>Identificação do Processo ou Abordagem</b>
Processo de Melhoria guiado por Defeitos ( <i>Defect-Driven Improvement Process</i> ).
<b>Descrição do Processo ou Abordagem</b>
<p>O processo tem ênfase na melhoria dos mecanismos de detecção de defeitos, tendo como resultado a redefinição da estratégia de testes adotada. Ele envolve as seguintes atividades:</p> <ul style="list-style-type: none"> <li>• Identificação dos defeitos de interesse (os de alta severidade encontrados nos testes finais e pelo cliente).</li> <li>• Seleção e ajuste da taxonomia de defeitos.</li> <li>• Coleta e classificação dos defeitos de acordo com a taxonomia.</li> <li>• Criação da nova estratégia de testes (obtida através da associação dos tipos de defeito às atividades de garantia da qualidade responsáveis por detectá-los e posteriormente determinando em qual destas atividades a detecção do defeito seria custosa).</li> </ul>

<ul style="list-style-type: none"> <li>• Criação do plano de melhoria da estratégia atual de testes (priorização a partir das atividades responsáveis pelo maior número de defeitos a serem detectados).</li> <li>• Apresentação dos resultados (<i>workshops</i> com a equipe de desenvolvimento).</li> </ul>
<b>Identificação de Conhecimento</b>
1. A classificação dos defeitos e sua posterior análise ajuda a melhorar as estratégias de detecção de defeitos.
<b>Descrição de Conhecimento</b>
Obtido através da aplicação do processo em três unidades organizacionais da ABB Corporate Research na Suécia, classificando e analisando aproximadamente 4000 defeitos. Diversas melhorias quantitativas puderam ser observadas, relacionadas à detecção mais cedo de defeitos (melhoria nos mecanismos de revisão e de testes de unidade). Em uma das unidades a quantidade de modificações em arquivos após o início dos testes foi reduzida a um sexto.
<b>Tipo de Estudo</b>
Estudo de Caso (em três unidades organizacionais da ABB Corporate Research, Suécia).

<b>Título do Artigo</b>
Integrating in-process software defect prediction with association mining to discover defect pattern
<b>Referência Completa</b>
Chang, C., Chu, C., Yeh, Y., Integrating in-process software defect prediction with association mining to discover defect pattern, Journal on Information and Software Technology 51( 2), 375 – 384, 2009.
<b>Fonte</b>
EI Compendex, Web of Science.
<b>Esquema de Classificação de Defeitos</b>
Defeitos são classificados de acordo com as seguintes características: id, descrição, ação que gerou o defeito, ação onde o defeito foi detectado, ação utilizada para remover o defeito, severidade do defeito e o módulo que causou o defeito (localização). Não há uma taxonomia para tipos de defeitos definida.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não há. As ações da WBS (work breakdown structure) são consideradas as causas dos defeitos.
<b>Identificação do Processo ou Abordagem</b>
ARDP – Association Rule based Defect Prediction.
<b>Descrição do Processo ou Abordagem</b>
<p>A abordagem ARDP (<i>Association Rule based Defect Prediction</i>) é uma evolução da abordagem ABDP (Action Based Defect Prevention, que neste artigo foi renomeada pelos autores para Action Based Defect Prediction, refletindo melhor a natureza da abordagem) (Chang e Chu, 2007). A evolução é a utilização da técnica de mineração de regras de associação (<i>association rule mining</i>) ao invés do uso de árvores de decisão. Assim, utilizando as regras de associação obtidas de ações já realizadas no projeto é possível prever as ações que geram um número grande de defeitos.</p> <p>A abordagem contém quatro componentes, ilustrados na Figura abaixo. Nesta Figura, a coleção das ações (<i>action collection</i>) diz respeito à manutenção das ações e das informações referentes a estas ações. O relato dos defeitos (<i>defect reporting</i>) captura os defeitos produzidos pelas diferentes ações. A geração de regras (<i>rule generation</i>) diz respeito à aplicação da técnica de mineração de regras de associação, permitindo relacionar características das atividades com os defeitos. Por fim, a previsão de ações (<i>action prediction</i>) utiliza as regras de associação para estimar se as ações que ainda serão realizadas no</p>

projeto geram um grande número de defeitos, de acordo com as informações a respeito destas ações mantidas na coleção das ações.



Para exemplificar uma regra, uma ação sujeita a um número grande defeitos poderia ser uma ação nova, de complexidade alta, que manipula o projeto detalhado da aplicação sendo construída.

As regras associando as características das atividades com os defeitos representam informação que pode apoiar a análise causal dos defeitos.

Identificação de Conhecimento
1. ARDP pode ser utilizada para obter um conjunto de regras de associação entre características de atividades e defeitos produzidos por estas atividades. Assim, permitindo identificar e replanejar atividades sujeitas à introdução de muitos defeitos.
Descrição de Conhecimento
1. Avaliado através de estudo experimental descrito no artigo.
Tipo de Estudo
Estudo experimental envolvendo 66 iterações de uso da abordagem para avaliar sua eficiência (no projeto AMS-COMFT, contendo 7 módulos, 22 tarefas e 682 ações). O projeto AMS-COMFT trata de um sistema de gerenciamento de atendimentos para o ministério de finanças de Taiwan.



## B4 – Informações Extraídas dos Artigos Recuperados na Revisão de Julho de 2010

<b>Título do Artigo</b>
A universal fault diagnostic expert system based on Bayesian Network
<b>Referência Completa</b>
Han, T., Li, B., Xu, L., A universal fault diagnostic expert system based on Bayesian Network, Proceedings Int. Conf. on Computer Science and Software Engineering, CSSE 2008, 260 – 263, 2008.
<b>Fonte</b>
IEEE.
<b>Esquema de Classificação de Defeitos</b>
Não tem. Lida com causas de sintomas em máquinas físicas para a produção de chips. O primeiro passo da montagem da rede bayesiana envolve listar todos os sintomas e todas as causas.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem. O primeiro passo da montagem da rede bayesiana envolve listar todos os sintomas e todas as causas.
<b>Identificação do Processo ou Abordagem</b>
Sistema Especialista Baseado em Redes Bayesianas.
<b>Descrição do Processo ou Abordagem</b>
Lida com causas de sintomas em máquinas físicas para a produção de chips. O primeiro passo da montagem da rede bayesiana envolve listar todos os sintomas e todas as causas. O segundo passo é o preenchimento de um questionário de relacionamentos causais entre os sintomas e as causas. O terceiro passo é o preenchimento das tabelas de probabilidade pelos especialistas. Posteriormente, com base nestas probabilidades, redes Bayesianas são utilizadas como máquina de inferência para encontrar as causas dos sintomas.
<b>Identificação de Conhecimento</b>
1. Redes Bayesianas construídas com base na opinião de especialistas, aplicadas ao domínio de máquinas para produção de chips, apóiam o diagnóstico correto de causas de sintomas.
<b>Descrição de Conhecimento</b>
Obtido através de um estudo de caso em que o sistema especialista foi aplicado ao domínio de máquinas de uma fábrica de chips.

Tipo de Estudo
Estudo de caso.

Título do Artigo																																																																																																																									
An Analysis of Missed Structure Field Handling Bugs																																																																																																																									
Referência Completa																																																																																																																									
Rao, R., An Analysis of Missed Structure Field Handling Bugs, Sixth International Conference on Software Engineering Research, Management and Applications, SERA '08., 41-48, 2008.																																																																																																																									
Fonte																																																																																																																									
IEEE																																																																																																																									
Esquema de Classificação de Defeitos																																																																																																																									
<p>O artigo analisa um tipo específico de defeito de código MSFH, que são omissões associadas a campos de estruturas de programação (sejam estruturas de programação estruturada ou classes de programação orientada a objetos). Assim, é criada uma taxonomia própria para este tipo de defeito, conforme a tabela abaixo.</p> <div><p>Table 1. Taxonomy of MSFH bugs</p><table><tr><th>Context</th><th>Bug class</th><th>bug cnt.</th><th>Cl. %</th></tr><tr><td rowspan="6">Context independent</td><td>Missing Initialization(I)</td><td>15</td><td>26</td></tr><tr><td>Missing Resource Releasing(R)</td><td>6</td><td>83</td></tr><tr><td>Missing Dumping(D)</td><td>4</td><td>75</td></tr><tr><td>Missing Deep Copying(Y)</td><td>1</td><td>0</td></tr><tr><td>Missing Copying(C)</td><td>4</td><td>0</td></tr><tr><td>Missing Comments(M)</td><td>2</td><td>0</td></tr><tr><td rowspan="5">Communication</td><td>Missing Addition of a Field(A)</td><td>2</td><td>0</td></tr><tr><td>Missing Field Width for Types(T)</td><td>3</td><td>66</td></tr><tr><td>Missing Field Width for Enums(E)</td><td>3</td><td>66</td></tr><tr><td>Missing Byte Order Conversion(B)</td><td>5</td><td>40</td></tr><tr><td>Missing Packing(P)</td><td>6</td><td>33</td></tr><tr><td rowspan="3">Configuration</td><td>Missing Validation(V)</td><td>6</td><td>83</td></tr><tr><td>Missing Change Handling(G)</td><td>6</td><td>50</td></tr><tr><td>Missing Attributes(H)</td><td>4</td><td>50</td></tr></table></div>		Context	Bug class	bug cnt.	Cl. %	Context independent	Missing Initialization(I)	15	26	Missing Resource Releasing(R)	6	83	Missing Dumping(D)	4	75	Missing Deep Copying(Y)	1	0	Missing Copying(C)	4	0	Missing Comments(M)	2	0	Communication	Missing Addition of a Field(A)	2	0	Missing Field Width for Types(T)	3	66	Missing Field Width for Enums(E)	3	66	Missing Byte Order Conversion(B)	5	40	Missing Packing(P)	6	33	Configuration	Missing Validation(V)	6	83	Missing Change Handling(G)	6	50	Missing Attributes(H)	4	50																																																																							
Context	Bug class	bug cnt.	Cl. %																																																																																																																						
Context independent	Missing Initialization(I)	15	26																																																																																																																						
	Missing Resource Releasing(R)	6	83																																																																																																																						
	Missing Dumping(D)	4	75																																																																																																																						
	Missing Deep Copying(Y)	1	0																																																																																																																						
	Missing Copying(C)	4	0																																																																																																																						
	Missing Comments(M)	2	0																																																																																																																						
Communication	Missing Addition of a Field(A)	2	0																																																																																																																						
	Missing Field Width for Types(T)	3	66																																																																																																																						
	Missing Field Width for Enums(E)	3	66																																																																																																																						
	Missing Byte Order Conversion(B)	5	40																																																																																																																						
	Missing Packing(P)	6	33																																																																																																																						
Configuration	Missing Validation(V)	6	83																																																																																																																						
	Missing Change Handling(G)	6	50																																																																																																																						
	Missing Attributes(H)	4	50																																																																																																																						
Esquema de Classificação de Causas de Defeitos																																																																																																																									
<p>O artigo mapeou as possíveis causas para cada um dos tipos de defeitos MSFH (omissões associadas a campos de estruturas de programação) da taxonomia, conforme a tabela abaixo.</p> <div><p>Table 2. Causes and their applicability</p><table><tr><th></th><th>I</th><th>R</th><th>D</th><th>Y</th><th>C</th><th>M</th><th>A</th><th>T</th><th>E</th><th>B</th><th>P</th><th>V</th><th>G</th><th>H</th></tr><tr><td>Focusing on Mainline</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr><tr><td>Code Spread</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td>•</td><td></td><td>•</td><td>•</td><td></td></tr><tr><td>Language Design</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td></tr><tr><td>Implicit Requirements</td><td></td><td></td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td></td></tr><tr><td>Multiproduct Nature</td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td></tr><tr><td>Implicit Assumptions</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td><td></td><td></td></tr><tr><td>Defaults</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td></td><td>•</td><td></td><td></td><td>•</td></tr></table></div>			I	R	D	Y	C	M	A	T	E	B	P	V	G	H	Focusing on Mainline	•	•	•	•	•	•	•	•	•	•	•	•	•	•	Code Spread	•	•	•	•	•					•		•	•		Language Design								•	•	•	•				Implicit Requirements			•									•	•		Multiproduct Nature							•	•	•	•	•				Implicit Assumptions	•											•			Defaults								•	•		•			•
	I	R	D	Y	C	M	A	T	E	B	P	V	G	H																																																																																																											
Focusing on Mainline	•	•	•	•	•	•	•	•	•	•	•	•	•	•																																																																																																											
Code Spread	•	•	•	•	•					•		•	•																																																																																																												
Language Design								•	•	•	•																																																																																																														
Implicit Requirements			•									•	•																																																																																																												
Multiproduct Nature							•	•	•	•	•																																																																																																														
Implicit Assumptions	•											•																																																																																																													
Defaults								•	•		•			•																																																																																																											
Identificação do Processo ou Abordagem																																																																																																																									

Análise individual de defeitos MSFH (Missing Structure Field Handling Bugs).
<b>Descrição do Processo ou Abordagem</b>
Análise individual de 67 defeitos MSFH, de dois sistemas de software de grande porte (mais de 10 milhões de linhas de código). Eles foram classificados, depois suas causas identificadas junto à equipe de desenvolvimento. Posteriormente foram identificadas possíveis soluções para cada uma destas causas e mapeadas em uma tabela.
<b>Identificação de Conhecimento</b>
1. Através da análise de defeitos MSFH chegou se a causas destes defeitos e a possíveis soluções para estas causas. Embora estas causas e soluções para este tipo de defeito sejam um conhecimento útil, a validade externa do estudo empírico é limitada, uma vez que apenas 67 defeitos foram analisados e em apenas dois sistemas.
<b>Descrição de Conhecimento</b>
Obtido através da análise individual de 67 defeitos MSFH, de dois sistemas de software de grande porte (mais de 10 milhões de linhas de código).
<b>Tipo de Estudo</b>
Estudo empírico, baseado em dados de dois projetos de grande porte.

<b>Título do Artigo</b>
Software defect prediction using intertransaction association rule mining
<b>Referência Completa</b>
Chang, C., Chu, C., Software defect prediction using intertransaction association rule mining, International Journal of Software Engineering and Knowledge Engineering, 19, 747-764, 2009.
<b>Fonte</b>
INSPEC, EI Compendex, Web of Science.
<b>Esquema de Classificação de Defeitos</b>
Defeitos são classificados de acordo com as seguintes características: id, descrição, ação que gerou o defeito, ação onde o defeito foi detectado, ação utilizada para remover o defeito, severidade do defeito e o módulo que causou o defeito (localização). Não há uma taxonomia para tipos de defeitos definida.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não há. As ações da WBS (work breakdown structure) são consideradas as causas dos defeitos.
<b>Identificação do Processo ou Abordagem</b>
InterARDP – Intertransaction Association Rule based Defect Prediction.
<b>Descrição do Processo ou Abordagem</b>
<p>A abordagem ARDP (<i>Intertransaction Association Rule-based Defect Prediction</i>) é uma evolução da abordagem ARDP (<i>Association Rule-based Defect Prediction</i>) (Chang <i>et al.</i>, 2009). A evolução é a utilização de outra técnica de mineração de regras de associação. Assim, utilizando as regras de associação obtidas de ações já realizadas no projeto é possível prever as ações que geram um número grande de defeitos.</p> <p>As regras associam as características das atividades com os defeitos e representam informação que pode apoiar também a análise causal dos defeitos.</p>
<b>Identificação de Conhecimento</b>
1. InterARDP pode ser utilizada para obter um conjunto de regras de associação entre características de atividades e defeitos produzidos por estas atividades. Assim, permitindo identificar e replanejar atividades sujeitas à introdução de muitos defeitos. As regras de associação também podem ser utilizadas para apoiar análise causal.
<b>Descrição de Conhecimento</b>
1. Avaliado através de estudo experimental descrito no artigo.
<b>Tipo de Estudo</b>
Estudo experimental envolvendo 66 iterações de uso da abordagem para avaliar

sua eficiência (no projeto AMS-COMFT, contendo 7 módulos, 22 tarefas e 682 ações). O projeto AMS-COMFT trata de um sistema de gerenciamento de atendimentos para o ministério de finanças de Taiwan.

<b>Título do Artigo</b>
Common Trends in Software Fault and Failure Data
<b>Referência Completa</b>
Hamill, M., Goseva-Popstojanova, K. Common Trends in Software Fault and Failure Data, IEEE Transactions on Software Engineering, 35, 484 -496, 2009.
<b>Fonte</b>
IEEE
<b>Esquema de Classificação de Defeitos</b>
<p>No estudo deste artigo os defeitos são agrupados como diferentes fontes de falhas, nas seguintes 12 categorias:</p> <p>Defeito de Requisitos, Defeito de Projeto, Defeito de Código, Problema de Dados, Defeito do Processo, Defeito de Integração, Defeito de Conformidade Procedural, Defeito de Entrada/Saída, Defeito de Build/Package/Merge, Defeito de Fabricação, Defeito de Simulação, Outros.</p>
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Não tem.
<b>Descrição do Processo ou Abordagem</b>
Não tem.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. Sistemas de controle de mudanças, embora não projetados com este propósito, fornecem informações muito úteis a respeito de defeitos e falhas.</li> <li>2. O numero de falhas relacionadas a defeitos que em mais de um arquivo é consistente entre os estudos. Defeitos em mais de dois arquivos: GCC (58%), NASA_conj1 (62%), Nasa_conj2 (60%). Defeitos em mais de três arquivos: GCC (31%), NASA_conj1 (44%), Nasa_conj2 (33%).</li> <li>3. Um numero significativo de falhas está relacionado a defeitos em diversos componentes: GCC (33%), NASA (6-36%).</li> <li>4. O numero de falhas relacionado a defeitos em mais de um item de configuração é consistente entre os dois conjuntos de dados independentes da NASA. Defeitos em mais de dois itens de configuração: NASA_conj1 (12%), Nasa_conj2 (19%). Defeitos em mais de três itens de configuração: NASA_conj1 (5%), Nasa_conj2 (4%).</li> <li>5. A maioria dos defeitos está contida em um percentual pequeno de arquivos. No GCC 80% dos defeitos se encontravam em 20% dos arquivos.</li> <li>6. O numero de solicitações de mudança para um item de configuração aumenta com o número de arquivos do item de configuração.</li> <li>7. Defeitos de requisitos, código e dados são os mais comuns. Nos dois projetos da NASA: Requisitos 33%, Código 33% e Dados 14%.</li> <li>8. O percentual de defeitos de codificação, interface e integração é próximo ou até</li> </ol>

<p>mesmo excede o percentual de defeitos de requisitos e projeto. Em ambos os conjuntos de dados da NASA isto ocorreu e os defeitos originados nas fases de requisitos e projeto representaram menos de 39 por cento do total, significativamente menos do que os 60-70% encontrados em Endres (1975) e Basili e Perricone (1984). A mesma situação ocorreu em outros estudos mais recentes, como (Leszak <i>et al.</i>, 2002). Estes resultados contradizem ainda a lei citada em (Rombach e Endres, 2003): “Defeitos são mais freqüentes durante as fases de requisitos e projeto”.</p>
<p><b>Descrição de Conhecimento</b></p>
<p>1-8. Conhecimento obtido em um estudo realizado utilizando dados extraídos de um sistema de controle de mudanças em projetos de grande porte: uma aplicação open-source (GCC) e dois projetos da NASA (NASA_conj1 e NASA_conj2). Adicionalmente, os resultados são comparados a outros estudos de análise de defeitos da literatura.</p>
<p><b>Tipo de Estudo</b></p>
<p>Estudo empírico utilizando dados extraídos de sistemas de controle de mudanças em projetos de grande porte (Open Source e NASA).</p>



<b>Título do Artigo</b>
Defect prevention with orthogonal defect classification
<b>Referência Completa</b>
Shenvi, A. A (2009). Defect prevention with orthogonal defect classification. Proceedings of the 2nd India Software Engineering Conference, ISEC 2009, 83 – 87.
<b>Fonte</b>
ACM, EI Compendex.
<b>Esquema de Classificação de Defeitos</b>
<p>ODC. Para cada defeito os seguintes atributos são capturados:</p> <ul style="list-style-type: none"> <li>• Atributos de abertura. Estes atributos podem ser capturados no momento da detecção do defeito. São eles: Atividade, Trigger (Gatilho, o que levou à detecção do defeito) e Impacto.</li> <li>• Atributos de fechamento. São atributos que podem ser capturados depois da correção do defeito. São eles: Alvo, Tipo, Qualificador, Fonte e Idade.</li> </ul> <p>Os tipos de defeitos padrão de ODC são interface, funcionalidade, build/package/merge, atribuição, documentação, verificação, algoritmo e timing/serialização.</p>
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Prevenção de defeitos utilizando ODC. De fato a abordagem detalha ODC, a identificação das causas de defeitos e a prevenção em si são realizadas informalmente seguindo o arcabouço do processo tradicional de prevenção de defeitos (Jones, 1985).
<b>Descrição do Processo ou Abordagem</b>
<p>Prevenção de defeitos neste artigo é descrita de forma abstrata, sem prover detalhes das técnicas efetivamente utilizadas para encontrar as causas (embora mencione Pareto, mesmo que de forma equivocada, já que a figura é de fato um diagrama de causa e efeito e não um Pareto). As atividades são: identificação dos defeitos, classificação dos defeitos, análise causal, planejamento e monitoramento de ações, medição, verificação dos resultados e aprendizado. Não são fornecidos detalhes suficientes sobre como realizar estas atividades. O momento da aplicação é em retrospectiva, depois da realização completa de uma iteração do projeto e não após cada atividade.</p> <p>As causas são encontradas informalmente através de um brainstorm bom base somente na assinatura de defeitos (distribuição dos tipos de defeitos em relação às</p>

atividades de desenvolvimento).
<b>Identificação de Conhecimento</b>
<p>Na experiência do Philips Innovation Campus de Bangalore, descrita no artigo:</p> <ol style="list-style-type: none"> <li>1. O percentual de defeitos de funcionalidade (alvo da sessão de análise causal) foi reduzido de 28% a 12% no projeto descrito no artigo. Na unidade organizacional de DVD a redução foi de 14% a 6% em 3 anos.</li> <li>2. As causas identificadas no caso específico do projeto descrito no artigo foram falta de conhecimento no domínio, espalhamento de requisitos, mal entendimento dos requisitos, requisitos implícitos não mencionados, omissão ou fato incorreto nos requisitos e tendência humana (foco na codificação).</li> <li>3. As ações implementadas para tratar estas causas foram elaborar mais a rastreabilidade, realizar workshops de requisitos, fornecer treinamentos no domínio e melhorar os mecanismos de revisão dos requisitos e do projeto antes da codificação.</li> <li>4. Defeitos de funcionalidade puderam ser encontrados mais cedo no processo de desenvolvimento.</li> <li>5. A assinatura de defeitos (distribuição dos tipos pelas atividades de desenvolvimento) ajuda a selecionar defeitos para análise.</li> <li>6. Para o propósito de prevenção de defeitos os atributos Tipo, Atividade e Qualificador (ou Natureza) foram suficientes.</li> <li>7. Lição aprendida. Defeitos de build/package/merge precisam ser capturados na fase de integração.</li> </ol>
<b>Descrição de Conhecimento</b>
Experiência da aplicação da prevenção de defeitos com ODC no Philips Innovation Campus de Bangalore.
<b>Tipo de Estudo</b>
Relato de Experiência.

<b>Título do Artigo</b>
Defect classification as problem classification for quality control in the software project management by DTL
<b>Referência Completa</b>
Hanchate, D., Sayyad, S., Shinde, S, Defect classification as problem classification for quality control in the software project management by DTL, 2nd International Conference on Computer Engineering and Technology (ICCET), V7-623 -V7-627, 2010.
<b>Fonte</b>
IEEE
<b>Esquema de Classificação de Defeitos</b>
O artigo classifica os defeitos entre as seguintes oito categorias: (1) Defeitos de Testes, (2) Defeitos de Desenvolvimento, (3) Defeitos de Projeto, (4) Defeitos Relacionados à Falta de Ferramentas, (5) Falta de Estratégia Apropriada, (6) Falta de Políticas e Apoio Organizacional, (7) Falta de Habilidades de Gerência e (8) Falta de Treinamento. Entretanto, para se manter consistente com a definição de defeito da IEEE (IEEE, 2010), nem todas estas categorias poderiam ser utilizadas para defeitos.
<b>Esquema de Classificação de Causas de Defeitos</b>
O artigo classifica as causas nas categorias utilizadas normalmente para manufatura: (1) tarefa, (2) tecnologia, (3) processo, (4) pessoas, (5) políticas e (6) produto.
<b>Identificação do Processo ou Abordagem</b>
Aplicação de Decision Tree Learning à análise causal de defeitos.
<b>Descrição do Processo ou Abordagem</b>
A abordagem elabora uma árvore de decisão para cada uma das categorias de causas, sendo a resposta de cada árvore ‘sim’ ou ‘não’. Os nós da árvore são as categorias de defeitos (embora não sejam categorias válidas) e para cada defeito deve se julgar se ele é baixo, médio ou alto, para caminhar na árvore de decisão. Os dados de treinamento do exemplo são fictícios e o não há nenhuma evidência da utilidade da árvore.
<b>Identificação de Conhecimento</b>
1. Árvores de decisão podem ser montadas com base em dados de defeitos e causas, mas não há nenhuma evidência da utilidade destas árvores.
<b>Descrição de Conhecimento</b>
Obtido através de uma prova de conceito com dados fictícios e utilizando de forma inconsistente os termos defeito, problema e causa.
<b>Tipo de Estudo</b>
Prova de conceito com dados fictícios.

Título do Artigo																																							
IEEE Standard Classification for Software Anomalies																																							
Referência Completa																																							
IEEE, IEEE Standard Classification for Software Anomalies. <i>IEEE Std 1044-2009</i> , C1 -15, 2010.																																							
Fonte																																							
IEEE, INSPEC.																																							
Esquema de Classificação de Defeitos																																							
O padrão define os seguintes atributos para classificar defeitos de software:																																							
<table border="1"> <thead> <tr> <th>Attribute</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>Defect ID</td><td>Unique identifier for the defect.</td></tr> <tr> <td>Description</td><td>Description of what is missing, wrong, or unnecessary.</td></tr> <tr> <td>Status</td><td>Current state within defect report life cycle.</td></tr> <tr> <td>Asset</td><td>The software asset (product, component, module, etc.) containing the defect.</td></tr> <tr> <td>Artifact</td><td>The specific software work product containing the defect.</td></tr> <tr> <td>Version detected</td><td>Identification of the software version in which the defect was detected.</td></tr> <tr> <td>Version corrected</td><td>Identification of the software version in which the defect was corrected.</td></tr> <tr> <td>Priority</td><td>Ranking for processing assigned by the organization responsible for the evaluation, resolution, and closure of the defect relative to other reported defects.</td></tr> <tr> <td>Severity</td><td>The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering.</td></tr> <tr> <td>Probability</td><td>Probability of recurring failure caused by this defect.</td></tr> <tr> <td>Effect</td><td>The class of requirement that is impacted by a failure caused by a defect.</td></tr> <tr> <td>Type</td><td>A categorization based on the class of code within which the defect is found or the work product within which the defect is found.</td></tr> <tr> <td>Mode</td><td>A categorization based on whether the defect is due to incorrect implementation or representation, the addition of something that is not needed, or an omission.</td></tr> <tr> <td>Insertion activity</td><td>The activity during which the defect was injected/inserted (i.e., during which the artifact containing the defect originated).</td></tr> <tr> <td>Detection activity</td><td>The activity during which the defect was detected (i.e., inspection or testing).</td></tr> <tr> <td>Failure reference(s)</td><td>Identifier of the failure(s) caused by the defect.</td></tr> <tr> <td>Change reference</td><td>Identifier of the corrective change request initiated to correct the defect.</td></tr> <tr> <td>Disposition</td><td>Final disposition of defect report upon closure.</td></tr> </tbody> </table>		Attribute	Definition	Defect ID	Unique identifier for the defect.	Description	Description of what is missing, wrong, or unnecessary.	Status	Current state within defect report life cycle.	Asset	The software asset (product, component, module, etc.) containing the defect.	Artifact	The specific software work product containing the defect.	Version detected	Identification of the software version in which the defect was detected.	Version corrected	Identification of the software version in which the defect was corrected.	Priority	Ranking for processing assigned by the organization responsible for the evaluation, resolution, and closure of the defect relative to other reported defects.	Severity	The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering.	Probability	Probability of recurring failure caused by this defect.	Effect	The class of requirement that is impacted by a failure caused by a defect.	Type	A categorization based on the class of code within which the defect is found or the work product within which the defect is found.	Mode	A categorization based on whether the defect is due to incorrect implementation or representation, the addition of something that is not needed, or an omission.	Insertion activity	The activity during which the defect was injected/inserted (i.e., during which the artifact containing the defect originated).	Detection activity	The activity during which the defect was detected (i.e., inspection or testing).	Failure reference(s)	Identifier of the failure(s) caused by the defect.	Change reference	Identifier of the corrective change request initiated to correct the defect.	Disposition	Final disposition of defect report upon closure.
Attribute	Definition																																						
Defect ID	Unique identifier for the defect.																																						
Description	Description of what is missing, wrong, or unnecessary.																																						
Status	Current state within defect report life cycle.																																						
Asset	The software asset (product, component, module, etc.) containing the defect.																																						
Artifact	The specific software work product containing the defect.																																						
Version detected	Identification of the software version in which the defect was detected.																																						
Version corrected	Identification of the software version in which the defect was corrected.																																						
Priority	Ranking for processing assigned by the organization responsible for the evaluation, resolution, and closure of the defect relative to other reported defects.																																						
Severity	The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering.																																						
Probability	Probability of recurring failure caused by this defect.																																						
Effect	The class of requirement that is impacted by a failure caused by a defect.																																						
Type	A categorization based on the class of code within which the defect is found or the work product within which the defect is found.																																						
Mode	A categorization based on whether the defect is due to incorrect implementation or representation, the addition of something that is not needed, or an omission.																																						
Insertion activity	The activity during which the defect was injected/inserted (i.e., during which the artifact containing the defect originated).																																						
Detection activity	The activity during which the defect was detected (i.e., inspection or testing).																																						
Failure reference(s)	Identifier of the failure(s) caused by the defect.																																						
Change reference	Identifier of the corrective change request initiated to correct the defect.																																						
Disposition	Final disposition of defect report upon closure.																																						
Os atributos tipo e natureza ( <i>mode</i> ) estão diretamente relacionados com a taxonomia de defeitos e, ainda de acordo com o padrão, podem assumir os seguintes valores:																																							
<ul style="list-style-type: none"> <li>• Tipo: dados, interface, lógica, descrição, sintaxe e padrões e outros.</li> <li>• Natureza (<i>mode</i>): incorreto, omitido e incluído sem ser necessário.</li> </ul>																																							
Esquema de Classificação de Causas de Defeitos																																							
Não tem.																																							
Identificação do Processo ou Abordagem																																							
Não tem.																																							
Descrição do Processo ou Abordagem																																							
Não tem.																																							
Identificação de Conhecimento																																							
<ol style="list-style-type: none"> <li>1. Atualização das definições padrão para os termos ‘defeito’, ‘erro’, ‘falha’, ‘falta’ e ‘problema’.</li> <li>2. Atributos para a classificação de defeitos.</li> <li>3. Possíveis valores para os atributos.</li> </ol>																																							

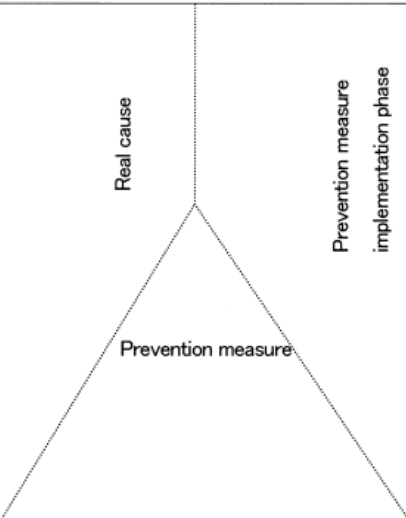
<b>Descrição de Conhecimento</b>
Padrão estabelecido por especialistas do IEEE.
<b>Tipo de Estudo</b>
Não tem.

<b>Título do Artigo</b>
Applying DPPI: A Defect Causal Analysis Approach Using Bayesian Networks
<b>Referência Completa</b>
Kalinowski, M., Travassos G.H., Mendes, E., Card, D.N., Applying DPPI: A Defect Causal Analysis Approach Using Bayesian Networks, 11th International Conference on Product Focused Software Development and Process Improvement (PROFES 2010), LNCS 6156, pp. 92–106, Limerick, Ireland, 2010.
<b>Fonte</b>
INSPEC.
<b>Esquema de Classificação de Defeitos</b>
O artigo sugere o esquema de classificação de defeitos detalhado nas diretrizes para implementação de análise causal de defeitos, que podem ser encontradas em (Kalinowski <i>et al.</i> , 2008a) e no capítulo 3.
<b>Esquema de Classificação de Causas de Defeitos</b>
O artigo sugere o esquema de classificação de causas de defeitos detalhado nas diretrizes para implementação de análise causal de defeitos, que podem ser encontradas em (Kalinowski <i>et al.</i> , 2008a) e no capítulo 3.
<b>Identificação do Processo ou Abordagem</b>
DPPI – Defect Prevention-based Process Improvement. Evolução da abordagem inicial DBPI (Kalinowski <i>et al.</i> , 2008b), contendo ajustes realizados com base na experiência de sua aplicação em um projeto real e do retorno obtido de especialistas da área.
<b>Descrição do Processo ou Abordagem</b>
Abordagem descrita em maiores detalhes no capítulo 4. A experiência de seu uso no projeto real está descrita no capítulo 5.
<b>Identificação de Conhecimento</b>
1. Em um estudo de caso envolvendo um projeto de desenvolvimento Web real de larga escala, foi possível aplicar as atividades da abordagem, indicando sua viabilidade. Adicionalmente, o uso da abordagem demonstrou ajudar a identificar corretamente as causas dos defeitos.
<b>Descrição de Conhecimento</b>
1. Obtido através de uma prova de conceito, aplicando a abordagem em um projeto real da Fundação COPPETEC.
<b>Tipo de Estudo</b>
Prova de conceito, aplicando a abordagem em um projeto real da Fundação COPPETEC.

## B5 – Informações Extraídas dos Artigos Adicionais

### Recuperados na Revisão de Julho de 2010 na SCOPUS

<b>Título do Artigo</b>
Analysis of software bug causes and its prevention
<b>Referência Completa</b>
Nakashima, T., Oyama, M., Hisada, H., Ishii, N., Analysis of software bug causes and its prevention, Journal on Information and Software Technology, 41, 1059-1068, 1999.
<b>Fonte</b>
SCOPUS
<b>Esquema de Classificação de Defeitos</b>
Este artigo não tem um esquema de classificação específico, ele classifica defeitos utilizando o método KJ (diagrama de afinidades) e como resultado nove categorias foram obtidas.
<b>Esquema de Classificação de Causas de Defeitos</b>
<p>As 200 causas identificadas para os 28 defeitos analisados no artigo foram organizadas em 23 itens, utilizando o método KJ. Os itens e sua distribuição obtida na aplicação do artigo se encontram na figura abaixo, extraída do artigo.</p>
<b>Identificação do Processo ou Abordagem</b>
Análise exhaustiva de defeitos de severidade alta.

Descrição do Processo ou Abordagem														
<p>A abordagem envolve três passos:</p> <ul style="list-style-type: none"> <li>• O preenchimento de uma planilha para cada defeito, pelo responsável pela introdução do defeito;</li> <li>• Entrevistas com os responsáveis pela introdução dos defeitos para encontrar as causas reais; e</li> <li>• Confirmação das causas reais.</li> </ul> <p>Desta forma, 200 causas reais foram encontradas para um conjunto de 28 defeitos. Estas causas foram então agrupadas em 23 itens utilizando o método KJ. Os 6 itens que mais contribuíram para os defeitos foram então analisadas e medidas de prevenção foram implementadas, conforme a tabela abaixo, extraída do artigo. A tabela destaca ainda as fases do desenvolvimento em que estas ações preventivas foram implementadas. Adicionalmente, o artigo descreve diretrizes para prevenir os defeitos de projeto de software encontrados na experiência.</p>														
1	2	3	4	5	6									
Insufficient understanding	No fully checked	It is assumed that they should be handled and corrected in the next phase.	Error in I/F specification.	Preoccupied thought.	No memos taken.	Function design	Structure design	Module design	Programming	Module debug	Integrated debug	Function debug	Testing	Total
x	x	x	x	x		Fully review.	x	x	x	x				4
x	x					Share the resources.	x	x	x	x				4
	x					Check boundary value or exceptional value.				x	x			3
	x					Share the problems among designers.	x	x	x	x	x	x	x	8
	x	x				Describe clearly problems or pending issues.	x	x	x					3
x			x			Describe precaution or expiration issues in detail.			x					1
	x		x	x		Be specific in document when ask other person to do a job.	x	x	x					3
	x		x	x		Confirm from third party's view.	x	x	x	x	x	x	x	8
3	6	2	4	4	1	6	6	7	5	3	3	2	2	

Identificação de Conhecimento														
<p>Na experiência relatada no artigo:</p> <ol style="list-style-type: none"> <li>1. 49,5% das causas dos defeitos eram referentes a categorias relacionadas ao descuido dos projetistas do software.</li> <li>2. A maior parte dos defeitos foi introduzida na fase de projeto (35,5%). E 81% dos defeitos foram introduzidos entre as fases de Especificação Funcional e Codificação.</li> <li>3. Houve uma redução de 32% na taxa de defeitos de severidade alta em um único ciclo</li> </ol>														



<p>de aplicação da abordagem. Adicionalmente, O número de defeitos causados por descuido do projetista (principal causa) foi reduzido de 49,5% do total para 29% do total. Os autores acreditam que reduções ainda maiores podem ser obtidas em ciclos posteriores de melhoria.</p> <p>4. A análise causal deve focar em encontrar as causas dos defeitos evitando acusar responsáveis por inserir os defeitos.</p>	
<b>Descrição de Conhecimento</b>	
Obtido ao aplicar a abordagem no desenvolvimento de software embarcado no Electronic Fund Transfer System Department da OMRON Corporation.	
<b>Tipo de Estudo</b>	
Relato de Experiência no Electronic Fund Transfer System Department da OMRON Corporation, no Japão.	

<b>Título do Artigo</b>
Optimum control limits for employing statistical process control in software process.
<b>Referência Completa</b>
Jalote, P., Saxena, A., Optimum control limits for employing statistical process control in software process, IEEE Transactions on Software Engineering, 28, 1126-1134, 2002.
<b>Fonte</b>
SCOPUS
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Processo para identificação de limites de controle ótimos para controlar atividades de engenharia com dados de defeitos processos de inspeção de software.
<b>Descrição do Processo ou Abordagem</b>
O processo recebe como entradas (i) a mudança esperada de desvio na densidade de defeitos em função de causas atribuíveis, (ii) o custo de um defeito não detectado, (iii) o custo de um alarme falso no diagrama e (iv) o número de revisões estáveis entre uma causa atribuível e a próxima. Com base nestas entradas ele calcula os limites de controle, ajustando sua sensibilidade para minimizar o custo total do processo.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. O gráfico XmR (<math>3\sigma</math>) não se mostra adequado para processos de software, onde limites de controle de maior sensibilidade são mais adequados, tendo em vista que em processos de software, gráficos de controle são utilizados também para controlar o projeto e não apenas para ajustes para os próximos projetos. Nos exemplos em que a abordagem foi aplicada, em função de uma análise de um modelo de custos, os gráficos XmR não se mostravam adequados.</li> <li>2. Os gráficos U são mais sensíveis do que os XmR e por este motivo são mais adequados para dados de defeitos provenientes de inspeções de software.</li> <li>3. Para controlar atividades de engenharia com base em dados de inspeções de software, limites de controle customizados e ótimos podem ser calculados em função de um modelo de custo para o processo de software. Entretanto, além do modelo de custos, dados sobre a estabilidade do processo são necessários para este cálculo.</li> <li>4. Como ponto de partida pode não ser interessante trabalhar com os limites de controle ótimos, uma vez que a sensibilidade excessiva pode não ser interessante em um momento de institucionalização de gráficos de controle estatístico.</li> <li>5. De acordo com dados citados no artigo, a InfoSys tem um processo de prevenção de</li> </ol>

defeitos estruturado e a prevenção de defeitos para uma atividade de desenvolvimento (analisar os defeitos e implementar as ações preventivas) envolve em média um esforço de apenas 5,2 homens-hora.
<b>Descrição de Conhecimento</b>
O processo foi aplicado na InfoSys (alguns dos dados de entrada do processo tiveram que ser estimados junto aos gerentes de projeto).
<b>Tipo de Estudo</b>
Estudo de Caso na InfoSys.

<b>Título do Artigo</b>
The Vital Few Versus the Trivial Many: Examining the Pareto Principle for Software.
<b>Referência Completa</b>
Gittens, M., Kim, Y., Godwin, D., The vital few versus the trivial many: Examining the Pareto principle for software, Proceedings of the 29th International Computer Software and Applications Conference (COMPSAC), 1, 179-185, 2005.
<b>Fonte</b>
SCOPUS
<b>Esquema de Classificação de Defeitos</b>
Não tem.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Não tem.
<b>Descrição do Processo ou Abordagem</b>
Não se aplica.
<b>Identificação de Conhecimento</b>
<ol style="list-style-type: none"> <li>1. No sistema analisado 80% dos defeitos encontrados nos testes se encontravam em 26% dos arquivos e todos os defeitos encontrados pelos usuários se concentravam em pouco mais de 10% dos arquivos. Assim, embora os números não correspondam aos percentuais de Pareto a idéia geral do princípio é reforçada.</li> <li>2. No sistema analisado não foi possível evidenciar concentração de complexidade em poucos arquivos. Assim, no sistema sendo avaliado, 80% da complexidade não pôde ser atribuída a 20% do código e é improvável que complexidade seja um motivo para a concentração de defeitos em arquivos. Esta observação independe da medida de complexidade utilizada, uma vez que elas mostraram ter uma forte correlação.</li> </ol>
<b>Descrição de Conhecimento</b>
Obtido através de um estudo empírico em um sistema com mais de 19 milhões de linhas de código (analisando a parte do código escrita em C/C++).
<b>Tipo de Estudo</b>
Estudo empírico em um sistema com mais de 19 milhões de linhas de código (analisando a parte do código escrita em C/C++).

<b>Título do Artigo</b>
Software defect analysis of a multi-release telecommunications system
<b>Referência Completa</b>
Leszak, M., Software defect analysis of a multi-release telecommunications system, PROFES, Lecture Notes in Computer Science, 3547, 98-114, 2005.
<b>Fonte</b>
SCOPUS
<b>Esquema de Classificação de Defeitos</b>
Considera defeitos como sendo solicitações de mudança (SM) aceitas para modificar arquivos de software. Para cada uma destas SMs os seguintes dados são capturados: severidade, release do produto em que o problema ocorreu, localização (módulo), atividade de detecção, numero de eventos de check-in relacionados à SM.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Não tem. É realizada uma análise de dados de defeitos (relacionando os com o tamanho dos arquivos, com as solicitações de mudança, etc), mas não uma análise das causas destes defeitos.
<b>Descrição do Processo ou Abordagem</b>
Não se aplica.
<b>Identificação de Conhecimento</b>
<p>No sistema embarcado para que a análise foi realizada:</p> <ol style="list-style-type: none"> <li>1. O número de defeitos e o tamanho do código do arquivo mostraram uma correlação muito fraca.</li> <li>2. A proporção de arquivos defeituosos tende a reduzir com as liberações.</li> <li>3. Tamanho e faltas em liberações anteriores, sozinhos, não são bons preditores para o número de defeitos de um arquivo em uma liberação.</li> <li>4. Defeitos encontrados pelos clientes por subsistema estão fortemente relacionados com os defeitos pré-entrega encontrados nestes subsistemas.</li> </ol>
<b>Descrição de Conhecimento</b>
Obtido através da análise de defeitos de 11 liberações (10.100 defeitos) de um sistema embarcado de larga escala na área de telecomunicações na Lucent Technologies (chamado Lambda-Unite MSS). Para a análise a evolução das mudanças foi utilizada (solicitações de mudança).
<b>Tipo de Estudo</b>
Estudo empírico.

<b>Título do Artigo</b>
Faults-slip-through - A concept for measuring the efficiency of the test process.
<b>Referência Completa</b>
Damm, L., Lundberg, L., Wohlin, C., Faults-slip-through - A concept for measuring the efficiency of the test process, Journal on Software Process Improvement and Practice, 11, 47-59, 2006.
<b>Fonte</b>
SCOPUS
<b>Esquema de Classificação de Defeitos</b>
Para cada defeito deve ser registrada a fase na qual deveria ter sido encontrado e a fase na qual foi encontrado, permitindo montar uma matriz.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Medição do FST ( <i>Faults-slip-Through</i> )
<b>Descrição do Processo ou Abordagem</b>
A mesma da descrição já realizada para o artigo (Damm e Lundberg, 2007), encontrado nas demais bases utilizadas na revisão sistemática.
<b>Identificação de Conhecimento</b>
<p>O relato de experiência destaca as seguintes lições aprendidas sobre melhoria de processos com base em medição:</p> <ol style="list-style-type: none"> <li>1. Medição FST pode ser utilizada para quantificar os benefícios de adicionar atividades de melhoria da qualidade no início do desenvolvimento.</li> <li>2. A medida PIQ permite identificar problemas no processo de desenvolvimento. A medida POQ, por sua vez, permite identificar a origem destes problemas.</li> </ol>
<b>Descrição de Conhecimento</b>
Obtido a partir da experiência de implementar medição FST na Ericsson.
<b>Tipo de Estudo</b>
Relato de Experiência (na Ericsson da Suécia).

<b>Título do Artigo</b>
The When-Who-How analysis of defects for improving the quality control process
<b>Referência Completa</b>
Jalote, P., Munshi, R., Probsting, T., The When-Who-How analysis of defects for improving the quality control process, Journal of Systems and Software, 80, 584-589, 2007.
<b>Fonte</b>
SCOPUS
<b>Esquema de Classificação de Defeitos</b>
O artigo menciona que informações comumente encontradas a respeito de defeitos em repositórios de defeitos são: componente onde o defeito se encontra, quem encontrou o defeito, método utilizado para encontrar o defeito, sintomas, data do registro e severidade.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem. O artigo apenas analisa os defeitos para melhorar os mecanismos de detecção, não visa identificar causas.
<b>Identificação do Processo ou Abordagem</b>
When-Who-How Analysis
<b>Descrição do Processo ou Abordagem</b>
O processo visa melhorar os mecanismos de controle da qualidade e não a prevenção dos defeitos em si. Assim, os defeitos são combinados nas seguintes três dimensões When (quando foram encontrados), Who (que grupo de pessoas foi responsável por encontrar o defeito) e How (como o defeito foi detectado). Depois os dados destas análises são combinados para obter um entendimento sobre o controle da qualidade em relação a tempo, equipes e técnicas de remoção de defeitos.
<b>Identificação de Conhecimento</b>
1. A aplicação da abordagem When-Who-How em uma liberação do Windows (analisando os componentes que tinham mais de 1000 defeitos) mostrou que a análise individual de cada uma das três dimensões da abordagem pode prover insights interessantes sobre os mecanismos de controle da qualidade, mas que a combinação das dimensões amplia consideravelmente as possibilidades de análise.
<b>Descrição de Conhecimento</b>
Obtido através da aplicação da abordagem When-Who-How em uma liberação do Windows (analisando os componentes que tinham mais de 1000 defeitos).
<b>Tipo de Estudo</b>
Estudo de Caso.

Título do Artigo	
Improvement of causal analysis using multivariate statistical process control	
Referência Completa	
Chang, C., Chu, C., Improvement of causal analysis using multivariate statistical process control, Software Quality Journal, 16, 377-409, 2008.	
Fonte	
SCOPUS	
Esquema de Classificação de Defeitos	
O esquema de classificação de defeitos do artigo envolve sua descrição, a atividade em que o defeito foi inserido, a atividade em que ele foi detectado, o tipo de defeito, a natureza do defeito (chamada no artigo de <i>qualifier</i> , assim como em ODC), sua severidade e a fonte. Maiores detalhes na tabela abaixo, retirada do artigo.	
Attributes	Possible values
Defect_ID	The unique identification denoting the defect
Description	Description of the reported defect
Action_Generated	The action that generates the defect
Action_Removed	The action used to remove the defect
Severity	1: Exception, 2: Minor, 3: Average, 4: Major, 5: Critical
Defect_Type	0: None, 1: Calculation/logic, 2: Error checking, 3: Algorithm, 4: Function/class, 5: Interface, 6: Others
Target	The software development stage of the Action_Generated
Qualifier	0: Missing, 1: Incorrect, 2: Extraneous
Source	0: Produced, 1: Reused from library, 2: Outsourced
Esquema de Classificação de Causas de Defeitos	
Não há, causas são derivadas a partir de medidas e testes de hipótese.	
Identificação do Processo ou Abordagem	
Multilevel Software Cause Identification (MSCI) Approach.	
Descrição do Processo ou Abordagem	
<p>A abordagem trata de causas de uma forma genérica e não apenas causas relacionadas a defeitos de software. Ele sugere o uso de <i>multivariate statistical process control charts</i>. Embora gráficos de controle estatístico de uma única variável possam ser utilizados para monitorar medidas separadamente, alguns problemas podem não ser detectados (como exemplo o artigo cita uma situação em que duas variáveis têm correlação, mas variam em direções opostas).</p> <p>A segunda parte da abordagem envolve a identificação de causas da variação. É possível identificar através de testes estatísticos as medidas que mais contribuem para a</p>	



instabilidade detectada. A partir destas medidas, é então possível identificar as medidas base que mais contribuem para a variação na medida. Com base nestas medidas base é possível realizar suposições de causas e posteriormente avaliar estas suposições através do teste de hipóteses (utilizando como um grupo os dados estáveis e como outro grupo os dados instáveis).

Entretanto, a abordagem permite identificar somente as medidas base que mais contribuem para as instabilidades (como, por exemplo, grande número de defeitos na realização das tarefas) e testar hipóteses sobre estas instabilidades estatisticamente (como, por exemplo, o aumento do número de defeitos de um determinado tipo). Identificar as causas raiz para essas variações das medidas base não é uma tarefa trivial e não está efetivamente incluído na abordagem. Adicionalmente, para testar uma hipótese é preciso ter a medida referente ou coletar dados através de questionários (para ambos os grupos do teste de hipótese).

#### **Identificação de Conhecimento**

1. Embora gráficos de controle estatístico de uma única variável possam ser utilizados para monitorar medidas separadamente, alguns problemas podem necessitar de análise através de *multivariate statistical process charts* para que sejam detectados.
2. É possível desmembrar resultados de *multivariate statistical process charts* para chegar às medidas que mais colaboram para as instabilidades.
3. Testes de hipótese podem ser utilizadas para confirmar hipóteses sobre a variação das medidas base.
4. É possível aplicar a abordagem MSCI em projetos reais.

#### **Descrição de Conhecimento**

Obtido através da aplicação retroativa da abordagem em dados de um projeto real.

#### **Tipo de Estudo**

Estudo de caso com a aplicação retroativa da abordagem (no projeto AMS-COMFT). O projeto AMS-COMFT trata de um sistema de gerenciamento de atendimentos para o ministério de finanças de Taiwan.

<b>Título do Artigo</b>
A model for software rework reduction through a combination of anomaly metrics
<b>Referência Completa</b>
Damm, L., Lundberg, L., Wohlin, C., A model for software rework reduction through a combination of anomaly metrics, Journal of Systems and Software, 81, 1968-1982, 2008.
<b>Fonte</b>
SCOPUS
<b>Esquema de Classificação de Defeitos</b>
Para cada defeito deve ser registrada a fase na qual deveria ter sido encontrado e a fase na qual foi encontrado. O artigo também menciona a severidade, o tipo de defeito e sua localização. Em relação ao tipo de defeito, houve uma tentativa de adaptar os tipos de ODC (Chillarege <i>et al.</i> , 1992), mas no fim a classificação acabou não sendo realizada.
<b>Esquema de Classificação de Causas de Defeitos</b>
Não tem.
<b>Identificação do Processo ou Abordagem</b>
Modelo de Métricas de Anomalias para Redução de Retrabalho.
<b>Descrição do Processo ou Abordagem</b>
Utiliza as métricas PIQ e POQ, enquanto a métrica PIQ indica problemas no processo, a métrica POQ indica as origens destes problemas, permitindo realizar atividades de melhoria para reduzir o retrabalho.
<b>Identificação de Conhecimento</b>
<p>O relato de experiência menciona que não foi possível medir quantitativamente os benefícios obtidos com o uso do modelo e a implementação das melhorias sugeridas, uma vez que diversas outras iniciativas de melhoria se encontravam em andamento simultaneamente. Entretanto, as seguintes lições puderam ser aprendidas durante a experiência:</p> <ol style="list-style-type: none"> <li>1. A institucionalização de métricas de produtividade pode fazer com que pessoas reajam com um comportamento antiprodutivo;</li> <li>2. Um projeto com muitos defeitos, e pouco vazamento de defeitos entre fases indica problemas na estratégia de verificação e validação;</li> <li>3. A classificação de defeitos e de suas causas pode ser difícil e frequentemente as categorias, para que se tornem efetivamente úteis, podem se tornar muito dependentes do produto e do processo;</li> <li>4. Esquemas de triggers (gatilhos, o que levou o defeito a ser detectado) são difíceis de serem utilizados se não customizados para o contexto de verificação e validação da organização;</li> <li>5. A distribuição de defeitos entre módulos é fortemente dependente do grau de modificação deste módulo.</li> <li>6. A combinação de diferentes métricas de defeitos (utilizando esquemas de classificação</li> </ol>

de defeitos, de triggers, métricas PIQ, POQ, etc) pode se mostrar poderosa e extremamente sinérgica.
7. Diferentes projetos têm diferentes problemas, o que torna algumas métricas mais úteis do que outras em determinadas situações.
8. Medição FST pode ser utilizada para quantificar os benefícios de adicionar atividades de melhoria da qualidade no início do desenvolvimento.
9. A medida PIQ permite identificar problemas no processo de desenvolvimento. A medida POQ, por sua vez, permite identificar a origem destes problemas.
<b>Descrição de Conhecimento</b>
Obtido a partir da experiência de implementar o modelo de métricas de anomalias na Ericsson.
<b>Tipo de Estudo</b>
Relato de Experiência (na Ericsson da Suécia).

Título do Artigo			
Toward an understanding of bug fix patterns			
Referência Completa			
Pan, K., Kim, S., Whitehead, J., Toward an understanding of bug fix patterns, Journal on Empirical Software Engineering, 14, 286-315, 2009.			
Fonte			
SCOPUS			
Esquema de Classificação de Defeitos			
O artigo apresenta um esquema de classificação específico para defeitos de código, com base nos padrões de modificações realizadas em de código fonte.			
Category	Pattern Name		
Assignment (AS)	Change of assignment expression		Method call with different number or types of parameters
Class Field (CF)	Addition of a class field Change of class field declaration Removal of a class field	Method Declaration (MD)	Change of method declaration Addition of a method declaration Removal of a method declaration
If-related (IF)	Addition of an else branch Addition of precondition check Addition of precondition check with jump Addition of post-condition check Change of if condition expression Removal of an else branch Removal of an if predicate	Sequence (SQ)	Addition of operations in an operation sequence of field settings Addition of operations in an operation sequence of method calls to an object Addition or removal method call operations in a short construct body Removal of operations from an operation sequence of field settings Removal of operations from an operation sequence of method calls to an object
Loop (LP)	Change of loop condition Change of the expression that modifies the loop variable	Switch (SW)	Addition/removal of switch branch
Method Call (MC)	Method call with different actual parameter values Different method call to a class instance	Try (TY)	Addition/removal of a catch block Addition/removal of a try statement
Esquema de Classificação de Causas de Defeitos			
Não tem.			
Identificação do Processo ou Abordagem			
Identificação de padrões de defeitos através da análise de correções de falhas			
Descrição do Processo ou Abordagem			
Correções de falhas foram analisadas através do atendimento a solicitações de mudança e 27 padrões de defeitos de código foram identificados. Posteriormente informações sobre estes defeitos foram extraídos automaticamente aplicando uma ferramenta de extração em dados de correção de falhas nos seguintes sete projetos open-			

source Java: Eclipse, Columba, JEdit, Scarab, ArgoUML, Lucene e MegaMek.
<b>Identificação de Conhecimento</b>
<p>Na extração dos padrões dos 7 sistemas open-source analisados:</p> <ol style="list-style-type: none"> <li>1. As categorias de defeitos mais comuns eram relacionadas à chamadas de procedimentos (21,9 a 33,1%) e relacionados à comandos condicionais IF (19,7 a 33,9%).</li> <li>2. Os tipos mais comuns eram a chamada de métodos com valores de parâmetros inadequados (14,9 a 25,5%), condicional errado no IF (5,6 a 18,6%) e atribuição equivocada (6,0 a 14,2%).</li> <li>3. Os tipos de defeitos de código tendem a ter distribuições similares entre os diferentes projetos (correlação de Pearson maior do que 85%).</li> <li>4. As taxas de introdução de defeitos por desenvolvedor também tendem a ter distribuições similares e bastante próximas das distribuições dos projetos em si.</li> <li>5. Existem situações de código que causam dificuldades independente de seu contexto e dos desenvolvedores.</li> </ol>
<b>Descrição de Conhecimento</b>
Obtido através da extração automática de padrões de defeitos com base em alterações feitas no código de sete softwares open-source desenvolvidos em Java: Eclipse, Columba, JEdit, Scarab, ArgoUML, Lucene e MegaMek.
<b>Tipo de Estudo</b>
Estudo empírico.

## ANEXO C – SUMÁRIO DAS INFORMAÇÕES EXTRAÍDAS DOS ARTIGOS RECUPERADOS

*Este anexo apresenta um sumário das informações extraídas dos artigos recuperados na revisão sistemática.*

A partir das informações extraídas dos diversos artigos, foram geradas tabelas resumindo as seguintes informações derivadas diretamente dos objetivos da revisão:

- os principais processos e abordagens utilizados para prevenção de defeitos de software (Anexo C1);
- os esquemas de classificação de defeitos utilizados nas fontes de informação (Anexo C2);
- os esquemas de classificação de causas de defeitos utilizados nas fontes de informação (Anexo C3);
- conhecimentos gerados a respeito de prevenção (análise causal e resolução) de defeitos de software (Anexo C4).

É importante ressaltar que as tabelas do Anexo B complementam estas informações, fornecendo o contexto para as mesmas, descrevendo as abordagens e os conhecimentos e listando o tipo de estudo realizado. As tabelas de sumarização das informações se encontram nas subseções seguintes.

### C1 – Informações Extraídas sobre Processos e Abordagens

A Tabela C.1 sumariza os processos e abordagens que têm sido propostos e utilizados para prevenção (análise causal e resolução) de defeitos de software, de acordo os artigos analisados.

**Tabela C.1. Processos e Abordagens.**

Processos e Abordagens	Referência
Análise de Defeitos e suas Causas.	(Endres, 1975)

Abordagem integrada ao processo para prevenção de defeitos.	(Jones, 1985)
Análise causal de defeitos de software, baseado no conceito de em ciclos de qualidade, congruente com a abordagem de (Jones, 1985).	(Philips, 1986)
Processo tradicional de prevenção de defeitos.	(Mays, 1990) e (Mays <i>et al.</i> , 1990)
Abordagem para identificar relacionamentos de causa e efeito de defeitos de software.	(Nakajo e Kume, 1991)
O artigo cita três métodos de engenharia de software para análise de causa e efeito visando melhoria de processos: análise baseada em métricas, análise da árvore de defeito ( <i>fault tree analysis</i> ) e análise baseada em casos ( <i>case-based analysis</i> ).	(Nakajo, 1991)
Abordagem para assegurar qualidade de serviço utilizando medidas baseadas nos clientes. Utiliza análise causal de defeitos seguindo o processo tradicional de prevenção de defeitos.	(Pratt, 1991)
Abordagem para melhoria de processos baseada em defeitos, utilizando <i>attribute focusing</i> .	(Bhandari <i>et al.</i> , 1993)
Análise post-mortem de defeitos de software utilizando <i>Attribute Focusing</i> .	(Bhandari e Roth, 1993)
Processo de análise causal de defeitos (compatível com o processo de prevenção de defeitos de Mays (1990)).	(Card, 1993)
Processo tradicional de prevenção de defeitos (Jones, 1985).	(Collofello e Gosalla, 1993)
Filtragem de defeitos (extensão do processo tradicional com automação de ações corretivas para código fonte).	(Troster <i>et al.</i> , 1993)
Utilização de <i>surveys</i> para melhoria de processos de software. (Análise dos defeitos de processo revelados por <i>surveys</i> ).	(Wigglesworth, 1993)
Abordagem baseada em análise causal (com diagramas de Ishikawa) para produzir e melhorar <i>checklists</i> .	(Chernak, 1996)
Processo de análise causal de falhas de software e de atrasos de cronograma implantado na Italtel SIT BUCT Línea	(Damele <i>et al.</i> , 1996)

UT. Similar ao processo de prevenção de defeitos (Jones, 1985).	
Descreve três abordagens: análise causal avulsa ( <i>one-shot root cause analysis</i> ), análise causal após projeto ( <i>post-project root cause analysis</i> ) e ciclo de melhoria contínua de processos ( <i>continuous process improvement cycle</i> ).	(Grady, 1996)
Integração da tipologia de defeitos com métricas de contenção de fase para apoiar análise causal de defeitos.	(Kelsey, 1997)
Processo de análise causal de defeitos (compatível com o processo de prevenção de defeitos de Mays (1990)).	(Card, 1998)
Abordagem que junta controle estatístico de processos com análise causal.	(Hong <i>et al.</i> , 1999)
Análise exaustiva de defeitos de severidade alta.	(Nakashima <i>et al.</i> , 1999)
Metodologia própria baseada no CMM e similar à descrita por ENDRES (1975). O processo seguido, de acordo com os autores, é congruente com o processo genérico apresentado em (CARD, 1998).	(Leszak <i>et al.</i> , 2000) e (Leszak <i>et al.</i> , 2002)
A abordagem apresentada no artigo identifica causas de defeitos a partir da aplicação de filtros (utilizando análise de correlação) relacionando a presença das causas com a ocorrência dos defeitos.	(Wohlin <i>et al.</i> , 2000b)
Processo para identificação de limites de controle ótimos para controlar atividades de engenharia com dados de defeitos processos de inspeção de software.	(Jalote e Saxena, 2002)
Abordagem para controle da qualidade utilizando modelos causais baseados em redes bayesianas.	(Krause <i>et al.</i> , 2002)
O artigo cita os processo de análise causal de defeitos (CARD, 1998) e prevenção de defeitos (MAYS <i>et al.</i> , 1990), artigos analisados neste relatório como parte da revisão sistemática.	(van Moll <i>et al.</i> , 2002)
Apoio a análise causal pela ferramenta Reason.	(Vantine <i>et al.</i> , 2002)
Modelo dinâmico baseado em redes bayesianas e relações de causa e efeito de defeitos de software.	(Gras, 2004)
Análise causal de defeitos a partir de conhecimento de especialistas.	(Jacobs <i>et al.</i> , 2004) e (Jacobs <i>et al.</i> , 2005)



Uso de mapas causais para avaliação de projetos passados visando apoiar a gerência de riscos de novos projetos.	(Al-Shebab <i>et al.</i> , 2005)
Processo análogo ao processo de análise causal de defeitos de software apresentado em (Card, 1998), com detalhamento e experiências adicionais.	(Card, 2005)
Análogo ao processo de análise causal de defeitos de software apresentado em (Card, 1998).	(Jalote e Agrawal, 2005)
Medição do FST ( <i>Faults-slip-Through</i> )	(Damm <i>et al.</i> , 2006)
Análise baseada em defeitos (Fault-Based Analysis - FBA).	(Hayes <i>et al.</i> , 2006)
Processo de Gerenciamento de Defeitos (envolve prevenção de defeitos).	(Jantti <i>et al.</i> , 2006)
Técnicas de classificação de textos com base em linguagem natural para classificar falhas de projetos open-source.	(Li <i>et al.</i> , 2006)
Metodologia para analisar defeitos com base em leitura de código e análise do histórico de mudanças deste código.	(Nakamura <i>et al.</i> , 2006)
Processo de Abstração de Erros (Error Abstraction Process – EAP).	(Walia <i>et al.</i> , 2006)
Usa análise causal de defeitos para estabilizar processos.	(Wang <i>et al.</i> , 2006)
ABDP – Action-based Defect Prevention.	(Chang e Chu, 2007)
Medição do FST ( <i>Faults-slip-Through</i> ).	(Damm e Lundberg, 2007)
When-Who-How Analysis	(Jalote <i>et al.</i> , 2007)
Multilevel Software Cause Identification (MSCI) Approach.	(Chang e Chu, 2008)
Modelo de Métricas de Anomalias para Redução de Retrabalho.	(Damm <i>et al.</i> , 2008)
MiniDMAIC.	(Gonçalves <i>et al.</i> , 2008)
Sistema Especialista Baseado em Redes Bayesianas (aplicado à diagnóstico de causas na produção de chips).	(Han <i>et al.</i> , 2008)
DBPI – Defect prevention Based Process Improvement.	(Kalinowski <i>et al.</i> , 2008)
Análise individual de defeitos MSFH (Missing Structure Field Handling Bugs).	(Rau, 2008)
Processo de Melhoria guiado por Defeitos (Defect-	(Robinson <i>et al.</i> , 2008)

Driven Improvement Process).	
ARDP – Association Rule based Defect Prediction (evolução da ABDP - (Chang e Chu, 2007)).	(Chang <i>et al.</i> , 2009)
InterARDP – Intertransaction Association Rule based Defect Prediction (evolução de ARDP).	(Chang e Chu, 2009)
Identificação de padrões de defeitos através da análise de correções de falhas.	(Pan <i>et al.</i> , 2009)
Prevenção de defeitos utilizando ODC. De fato a abordagem detalha ODC, a identificação das causas de defeitos e a prevenção em si são realizadas informalmente seguindo o arcabouço do processo tradicional de prevenção de defeitos (Jones, 1985).	(Shenvi, 2009)
Aplicação de Decision Tree Learning à análise causal de defeitos.	(Hanchate <i>et al.</i> , 2010)
DPPI – Defect Prevention-based Process Improvement. Evolução da abordagem inicial DBPI (Kalinowski <i>et al.</i> , 2008), contend ajustes realizados com base na experiência de sua aplicação em um projeto real e do retorno obtido de especialistas da área.	(Kalinowski <i>et al.</i> , 2010)

## C2 – Informações Extraídas sobre Esquemas de Classificação de Defeitos

A Tabela C.2 resume os esquemas de classificação de defeitos utilizados nos processos e abordagens de análise causal e resolução dos artigos analisados.

**Tabela C.2. Esquemas de Classificação de Defeitos.**

Esquema de Classificação de Defeitos	Referência
<p>O esquema envolve o tipo de defeito, dados de identificação, descrição do defeito e dados da correção. O artigo menciona ainda que os seguintes dados adicionais deveriam ser determinados sobre os defeitos através de sua análise: localização, momento de inserção, responsável, tipo, causa, como prevenir e como detectar.</p> <p>Foram criados três grupos (A, B e C) de tipos de defeitos para defeitos de codificação:</p> <ul style="list-style-type: none"> <li>• Os defeitos do grupo A são referentes ao mal entendimento do problema ou à escolha inadequada de um algoritmo para resolvê-lo.</li> <li>• Os defeitos do grupo B são referentes a implementações incorretas ou incompletas.</li> <li>• Os defeitos do grupo C são referentes a descuidos e que não deveriam estar presentes (como não aderência a padrões, mensagens escritas erradamente, entre outros).</li> </ul> <p>Cada um dos grupos possui então um refinamento dos tipos de defeitos (em menor granularidade) que podem ocorrer nos mesmos.</p>	(Endres, 1975)
<p>Não tem um esquema, apenas menciona que a fase em que o defeito foi inserido deve ser determinada (no artigo ela é determinada como parte da reunião de análise causal).</p>	(Jones, 1985), (Mays, 1990) e (Mays <i>et al.</i> , 1990)
<p>As categorias de defeitos utilizadas no artigo são: defeitos internos do módulo, defeitos de interface do módulo e defeitos de funcionalidade do módulo.</p>	(Nakajo e Kume, 1991)
<p>Não tem. Mas o autor menciona que o mecanismo de captura de</p>	(Pratt,

defeitos deve prover informações de tempo real, consistentes e precisas sobre o defeito. Exemplos destas informações citadas no artigo são: descrição do defeito, quando e aonde ocorreu, quando e aonde foi identificado e como se manifestou.	1991)
<p>ODC (Chillarege <i>et al.</i>, 1992), com atributos para relacionar um defeito a um conjunto de atividades do processo, conforme já sugerido no próprio ODC. Atributos do ODC: tipo de defeito, omitido/incorreto, gatilho (<i>trigger</i> - como foi detectado), origem (nova função ou correção) e impacto. Atributos para relacionar às atividades do processo: fase de detecção, fase de introdução e componente (localização).</p> <p>Os tipos são os definidos no ODC: interface, funcionalidade, build/package/merge, atribuição, documentação, verificação, algoritmo e timing/serialização.</p>	(Bhandari <i>et al.</i> , 1993)
Um questionário múltipla-escolha foi preenchido para cada defeito a ser analisado. O questionário continha perguntas sobre: componente, como foi inserido, onde deveria ter sido detectado, porque não foi detectado, qual a melhor forma de detectar esse tipo de defeito e se casos de teste seriam escritos para detectá-lo.	(Bhandari e Roth, 1993)
<p>As seguintes informações são obtidas para cada defeito: Tipo de defeito, fase em que o defeito foi encontrado, fase em que o defeito foi inserido, causa do defeito, soluções para prevenir que o defeito re-ocorra no futuro. Além disto os defeitos são priorizados de acordo com três categorias de severidade: críticos, relevantes e pouco relevantes.</p> <p>Os tipos de defeitos utilizados no artigo são específicos para defeitos de manutenção de software. Eles foram determinados baseado na experiência dos autores analisando centenas de defeitos de manutenção. Os tipos são: defeitos de projeto, incompatibilidade de interface, sincronização incorreta de código proveniente de projetos paralelos, remendo (<i>patch</i>) incorreto de objetos e exaustão de recursos do sistema.</p>	(Collofello e Gosala, 1993)
ODC (orthogonal defect classification) (Chillarege <i>et al.</i> , 1992). Os tipos de defeito são determinados de acordo com a organização.	(Chernak, 1996)
Modelo de classificação próprio da Hewlett-Packard (Grady, 1992). Registrando para cada defeito sua origem, seu tipo e o seu modo, de acordo com a figura a seguir, extraída do artigo.	(Grady, 1996)

<p>Sugere registrar além do tipo de defeito o momento em que eles foram encontrados, permitindo derivar métricas de contenção de fase (phase containment metrics).</p> <p>Em relação aos tipos de defeitos, uma tipologia de defeitos (<i>defect typology</i>), derivada da tipologia normalmente utilizada em <i>checklists</i> de inspeção (não referenciada no artigo). A nova tipologia é genérica de modo que possa ser aplicada a todas as fases de desenvolvimento. A tipologia segue.</p> <p><b>Injection Phase Types:</b></p> <table border="0"> <tr> <td>CRQ</td> <td>customer requirements were inadequately or incorrectly specified</td> </tr> <tr> <td>FRQ</td> <td>functional requirements were inadequately or incorrectly specified</td> </tr> <tr> <td>DRQ</td> <td>design requirements were inadequately or incorrectly specified</td> </tr> <tr> <td>CMD</td> <td>linguistic or algorithmic defects were introduced during coding or module development</td> </tr> <tr> <td>TS</td> <td>test specifications &amp; plans were inadequate or incorrect</td> </tr> <tr> <td>COM</td> <td>the human or document interfaces between phases of cycle were inadequate</td> </tr> </table> <p><b>Operational Context Types:</b></p> <table border="0"> <tr> <td>OP</td> <td>a defect occurs in a user-initiated operation</td> </tr> <tr> <td>FN</td> <td>a defect occurs in a function (discrete code path) supporting an operation</td> </tr> <tr> <td>UI</td> <td>a defect occurs in the user interface</td> </tr> <tr> <td>IN</td> <td>a defect occurs in the installation of the product</td> </tr> <tr> <td>UE</td> <td>a defect is dependent upon the user environment</td> </tr> <tr> <td>SC</td> <td>the product does not scale to some specific environment</td> </tr> <tr> <td>SCM</td> <td>a defect occurs due to systems incompatibility</td> </tr> </table> <p><b>Location Types:</b></p> <table border="0"> <tr> <td>IN</td> <td>a defect is located in internal code interfaces</td> </tr> <tr> <td>EN</td> <td>a defect is located in external code interfaces</td> </tr> <tr> <td>EH</td> <td>a defect is located in error/exception/recovery handling logic</td> </tr> <tr> <td>RM</td> <td>a defect is located in resource (internal or external) use</td> </tr> <tr> <td>DI</td> <td>a defect is located in data/structure initialization or use</td> </tr> </table> <p>De acordo com o autor a tipologia categoriza tanto sintomas de um defeito como suas causas.</p>	CRQ	customer requirements were inadequately or incorrectly specified	FRQ	functional requirements were inadequately or incorrectly specified	DRQ	design requirements were inadequately or incorrectly specified	CMD	linguistic or algorithmic defects were introduced during coding or module development	TS	test specifications & plans were inadequate or incorrect	COM	the human or document interfaces between phases of cycle were inadequate	OP	a defect occurs in a user-initiated operation	FN	a defect occurs in a function (discrete code path) supporting an operation	UI	a defect occurs in the user interface	IN	a defect occurs in the installation of the product	UE	a defect is dependent upon the user environment	SC	the product does not scale to some specific environment	SCM	a defect occurs due to systems incompatibility	IN	a defect is located in internal code interfaces	EN	a defect is located in external code interfaces	EH	a defect is located in error/exception/recovery handling logic	RM	a defect is located in resource (internal or external) use	DI	a defect is located in data/structure initialization or use	<p>(Kelsey, 1997)</p>
CRQ	customer requirements were inadequately or incorrectly specified																																				
FRQ	functional requirements were inadequately or incorrectly specified																																				
DRQ	design requirements were inadequately or incorrectly specified																																				
CMD	linguistic or algorithmic defects were introduced during coding or module development																																				
TS	test specifications & plans were inadequate or incorrect																																				
COM	the human or document interfaces between phases of cycle were inadequate																																				
OP	a defect occurs in a user-initiated operation																																				
FN	a defect occurs in a function (discrete code path) supporting an operation																																				
UI	a defect occurs in the user interface																																				
IN	a defect occurs in the installation of the product																																				
UE	a defect is dependent upon the user environment																																				
SC	the product does not scale to some specific environment																																				
SCM	a defect occurs due to systems incompatibility																																				
IN	a defect is located in internal code interfaces																																				
EN	a defect is located in external code interfaces																																				
EH	a defect is located in error/exception/recovery handling logic																																				
RM	a defect is located in resource (internal or external) use																																				
DI	a defect is located in data/structure initialization or use																																				
<p>O artigo menciona três dimensões úteis para classificar defeitos: momento de introdução, momento de detecção e tipo de defeito. Cita como tipos de defeitos típicos os presentes no esquema ODC: interface,</p>	<p>(Card, 1998)</p>																																				

computacional, lógico, de inicialização, estrutura de dados e documentação. Menciona que os tipos de defeito devem ser ajustados para atender às necessidades da organização.																												
O artigo menciona que informações como a natureza do defeito, o momento de inserção, o momento e método de detecção e a quantidade de retrabalho envolvidos apóiam análise causal, provendo a base para melhoria de processos.	(Fairley, 1999)																											
Utiliza, mas não tem no artigo. Apenas cita como exemplos de categorias erros de documentação, violação de padrões e projeto mal codificado, juntos responsáveis por 65% dos defeitos em dados de 32 inspeções no <i>Motorola Singapore Software Center</i> .	(Hong <i>et al.</i> , 1999)																											
Este artigo não tem um esquema de classificação específico, ele classifica defeitos utilizando o método KJ (diagrama de afinidades) e como resultado nove categorias foram obtidas.	(Nakashima <i>et al.</i> , 1999)																											
<p>As seguintes informações são armazenadas para cada defeito: atividade de detecção, localização do defeito, causas do defeito, classificação do defeito e análise de barreira (como detectar mais cedo ou prevenir a ocorrência do defeito).</p> <p>A classificação de defeitos é própria, dividindo os defeitos em três classes: implementação, interface e externos. Cada uma destas classes é então subdividida nos tipos de defeitos ilustrados na tabela a seguir.</p> <table><tr><th>Implementation</th><th>Interface</th><th>External</th></tr><tr><td>1. Data design/usage</td><td>9. Data design/usage</td><td>16. Development environment</td></tr><tr><td>2. Resource allocation/usage</td><td>10. Functionality design/usage</td><td>17. Test environment (tools/infrastructure)</td></tr><tr><td>3. Exception handling</td><td>11. Communication protocol</td><td>18. Test environment (test cases/suites)</td></tr><tr><td>4. Algorithm</td><td>12. Process coordination</td><td>19. Concurrent work (other releases)</td></tr><tr><td>5. Functionality</td><td>13. Unexpected interactions</td><td>20. Previous (inherited from previous release)</td></tr><tr><td>6. Performance</td><td>14. Change coordination</td><td>21. Other</td></tr><tr><td>7. Language pitfalls</td><td>15. Other</td><td></td></tr><tr><td>8. Other</td><td></td><td></td></tr></table> <p>A classificação conta ainda com: (1) a natureza do defeito: incorreto, incompleto e outra; (2) a redefinição da severidade do defeito (se necessário) e (3) o esforço para reproduzir, investigar e corrigir o defeito.</p> <p>Para o esforço uma escala uniforme foi utilizada: zero, menos de um dia, um a cinco dias, cinco a vinte dias e mais de vinte dias.</p>	Implementation	Interface	External	1. Data design/usage	9. Data design/usage	16. Development environment	2. Resource allocation/usage	10. Functionality design/usage	17. Test environment (tools/infrastructure)	3. Exception handling	11. Communication protocol	18. Test environment (test cases/suites)	4. Algorithm	12. Process coordination	19. Concurrent work (other releases)	5. Functionality	13. Unexpected interactions	20. Previous (inherited from previous release)	6. Performance	14. Change coordination	21. Other	7. Language pitfalls	15. Other		8. Other			(Leszak <i>et al.</i> , 2000)
Implementation	Interface	External																										
1. Data design/usage	9. Data design/usage	16. Development environment																										
2. Resource allocation/usage	10. Functionality design/usage	17. Test environment (tools/infrastructure)																										
3. Exception handling	11. Communication protocol	18. Test environment (test cases/suites)																										
4. Algorithm	12. Process coordination	19. Concurrent work (other releases)																										
5. Functionality	13. Unexpected interactions	20. Previous (inherited from previous release)																										
6. Performance	14. Change coordination	21. Other																										
7. Language pitfalls	15. Other																											
8. Other																												
Não tem. O artigo cita ODC ( <i>orthogonal defect classification</i> ) (Chillarege <i>et al.</i> , 1992), mas a abordagem apresentada pode ser aplicada somente a um tipo de defeito por vez.	(Paul <i>et al.</i> , 2000)																											
O mesmo utilizado em (LESZAK <i>et al.</i> , 2000), artigo analisado como	(Leszak <i>et</i>																											

parte desta revisão sistemática.	<i>al.</i> , 2002)
O artigo apenas comenta que diversos esquemas de classificação de defeitos podem ser utilizados, dependendo do propósito da análise de dados. Assim, diferentes atributos tais como localização, fase de introdução, sintoma e causa podem ser utilizados (Fr, 2001). Ele cita ainda um esquema específico, o Orthogonal Defect Classification (ODC) (Chillarege <i>et al.</i> , 1992).	(van Moll <i>et al.</i> , 2002)
<p>O artigo sugere três dimensões para classificar defeitos:</p> <ul style="list-style-type: none"> <li>• Momento de introdução;</li> <li>• Momento de detecção;</li> <li>• Tipo de defeito.</li> </ul> <p>Os tipos de defeito comumente utilizados são: interface, computacional, lógico, inicialização e estrutura de dados. Segundo o autor ODC (Chillarege <i>et al.</i>, 1992), que considera as dimensões acima citadas, oferece um esquema de classificação compreensível para análise causal de defeitos. Entretanto, o esquema de classificação de defeitos deve ser adaptado de maneira a apoiar as necessidades de análise da organização que está implementando análise causal de defeitos.</p>	(Card, 2005)
Menciona que organizações comumente registram além do tipo de defeito a fase de detecção e de inserção. Sugere que os tipos de defeito sejam adequados para a realidade da organização. Na experiência na InfoSys os seguintes tipos foram utilizados: Lógica, Padrões, Código Redundante, Interface Gráfica e Arquitetura.	(Jalote e Agrawal, 2005)
Considera defeitos como sendo solicitações de mudança (SM) aceitas para modificar arquivos de software. Para cada uma destas SMs os seguintes dados são capturados: severidade, release do produto em que o problema ocorreu, localização (módulo), atividade de detecção, numero de eventos de check-in relacionados à SM.	(Leszak, 2005)
Para cada defeito deve ser registrada a fase na qual deveria ter sido encontrado e a fase na qual foi encontrado, permitindo montar uma matriz.	(Damm <i>et al.</i> , 2006)
Não tem. Apresenta um estudo de caso onde questionou empresas se coletavam os seguintes dados a respeito de seus defeitos: fase de detecção, atividade que originou o defeito, tipo de defeito, causa do defeito, tempo e custo para corrigir o defeito.	(Jantti <i>et al.</i> , 2006)
A taxonomia para a classificação de defeitos se encontra descrita na	(Hayes <i>et</i>

tabela abaixo, extraída do artigo.

*al.*, 2006

Major Fault	Sub-Faults
1. Requirements	Originate in Requirements phase; found in Requirements Specifications
.1 Incompleteness	.1.1 Incomplete Decomposition .1.2 Incomplete Requirement Description
.2 Omitted/Missing	.2.1 Omitted Requirement .2.2 Missing External Constants .2.3 Missing Description of Initial System State
.3 Incorrect	.3.1 Incorrect External Constants .3.2 Incorrect Input or Output Descriptions .3.3 Incorrect Description of Initial System State .3.4 Incorrect Assignment of Resources
.4 Ambiguous	.4.1 Improper Translation .4.2 Lack of Clarity
.5 Infeasible	.-----
.6 Inconsistent	.6.1 External Conflicts .6.2 Internal Conflicts
.7 Over-specification	.-----
.8 Not Traceable	.-----
.9 [reserved for the future]	.-----
.10 Non-Verifiable	.-----
.11 Misplaced	.-----
.12 Intentional Deviation	.-----
.13 Redundant	.-----

A taxonomia é parecida com a definida por Shull (1998) e, segundo os autores, muitos artigos continham estes mesmos tipos de defeito e poucos artigos descreviam ‘novos’ tipos de defeito.

Classifica falhas de acordo com suas causas no código (os defeitos): memória, concorrência e semântica. As falhas de memória e semânticas por sua vez são subdivididas em subcategorias de acordo com a figura abaixo, extraída do artigo.

Memory Bug	Memory Leak	Failures to release unused memory.
	Uninitialized Memory Read	Read memory data before it is initialized.
	Dangling Pointer	Pointers still keep freed memory addresses.
	NULL Pointer Dereference	Dereference of a null pointer.
	Overflow	Illegal access beyond the buffer boundary.
	Double Free	One memory location is freed twice.
Semantic Bug	Missing Features	A feature is supposed to be but is not implemented.
	Missing Cases	A case in a functionality is not implemented.
	Corner Cases	Some boundary cases are considered incorrectly or ignored.
	Wrong Control Flow	The control flow is incorrectly implemented.
	Exception Handling	Do not have proper exception handling.
	Processing	Processing such as evaluation of expressions and equations is incorrect.
	Typo	Typographical mistakes.
	Other Wrong Functionality Implementation	Any other semantic bug that does not meet the design requirement.

(Li *et al.*, 2006)



<p>Sugerem um <i>template</i> contendo as seguintes informações (para defeitos encontrados em código fonte, sem documentação): problema, localização, defeito, falha, tempo para encontrar e corrigir, como foi corrigido e descrição com outras informações de contexto.</p> <p>Em relação à classificação dos defeitos, sugerem uma abordagem que primeiramente agrupa defeitos por informações de contexto similares e depois define uma categoria para estes grupamentos, abstraindo as características comuns.</p>	(Nakamura <i>et al.</i> , 2006)
<p>Defeitos são classificados de acordo com as seguintes características: id, descrição, ação que gerou o defeito, ação onde o defeito foi detectado, ação utilizada para remover o defeito, severidade do defeito e o módulo que causou o defeito (localização).</p>	(Chang e Chu, 2007)
<p>Para cada defeito deve ser registrada a fase na qual deveria ter sido encontrado e a fase na qual foi encontrado, permitindo montar uma matriz.</p>	(Damm e Lundberg, 2007)
<p>O artigo menciona que informações comumente encontradas a respeito de defeitos em repositórios de defeitos são: componente onde o defeito se encontra, quem encontrou o defeito, método utilizado para encontrar o defeito, sintomas, data do registro e severidade.</p>	(Jalote <i>et al.</i> , 2007)
<p>O artigo cita vários esquemas de classificação de defeitos (faltas) e deficiências e ambigüidades nestes esquemas. Os esquemas são os de Knuth (Knuth, 1989), de Beizer (Beizer, 1990), o de Gray (Gray, 1986), ODC (Orthogonal Defect Classification) (Chillarege <i>et al.</i>, 1992), o de Eisenstadt (Eisenstadt, 1997), e o de DeMillo e Mathur (DeMillo e Mathur, 1995). Ele destaca ODC como um esquema de classificação pouco ambíguo e de fácil utilização, mas destaca que diferentes autores têm proposto diferentes esquemas e que poucos argumentos são fornecidos para justificar suas escolhas particulares.</p>	(Ploski <i>et al.</i> , 2007)
<p>O esquema de classificação de defeitos do artigo envolve sua descrição, a atividade em que o defeito foi inserido, a atividade em que ele foi detectado, o tipo de defeito, a natureza do defeito (chamada no artigo de <i>qualifier</i>, assim como em ODC), sua severidade e a fonte. Maiores detalhes na tabela abaixo, retirada do artigo.</p>	(Chang e Chu, 2008)

Attributes	Possible values	
Defect_ID	The unique identification denoting the defect	
Description	Description of the reported defect	
Action_Generated	The action that generates the defect	
Action_Removed	The action used to remove the defect	
Severity	1: Exception, 2: Minor, 3: Average, 4: Major, 5: Critical	
Defect_Type	0: None, 1: Calculation/logic, 2: Error checking, 3: Algorithm, 4: Function/class, 5: Interface, 6: Others	
Target	The software development stage of the Action_Generated	
Qualifier	0: Missing, 1: Incorrect, 2: Extraneous	
Source	0: Produced, 1: Reused from library, 2: Outsourced	

Para cada defeito deve ser registrada a fase na qual deveria ter sido encontrado e a fase na qual foi encontrado. O artigo também menciona a severidade, o tipo de defeito e sua localização. Em relação ao tipo de defeito, houve uma tentativa de adaptar os tipos de ODC (Chillarege *et al.*, 1992), mas no fim a classificação acabou não sendo realizada.

(Damm *et al.*, 2008)

O artigo analisa um tipo específico de defeito de código MSFH, que são omissões associadas a campos de estruturas de programação (sejam estruturas de programação estruturada ou classes de programação orientada a objetos). Assim, é criada uma taxonomia própria para este tipo de defeito, conforme a tabela abaixo.

**Table 1. Taxonomy of MSFH bugs**

Context	Bug class	bug cnt.	Cl. %
Context independent	Missing Initialization(I)	15	26
	Missing Resource Releasing(R)	6	83
	Missing Dumping(D)	4	75
	Missing Deep Copying(Y)	1	0
	Missing Copying(C)	4	0
	Missing Comments(M)	2	0
Communication	Missing Addition of a Field(A)	2	0
	Missing Field Width for Types(T)	3	66
	Missing Field Width for Enums(E)	3	66
	Missing Byte Order Conversion(B)	5	40
	Missing Packing(P)	6	33
Configuration	Missing Validation(V)	6	83
	Missing Change Handling(G)	6	50
	Missing Attributes(H)	4	50

(Rao, 2008)

Adaptado do esquema de classificação de Beizer (Beizer, 1990). As oito categorias do esquema de Beizer foram aproveitadas: (i) Defeitos de requisitos, (ii) Funcionalidade implementada, (iii) Defeitos estruturais, (iv) Dados, (v) Implementação, (vi) Integração, (vii) Arquitetura do Software e (viii) Instalação e Testes. Além destas categoriais algumas subcategorias específicas foram acrescentadas por serem consideradas relevantes pela organização, tais como interface gráfica e defeitos de instalação.

(Robinson *et al.*, 2008)

<p>Defeitos são classificados de acordo com as seguintes características: id, descrição, ação que gerou o defeito, ação onde o defeito foi detectado, ação utilizada para remover o defeito, severidade do defeito e o módulo que causou o defeito (localização). Não há uma taxonomia para tipos de defeitos definida explicitamente no artigo, mas provavelmente a mesma taxonomia da definida em (Chang e Chu, 2008) foi utilizada, uma vez que se trata de dados do mesmo projeto.</p>	<p>(Chang <i>et al.</i>, 2009) e (Chang e Chu, 2009)</p>																				
<p>O artigo apresenta um esquema de classificação específico para defeitos de código, com base nos padrões de modificações realizadas em de código fonte.</p> <table border="1"> <thead> <tr> <th>Category</th><th>Pattern Name</th></tr> </thead> <tbody> <tr> <td>Assignment (AS)</td><td>Change of assignment expression</td></tr> <tr> <td>Class Field (CF)</td><td>Addition of a class field Change of class field declaration Removal of a class field</td></tr> <tr> <td></td><td>Method Declaration (MD) Change of method declaration Addition of a method declaration Removal of a method declaration</td></tr> <tr> <td>If-related (IF)</td><td>Addition of an else branch Addition of precondition check Addition of precondition check with jump Addition of post-condition check Change of if condition expression Removal of an else branch Removal of an if predicate</td></tr> <tr> <td></td><td>Sequence (SQ) Addition of operations in an operation sequence of field settings Addition of operations in an operation sequence of method calls to an object Addition or removal method call operations in a short construct body Removal of operations from an operation sequence of field settings Removal of operations from an operation sequence of method calls to an object</td></tr> <tr> <td>Loop (LP)</td><td>Change of loop condition Change of the expression that modifies the loop variable</td></tr> <tr> <td></td><td>Switch (SW) Addition/removal of switch branch</td></tr> <tr> <td>Method Call (MC)</td><td>Method call with different actual parameter values Different method call to a class instance</td></tr> <tr> <td></td><td>Try (TY) Addition/removal of a catch block Addition/removal of a try statement</td></tr> </tbody> </table>	Category	Pattern Name	Assignment (AS)	Change of assignment expression	Class Field (CF)	Addition of a class field Change of class field declaration Removal of a class field		Method Declaration (MD) Change of method declaration Addition of a method declaration Removal of a method declaration	If-related (IF)	Addition of an else branch Addition of precondition check Addition of precondition check with jump Addition of post-condition check Change of if condition expression Removal of an else branch Removal of an if predicate		Sequence (SQ) Addition of operations in an operation sequence of field settings Addition of operations in an operation sequence of method calls to an object Addition or removal method call operations in a short construct body Removal of operations from an operation sequence of field settings Removal of operations from an operation sequence of method calls to an object	Loop (LP)	Change of loop condition Change of the expression that modifies the loop variable		Switch (SW) Addition/removal of switch branch	Method Call (MC)	Method call with different actual parameter values Different method call to a class instance		Try (TY) Addition/removal of a catch block Addition/removal of a try statement	<p>(Pan <i>et al.</i>, 2009)</p>
Category	Pattern Name																				
Assignment (AS)	Change of assignment expression																				
Class Field (CF)	Addition of a class field Change of class field declaration Removal of a class field																				
	Method Declaration (MD) Change of method declaration Addition of a method declaration Removal of a method declaration																				
If-related (IF)	Addition of an else branch Addition of precondition check Addition of precondition check with jump Addition of post-condition check Change of if condition expression Removal of an else branch Removal of an if predicate																				
	Sequence (SQ) Addition of operations in an operation sequence of field settings Addition of operations in an operation sequence of method calls to an object Addition or removal method call operations in a short construct body Removal of operations from an operation sequence of field settings Removal of operations from an operation sequence of method calls to an object																				
Loop (LP)	Change of loop condition Change of the expression that modifies the loop variable																				
	Switch (SW) Addition/removal of switch branch																				
Method Call (MC)	Method call with different actual parameter values Different method call to a class instance																				
	Try (TY) Addition/removal of a catch block Addition/removal of a try statement																				
<p>ODC. Para cada defeito os seguintes atributos são capturados:</p> <ul style="list-style-type: none"> <li>Atributos de abertura. Estes atributos podem ser capturados no momento da detecção do defeito. São eles: Atividade, Trigger (Gatilho, o que levou à detecção do defeito) e Impacto.</li> <li>Atributos de fechamento. São atributos que podem ser capturados depois da correção do defeito. São eles: Alvo, Tipo, Qualificador, Fonte e Idade.</li> </ul> <p>Os tipos de defeitos padrão de ODC são interface, funcionalidade, build/package/merge, atribuição, documentação, verificação, algoritmo e timing/serialização.</p>	<p>(Shenvi, 2009)</p>																				
<p>O artigo classifica os defeitos entre as seguintes oito categorias: (1)</p>	<p>(Hanchate</p>																				

<p>Defeitos de Testes, (2) Defeitos de Desenvolvimento, (3) Defeitos de Projeto, (4) Defeitos Relacionados à Falta de Ferramentas, (5) Falta de Estratégia Apropriada, (6) Falta de Políticas e Apoio Organizacional, (7) Falta de Habilidades de Gerência e (8) Falta de Treinamento. Entretanto, para se manter consistente com a definição de defeito da IEEE (IEEE, 2010), nem todas estas categorias poderiam ser utilizadas para defeitos.</p>	<p><i>et al.</i>, 2010)</p>																																						
<p>O padrão define os seguintes atributos para classificar defeitos de software:</p> <table border="1"> <thead> <tr> <th>Attribute</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>Defect ID</td><td>Unique identifier for the defect.</td></tr> <tr> <td>Description</td><td>Description of what is missing, wrong, or unnecessary.</td></tr> <tr> <td>Status</td><td>Current state within defect report life cycle.</td></tr> <tr> <td>Asset</td><td>The software asset (product, component, module, etc.) containing the defect.</td></tr> <tr> <td>Artifact</td><td>The specific software work product containing the defect.</td></tr> <tr> <td>Version detected</td><td>Identification of the software version in which the defect was detected.</td></tr> <tr> <td>Version corrected</td><td>Identification of the software version in which the defect was corrected.</td></tr> <tr> <td>Priority</td><td>Ranking for processing assigned by the organization responsible for the evaluation, resolution, and closure of the defect relative to other reported defects.</td></tr> <tr> <td>Severity</td><td>The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering.</td></tr> <tr> <td>Probability</td><td>Probability of recurring failure caused by this defect.</td></tr> <tr> <td>Effect</td><td>The class of requirement that is impacted by a failure caused by a defect.</td></tr> <tr> <td>Type</td><td>A categorization based on the class of code within which the defect is found or the work product within which the defect is found.</td></tr> <tr> <td>Mode</td><td>A categorization based on whether the defect is due to incorrect implementation or representation, the addition of something that is not needed, or an omission.</td></tr> <tr> <td>Insertion activity</td><td>The activity during which the defect was injected/inserted (i.e., during which the artifact containing the defect originated).</td></tr> <tr> <td>Detection activity</td><td>The activity during which the defect was detected (i.e., inspection or testing).</td></tr> <tr> <td>Failure reference(s)</td><td>Identifier of the failure(s) caused by the defect.</td></tr> <tr> <td>Change reference</td><td>Identifier of the corrective change request initiated to correct the defect.</td></tr> <tr> <td>Disposition</td><td>Final disposition of defect report upon closure.</td></tr> </tbody> </table> <p>Os atributos tipo e natureza (<i>mode</i>) estão diretamente relacionados com a taxonomia de defeitos e, ainda de acordo com o padrão, podem assumir os seguintes valores:</p> <ul style="list-style-type: none"> <li>• Tipo: dados, interface, lógica, descrição, sintaxe e padrões e outros.</li> <li>• Natureza (<i>mode</i>): incorreto, omitido e incluído sem ser necessário.</li> </ul>	Attribute	Definition	Defect ID	Unique identifier for the defect.	Description	Description of what is missing, wrong, or unnecessary.	Status	Current state within defect report life cycle.	Asset	The software asset (product, component, module, etc.) containing the defect.	Artifact	The specific software work product containing the defect.	Version detected	Identification of the software version in which the defect was detected.	Version corrected	Identification of the software version in which the defect was corrected.	Priority	Ranking for processing assigned by the organization responsible for the evaluation, resolution, and closure of the defect relative to other reported defects.	Severity	The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering.	Probability	Probability of recurring failure caused by this defect.	Effect	The class of requirement that is impacted by a failure caused by a defect.	Type	A categorization based on the class of code within which the defect is found or the work product within which the defect is found.	Mode	A categorization based on whether the defect is due to incorrect implementation or representation, the addition of something that is not needed, or an omission.	Insertion activity	The activity during which the defect was injected/inserted (i.e., during which the artifact containing the defect originated).	Detection activity	The activity during which the defect was detected (i.e., inspection or testing).	Failure reference(s)	Identifier of the failure(s) caused by the defect.	Change reference	Identifier of the corrective change request initiated to correct the defect.	Disposition	Final disposition of defect report upon closure.	<p>(IEEE, 2010)</p>
Attribute	Definition																																						
Defect ID	Unique identifier for the defect.																																						
Description	Description of what is missing, wrong, or unnecessary.																																						
Status	Current state within defect report life cycle.																																						
Asset	The software asset (product, component, module, etc.) containing the defect.																																						
Artifact	The specific software work product containing the defect.																																						
Version detected	Identification of the software version in which the defect was detected.																																						
Version corrected	Identification of the software version in which the defect was corrected.																																						
Priority	Ranking for processing assigned by the organization responsible for the evaluation, resolution, and closure of the defect relative to other reported defects.																																						
Severity	The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering.																																						
Probability	Probability of recurring failure caused by this defect.																																						
Effect	The class of requirement that is impacted by a failure caused by a defect.																																						
Type	A categorization based on the class of code within which the defect is found or the work product within which the defect is found.																																						
Mode	A categorization based on whether the defect is due to incorrect implementation or representation, the addition of something that is not needed, or an omission.																																						
Insertion activity	The activity during which the defect was injected/inserted (i.e., during which the artifact containing the defect originated).																																						
Detection activity	The activity during which the defect was detected (i.e., inspection or testing).																																						
Failure reference(s)	Identifier of the failure(s) caused by the defect.																																						
Change reference	Identifier of the corrective change request initiated to correct the defect.																																						
Disposition	Final disposition of defect report upon closure.																																						

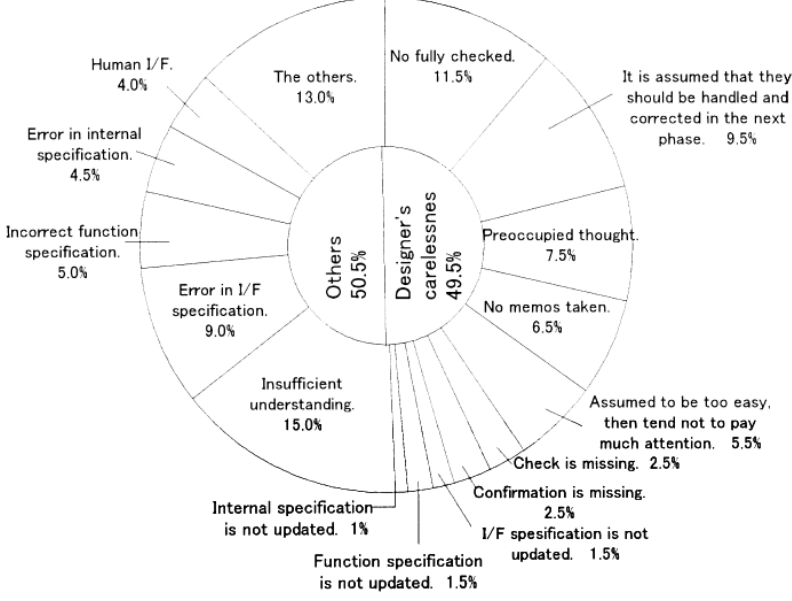
### C3 – Informações Extraídas sobre Esquemas de Classificação de Causas de Defeitos

A Tabela C.3 sumariza os esquemas de classificação de causas de defeitos que estão sendo utilizados nas abordagens de análise causal e resolução dos artigos analisados.

**Tabela C.3. Esquemas de Classificação de Causas de Defeitos.**

Esquema de Classificação de Causas de Defeitos	Referência
<p>As causas (tecnológicas ou organizacionais) são organizadas nos mesmos três grupos (A, B e C) que os defeitos:</p> <ul style="list-style-type: none"> <li>• Causas do grupo A são referentes a defeitos de mal entendimento do problema ou à escolha inadequada de um algoritmo para resolvê-lo.</li> <li>• Causas do grupo B são referentes a defeitos de implementações incorretas ou incompletas.</li> <li>• Causas do grupo C são referentes a defeitos de descuido e que não deveriam estar presentes (como não aderência a padrões, mensagens escritas erradamente, entre outros).</li> </ul> <p>Cada grupo possui então um refinamento dos tipos de causas (em menor granularidade).</p>	(Endres, 1975)
<p>O artigo cita quatro categorias de causas de defeitos, que de acordo com o mesmo poderiam ser utilizadas para qualquer defeito: Descuido, educação, comunicação e transcrição.</p>	(Jones, 1985)
<p>O artigo cita como as quatro categorias básicas de causas de defeitos: descuido, educação, comunicação e transcrição.</p>	(Mays, 1990)
<p>O artigo cita como as quatro categorias básicas de causas de defeitos: descuido, educação, comunicação e transcrição. O artigo menciona que, além destas quatro, a categoria ‘falha no processo’ tem sido utilizada em algumas organizações, mas na opinião dos autores as falhas no processo só devem ser consideradas depois que as causas humanas foram identificadas, idealmente quando as ações preventivas forem propostas.</p>	(Mays <i>et al.</i> , 1990)
<p>O artigo divide causas associadas a erros humanos nas seguintes categorias:</p> <ul style="list-style-type: none"> <li>• Erros de comunicação dentro de uma equipe de</li> </ul>	(Nakajo e Kume, 1991)

<p>desenvolvimento.</p> <ul style="list-style-type: none"> <li>• Erros de comunicação entre diferentes equipes de desenvolvimento.</li> <li>• Erros de entendimento dos requisitos.</li> <li>• Erros ao desenvolver requisitos.</li> </ul>	
<p>A CSC utiliza o mesmo esquema de classificação de defeitos sugerido por Kaoru Ishikawa (Ishikawa, 1976), contendo as seguintes quatro categorias: ferramentas, entradas, pessoas e métodos.</p>	(Card, 1993)
<p>Novamente o artigo cria seu próprio esquema de classificação, específico para causas de defeitos referentes a manutenção de software. O esquema envolve as seguintes categorias: conhecimento do sistema / experiência, comunicação, impactos de software, métodos e padrões, implantação, ferramentas de apoio, erro de origem humana.</p>	(Collofello e Gosalla, 1993)
<p>Não tem, sugere que causas identificadas sejam agrupadas em áreas relacionadas, criando assim um diagrama de afinidades (affinity diagram). Os diagramas de afinidades formam então a base para os diagramas de causa e efeito (Ishikawa, 1976).</p>	(Grady, 1996)
<p>São citadas as quatro categorias de causas descritas em (ISHIKAWA, 1976): (1) métodos, (2) ferramentas e ambiente, (3) pessoas e (4) entradas e requisitos. Novamente as categorias devem ser ajustadas para atender às necessidades da organização.</p>	(Card, 1998)
<p>Utiliza, mas não tem no artigo. Apenas cita como exemplos descuido, falta de conhecimento do domínio e projeto não suficientemente detalhado. Representando respectivamente 33%, 22% e 16% das causas nos dados de 32 inspeções no <i>Motorola Singapore Software Center</i>.</p>	(Hong <i>et al.</i> , 1999)
<p>As 200 causas identificadas para os 28 defeitos analisados no artigo foram organizadas em 23 itens, utilizando o método KJ. Os itens e sua distribuição obtida na aplicação do artigo se encontram na figura abaixo, extraída do artigo.</p>	(Nakashima <i>et al.</i> , 1999)

	
<p>As causas são classificadas de acordo com quatro classes, formando um espaço de causas de quatro dimensões:</p> <ul style="list-style-type: none"> <li>• Causas de atividade. São as atividades ou documentos em que o defeito foi introduzido. Além da atividade em si as causas de atividade podem ser qualificadas pela natureza da causa como incorreta, incompleta, ambígua, modificada, não atendendo às necessidades do usuário e não aplicável.</li> <li>• Causas relacionadas a fatores humanos. São elas: troca da coordenação, falta de conhecimento do domínio, falta de conhecimento do sistema, falta de conhecimento sobre ferramentas, falta de conhecimento sobre o processo, erro individual, introduzido na remoção de outro defeito, problema de comunicação, falta de percepção sobre a necessidade de documentação e não aplicável.</li> <li>• Causas de projeto. São elas: pressão de tempo, erro da gerência, causado por outro produto e não aplicável.</li> <li>• Causas de revisão. São elas: revisão incompleta (ou não realizada), não houve preparação suficiente, participação não adequada e não aplicável.</li> <li>• Outras causas.</li> </ul>	<p>(Leszak <i>et al.</i>, 2000) e (Leszak <i>et al.</i>, 2002)</p>
<p>O artigo apenas cita os esquemas de classificação de causas apresentados em (LESZAK <i>et al.</i>, 2000) e (MAYS <i>et al.</i>, 1990), artigos analisados neste relatório como parte da revisão sistemática.</p>	<p>(van Moll <i>et al.</i>, 2002)</p>
<p>Associa causas a categorias de risco em desenvolvimento com equipes geograficamente distribuídas (comunicação, processo, organização e tecnologia).</p>	<p>(Jacobs <i>et al.</i>, 2004) e (Jacobs <i>et al.</i>, 2005)</p>
<p>O artigo recita as quatro categorias de causa descritas por</p>	<p>(Card, 2005)</p>

<p>Ishikawa (1976): métodos, ferramentas/ambiente, pessoas e entradas (requisitos).</p> <p>Entretanto, o esquema de classificação das causas de defeitos deve ser adaptado de maneira a apoiar as necessidades de análise da organização que está implementando análise causal de defeitos.</p>	
<p>Menciona que na manufatura as categorias costumam ser pessoas, máquinas, métodos, materiais, medição e ambiente (5M e 1E) (Robitaille, 2004).</p> <p>Entretanto, sugere as seguintes categorias de causa para software: processo, pessoas e tecnologia. O artigo recita que estes são os fatores mais impactantes na qualidade e produtividade de software de acordo com (Jalote, 2000) e (SEI, 1995).</p>	<p>(Jalote e Agrawal, 2005)</p>
<p>Examinando 25 relatórios de problemas foram encontrados três tipos de causas comuns: processo inadequado, falta de entendimento dos requisitos e erro humano.</p>	<p>(Hayes <i>et al.</i>, 2006)</p>
<p>Tem uma taxonomia para classificação de erros cometidos em documentos de requisitos (Requirements Error Classification Taxonomy - RET). Estes erros são as causas dos defeitos, mas não é fornecida uma taxonomia para classificar causas desses erros. A classificação dos erros para documentos de requisitos se encontra descrita na figura abaixo, extraída do artigo.</p> <div data-bbox="319 1317 1037 1933"> <pre> graph TD     RE[Requirement Errors] --&gt; PE[People Errors]     RE --&gt; P[Process Errors]     RE --&gt; D[Documentation Errors]     subgraph PE_Box [People Errors]         PE1[1. Communication]         PE2[2. Participation]         PE3[3. Domain Knowledge]         PE4[4. Understanding Specific Application]         PE5[5. Process Execution]         PE6[6. Other human cognition]     end     subgraph P_Box [Process Errors]         P1[1. Inadequate method of achieving goal/objective]         P2[2. Management]         P3[3. Elicitation]         P4[4. Analysis]         P5[5. Traceability]     end     subgraph D_Box [Documentation Errors]         D1[1. Organization]         D2[2. No Standard Usage]         D3[3. Specification]     end </pre> </div> <p>De acordo com os autores o esquema foi elaborado com base na</p>	<p>(Walia <i>et al.</i>, 2006)</p>



literatura, em taxonomias de classificação de defeitos e dados históricos de projetos anteriores.																																																																																																																									
Não há. As ações da WBS (work breakdown structure) são consideradas as causas dos defeitos.	(Chang e Chu, 2007), (Change <i>et al.</i> , 2009) e (Chang e Chu, 2009)																																																																																																																								
Não tem, apenas associa tipos de defeitos às atividades responsáveis por sua detecção.	(Robinson <i>et al.</i> , 2007)																																																																																																																								
Não há, causas são derivadas a partir de medidas e testes de hipótese.	(Chang e Chu, 2008)																																																																																																																								
<p>O artigo mapeou as possíveis causas para cada um dos tipos de defeitos MSFH (omissões associadas a campos de estruturas de programação) da taxonomia, conforme a tabela abaixo.</p> <p><b>Table 2. Causes and their applicability</b></p> <table><tr><th></th><th>I</th><th>R</th><th>D</th><th>Y</th><th>C</th><th>M</th><th>A</th><th>T</th><th>E</th><th>B</th><th>P</th><th>V</th><th>G</th><th>H</th></tr><tr><td>Focusing on Mainline</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td></tr><tr><td>Code Spread</td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td><td></td><td>•</td><td></td><td>•</td><td>•</td><td></td></tr><tr><td>Language Design</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td></tr><tr><td>Implicit Requirements</td><td></td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td></td></tr><tr><td>Multiproduct Nature</td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td>•</td><td>•</td><td>•</td><td></td><td></td><td></td></tr><tr><td>Implicit Assumptions</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td><td></td><td></td></tr><tr><td>Defaults</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>•</td><td>•</td><td></td><td>•</td><td></td><td></td><td>•</td></tr></table>		I	R	D	Y	C	M	A	T	E	B	P	V	G	H	Focusing on Mainline	•	•	•	•	•	•	•	•	•	•	•	•	•	•	Code Spread	•	•	•	•	•					•		•	•		Language Design								•	•	•	•				Implicit Requirements		•										•	•		Multiproduct Nature							•	•	•	•	•				Implicit Assumptions	•											•			Defaults								•	•		•			•	(Rao, 2008)
	I	R	D	Y	C	M	A	T	E	B	P	V	G	H																																																																																																											
Focusing on Mainline	•	•	•	•	•	•	•	•	•	•	•	•	•	•																																																																																																											
Code Spread	•	•	•	•	•					•		•	•																																																																																																												
Language Design								•	•	•	•																																																																																																														
Implicit Requirements		•										•	•																																																																																																												
Multiproduct Nature							•	•	•	•	•																																																																																																														
Implicit Assumptions	•											•																																																																																																													
Defaults								•	•		•			•																																																																																																											
O artigo classifica as causas nas categorias utilizadas normalmente para manufatura: (1) tarefa, (2) tecnologia, (3) processo, (4) pessoas, (5) políticas e (6) produto.	(Hanchate, 2010)																																																																																																																								

## **C4 – Informações Extraídas sobre Outros Conhecimentos Relacionados a Análise Causal de Defeitos**

Para tornar outros conhecimentos identificados sobre prevenção (análise causal e resolução) de defeitos mais aplicáveis eles foram estruturados abaixo de modo a separar o conhecimento genérico, considerando o processo como um todo, do conhecimento sobre atividades e tarefas específicas. As subseções seguintes refletem esta separação.

É importante ressaltar que o conhecimento não deve ser utilizado de forma desassociada do contexto em que ele foi produzido, uma vez que muito do conhecimento é referente a afirmações que não contam com avaliações apropriadas.

### **Conhecimento Genérico sobre Prevenção de Defeitos de Software**

A Tabela C.4 agrupa o conhecimento genérico a respeito de prevenção de defeitos de software. Um maior detalhamento sobre o conhecimento e como o mesmo foi obtido encontra-se no Anexo B.

**Tabela C.4. Conhecimento Genérico a sobre Prevenção de Defeitos.**

<b>Conhecimento</b>	<b>Referência</b>
Análise causal de defeitos pode contribuir para estabelecer um catálogo de causas para defeitos.	(Endres, 1975)
Na experiência da IBM, o processo de prevenção de defeitos resultou em uma menor injeção de defeitos em cada atividade do desenvolvimento, reduzindo o número de defeitos encontrados nos testes em até 50%.	(Jones, 1985)
Na experiência da IBM, o processo de prevenção de defeitos resultou em melhorias nas áreas de comunicação e na percepção da qualidade.	(Jones, 1985)
A análise causal de defeitos deve enfatizar a prevenção de defeitos e não a detecção de defeitos.	(Jones, 1985)
Na experiência da IBM a implementação do processo de prevenção de defeitos mostrou um baixo custo.	(Jones, 1985)
Na experiência da IBM, análise causal de defeitos resultou em	(Philips, 1986)

50% de redução na taxa de defeitos encontrados no ciclo de vida.	
O processo de prevenção de defeitos pode ser aplicado nas diferentes fases de desenvolvimento. Na experiência da IBM ele foi aplicado a projeto, codificação e testes.	(Mays, 1990)
O processo de prevenção de defeitos apresenta ganhos e baixo custo. Na experiência (de mais de vinte e cinco organizações de sete laboratórios de desenvolvimento) da IBM houve redução na taxa de defeitos em mais de 50% e o custo foi de menos de meio por cento dos recursos da área de produto, em um período de 3 anos.	(Mays, 1990) e (Mays <i>et al.</i> , 1990)
Há quatro elementos chave no processo de prevenção de defeitos: (1) análise causal sistemática, (2) equipe de ação gerenciada, (3) reuniões início de fase e (4) ferramentas para coleção de dados e acompanhamento de ações.	(Mays <i>et al.</i> , 1990)
<i>Attribute focusing</i> (AF) é uma abordagem útil para análise de dados de defeitos durante o desenvolvimento, por permitir uma rápida análise de defeitos. Entretanto, defeitos que podem ser descobertos apenas através de sua descrição textual não serão encontrados por AF. A abordagem pode ser combinada de forma sinérgica com análise causal de defeitos.	(Bhandari e Roth, 1993) e (Bhandari <i>et al.</i> , 1993)
Uma premissa para a análise causal de defeitos é que os produtores (desenvolvedores e mantenedores), por possuírem mais intimidade com o processo e o produto, são os melhores para determinar como defeitos foram introduzidos no produto e como alterar o processo para prevenir que os mesmos sejam introduzidos novamente.	(Card, 1993), (Card, 1998) e (Card, 2005)
Análise causal de defeitos é uma tecnologia que ajuda a aumentar a maturidade das organizações, por facilitar: <ul style="list-style-type: none"> <li>• A percepção da qualidade;</li> <li>• O comprometimento com o processo;</li> <li>• O reconhecimento pela necessidade de medição de qualidade.</li> </ul>	(Card, 1993), (Card, 1998) e (Card, 2005)
Na CSC (Computer Sciences Corporation) com um investimento de 1,5% do orçamento do projeto foi possível obter uma redução nas taxas de defeitos em mais de 50% ao longo de dois anos.	(Card, 1993), (Card, 1998) e (Card, 2005)
Na AG Communication Systems Corporation, a aplicação de análise causal de defeitos ao processo de manutenção de software	(Collofello e Gosalla, 1993)

contribuiu para uma redução de defeitos na organização.	
Outras formas de estabelecer e aumentar a maturidade de processos de software podem ser empregadas de forma sinérgica. O modelo de maturidade do Software Engineering Institute (SEI) é um exemplo de arcabouço conceitual para melhoria de processos.	(Dalal <i>et al.</i> , 1993)
Análise causal de defeitos tem um papel central e crucial para melhorar o processo de desenvolvimento de software.	(Dalal <i>et al.</i> , 1993)
O processo tradicional de prevenção de defeitos, por focar em defeitos individuais, muitas vezes não captura defeitos sistemáticos.	(Troster <i>et al.</i> , 1993)
Filtragem de defeitos é uma abordagem eficiente para prevenir defeitos de código.	(Troster <i>et al.</i> , 1993)
Surveys ao longo do desenvolvimento podem ser utilizados para prevenir defeitos de processo com base em dados qualitativos.	(Wigglesworth, 1993)
Na experiência da Italtel SIT BUCT Línea UT a análise causal de defeitos contribuiu para um melhor entendimento das práticas de desenvolvimento e para a adoção de diversas ações corretivas.	(Damele <i>et al.</i> , 1996)
Na experiência da Italtel SIT BUCT Línea UT a análise causal de defeitos se mostrou poderosa em destacar oportunidades de melhoria.	(Damele <i>et al.</i> , 1996)
Na experiência da HP, análise causal de defeitos trouxe retornos imediatos, particularmente em visualizar o progresso. Defeitos de interface passaram de 20% para 5% do total de defeitos de software.	(Grady, 1996)
Não é difícil institucionalizar o uso de um modelo de defeitos e do processo de análise causal.	(Grady, 1996)
O uso de análise causal de defeitos é um meio de obter um balanceamento ideal entre reagir a defeitos e prevenir a ocorrência de defeitos no futuro.	(Grady, 1996)
Realizar análise causal de forma mais madura ajuda a aumentar a probabilidade de sucesso ao implementar mudanças no processo.	(Grady, 1996)
Análise causal de defeitos difere de outros métodos de avaliação de processos por apontar ações específicas ao invés de sugerir amplas áreas que requerem a atenção da equipe de melhoria de processos.	(Card, 1998) e (Card, 2005)

Apoio da gerência e respeito pelas habilidades técnicas dos desenvolvedores de software são importantes para o sucesso da implementação de análise causal de defeitos.	(Card, 1998) e (Card, 2005)
Com análise causal de defeitos as taxas de ocorrência de defeitos são reduzidas em média 50%. (Dados da IBM e da CSC).	(Card, 1998) e (Card, 2005)
O custo da implementação de análise causal varia de 0,5% a 1,5% do investimento do projeto. (Dados da IBM e da CSC)	(Card, 1998) e (Card, 2005)
Embora análise causal de defeitos seja exigida apenas nos níveis mais altos de maturidade dos modelos de qualidade, ela pode ajudar a melhorar os processos de organizações bem antes que estes níveis sejam atingidos.	(Card, 1998) e (Card, 2005)
Três princípios guiam as abordagens de análise causal de defeitos para a melhoria da qualidade: <ul style="list-style-type: none"> <li>• Reduzir defeitos para aumentar a qualidade.</li> <li>• Aplicar expertise local.</li> <li>• Focar em erros sistemáticos.</li> </ul>	(Card, 1998) e (Card, 2005)
Os pré-requisitos para se aplicar análise causal de defeitos são: <ul style="list-style-type: none"> <li>• Defeitos existem;</li> <li>• Defeitos foram documentados;</li> <li>• Existe vontade de prevenir a ocorrência dos defeitos em projetos futuros;</li> <li>• Um processo de software básico precisa estar definido para prover o arcabouço para as ações que devem ser implementadas.</li> </ul>	(Card, 1998) e (Card, 2005)
Na experiência do Motorola Singapore Software Center notou-se que mesmo em um único ciclo de produção de software, muitos eventos são numerosos e repetitivos, oferecendo oportunidades para melhoria de processos com ganhos imediatos para o ciclo em questão.	(Hong, 1999)
Na OMRON Corporation a aplicação de análise causal de defeitos resultou em uma redução de 32% na taxa de defeitos de severidade alta em um único ciclo de aplicação da abordagem. Adicionalmente, O número de defeitos causados por descuido do projetista (principal causa da análise em questão) foi reduzido de 49,5% do total para 29% do total. Os autores acreditam que reduções ainda maiores podem ser obtidas em ciclos posteriores de melhoria.	(Nakashima <i>et al.</i> , 1999)
Na experiência da Lucent Technology a análise causal de	(Leszak <i>et al.</i> ,

defeitos mostrou possuir esforço baixo e tolerável em relação aos seus aparentes benefícios.	2000)
É possível extrair informação valiosa de produtos existentes para identificar e entender as causas de defeitos de software através de análise de correlação.	(Wohlin <i>et al.</i> , 2000b)
De acordo com dados citados no artigo, a InfoSys tem um processo de prevenção de defeitos estruturado e a prevenção de defeitos para uma atividade de desenvolvimento (analisar os defeitos e implementar as ações preventivas) envolve em média um esforço de apenas 5,2 homens-hora.	(Jalote e Saxena, 2002)
Na experiência na Philips Software Centre, redes bayesianas, alimentadas a partir de dados de projetos pré-existent e complementadas com conhecimento de especialistas se mostraram adequadas para simular relacionamentos de causa e efeito e controlar a entrega de objetivos de qualidade.	(Krause <i>et al.</i> , 2002)
A detecção e a prevenção de defeitos se beneficiam de um ciclo de vida de desenvolvimento bem definido.	(van Moll <i>et al.</i> , 2002)
Na experiência da Motorola Labs, redes bayesianas, com seus fatores e respectivos relacionamentos de causa e efeito elicitados junto a especialistas, se mostraram adequadas para modelar conhecimento sobre relações de causa e efeito e fazer previsões.	(Gras, 2004)
Desenvolvimento geograficamente distribuído aumenta a probabilidade de injeção de defeitos, apresentando novas causas para defeitos e uma maior probabilidade de ocorrência para algumas causas existentes.	(Jacobs <i>et al.</i> , 2004) e (Jacobs <i>et al.</i> , 2005)
Mapas causais permitem mostrar as relações entre diferentes causas, sendo uma boa representação gráfica para discussão sobre o que deu errado dentro de um projeto.	(Al-Shebab <i>et al.</i> , 2005)
Gerentes e desenvolvedores tendem a ter visões bastante diferentes sobre o que deu errado dentro de um projeto.	(Al-Shebab <i>et al.</i> , 2005)
Diversas experiências mostram um declínio significativo na taxa de defeitos, tanto para projetos envolvendo centenas de pessoas (Mays <i>et al.</i> , 1990) (Leszak <i>et al.</i> , 2002) quanto para projetos menores (Dangerfield <i>et al.</i> , 1992) (Yu, 1998).	(Card, 2005)

Reduzir fontes sistemáticas de defeitos normalmente também provê benefícios de redução de custos e tempos de desenvolvimento.	(Card, 2005)
<p>Na experiência da InfoSys com análise causal de defeitos foi possível:</p> <ul style="list-style-type: none"> <li>• Reduzir a taxa de defeitos de 0,33 defeitos por homem-hora para 0,1 entre as primeiras duas iterações do projeto.</li> <li>• Reduzir o esforço de retrabalho de 15% para menos de 5% entre as primeiras duas iterações do projeto.</li> </ul>	(Jalote e Agrawal, 2005)
O artigo sugere que o retorno da análise causal de defeitos em um projeto incremental é maior para as primeiras iterações. Nas iterações seguintes as principais oportunidades de melhoria podem já ter sido implementadas.	(Jalote e Agrawal, 2005)
Prevenção de defeitos é considerada uma das maiores contribuições da década de 70 para a engenharia de software.	(Boehm, 2006)
No longo prazo a gerência de defeitos deve ser mais pró-ativa do que reativa. Focar em defeitos antes que eles ocorram é mais útil do que corrigir um grande número de erros repetitivos.	(Jantti <i>et al.</i> , 2006)
A aplicação da abordagem When-Who-How em uma liberação do Windows (analisando os componentes que tinham mais de 1000 defeitos) mostrou que a análise individual de cada uma das três dimensões da abordagem pode prover insights interessantes sobre os mecanismos de controle da qualidade, mas que a combinação das dimensões amplia consideravelmente as possibilidades de análise.	(Jalote <i>et al.</i> , 2007)
A classificação dos defeitos e sua posterior análise ajuda a melhorar as estratégias de detecção de defeitos.	(Robinson <i>et al.</i> , 2008)
MiniDMAIC facilita a implementação do processo de análise de causas de problemas e resolução do CMMI, pode ajudar organizações a atingir maiores níveis de maturidade, aumentar a satisfação de seus usuários e estabilizar e melhorar processos.	(Gonçalves <i>et al.</i> , 2008)
As abordagens ARDP e InterARDP podem ser utilizadas para obter um conjunto de regras de associação entre características de atividades e defeitos produzidos por estas atividades. Assim, permitindo identificar e replanejar atividades sujeitas à introdução de muitos defeitos. As regras de associação também podem ser utilizadas para apoiar análise causal.	(Chang <i>et al.</i> , 2009) e (Chang e Chu, 2009)

Sistemas de controle de mudanças, embora não projetados para este propósito, podem fornecer informações úteis a respeito de defeitos e falhas.	(Hamill e Goseva-Popstojanova, 2009)
A aplicação de prevenção de defeitos no Philips Innovation Campus (em Bangalore) o percentual de defeitos de funcionalidade (alvo da sessão de análise causal) foi reduzido de 28% a 12% no projeto descrito no artigo, em um único ciclo de melhoria. Na unidade organizacional de DVD a redução foi de 14% a 6% em 3 anos.	(Shenvi, 2009)
A aplicação da abordagem DPPI a um projeto de desenvolvimento Web real de larga escala se mostrou viável. Adicionalmente, o uso da abordagem ajudou a identificar corretamente as causas dos defeitos.	(Kalinowski, 2010)

### Conhecimento sobre Atividades e Tarefas Específicas

Muito do conhecimento extraído é relacionado a atividades e tarefas específicas. As tabelas a seguir contêm o conhecimento gerado nos artigos da revisão sistemática referente às atividades do processo de prevenção de defeitos (Mays *et al.*, 1990): Reunião de análise causal, implementação de ações, reuniões de início de fase e coleção de dados e acompanhamento. Para as reuniões de início de fase nenhum conhecimento específico foi identificado. Para as demais atividades o conhecimento identificado se encontra a seguir.

A Tabela C.5 lista o conhecimento sobre a reunião de análise causal, estruturado de acordo com os passos (Card, 2005) da reunião de análise causal onde o conhecimento se aplica.

**Tabela C.5. Conhecimento sobre a Reunião de Análise Causal.**

Classificar e Selecionar Amostras de Defeitos	Referência
Na experiência da IBM de Boeblingen, 78% dos defeitos estavam concentrados em 21% dos módulos.	(Endres, 1975)
Na experiência da IBM de Boeblingen, 85% dos defeitos puderam ser corrigidos modificando um único módulo.	(Endres, 1975)
A combinação de uma taxonomia com métricas de contenção apóia em determinar aonde um defeito ocorreu, porque foi introduzido e quando foi detectado.	(Kelsey, 1997)



A análise de mais de 20 defeitos em uma única sessão não é recomendada. O moderador deve selecionar a amostra de defeitos a serem analisados antes da reunião.	(Card, 1998)
Na experiência da OMRON Corporation descrita no artigo, a maior parte dos defeitos foi introduzida na fase de projeto (35,5%). E 81% dos defeitos foram introduzidos entre as fases de Especificação Funcional e Codificação.	(Nakashima <i>et al.</i> , 1999)
A tendência de defeitos é difícil de ser investigada quando o esquema de classificação é complicado e a amostra de defeitos é pequena.	(Leszak, 2002).
No sistema analisado 80% dos defeitos encontrados nos testes se encontravam em 26% dos arquivos e todos os defeitos encontrados pelos usuários se concentravam em pouco mais de 10% dos arquivos. Assim, embora os números não correspondam aos percentuais de Pareto a idéia geral do princípio é reforçada.	(Gittens <i>et al.</i> , 2005)
No sistema analisado não foi possível evidenciar concentração de complexidade em poucos arquivos. Assim, no sistema sendo avaliado, 80% da complexidade não pôde ser atribuída a 20% do código e é improvável que complexidade seja um motivo para a concentração de defeitos em arquivos. Esta observação independe da medida de complexidade utilizada, uma vez que elas mostraram ter uma forte correlação.	(Gittens <i>et al.</i> , 2005)
No sistema embarcado para que a análise foi realizada o número de defeitos e o tamanho do código do arquivo mostraram uma correlação muito fraca.	(Leszak, 2005)
Defeitos encontrados pelos clientes por subsistema estão fortemente relacionados com os defeitos pré-entrega encontrados nestes subsistemas.	(Leszak, 2005)
A medida PIQ (Phase Input Quality) permite identificar problemas no processo de desenvolvimento. A medida POQ (Phase Output Quality), por sua vez, permite identificar a origem destes problemas. Assim, análises causas adicionais podem ser realizadas com amostras de defeitos selecionados das fases responsáveis pelo vazamento de defeitos (indicadas pelo POQ).	(Damm <i>et al.</i> , 2006)

ODC é um esquema de classificação pouco ambíguo e de fácil utilização.	(Ploski <i>et al.</i> , 2007)
Diferentes autores têm proposto diferentes esquemas de classificação fornecendo poucos argumentos para suas escolhas particulares.	(Ploski <i>et al.</i> , 2007)
A classificação de defeitos e de suas causas pode ser difícil e frequentemente as categorias, para que se tornem efetivamente úteis, podem se tornar muito dependentes do produto e do processo.	(Damm <i>et al.</i> , 2008)
Esquemas de <i>triggers</i> (gatilhos, o que levou o defeito a ser detectado) são difíceis de serem utilizados para a classificação se não customizados para o contexto de verificação e validação da organização.	(Damm <i>et al.</i> , 2008)
A maioria dos defeitos está contida em um percentual pequeno de arquivos. No sistema analisado 80% dos defeitos se encontravam em 20% dos arquivos.	(Hamill e Goseva-Popstojanova, 2009)
O percentual de defeitos de codificação, interface e integração é próximo ou até mesmo excede o percentual de defeitos de requisitos e projeto. Em ambos os conjuntos de dados da NASA isto ocorreu e os defeitos originados nas fases de requisitos e projeto representaram menos de 39 por cento do total, significativamente menos do que os 60-70% encontrados em Endres (1975) e Basili e Perricone (1984). A mesma situação ocorreu em outros estudos mais recentes, como (Leszak <i>et al.</i> , 2002). Estes resultados contradizem ainda a lei citada em (Rombach e Endres, 2003): “Defeitos são mais freqüentes durante as fases de requisitos e projeto”.	(Hamill e Goseva-Popstojanova, 2009)
Defeitos de requisitos, código e dados são os mais comuns. Nos dois projetos da NASA: Requisitos 33%, Código 33% e Dados 14%.	(Hamill e Goseva-Popstojanova, 2009)
Na análise de defeitos de código de sete sistemas open-source Java realizada no artigo, as categorias de defeitos mais comuns foram relacionadas a chamadas de procedimentos (21,9 a 33,1%) e relacionados a comandos condicionais IF (19,7 a 33,9%).	(Pan <i>et al.</i> , 2009)
Na análise de defeitos de código de sete sistemas open-source	(Pan <i>et al.</i> , 2009)

Java realizada no artigo, os tipos mais comuns foram a chamada de métodos com valores de parâmetros inadequados (14,9 a 25,5%), condicional errado no IF (5.6 a 18.6%) e atribuição equivocada (6,0 a 14,2%).	
A assinatura de defeitos (distribuição dos tipos pelas atividades de desenvolvimento) ajuda a selecionar defeitos para análise.	(Shenvi, 2009)
Para o propósito de prevenção de defeitos os atributos Tipo, Atividade e Qualificador (ou Natureza) de ODC foram suficientes.	(Shenvi, 2009)
<b>Conhecimento sobre Encontrar Erros Sistemáticos</b>	<b>Referência</b>
Normalmente uma pequena fração de erros (menos de 3%) é responsável por aproximadamente dois terços das falhas de software reportadas (Cobb e Mills, 1990).	(Linger, 1993)
A abordagem EAP ( <i>Error Abstraction Process</i> ) pode ser utilizada para apoiar a identificação de erros sistemáticos em defeitos de software.	(Walia <i>et al.</i> , 2006)
Na análise de defeitos de código de sete sistemas open-source Java realizada no artigo, os tipos de defeitos de código tiveram distribuições similares entre os diferentes projetos (correlação de Pearson maior do que 85%). As taxas de introdução de defeitos por desenvolvedor também tendem a ter distribuições similares e bastante próximas das distribuições dos projetos em si. Isto leva a crer que existem erros sistemáticos associados a código independente de seu contexto e dos desenvolvedores.	(Pan <i>et al.</i> , 2009)
Exemplos de erros sistemáticos podem ser encontrados em diversos dos artigos analisados. Entretanto, cada um dos erros sistemáticos está vinculado ao contexto específico da análise realizada, sem validade externa.	-
<b>Determinar as Causas Principais e Propor Ações</b>	<b>Referência</b>
Na experiência da IBM de Boeblingen, aproximadamente metade de todos os defeitos eram referentes ao mal entendimento do problema e entre 60-70% dos defeitos eram relacionados a requisitos ou projeto.	(Endres, 1975)
Tentar defender seus próprios processos na hora da identificação das causas é considerado uma armadilha para a equipe de	(Card, 1993)

análise causal.	
Na experiência da AG Communication Systems Corporation, aproximadamente 80% dos defeitos gerados durante manutenção de software eram causados por conhecimento e/ou experiência insuficiente, problemas de comunicação ou falha em considerar todos os impactos da mudança.	(Collofello e Gosalla, 1993)
O diagrama de causa e efeito (Ishikawa, 1976) tem sido considerado uma ferramenta útil para determinar causas de defeitos por diversas organizações de software.	(Dalal <i>et al.</i> , 1993)
Frequentemente as causas são atribuídas a fatores humanos (no artigo em questão 49,5%). A análise causal deve focar em encontrar as causas dos defeitos evitando acusar responsáveis por inserir estes defeitos.	(Nakashima <i>et al.</i> , 1999)
Há uma significativa influência de fatores humanos na introdução de defeitos.	(Leszak <i>et al.</i> , 2000)
Diagramas de causa e efeito (Ishikawa, 1976) foram considerados uma ferramenta útil para obter as causas de defeitos baseado no conhecimento de especialistas.	(Jacobs <i>et al.</i> , 2004) e (Jacobs <i>et al.</i> , 2005)
<p>O artigo cita três condições que de acordo com Babbie (1986) precisam ser satisfeitas para que se tenha um relacionamento causal:</p> <ul style="list-style-type: none"> <li>• Deve haver uma correlação entre a hipotética causa e o seu efeito;</li> <li>• A causa precisa preceder o efeito;</li> <li>• O mecanismo ligando a causa ao defeito precisa ser identificado.</li> </ul>	(Card, 2005)
Em resultado de uma análise de 29.000 falhas em projetos open source, a maior parte dos defeitos ocasionando as falhas (mais de 80%) eram semânticos. A maior parte dos erros semânticos por sua vez são causados por implementação de funcionalidades de modo a não atender aos requisitos.	(Li <i>et al.</i> , 2006)
Na experiência descrita no artigo, redes Bayesianas construídas com base na opinião de especialistas, aplicadas ao domínio de máquinas para produção de chips apoiaram o diagnóstico correto de causas de sintomas destas máquinas.	(Han <i>et al.</i> , 2008)
Através da análise de defeitos de código do tipo MSFH ( <i>Missing</i>	(Rao, 2008)

<i>structure Field Handling</i> ) chegou se a causas destes defeitos e soluções para estas causas. Embora estas causas para este tipo de defeito e as soluções sejam um conhecimento útil, a validade externa do estudo empírico é limitada, uma vez que apenas 67 defeitos foram analisados e em apenas dois sistemas.	
Árvores de decisão podem ser montadas com base em dados de defeitos e causas, mas não há nenhuma evidência da utilidade destas árvores. A inferência de cada uma das árvores de decisão (uma para cada categoria de causa) serviria para retonar sim ou não para aquela categoria de causa.	(Hanchate <i>et al.</i> , 2010)
Exemplos de causas e ações propostas para tratá-las podem ser encontrados em diversos dos artigos analisados. Entretanto, as causas e ações estão vinculadas ao contexto específico da análise realizada, sem validade externa.	-
Documentação da Reunião	Referência
Não documentar e monitorar o processo de análise causal de defeitos é considerado uma armadilha para a equipe de análise causal.	(Card, 1993) e (Card, 2005)

A Tabela C.6 lista o conhecimento sobre a atividade de implementação das ações.

**Tabela C.6. Conhecimento sobre a Implementação das Ações.**

Conhecimento sobre Implementação das Ações	Referência
Demorar para implementar as ações (as ações devem ser gerenciadas, se as ações não são tomadas as propostas são inúteis) é considerada uma armadilha para a equipe de análise causal.	(Card, 1993) e (Card, 2005)

A Tabela C.7 lista o conhecimento sobre a atividade de coleção de dados de defeitos e acompanhamento de atividades de engenharia a partir de seus dados de defeitos.

**Tabela C.7. Conhecimento sobre a Atividade de Coleção de Dados de Defeitos e Acompanhamento de Atividades de Engenharia.**

Conhecimento sobre Coleção de Dados de Defeitos e Acompanhamento	Referência
--	------------


A coleção de dados deve ser iniciada de forma simples e o quanto antes.	(Bush, 1990)
Dados crus de incidência de defeitos não provêem meios para melhoria.	(Bush, 1990)
Abordagens estatísticas podem ser empregadas para controle de processos de desenvolvimento de software e trazer ganhos significativos.	(Dalal <i>et al.</i> , 1993)
O número de defeitos não é informação suficiente para apoiar a controlar e melhorar o processo de software.	(Dalal <i>et al.</i> , 1993)
O gráfico U é adequado para controlar estatisticamente o número de defeitos detectados por unidade inspecionada ou testada (Montgomery, 1991).	(Hong, 1999)
O gráfico XmR ( $3\sigma$ ) não se mostra adequado para processos de software, onde limites de controle de maior sensibilidade são mais adequados, tendo em vista que em processos de software, gráficos de controle são utilizados também para controlar o projeto e não apenas para ajustes para os próximos projetos. Nos exemplos em que a abordagem do artigo foi aplicada, em função de uma análise de um modelo de custos, os gráficos XmR não se mostravam adequados.	(Jalote e Saxena, 2002)
Os gráficos U são mais sensíveis do que os XmR e por este motivo são mais adequados para dados de defeitos provenientes de inspeções de software.	(Jalote e Saxena, 2002)
Para controlar atividades de engenharia com base em dados de inspeções de software, limites de controle customizados e ótimos podem ser calculados em função de um modelo de custo para o processo de software. Entretanto, além do modelo de custos, dados sobre a estabilidade do processo são necessários para este cálculo.  Como ponto de partida pode não ser interessante trabalhar com os limites de controle ótimos, uma vez que a sensibilidade excessiva pode não ser interessante em um momento de institucionalização de gráficos de controle estatístico.	(Jalote e Saxena, 2002)
Para gráficos de controle a respeito de defeitos o gráfico U costuma ser adequado. Ele se aplica a dados com distribuição de Poisson, com uma área de oportunidade variada (por exemplo,	(Card, 2005)

tamanho de um documento) para que o evento ocorra (por exemplo, detecção de um defeito).	
A distribuição dos tipos de defeitos pode ser um indicador de curto prazo mais sensível para mudanças de processos do que a taxa de defeitos em si.	(Card, 2005)
Embora gráficos de controle estatístico de uma única variável possam ser utilizados para monitorar medidas separadamente, alguns problemas podem necessitar de análise através de multivariate statistical process charts para que sejam detectados. Posteriormente, é possível desmembrar resultados de multivariate statistical process charts para chegar às medidas que mais colaboram para as instabilidades.	(Chang e Chu, 2008)
Na experiência da Ericsson da Suécia, a medição FST (Faults-slip-through) pôde ser utilizada para quantificar os benefícios de adicionar atividades de melhoria da qualidade (incluindo prevenção de defeitos) no início do desenvolvimento.	(Damm <i>et al.</i> , 2006), (Damm e Lundberg, 2007) e (Damm <i>et al.</i> , 2008)
A combinação de diferentes métricas de defeitos (utilizando esquemas de classificação de defeitos, de triggers, métricas PIQ, POQ, etc) pode se mostrar poderosa e extremamente sinérgica.	(Damm <i>et al.</i> , 2008)

# ANEXO D – TEMPLATE PARA APOIAR A APLICAÇÃO MANUAL DE DPPI

*Este anexo apresenta um template que pode ser utilizado para apoiar a aplicação manual de DPPI.*

## D1 – DPPI Template

	<b>DPPI Template</b>	
	Development Activity:	
	Project:	
	Module (if applicable):	

Development Activity Analysis			
Moderator:	<Name of the Moderator>		
Date:	<Date of the Analysis>		
Analyze Development Activity Results			
Defects Per Unit of Size:	U-chart	Stable (Yes/No)	
	<Paste U-chart of defects per unit of size here>	<Yes/No>	
		Brief Textual Interpretation	
		<Briefly describe main observations>	
Defects Per Inspection Hour:	U-chart	Stable (Yes/No)	
	<Paste U-chart of defects per inspection hour here>	<Yes/No>	
		Brief Textual Interpretation	
		<Briefly describe main observations>	
Phase Input Quality:	Individuals (xMR) Chart	Stable (Yes/No)	
	<Paste individuals SPC chart of Phase Input Quality>	<Yes/No>	
		Brief Textual Interpretation	
		<Briefly describe main observations>	
Phase Output Quality:	Individuals (xMR) Chart	Stable (Yes/No)	
	<Paste individuals SPC chart of Phase Output Quality>	<Yes/No>	
		Brief Textual Interpretation	
		<Briefly describe main observations>	
Establish Quantitative Improvement Goals			
Defect Rate Improvement Goals		Phase Input/Output Quality Improvement Goals	



<e.g. "reducing the defect rate by X percent">		<e.g. "improving the phase output quality (POQ) by X percent">	
<b>DCA Preparation</b>			
Moderator:	<Name of the Moderator>		
Inspectors:	<Name of the Inspectors - in case they supported the DCA preparation>		
Authors:	<Name of the Authors - in case they supported the DCA preparation>		
Date:	<DCA Preparation Date>		
<b>Apply Pareto Chart and Select Samples</b>			
Pareto Chart		Main Defect Categories	
<Paste Pareto chart for defect data here>		<Main defect categories identified in the Pareto Chart>	
<b>Find Systematic Errors<sup>1</sup></b>			
Defect Category	Systematic Error	Related Defects	
<name of the category>	<Short description>	<Ideally a link to the set of defects related to the systematic error>	
<b>DCA Meeting</b>			
Moderator:	<Name of the Moderator>		
Authors:	<Name of the Authors>		
SEPG Members:	<Name of the SEPG Members>		
Date:	<Date of the DCA Meeting>		
<b>Identify Main Causes<sup>2</sup></b>			
Systematic Error	Cause-effect Diagram	Main Cause(s)	
< Sys. Error found during DCA Preparation>	<Paste cause-effect diagram resulting from analyzing the systematic error here>	<Main cause(s) identified in the cause-effect diagram>	
<b>Propose Actions to Prevent Causes<sup>3</sup></b>			
Cause	Action		
<Name of the cause>	<Short description of the action to be taken to prevent the cause>		

<sup>1</sup> There could be zero, one or more systematic errors for each of the defect categories.

<sup>2</sup> Each of the found systematic errors should be analyzed in order to find its main cause(s).

<sup>3</sup> Each of the identified main causes should be brainstormed in order to find action proposals to prevent it from recurring. The consensus actions should be registered.

Development Activity Improvement				
SEPG Members:	<List of SEPG members responsible for implementing the action proposals>			
Manage Actions to Conclusion				
Action	Estimated Due Date	Due Date	Observation	Status
<Identification of the action>	<Est. Date>	<Due Date>	<e.g. description of the progress of the task or of what has effectively been done>	<Not Started/In Progress/Done>
Communicate Implemented Actions				
Communication Vehicle:	<e.g. presentation, internal circular, notification message, etc>			
Communicator:	<Name of the person that effectively communicated the changes to the team responsible for the development activity>			
Listeners:	<Name of people who attended/received the communication>			
Date:	<Date of the communication>			

## ANEXO E – A FERRAMENTA DPPI FRAMEWORK

*Este anexo apresenta resumidamente os requisitos, o projeto e as funcionalidades da ferramenta DPPI Framework, desenvolvida para apoiar a aplicação de DPPI em projetos de desenvolvimento de software reais.*

### E1 – Descrição Geral do Problema

**Necessidade:** Viabilizar a aplicação da abordagem DPPI para análise causal de defeitos de software a um baixo custo operacional.

**A quem afeta:** Organizações de software que desejem utilizar DPPI para realizar análise causal de defeitos.

**Impacto:** Facilitar a aplicação da abordagem DPPI, permitindo que organizações de software tenham acesso a esta abordagem e apliquem análise causal de defeitos, utilizando as melhores práticas e em conformidade com modelos de maturidade.

**Benefícios:** Diminuir os custos relacionados à aplicação da abordagem DPPI, manter registros da aplicação da abordagem, dar mais agilidade e flexibilidade à realização da análise causal de defeitos.

### E2 – Escopo

O escopo da ferramenta compreende apoiar a abordagem DPPI para análise causal de defeitos, que se inicia depois que os defeitos já foram encontrados (e, normalmente, até já corrigidos). O mesmo não tratará de apoio para as atividades de identificação de defeitos (inspeções e testes), tendo como fronteira com estas atividades apenas a importação de listas de defeitos. Como o software será utilizado somente na presença do moderador da análise causal, inicialmente, a gestão de usuários também estará fora do escopo.

Todas as atividades de DPPI serão apoiadas pela ferramenta, entretanto, na atividade de melhoria a ferramenta apenas permitirá registrar o status das ações de melhoria identificadas, não provendo um apoio adicional para o gerenciamento efetivo da implementação destas ações, pois já existem ferramentas mais elaboradas para este fim.

## **E3 – Requisitos Não Funcionais e Funcionais**

### **Requisitos Não-Funcionais**

- **RNF1:** O software deve ser projetado para possibilitar que esteja disponível 24hs por dia, 7 dias por semana (24x7), uma vez que uma análise causal poderá ser iniciada a qualquer momento e em qualquer projeto.
- **RNF2:** Todas as falhas do software devem ser apresentadas seguindo o mesmo padrão de interface do próprio software. Nenhum detalhe técnico deverá ser apresentado ao usuário do software (versões de BD, SO, servidor web, tecnologias empregadas, etc).
- **RNF3:** O software deve assegurar um tempo de resposta máximo de 10 segundos para as requisições dos usuários.
- **RNF4:** O software deve ser implementado como uma aplicação Web.
- **RNF5:** O software deve ser compatível com os browsers IE (versão 7.0 ou superior) e Firefox (3.0 ou superior)

### **Requisitos Funcionais**

- **RF1:** O software deve permitir ao moderador caracterizar sessões de análise causal.
- **RF2:** O software deve permitir ao moderador importar a lista de defeitos que será utilizada em uma sessão de análise causal.
- **RF3:** O software deve permitir ao moderador visualizar os resultados da atividade de desenvolvimento.
- **RF4:** O software deve permitir ao moderador estabelecer objetivos quantitativos de melhoria.
- **RF5:** O software deve permitir ao moderador visualizar defeitos por categoria e registrar as principais categorias de defeitos.
- **RF6:** O software deve permitir ao moderador registrar erros sistemáticos para cada categoria de defeitos.

- **RF7:** O software deve permitir ao moderador associar defeitos a cada erro sistemático.
- **RF8:** O software deve permitir ao moderador consultar os defeitos associados a cada erro sistemático.
- **RF9:** O software deve permitir ao moderador registrar as causas para cada erro sistemático.
- **RF10:** O software deve permitir ao moderador, dado um erro sistemático, consultar a inferência bayesiana de probabilidades de causas para a categoria de defeitos do erro sistemático.
- **RF11:** O software deve permitir ao moderador registrar propostas de ação para tratar as causas identificadas.
- **RF12:** O software deve permitir ao moderador registrar o status da implementação das ações.
- **RF13:** O software deve permitir ao moderador registrar a ocorrência da comunicação das ações implementadas para a equipe.

## **E4 – Casos de Uso**

Durante a execução da abordagem DPPI, diversos papéis estão envolvidos: o moderador, os inspetores, os autores do documento, o grupo de processos da empresa. Apesar disso, apenas o moderador interage com a ferramenta em si, sendo assim o único ator do sistema. O diagrama de casos de uso encontra-se na Figura E.1.

Os Casos de Uso da ferramenta de apoio a DPPI foram desenvolvidos para contemplar cada uma das atividades da abordagem DPPI. Dessa forma, o mapeamento entre as atividades descritas no capítulo 4 e a implementação foi facilitado. A descrição dos casos de uso segue.

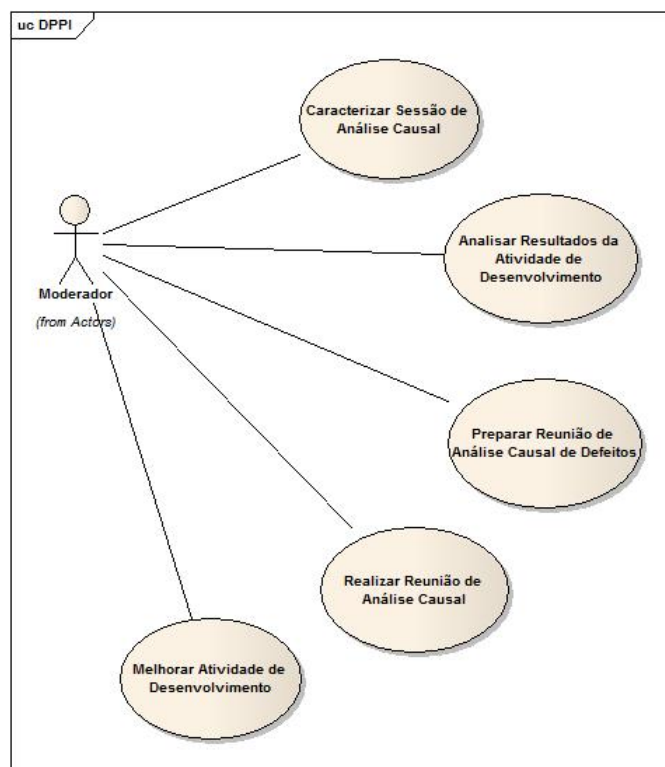


Figura E.1. Diagrama de Casos de Uso para a Ferramenta DPPI Framework.

- Caso de Uso 01

UC01 – Caracterizar Sessão de Análise Causal de Defeitos	
<b>Objetivo</b>	Permitir que o Moderador forneça as informações básicas para a caracterização de uma Reunião de Análise Causal de Defeitos
<b>Pré-condições</b>	Nenhuma
<b>Pós-condições sucesso</b>	Reunião de Análise Causal Caracterizada
<b>Ator Primário</b>	Moderador
<b>Triggers</b>	Usuário seleciona a opção 'Caracterizar Reunião de Análise Causal' a partir da página inicial do sistema
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Ator seleciona o projeto, o módulo do projeto e a atividade de desenvolvimento para a reunião de análise causal, a partir das listas destes campos <b>[A1]</b>.</li> <li>2. Sistema guarda as informações da caracterização do projeto.</li> <li>3. Ator informa arquivo XML com a lista de defeitos que serão utilizados durante a reunião.</li> <li>4. Sistema guarda as informações dos defeitos.</li> <li>5. Ator seleciona a opção "Iniciar Reunião de Análise Causal de Defeitos".</li> </ol>
<b>Fluxo Alternativo</b>	<p><b>[A1]</b> Ator não encontra o projeto, módulo ou atividade de desenvolvimento nas listas</p> <ol style="list-style-type: none"> <li>1. Ator seleciona a opção "Adicionar Projeto", "Adicionar Módulo" ou "Adicionar Atividade de Desenvolvimento".</li> <li>2. Sistema apresenta uma janela para entrada de dados novos.</li> <li>3. Ator cadastra a informação nova.</li> <li>4. Sistema reapresenta a tela principal, com a informação adicionada pelo Ator.</li> </ol>
<b>Requisitos Funcionais</b>	<b>RF1, RF2</b>

- Caso de Uso 02

UC02 – Analisar Resultados da Atividade de Desenvolvimento	
<b>Objetivo</b>	Permitir que o Moderador analise os resultados da atividade de desenvolvimento
<b>Pré-condições</b>	Ter realizado o UC 01
<b>Pós-condições sucesso</b>	Resultados da atividade de desenvolvimento analisados
<b>Ator Primário</b>	Moderador
<b>Triggers</b>	Ator seleciona a opção “Iniciar DPPI” na atividade anterior (UC 01).
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Ator carrega duas imagens com os gráficos U correspondentes.</li> <li>2. Sistema apresenta os gráficos correspondentes.</li> <li>3. Ator define se cada um dos gráficos é estável ou não e escreve um pequeno texto de interpretação sobre os gráficos.</li> <li>4. Ator insere os objetivos quantitativos de melhoria.</li> <li>5. Sistema guarda as informações</li> </ol>
<b>Requisitos Funcionais</b>	<b>RF3, RF4</b>

- Caso de Uso 03

UC03 – Preparar Reunião de Análise Causal de Defeitos	
<b>Objetivo</b>	Permitir que o Moderador prepare a reunião de análise causal
<b>Pré-condições</b>	Ter realizado o UC 02
<b>Pós-condições sucesso</b>	Reunião de análise causal estará pronta para ser iniciada
<b>Ator Primário</b>	Moderador
<b>Triggers</b>	Ator seleciona a opção “Próximo passo” na atividade anterior (UC 02).
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Ator insere o nome dos inspetores e autores presentes.</li> <li>2. Ator carrega imagem com os gráficos de Pareto.</li> <li>3. Sistema apresenta o gráfico correspondente.</li> <li>4. Ator define a Categoria de Defeitos que pretende utilizar, entre “Tipo do Defeito” e “Natureza do Defeito”. <b>[A1]</b></li> <li>5. Ator analisa o gráfico de Pareto e escreve um texto com as categorias de defeitos principais.</li> <li>6. Ator adiciona um Erro Sistemático, definido o tipo da categoria dentre as opções de categoria definidas no passo 4.</li> <li>7. Para cada Erro Sistemático, o ator seleciona a opção “Associar Defeitos” e define os defeitos relacionados com aquele Erro Sistemático.</li> <li>8. Sistema guarda as informações</li> </ol>
<b>Fluxo Alternativo</b>	<p><b>[A1]</b> Ator seleciona a Categoria de Defeitos que deseja ver os defeitos correspondentes e seleciona a opção “Mostrar defeitos”</p> <ol style="list-style-type: none"> <li>1. Sistema apresenta uma tela com todos os defeitos daquela inspeção que são da Categoria de Defeito escolhida.</li> </ol>
<b>Requisitos Funcionais</b>	<b>RF5, RF6, RF7</b>

- Caso de Uso 04

UC04 – Realizar Reunião de Análise Causal de Defeitos	
<b>Objetivo</b>	Permitir que o Moderador execute a reunião de análise causal
<b>Pré-condições</b>	Ter realizado o UC 03
<b>Pós-condições sucesso</b>	Reunião de análise causal estará concluída
<b>Ator Primário</b>	Moderador
<b>Triggers</b>	Ator seleciona a opção “Próximo passo” na atividade anterior (UC 03).
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Ator insere o nome dos membros do grupo de processos da organização e dos autores presentes.</li> <li>2. Sistema apresenta uma lista com os Erros Sistemáticos adicionados no UC 03 [A1].</li> <li>3. Ator seleciona a opção “Adicionar Causas”</li> <li>4. Sistema apresenta uma janela para que o ator adicione uma causa</li> <li>5. Ator define a categoria da causa e clica em “Adicionar”.</li> <li>6. Sistema apresenta a lista de causas adicionadas na tela.</li> <li>7. Para cada causa, Ator seleciona a opção “Adicionar Ações”.</li> <li>8. Sistema apresenta uma janela para que o ator adicione uma ação.</li> <li>9. Ator define a ação e clica em “Adicionar”.</li> <li>10. Sistema apresenta a lista de causas já com a ação adicionada.</li> </ol>
<b>Fluxo Alternativo</b>	<p>[A1] Ator seleciona a opção “Mostrar Tabela de Causa-Efeito Probabilístico”</p> <ol style="list-style-type: none"> <li>1. Sistema apresenta uma janela com as probabilidades de causa para erros da mesma categoria.</li> </ol>
<b>Requisitos Funcionais</b>	RF8, RF9, RF10, RF11

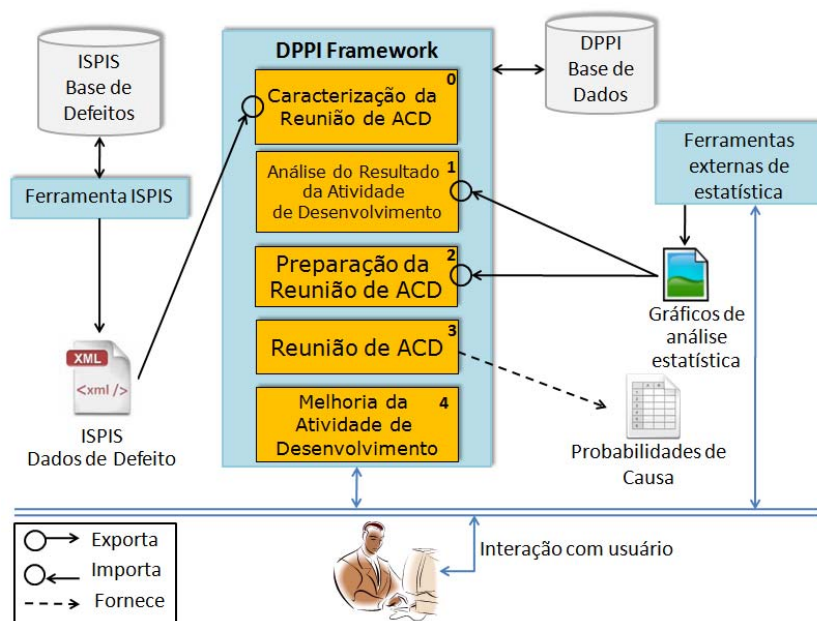
- Caso de Uso 05

UC05 –Melhorar Atividade de Desenvolvimento	
<b>Objetivo</b>	Permitir que o Moderador gerencie as ações de melhoria para a atividade de desenvolvimento
<b>Pré-condições</b>	Ter realizado o UC 04
<b>Pós-condições sucesso</b>	Fim da execução da abordagem DPPI
<b>Ator Primário</b>	Moderador
<b>Triggers</b>	Ator seleciona a opção “Próximo passo” na atividade anterior (UC 04).
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Ator insere o nome dos membros do grupo de processos da organização presentes.</li> <li>2. Sistema apresenta a lista de ações definidas no UC 04, com os campos: data de execução estimada, data de execução, observação e status.</li> <li>3. Ator preenche os campos do passo anterior.</li> <li>4. Para o conjunto de ações preenchidas do passo anterior, o Ator insere as seguintes informações: veículo de comunicação, comunicador, ouvintes e data da comunicação.</li> </ol>
<b>Requisitos Funcionais</b>	RF12, RF13



## E5 – Arquitetura da Ferramenta

A arquitetura de DPPI é apresentada na Figura E.2. Nela, podemos visualizar com mais facilidade a entrada e saída de dados das atividades de DPPI. Podemos perceber também a adição da atividade “Caracterização da Reunião de Análise Causal”. Ela foi necessária para podermos inserir na ferramenta as informações relevantes para contextualizar a sessão de análise causal. São coletadas informações referentes ao projeto, módulo do projeto e atividade de desenvolvimento. Como esta atividade deve ser realizada antes das 4 atividades da abordagem DPPI, ela foi representada na Figura E.2 como atividade 0.



**Figura E.2. Arquitetura da Ferramenta DPPI Framework.**

No momento da caracterização da sessão de análise causal, é carregada uma lista de defeitos para ser utilizada durante a execução de DPPI. Através da adição de uma lista de defeitos, será possível identificar erros sistemáticos e os relacionar com um conjunto de defeitos. Esta lista é proveniente de ISPIS (Kalinowski *et al.*, 2004), uma infra-estrutura que possibilita a realização sistematizada de inspeções assíncronas sobre artefatos construídos durante o processo de desenvolvimento de software. Numa inspeção é gerada uma lista de defeitos nos artefatos inspecionados, que é passada para o autor do documento executar as correções necessárias. Esta lista de defeitos, que pode ser exportada no formato XML por ISPIS, possui os seguintes campos: (i) descrição (ii) localização (iii) severidade (iv) classificação (v) tipo da taxonomia. Na Figura E.3, vemos um exemplo de lista de defeitos exportada por ISPIS no formato XML.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<inspection>
  <discrepancy>
    <location>UC 02</location>
    <classification>Fato Incorreto</classification>
    <taxonomytype></taxonomytype>
    <description>O ator não pode executar a ação "Apagar Projeto"</description>
    <severity>3</severity>
  </discrepancy>
  <discrepancy>
    <location>UC 05</location>
    <classification>Ambiguidade</classification>
    <taxonomytype></taxonomytype>
    <description>"Em tempo real" é ambíguo. O tempo deve ser estabelecido</description>
    <severity>2</severity>
  </discrepancy>
  <discrepancy>
    <location>UC 09</location>
    <classification>Omissão</classification>
    <taxonomytype></taxonomytype>
    <description>O sistema apresenta mais dados do que foram descritos</description>
    <severity>1</severity>
  </discrepancy>
</inspection>

```

**Figura E.3. Exemplo de Lista de Defeitos exportada da Ferramenta ISPIS.**

Ao importar a lista em DPPI, o mapeamento entre os campos é feito segundo a Figura E.4. Os campos descrição, localização e severidade são obrigatórios. O defeito deve possuir ou uma classificação, ou um tipo de taxonomia.

ISPIS	DPPI
Descrição	Descrição
Localização	Localização
Severidade	Severidade
Classificação	Natureza do Defeito
Tipo da Taxonomia	Tipo do Defeito

**Figura E.4. Mapeamento dos Campos de ISPIS e DPPI.**

## E6 – Diagrama de Classes do Domínio

O diagrama de classes do domínio que orientou o desenvolvimento encontra-se na Figura E.5.

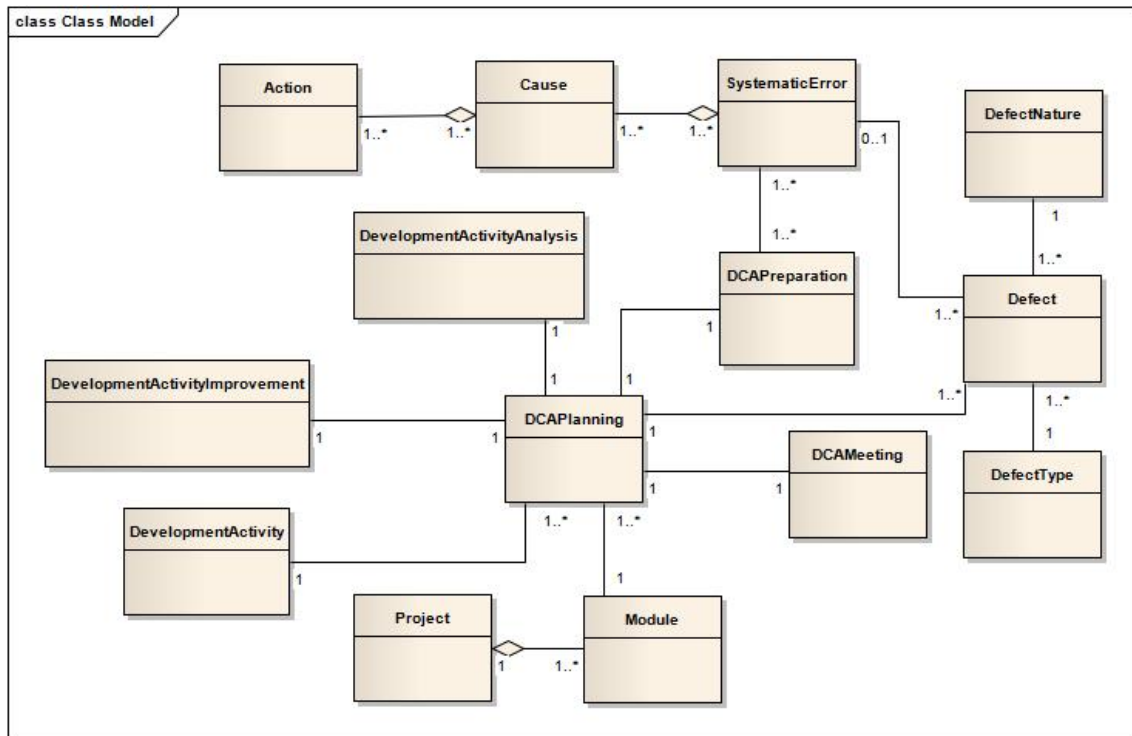


Figura E.5. Diagrama de Classes do Domínio para a Ferramenta DPPI Framework.

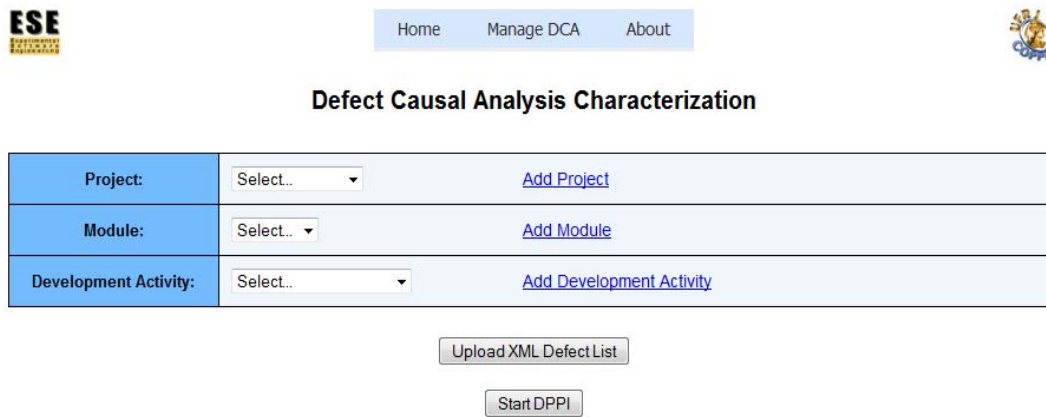
## E7 – Funcionalidade

As quatro atividades de DPPI (descritas no capítulo 4) foram projetadas para serem executadas com suas tarefas organizadas, guiando sua execução na ordem correta. Cada uma das atividades é apresentada em uma página web. Ao longo da execução das atividades, é mostrado no canto superior direito da página qual atividade da abordagem está sendo executada. Nas sessões seguintes, serão apresentadas imagens de cada página web.

### E7.1 Caracterização da Reunião de Análise Causal

O primeiro passo da ferramenta é a caracterização do projeto. Neste passo, são coletadas as informações necessárias para iniciar uma reunião de análise causal de defeitos. O usuário define o projeto, o módulo do projeto e a atividade de desenvolvimento na qual a sessão de análise causal será realizada. Após estas definições, neste momento é necessário importar na ferramenta um arquivo XML contendo a lista de defeitos da inspeção realizada

previamente. A ferramenta então lê esta lista de defeitos e cadastra estes defeitos no banco de dados, relacionando-os a esta reunião de análise causal em particular. Esta lista é de suma importância para a execução da abordagem DPPI, pois a partir dela são identificados os erros sistemáticos. Na Figura E.6, podemos ver a página de Caracterização da análise causal.



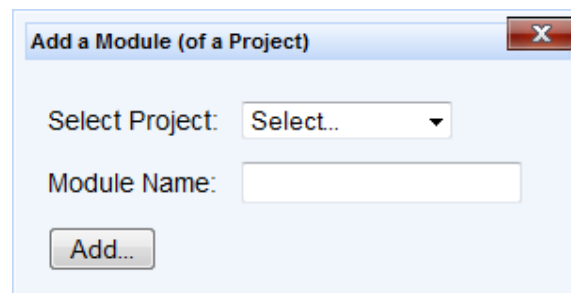
The screenshot shows a web application interface for "Defect Causal Analysis Characterization". At the top, there is a navigation bar with links for "Home", "Manage DCA", and "About". To the left is the "ESE" logo, and to the right is a small circular icon. The main heading is "Defect Causal Analysis Characterization". Below this, there is a form with three rows, each with a label, a dropdown menu, and a link:

Project:	Select...	<a href="#">Add Project</a>
Module:	Select...	<a href="#">Add Module</a>
Development Activity:	Select...	<a href="#">Add Development Activity</a>

Below the form, there are two buttons: "Upload XML Defect List" and "Start DPPI".

**Figura E.6. Caracterização da Reunião de Análise Causal.**

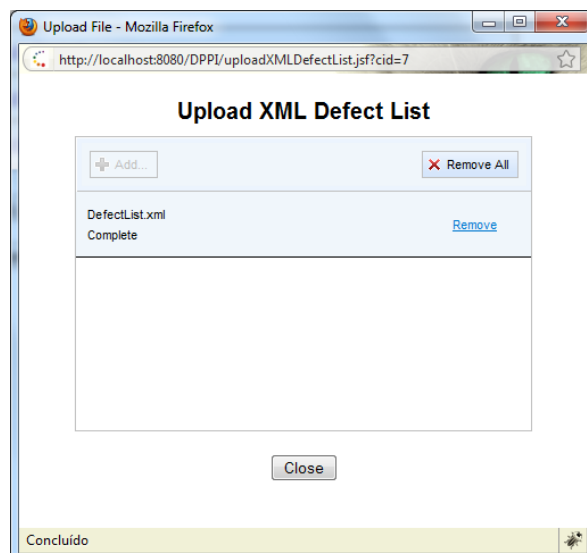
Um exemplo da janela de adição de informações é apresentado na Figura E.7.



The screenshot shows a dialog box titled "Add a Module (of a Project)". It contains two input fields: "Select Project:" with a dropdown menu showing "Select..." and "Module Name:" with a text input field. Below these fields is an "Add..." button.

**Figura E.7. Adição de um Módulo.**

A janela de importação da lista de defeitos em XML é apresentada na Figura E.8. Ela é acessada através do botão "Upload XML Defect List".




**Figura E.8. Importação da Lista de Defeitos.**


Com a lista de defeitos importada, é possível dar início a execução da abordagem clicando no botão “Start DPPI”.

## **E7.2 Análise da Atividade de Desenvolvimento**

Esta é a atividade inicial de DPPI. As informações inseridas na caracterização aparecem já preenchidas. O moderador então insere seu nome e a data em que a atividade está sendo executada. Em seguida, é iniciada a tarefa “Analisar os Resultados da Atividade de Desenvolvimento”. O moderador carrega os gráficos U correspondentes, gerados por uma ferramenta de estatística externa e verifica se estes estão estáveis ou não. Após isso, escreve um pequeno texto explicando sua interpretação sobre os gráficos. Em seguida, o moderador descreve os objetivos quantitativos de melhoria. A Figura E.9 representa esta etapa em seu momento inicial, ou seja, sem a adição de informações ou gráficos.



[Home](#)
[View External Statistical Tools](#)
[About](#)



**Development Activity Analysis**

1
2
3
4

<b>Development Activity:</b>	Development Activity	<b>Project:</b>	Project
<b>Module:</b>	Module	<b>Date:</b>	<input type="text"/>
<b>Moderator:</b>	<input type="text"/>		

Analyze Development Activity Results

Defects Per Unit of Size:

U-Chart

**Stable:**  
☐ Yes ☒ No

Brief Textual Interpretation

Defects Per Inspection Hour:

U-Chart

**Stable:**  
☐ Yes ☒ No

Brief Textual Interpretation

Establish Quantitative Improvement Goals

**Defect Rate Improvement Goal**

**Figura E.9. Apoio à Análise da Atividade de Desenvolvimento.**

Após inserir as informações necessárias, o moderador clica em “Next” para iniciar a preparação para a reunião de análise causal de defeitos.

### E7.3 Preparação da Análise Causal de Defeitos

Nesta página, foi implementada a atividade homônima da abordagem DPPI, e apresentada na Figura E.10. O moderador insere os nomes dos Inspetores e Autores (se presentes) e a data em que a atividade está sendo executada.

[Home](#)
[View External Statistical Tools](#)
[About](#)

1
2
3
4

### Defect Causal Analysis Preparation

Development Activity:	Development Activity	Project:	Project
Module:	Module	Date:	
Moderator:	Moderator		
Inspectors:			
Authors:			

#### Apply Pareto Chart and Select Samples

Pareto chart

Defect Type

Select...

Defect Nature

Select...

Choose:

☒ Defect Type
☐ Defect Nature

Show Defects

Main Defect Categories

Upload Pareto Chart

#### Find Systematic Errors

Defect Category:

Tipo de Defeito 1

Systematic Error Name:

Erro Sistemático 3

Add...

Defect Category ^	Systematic Error ^	Related Defects
Tipo de Defeito 1	Erro Sistemático 1	Associate Defects
Tipo de Defeito 1	Erro Sistemático 2	Associate Defects
Tipo de Defeito 1	Erro Sistemático 3	Associate Defects

Next

**Figura E.10. Apoio à Preparação para a Análise Causal de Defeitos.**

Em seguida, o moderador carrega o gráfico de Pareto (gerado externamente). Neste momento, é necessário escolher qual a categoria de defeitos que vai ser analisada durante a reunião de análise causal (Tipo de Defeito ou Natureza de Defeito). O moderador tem a possibilidade de definir um valor na lista de categorias possíveis e então selecionar a opção “Show Defects”. Uma nova janela irá surgir, com a lista de defeitos carregada, filtrada pela categoria escolhida. Com o auxílio do gráfico de Pareto, os participantes então definem as categorias principais de defeito.

Com a categoria definida, é possível executar a próxima tarefa: adição de erros sistemáticos. Isto é feito escolhendo sua categoria de defeito e inserindo o nome do erro sistemático. Ao clicar em “Add” logo ao lado do campo de entrada “Systematic Error

Name”, uma tabela aparecerá abaixo com os erros inseridos. Para cada erro, é possível clicar em “Associate Defects”, para relacionar os defeitos aos erros sistemáticos.

Após executar estas tarefas, o moderador clica em “Next” para iniciar a atividade Reunião de Análise Causal.

## **E7.4 Reunião de Análise Causal de Defeitos**

A atividade (ilustrada na Figura E.11) é iniciada com o preenchimento dos campos superiores. Neste caso, são inseridos os membros do grupo de engenharia de processos de software, os autores, e a data em que a atividade está sendo executada.

Podemos observar que o sistema apresenta uma tabela com os erros sistemáticos inseridos no passo anterior. Para cada um destes erros, é possível adicionar uma ou mais causas. Ao clicar em “Add Cause”, uma nova janela será aberta, onde o moderador irá definir uma categoria de causa dentre estas cinco: Pessoas, Métodos, Ferramentas, Entradas e Organização. Para auxiliar na escolha dessas causas, o moderador pode clicar em “Show Probabilistic Cause-Effect Table”. O sistema apresenta uma lista de causas referentes à categoria de defeito escolhida. Para cada uma dessas causas, é apresentada sua probabilidade de ocorrência de acordo com uma amostra de defeitos. Esta amostra contém todos os defeitos da categoria escolhida que pertencem à mesma atividade de desenvolvimento da reunião de análise causal atual. Ou seja, os defeitos utilizados em reuniões de análise causal previamente executadas farão parte da amostra, desde que pertençam a mesma atividade de desenvolvimento. Esta amostra é utilizada no treinamento da rede Bayesiana, para que sejam construídas as probabilidades de cada causa.

Para cada causa adicionada, surge uma nova linha na tabela logo abaixo, onde para cada causa é possível adicionar uma ou mais ações. Estas ações propostas serão gerenciadas na atividade seguinte, acessível ao clicar em “Next”.



## Defect Causal Analysis Meeting

Development Activity:	Development Activity	Project:	Project
Module:	Module	Date:	<input type="text"/>
Moderator:	Moderator		
SEPG Members:	<input type="text"/>		
Authors:	<input type="text"/>		

Identify Main Causes		
Systematic Error ^	Cause-effect Diagram	Main Causes
Erro Sistemático 1	<input type="button" value="Show Probabilistic Cause-effect Table"/>	<a href="#">Add Cause</a> <ul style="list-style-type: none"> <li>• Causa 1</li> <li>• Causa 2</li> <li>• Causa 3</li> </ul>
Erro Sistemático 2	<input type="button" value="Show Probabilistic Cause-effect Table"/>	<a href="#">Add Cause</a> <ul style="list-style-type: none"> <li>• Causa 4</li> </ul>
Erro Sistemático 3	<input type="button" value="Show Probabilistic Cause-effect Table"/>	<a href="#">Add Cause</a> <ul style="list-style-type: none"> <li>• Causa 5</li> <li>• Causa 6</li> </ul>

Propose Actions to Prevent Causes	
Cause ^	Action
Causa 1	<a href="#">Add Action</a> <ul style="list-style-type: none"> <li>• Action 1</li> </ul>
Causa 2	<a href="#">Add Action</a> <ul style="list-style-type: none"> <li>• Action 2</li> </ul>
Causa 3	<a href="#">Add Action</a> <ul style="list-style-type: none"> <li>• Action 3</li> <li>• Action 4</li> </ul>
Causa 4	<a href="#">Add Action</a>
Causa 5	<a href="#">Add Action</a>
Causa 6	<a href="#">Add Action</a>

Figura E.11. Apoio à Reunião de Análise Causal de Defeitos.

## E7.5 Melhoria da Atividade de Desenvolvimento

A atividade final da abordagem DPPI (ilustrada na Figura E.12) é iniciada com o preenchimento dos campos superiores. Neste caso, são inseridos novamente os membros do grupo de engenharia de processos de software e a data em que a atividade está sendo executada.

Nesta atividade, a lista de todas as ações inseridas no passo anterior (independente da causa a que foi relacionada) é apresentada. Cada uma das ações possui os seguintes campos a serem preenchidos: data estimada para execução, data de execução, observação e status (Não iniciado, Em execução, Concluído). O moderador preenche estas informações ao longo da execução das ações, de forma a ter um controle sobre a execução de cada uma

delas. Para voltar a esta tela no futuro, basta que o moderador selecione, no momento da caracterização da sessão de análise causal, o botão “Manage DCA”, no menu superior. Será apresentada uma lista com as sessões de análise causal realizadas, onde ao selecionar uma delas, esta última atividade será recuperada, permitindo assim o gerenciamento das ações.

Por fim, é necessário comunicar a implementação das ações. Quando desejar, o moderador preenche os seguintes dados: veículo de comunicação, comunicador, ouvintes e data da comunicação.

**ESE**

Home View External Statistical Tools About

1 2 3 4

**Development Activity Improvement**

Development Activity:	Development Activity	Project:	Project
Module:	Module	Date:	
SEPG Members:			

**Manage Actions to Conclusion**

Action ^	Estimated Due Date	Due Date	Observation	Status	Save
Action 1				Not Started	<a href="#">Save Action</a>
Action 2				Not Started	<a href="#">Save Action</a>
Action 3				Not Started	<a href="#">Save Action</a>
Action 4				Not Started	<a href="#">Save Action</a>

**Communicate Implemented Actions**

Communication Vehicle:	
Communicator:	
Listeners:	
Date	

Finish

**Figura E.12. Apoio à Melhoria da Atividade de Desenvolvimento.**

Ao comunicar a execução das ações, é encerrada a sessão de análise causal. O moderador então clica em “Finish”, dando fim à execução de DPPI.

## **ANEXO F – INSTRUMENTOS DO ESTUDO EXPERIMENTAL**

*Este anexo apresenta os instrumentos utilizados na condução do estudo experimental realizado para avaliar o uso da abordagem de DPPI para a identificação de causas de defeitos.*

### **F1 – Consentimento de Participação**

Eu declaro ter mais de 18 anos de idade e que concordo em participar de um estudo conduzido pelo pesquisador Marcos Kalinowski e pelo Prof. Guilherme Horta Travassos. Este estudo visa ajudar a compreender aspectos relativos à análise causal de defeitos de software.

#### **PROCEDIMENTO**

Eu entendo que serei solicitado a identificar causas associadas a erros sistemáticos ocorridos ao longo do desenvolvimento de um sistema web. Neste exercício alguns métodos experimentais serão aplicados visando avaliar aspectos da identificação das causas.

Os pesquisadores conduzirão o estudo consistindo da coleta, análise e relato dos dados do exercício. Eu entendo que não tenho obrigação alguma em contribuir com informação sobre meu desempenho no exercício, e que posso solicitar a retirada de meus resultados do experimento a qualquer momento. Eu entendo também que quando os dados forem coletados e analisados, meu nome será removido dos dados e que este não será utilizado em nenhum momento durante a análise ou quando os resultados forem apresentados.

#### **CONFIDENCIALIDADE**

Toda informação coletada neste estudo é confidencial, e meu nome não será identificado em momento algum. Da mesma forma, me comprometo a manter sigilo das tarefas solicitadas e dos documentos apresentados e que fazem parte do experimento.

## BENEFÍCIOS, LIBERDADE DE DESISTÊNCIA

Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada a minha pessoa não seja incluída no estudo. Eu entendo que participo do estudo experimental de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas, ferramentas e processos para a Engenharia de Software.

## RESPONSÁVEIS

Marcos Kalinowski

Prof. Guilherme Horta Travassos

Programa de Engenharia de Sistemas e Computação

COPPE/UFRJ

Nome (em letra de forma): \_\_\_\_\_

Assinatura: \_\_\_\_\_ Data: \_\_\_\_\_

## F2 – Caracterização dos Participantes

Nome \_\_\_\_\_ Nível (Ms.c/D.Sc.): \_\_\_\_\_

### Experiência Prática Com Desenvolvimento De Software

Qual é sua experiência anterior com desenvolvimento de software na prática? (marque aqueles itens que melhor se aplicam)

- ☐ nunca desenvolvi software.
- ☐ tenho desenvolvido software para uso próprio.
- ☐ tenho desenvolvido software como parte de uma equipe, relacionado a um curso.
- ☐ tenho desenvolvido software como parte de uma equipe, na indústria.

Por favor, explique sua resposta. Inclua o número de semestres ou número de anos de experiência relevante em desenvolvimento (E.g. “Eu trabalhei por 10 anos como programador na indústria”)

---

---

---

### Experiência com Desenvolvimento de Software

Por favor, indique o grau de sua experiência nesta seção seguindo a escala de 5 pontos abaixo:

- 1 = nenhum
- 2 = estudei em aula ou em livro
- 3 = pratiquei em 1 projeto em sala de aula
- 4 = usei em 1 projeto na indústria
- 5 = usei em vários projetos na indústria

- |   |   |   |   |   |   |
|---|---|---|---|---|---|
| • Experiência com gerenciamento de projeto de software? | 1 | 2 | 3 | 4 | 5 |
| • Experiência com requisitos de software?               | 1 | 2 | 3 | 4 | 5 |
| • Experiência inspecionando requisitos de software?     | 1 | 2 | 3 | 4 | 5 |

### **Experiência em Diferentes Domínios**

Nós usaremos esta seção para compreender quão familiar você está com diferentes domínios de aplicação.

Por favor, indique o grau de experiência nesta seção seguindo a escala de 3 pontos abaixo:

1 = Eu não tenho familiaridade com a área.

3 = Eu tenho alguma familiaridade com a área.

5 = Eu sou muito familiar com esta área.

#### Quanto você sabe sobre...

- |                               |   |   |   |
|-------------------------------|---|---|---|
| • Acompanhamento de Projetos? | 1 | 3 | 5 |
| • Finanças Corporativas?      | 1 | 3 | 5 |

## F3 – Instrumentos para a Operação do Experimento

### F3.1 Orientações para a Operação do Experimento Entregues aos Participantes da Abordagem *ad hoc*

#### Operação do Experimento

Juntamente com este documento você recebeu: (1) um documento de requisitos de um módulo do SIGIC, e (2) listas de defeitos encontrados durante a inspeção deste documento associadas a erros sistemáticos.

A fim de prevenir a repetição destes erros sistemáticos em projetos futuros você foi solicitado a identificar as principais causas associadas aos erros sistemáticos, listando no máximo 3 causas para cada erro sistemático. Para isto poderá fazer uso da descrição dos defeitos contida nas listas de defeitos e deverá preencher o *template* abaixo, onde as possíveis categorias de causa são: entradas, ferramentas, métodos, organização e pessoas.

#### 1.1. Erro Sistemático: Omissão de Campos e Informações

Principais Causas Encontradas:

Categoria de Causa	Causa

#### 1.2. Erro Sistemático: Omissão de Regras de Negócio

Principais Causas Encontradas:

Categoria de Causa	Causa

#### 1.3. Erro Sistemático: Omissão de Detalhes de Navegação/Fluxo

Principais Causas Encontradas:

Categoria de Causa	Causa

--	--

#### 1.4. Erro Sistemático: Omissão de Definições e Referências

Principais Causas Encontradas:

Categoria de Causa	Causa

#### 1.5. Erro Sistemático: Inserção de Informação Ambígua (MAP) ou Inserção de Informação Inconsistente (MFI)

Principais Causas Encontradas:

Categoria de Causa	Causa



## F3.2 Orientações para a Operação do Experimento Entregues aos Participantes da Abordagem de DPPI

### Operação do Experimento

Juntamente com este documento você recebeu: (1) um documento de requisitos de um módulo do SIGIC, (2) listas de defeitos encontrados durante a inspeção deste documento associadas a erros sistemáticos, e (3) diagramas de causa e efeito probabilístico identificando a probabilidade de ocorrência de cada uma das causas de acordo com dados históricos do SIGIC.

A fim de prevenir a repetição destes erros sistemáticos em projetos futuros você foi solicitado a identificar as principais causas associadas aos erros sistemáticos, listando no máximo 3 causas para cada erro sistemático. Para isto poderá fazer uso da descrição dos defeitos contida nas listas de defeitos e deverá preencher o *template* abaixo.

#### 1.1. Erro Sistemático: Omissão de Campos e Informações

Principais Causas Encontradas:

Categoria de Causa	Causa

#### 1.2. Erro Sistemático: Omissão de Regras de Negócio

Principais Causas Encontradas:

Categoria de Causa	Causa

#### 1.3. Erro Sistemático: Omissão de Detalhes de Navegação/Fluxo

Principais Causas Encontradas:

Categoria de Causa	Causa

--	--

#### 1.4. Erro Sistemático: Omissão de Definições e Referências

Principais Causas Encontradas:

Categoria de Causa	Causa

#### 1.5. Erro Sistemático: Inserção de Informação Ambígua (MAP) ou Inserção de Informação Inconsistente (MFI)

Principais Causas Encontradas:

Categoria de Causa	Causa

## F3.3 Listas de Defeitos Associados aos Erros Sistemáticos

### Módulo MAP

#### Erro Sistemático: Omissão de Campos e Informações

Localização	Descrição do Defeito
Pag. 13 - RF49	Que tipo de parâmetros e como eles atuam sobre as atividades de acompanhamento de projetos?
Pag. 14 - RF58	Que dados ou que tipo de dados a respeito das parcelas poderão ser consultados.
UC27	No passo 1 do fluxo principal, a informação "Grupo" é apresentada. Parece-me que não existe nenhuma referência a grupo no dicionário de dados, de onde vem esta informação?
UC27	No passo 1 do fluxo principal, onde é apresentada a informação "Projeto" não deveria poder ser "Curso" também?
UC27	A RN1 não informa o que o Gerente de Projetos Vinculados visualiza.
UC27	A RN5 menciona "status vencida" para cobrança. Quais seriam os possíveis status para cobrança? Não vi nada no dicionário de dados.
UC28	No passo 6 do fluxo principal (mas vale para todo o documento), me parece que vários campos que são somente leitura não estão especificados como tal, como por exemplo, Valor a Receber.
UC29	No passo 3 do fluxo principal, os campos "% de Despesas Operacionais" e "% de Overhead" não estão definidos no dicionário de dados.
UC29 - Passo 2 do Fluxo Principal	No passo 2 é mencionado o campo 'código do plano de aplicação especial'. Este campo não é listado no passo 1 para ser exibido.
UC30 - Passo 1 do Fluxo Principal	Não é especificado o formato do campo período. Exemplo: DD/MM/AAAA ou MM/AAAA.
UC30 - Passo 1 do Fluxo Principal	Não está especificado se o campo tipo é uma lista ou não.
UC33	No passo 1 do fluxo principal, não deveria existir algum agrupamento para os campos de filtro (com 'OUs' entre eles)? Por exemplo, só faz sentido o preenchimento do campo CPF ou CNPJ ou os outros campos, já que CPF e CNPJ só podem retornar 1 registro. Tem sido assim para os outros módulos.
UC34 - Passo 2 do Fluxo Principal	Não estão definidas as informações que devem ser exibidas na lista de projetos.

UC35 - Passo 1 do Fluxo Principal	Não é definido o formato do campo 'período' ('de' e 'até'). Por exemplo, se é 'DD/MM/AAAA' ou 'MM/AAAA'.
UC36 - Passo 1 do Fluxo Principal	O campo 'status' só lista os status de projetos, não mencionando fundos e cursos. Isto sugere omissão. Caso não seja, há um indicativo de ambigüidade na busca, pois o sistema poderia consultar fundos com o campo 'status' informando 'projeto ativo', o que não parece ser válido.
UC37	No passo 1 do fluxo principal, Antecedência de Aviso de Cobrança de Parcela seria em número de dias? Olhar outros campos deste UC em que a unidade também não está definida.
UC39	No passo 5 do fluxo principal, quais dados do projeto devem ser apresentados?
Pag. 128 - Sec. 11.15	No passo 2, qual o significado de cada linha de uma rubrica? Mais do que informar o número de linhas deve-se informar um "rótulo", não? O que este "rótulo" significa?
Pag. 139 - Sec. 13	Quais seriam os possíveis valores para o status da cobrança?

### Erro Sistemático: Omissão de Regras de Negócio

Localização	Descrição do Defeito
Pag. 13 - RF54	O ajuste no valor do overhead não tem uma faixa de valores pré-determinada onde o valor do overhead pode flutuar?
UC28	Na RN2 em "soma das colunas Saldo + Valor a Receber" estariam relacionados a rubrica de origem?
UC28 - Passo 8 do Fluxo Principal	Embora os campos 'valor contratado' e 'valor recebido' tenha sido informados como sendo somente leitura, não está claro se o preenchimento dos mesmos é obrigatório para o acionamento da opção 'Ajustar Cronograma de Desembolso para o Projeto'. Também não é mencionada qualquer influência destas informações no procedimento de remanejamento.
UC29 - pag81	no passo 1, faltou definir o que seria soma dos % anteriores.
UC33 - Passo 3 do Fluxo Principal	A informação 'valor' carece de detalhes quanto ao seu valor na lista de inadimplentes. Não está especificado se o valor refere-se, por exemplo, à soma do valor de todas as parcelas com as quais o aluno é inadimplente
UC33-PAG85	No passo 3, não foi definido X.
UC34-PAG87	No passo 1, não foi definido X.
uc35-pa90	No passo 1 de A3, não foi definido X.
UC35-PAG89	No passo 3, não foi definido X.
UC36 - [RN1]	Não está definido se, caso o usuário seja um coordenador de projeto, as listas de projetos, fundos e cursos exibirão todos os itens sem restrição, ou se exibirão apenas aqueles que estão de fato pertinentes ao coordenador.

UC40 - Passo 1 de [A4]	Não está definido no passo 1 de [A4], ou em alguma regra de negócio, como identificar a condição para que os valores estejam incorretos. Esta condição está definida no passo 1 de [A4].
Pag. 128 - Sec. 11.15	Consegui chegar ao valor 0,456 da CPMF mas como calcular TD? Não consegui chegar ao valor 1,62. Seria interessante completar o exemplo com TODAS as contas para se chegar aos valores apresentados.

### Erro Sistemático: Omissão de Detalhes de Navegação/Fluxo

Localização	Descrição do Defeito
UC28 - Passo 2 do Fluxo Principal	Não está explícito no passo 2 do fluxo principal, assim como em nenhuma regra de negócio, como o sistema deve proceder com a busca caso não sejam fornecidos nenhum filtro.
UC28 - Passo 8 do Fluxo Principal	Não está especificado qual valor (ou quais valores) o ator deve preencher: valor contratado, recebido ou de remanejamento.
UC29 - [A7]	Em [A7] não há está presente no último passo as instruções para prosseguir com o fluxo.
UC30 - Passo 2 do Fluxo Principal	Não está explícito no passo 2 do fluxo principal, assim como em nenhuma regra de negócio, como o sistema deve proceder com a busca caso não sejam fornecidos nenhum filtro.
UC35 - Passo 1 de [A3]	Não estão especificados quais passos (ou quais regras de negócio) de UC39 estão relacionados ao cálculo da informação 'overhead calculado'. A presença desta informação pode evitar ambiguidades.
UC39	No fluxo principal faltou o passo em que o ator preenche a distribuição real e o fluxo alternativo para o caso de problemas de validação.
UC40 - Passo 9 do Fluxo Principal	Não está definido como as informações calculadas devem ser exibidas para o ator.

### Erro Sistemático: Omissão de Definições e Referências

Localização	Descrição do Defeito
Pag. 7 - sec. 4	A definição de "Tipo de Distribuição" utiliza o conceito "recebimento". Antes de verificar como o conceito ?recebimento? era utilizado no documento achei que era sinônimo de parcela. Talvez valesse definir recebimento no glossário.
UC20	Os atores Gerência Financeira e Gerência Administrativa não estão definidos na seção de descrição dos atores (seção 9.1).
UC31	Não existe um requisito associado ao UC31?
UC36-PAG93	Onde está definido o padrão do documento do fluxo A4?

## Erro Sistemático: Inserção de Informação Ambígua

Localização	Descrição do Defeito
UC27	Na descrição do objetivo do caso de uso, ficou claro que o Gerente de Dívidas Ativas visualiza as parcelas que estão vencidas, mas não ficou claro o que os outros atores visualizam: só as que irão vencer ou as que venceram também? A RN4 deu um indicativo para isto, mas precisaria estar definido no objetivo ou no fluxo principal do caso de uso.
UC27 - pag78	A RN 3 está confusa. Ela possui 2 pontos de saída mesmo?
UC28 - Passo 6 do Fluxo Principal	'Lista de Rubricas cadastradas para o Projeto' é mencionada tanto no agrupamento 'origem', quanto no agrupamento 'destino'. Isto pode gerar dúvidas quanto à diferenciação das duas informações, ou ainda quais informações associadas à lista do grupo 'destino' devem ser exibidas.
UC29 - [RN2]	A informação 'distribuição de overhead' não é mencionada pelos passos do fluxo principal relacionados ao procedimento de atualização.
UC32-PAG84	Rever o A1. Deve ser contextualizado no caso de uso.
UC33-PAG86	A definição do fluxo alternativo nos passos 4 e 5 parece confusa.
UC34 - Passo 3 do Fluxo Principal	Não é possível inferir à qual opção 'detalhar' refere-se o passo 3 do fluxo principal de UC34.
UC36	Na RN3, não entendi ?saldos do último dia definido no parâmetro?. Seria o saldo até o dia definido no parâmetro ou seria o saldo daquele dia definido no parâmetro?
UC36 - [A4]	A informação 'dados de cálculo de saldo de projetos' em [A4] é imprecisa e não foi utilizada/definida nos fluxos principal e alternativos, o que pode gerar ambiguidade.
UC39 - Passo 5 do Fluxo Principal	Não estão definidos quais dados de projeto devem ser exibidos.
UC39 - Passos 6, 7, 8 e 9 do Fluxo Principal	No passo 6 está especificado que uma tabela deve ser apresentada com três colunas: valor orçado/Recebido, valor real e diferença. Nos passos 7, 8 e 9 são especificadas, para cada coluna, quais linhas devem ser exibidas. Como as linhas exibem informações diferentes para cada coluna, como gerar uma única tabela com esta especificação? Isto pode gerar ambiguidade.
UC39 - Passo 10 do Fluxo Principal	Não está clara a relação entre 'fundos' e 'plano de aplicação', sendo este último informado entre parênteses. Estes termos, à princípio, não seriam distintos?
UC40 - Passo 6 do Fluxo Principal	Embora sejam definidos no passo 5 quais informações do fundo devem ser exibidas na tabela, há uma exemplo de tabela ao final do UC40 que se assemelha àquela que deve ser exibida no passo 6. Não há nada explícito na especificação que associe a tabela do passo 6 ao exemplo de tabela. Isto pode gerar ambiguidade. Ainda que, de fato, a tabela ao final corresponda a um exemplo da tabela do passo 6, as colunas na primeira não são iguais às colunas que o passo 5 define para a segunda. Neste último caso haveria um indicativo de informação inconsistente.

## Módulo MFI

### Erro Sistemático: Omissão de Campos e Informações

Localização	Descrição do Defeito
UC02 - Fluxo Principal - Passo 3	Não está claro se os campos são editáveis ou não.
UC02 - Fluxo Principal - Passo 6	Não está sendo informado se o campo do número do borderô é editável ou não.
UC03 - Fluxo Principal - Passo 3	Não sendo informado se os campos são editáveis ou não.
UC3 - Fluxo principal (passo 1)	em "Lista com cada tipo de encargo" quais seriam os tipos de encargo?
UC3 - Fluxo principal (passo 4)	Não deveria ter também:  - Quantidade total de movimentos (somente leitura) - Quantidade total de movimentos selecionados (somente leitura)
UC3 - RN8	Onde esta data limite está definida?
UC5 - Fluxo Alternativo 3 (passo 4)	Em que consiste o registro do pagamento do movimento? Apenas a atualização do status do protocolo para ?Pago??
UC07 - Fluxo Principal - Passo 1	Não está claro qual informação de favorecido deve ser considerada.
UC13 - Fluxo principal (passo 3)	O que deve ser exibido na coluna ofício de troca? Um código? Link para download?
UC13 - Fluxo principal (passo 3)	De onde vem o valor investido na conta origem? como calcula este valor?

### Erro Sistemático: Omissão de Regras de Negócio

Localização	Descrição do Defeito
UC2 - Fluxo Alternativo 4 (passo 3)	Em que consiste ?registra a liberação do movimento?? Apenas a mudança de status do protocolo?
UC2 - RN13	A conta fonte 133.212-5 do UNIBANCO seria um parâmetro do sistema? Faltou esta definição.
UC2 - RN13	Qual é o formato do pedido de liberação ao Gerente Financeiro para confirmação da liberação do movimento?

UC2 - RN16	O que significa ?possibilitar gasto de overhead?? Qual o fato que possibilita este gasto?
UC2 - RN6	O sistema é responsável por gerar o ofício de troca? Ele é gerado neste momento?
UC02 - RN6	"Nada deve ser feito" não é suficiente para descrever o cenário estabelecido em RN6.
UC03 - RN6	"Nada deve ser feito" não é suficiente para descrever o cenário estabelecido em RN6.
UC09 - Fluxo principal (passo 9)	Registrar o pagamento consiste em associar o cheque ao encargo ou em algo mais? Seria interessante um detalhamento para o ?registra o pagamento do movimento? nesse caso.
UC11 - Fluxo principal (passo 6)	Faltou mencionar a atualização dos saldos como no passo 9 do UC2, não?
UC12 - Fluxo principal (passo 6)	Faltou mencionar a atualização dos saldos como no passo 9 do UC2, não?

### Erro Sistemático: Omissão de Detalhes de Navegação/Fluxo

Localização	Descrição do Defeito
UC02 - Fluxo Principal - Passo 5	Não está sendo informado como o sistema deve proceder caso o usuário acione a opção de Liberar movimentos sem ter selecionado algum movimento.
UC03 - Fluxo Principal - Passo 5	Não é informado como o sistema deve proceder caso nenhum movimento seja selecionado e a opção de liberação seja acionada.
UC03 - RN7	Não é informado como a mensagem deve ser exibida de modo que o usuário possa lê-la. O passo 8 do fluxo referencia RN7, mas se o fluxo continuar até o passo 11, o sistema voltará ao passo 3 sem ter permitido ao usuário ler a mensagem expressa em RN7.
UC05 - Fluxo Principal - Passo 4	Quando mais de um movimento for selecionado como o sistema deve fazer? Apresentar as telas dos fluxos A3 e A4 para cada movimento?
UC05 - Fluxo Principal - Passo 4	Não é informado como o sistema deve proceder caso nenhum movimento seja selecionado e a opção "Confirmar Pagamento" seja acionada.
UC06 - Fluxo Principal - Passo 5	Não é informado como o sistema deve proceder caso nenhum movimento seja selecionado e a opção "Gerar Voucher" seja acionada.
UC09 - Fluxo Principal - Passo 5	Não é informado como o sistema deve proceder caso nenhum encargo seja selecionado e a opção "Confirmar Pagamento" seja acionada.
UC13 - Fluxo Principal - Passo 5	Não é informado como o sistema deve proceder caso nenhuma linha seja selecionada e a opção "Confirmar Troca" seja acionada.
UC14 - A2 - Passo	Não é informado qual o passo seguinte que deve ser seguido ao sair de A2.



3	
UC14 - A3 - Passo 3	Não é informado qual o passo seguinte que deve ser seguido ao sair de A3.

### Erro Sistemático: Omissão de Definições e Referências

Localização	Descrição do Defeito
Sec. 7 - RF13	- O que representa este ajuste?  - "a partir dos movimentos e encargos" significa o que nesse contexto? O que é necessário dos movimentos e encargos para possibilitar ou criar a necessidade do ajuste?
UC02 - Fluxo principal (passo 1)	O termo Conta Origem deveria ser definido no glossário.
UC02 - Fluxo principal (passo 6)	Definir "conta fonte" no glossário.
UC02 - RN15	No glossário não está descrito o que é "gasto de overhead" ou "overhead".
UC03 - Fluxo principal (passo 9)	A RN15 do UC2 não se aplica neste caso?

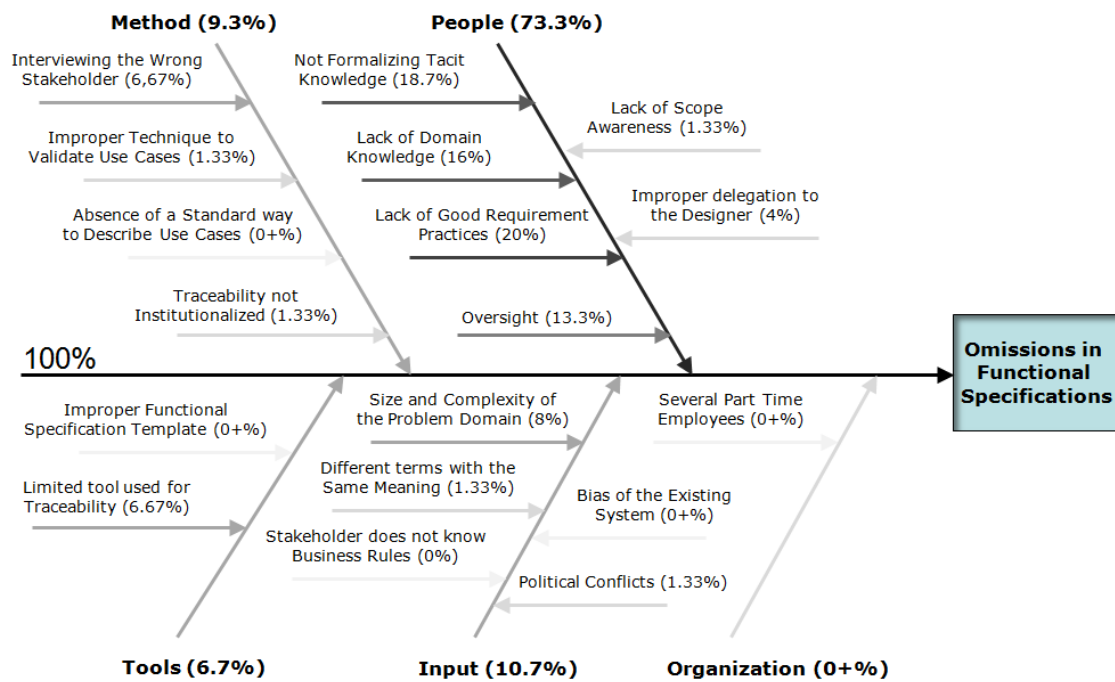
### Erro Sistemático: Inserção de Informação Inconsistente

Localização	Descrição do Defeito
UC02 - RN1	RN1 menciona que apenas protocolos relacionados a movimentos podem ser exibidos. Entretanto, esta regra está sendo referenciada em A1, cujo contexto não envolve protocolos.
UC02 - Fluxo Principal - Passo 3	O nome da opção que consulta as informações de saldo é "Consultar Saldo de Centro de Custo", mas em A6 diz-se que a opção chama-se "Saldo do Centro de Custo". Se as informações forem dispostas em uma tabela, "Saldo do Centro de Custo" poderia ser o nome da coluna.
UC2 - RN9	Quando existe uma solicitação de transferência e não há liberação do protocolo, qual seria o status desse protocolo que não é liberado? Sempre assumiria o status "pago"?
UC4 - Fluxo Principal (passo 2)	Seria mesmo para cada protocolo a ser liberado? Não é apenas selecionado um protocolo no UC2, passo 3?
UC05 - RN2	Em RN2 diz-se que o número do voucher pode ser alterado na tela, mas no passo 1 de A3 é dito que não.
UC 7 - passo 1, fluxo	Ao apresentar no filtro de busca os campos "Favorecido", "Borderô", "Cheque" e "Voucher",

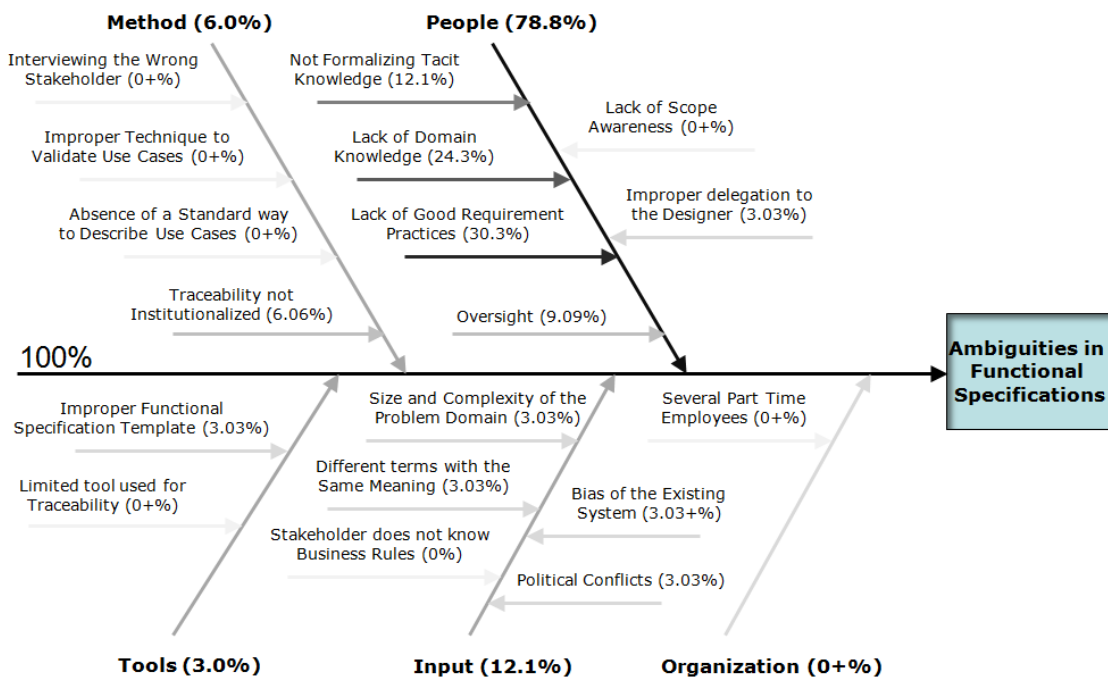
principal (página 27)	não existe descrição se os mesmos serão campos editáveis e que tipo de valores podem receber. Também existe uma combo logo abaixo esses campos que indica a modalidade de pagamento (opções: Todos, Boleto, Borderô, Cheque, Débito Automático, Fundo Fixo de Caixa). Não fica claro se existe ou não algum tipo de relação dos campos "Borderô", "Cheque" e "Voucher" com a combo modalidade de pagamento.
UC 12 - objetivo (página 35)	O objetivo especificado diz que "O sistema deve possibilitar o cancelamento de movimentos já pagos pela Tesouraria.", porém o RF12, ao qual ele se encontra ligado, diz que "O software deve permitir o cancelamento de movimentos a qualquer momento, seja antes da sua liberação, antes do seu pagamento ou após seu pagamento."
Todos os casos de uso que utilizem centro de custo (listados abaixo)	Em vários casos de uso, é utilizado o campo centro de custo, que seria formado por um campo editável + opção de busca. Caso a opção de busca seja selecionada, abre-se uma nova tela onde é informado o filtro da busca: curso, congresso, projeto, fundo, etc. Caso o usuário opte por digitar uma informação, o caso de uso não informa o que deveria ser digitado, número, título, etc. Ucs que utilizam centro de custo: UC 2, UC 3, UC 5 e UC 7

### F3.3 Diagramas de Causa e Efeito Probabilísticos

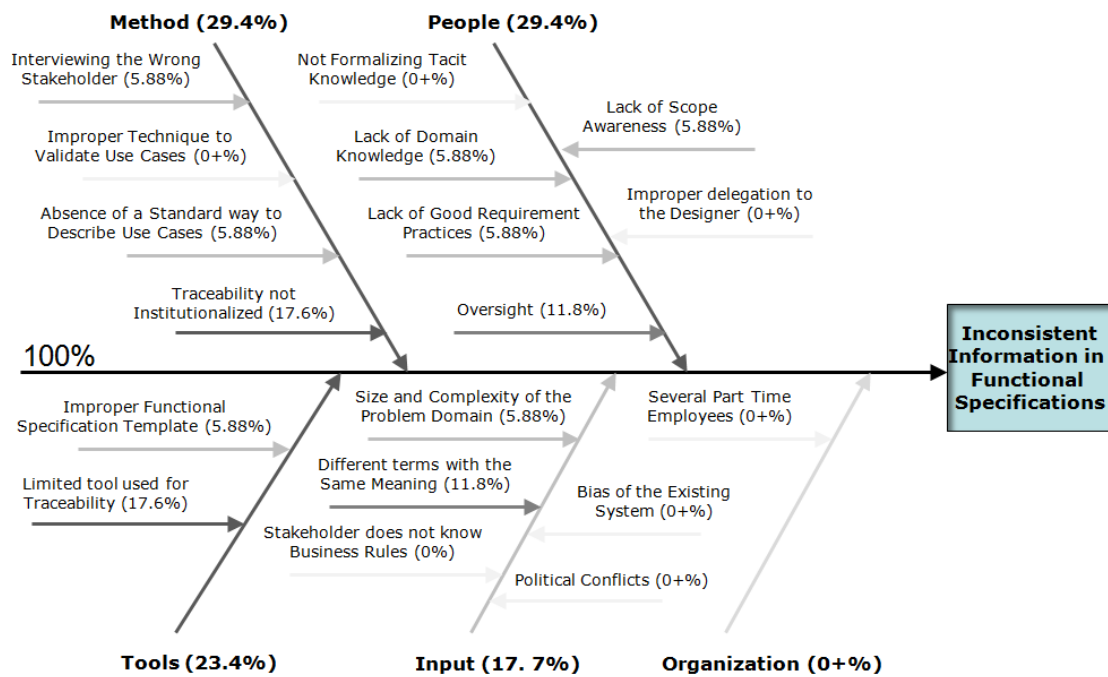
#### Tipo de Defeito: Omissão



#### Tipo de Defeito: Ambiguidade



## Tipo de Defeito: Informação Inconsistente



## F4 - Questionário de Acompanhamento

Nome:

Estas são perguntas que precisamos fazer de forma a tornar mais eficazes os dados obtidos ao longo do estudo.

1. Em relação à complexidade da tarefa realizada:

☐ Eu considere a tarefa muito complexa.

☐ Eu considere a tarefa complexa.

☐ Eu considere a tarefa simples.

☐ Eu considere a tarefa muito simples.

Que dificuldades você encontrou para realizar esta tarefa?

---

---

2. Em relação à sua satisfação na realização da tarefa:

☐ Eu fiquei totalmente satisfeito.

☐ Eu fiquei largamente satisfeito.

☐ Eu fiquei parcialmente satisfeito.

☐ Eu fiquei insatisfeito.

Em sua opinião, o que poderia tornar a realização desta tarefa mais agradável?

---

---

3. Em relação à sua satisfação com a corretude dos resultados obtidos:

☐ Eu fiquei totalmente satisfeito.

☐ Eu fiquei largamente satisfeito.

☐ Eu fiquei parcialmente satisfeito.

☐ Eu fiquei insatisfeito.

Em sua opinião, o que poderia ser feito para melhorar a corretude na identificação das principais causas dos defeitos?

---

---

4. Considerando uma escala de 0 (péssimo) a 10 (excelente), como você avaliaria sua corretude na identificação das principais causas dos defeitos? \_\_\_\_.

5. Descreva os procedimentos que utilizou para identificar as causas dos defeitos.

---

---

---

6. Quanto tempo você gastou para realizar a tarefa (em minutos): \_\_\_\_\_.

O que você acredita que poderia ser feito para realizar a tarefa em menos tempo?

---

---