Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estudo sistemático em dependabilidade e métodos ágeis: uma análise de falhas e defeitos

Renato dos Santos Leal
Artur de Azevedo Braga

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof.ª Dr.ª Genaína Nunes Rodrigues

Brasília
2013

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenadora: Prof.ª Dr.ª Maristela Terto de Holanda

Banca examinadora composta por:

Prof.ª Dr.ª Genaína Nunes Rodrigues (Orientadora) — CIC/UnB
Prof. Dr. Vander Ramos Alves — CIC/UnB
Prof. Dr.ª Carina Frota Alves — CIn/UFPE

CIP — Catalogação Internacional na Publicação

Leal, Renato dos Santos.

Estudo sistemático em dependabilidade e métodos ágeis: uma análise de falhas e defeitos / Renato dos Santos Leal, Artur de Azevedo Braga. Brasília : UnB, 2013.
74 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2013.

1. dependabilidade, 2. confiabilidade, 3. métodos ágeis, 4. falha, 5. defeito

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estudo sistemático em dependabilidade e métodos ágeis: uma análise de falhas e defeitos

Renato dos Santos Leal
Artur de Azevedo Braga

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.ª Dr.ª Genaína Nunes Rodrigues (Orientadora)
CIC/UnB

Prof. Dr. Vander Ramos Alves
CIC/UnB

Prof. Dr.ª Carina Frota Alves
CIn/UFPE

Prof.ª Dr.ª Maristela Terto de Holanda
Coordenadora do Bacharelado em Ciência da Computação

Brasília, 17 de setembro de 2013

Dedicatória

Este trabalho é dedicado às nossas famílias, que nos apoiam em cada decisão sem nos faltar em nenhum momento. Dedicamos também às pessoas que estiveram conosco e nos ajudaram durante a graduação, e à toda comunidade que contribui para a evolução dos métodos ágeis.

Agradecimentos

Agradecemos às nossas famílias, que estão sempre ao nosso lado em cada novo desafio que enfrentamos, nos educam e sustentam acreditando sempre nos nosso sucesso, e nos deram a oportunidade de cursar a graduação na Universidade de Brasília.

À nossa orientadora, Professora Doutora Genaína Nunes Rodrigues, que aceitou enfrentar conosco este desafio e nos acompanhou durante todo o processo, sempre nos ajudando para a nossa excelência e sucesso.

Agradecemos também à Empresa Júnior de Computação - CJR, que foi um grande divisor de águas em nosso desenvolvimento, abriu nossas portas para o empreendedorismo e contribuiu imensuravelmente para o nosso crescimento, profissional e pessoal. À Federação das Empresas Juniores do Distrito Federal - Concentro e à Confederação Nacional de Empresas Juniores - Brasil Júnior, que nos ensinaram que o todo é maior do que a soma das partes, e que possuímos um potencial enorme para transformar a nossa sociedade.

Por fim, agradecemos a todos aqueles que marcaram nossa história, contribuíram para o nosso crescimento e para a conclusão deste ciclo, que estarão sempre ao nosso lado, pois sabemos que esta é apenas mais uma etapa do processo de desenvolvimento.

Abstract

Nos últimos anos os métodos ágeis têm ganhado cada vez mais espaço no cenário mundial de desenvolvimento de software, tanto em empresas de pequeno como grande porte. Apesar dos grandes avanços destas metodologias, que possuem um foco em colaboração entre indivíduos e responder rápido a mudanças, há poucos estudos publicados acerca dos impactos que a utilização de métodos ágeis causa na confiabilidade do software. Este trabalho apresenta um estudo sistemático sobre a literatura existente, analisando principalmente os efeitos que as práticas ágeis têm sobre a ocorrência de falhas e defeitos no desenvolvimento do software e quais etapas do desenvolvimento possuem maior relação com elas.

Palavras-chave: dependabilidade, confiabilidade, métodos ágeis, falha, defeito

Abstract

In recent years agile methods have been gaining more space over the world stage of software development, within both large and small companies. Despite the great advances on these methodologies, which focus on collaboration among individuals and responding fast to changes, there are few published studies about the impact using agile methods causes in software reliability. This work presents a mapping study on the existing literature, analysing mainly the effect agile practices have on the occurrence of faults and failures within software development and what stages of development possess major relation with them.

Keywords: dependability, reliability, agile methods, fault, failure

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

Nos últimos anos as metodologias ágeis vem apresentando um grande avanço, em termos de utilização pela indústria e academia, tanto no Brasil [40] como no mundo [55]. Muito deste sucesso é pautado nas suposições de que tais metodologias e suas práticas além de melhorar a gerência do projeto como um todo, elevam a qualidade do produto e a satisfação do cliente. Muitos estudos abordam esta temática trazendo principalmente os benefícios encontrados ao utilizá-las, no entanto ao fazer uma busca mais profunda percebe-se que tais estudos muitas vezes trazem resultados e colocações apenas quanto ao sucesso e falha de projetos, a qualidade em geral, ou mesmo a produtividade das equipes. Estudos comparatórios entre tais métodos e metodologias tradicionais são abundantes, e também é possível encontrar estudos secundários de excelente qualidade e aceitação como o trabalho apresentado por Dyba et al. [7].

Tendo isso em mente, é preocupante perceber que poucos dos estudos realizados pela comunidade acadêmica e profissional se voltam para uma análise mais profunda dos termos de dependabilidade de software quando falamos de metodologias ágeis. A inexistência de trabalhos considerando estes dois grandes temas é, talvez, fruto da dificuldade encontrada por pesquisadores ao tentar associar os conceitos de dependabilidade com os métodos iterativos e da falta de técnicas de avaliação de dependabilidade quando utilizamos tais metodologias [50] [14], dado que muitas vezes são prezadas questões como mudança contínua, diminuição da documentação e escopo variável de acordo com o tempo do projeto.

1.2 Problema

O problema a ser resolvido neste trabalho, de forma geral, é identificar estudos e evidências na literatura especializada que reúna os principais estudos que abordem os temas de dependabilidade e métodos ágeis, considerando principalmente a análise dos conceitos de falhas e defeitos geradas por projetos que apliquem as técnicas de dependabilidade para tornar mais robustos os softwares desenvolvidos por metodologias ágeis.

1.3 Objetivos

1.3.1 Objetivos Gerais

Este trabalho tem como objetivo geral investigar e reunir de forma sistemática o conhecimento disponível na literatura especializada sobre os temas, através de um agrupamento dos estudos relevantes que abordam a ocorrência de falhas e defeitos com a utilização de métodos ágeis. Outros objetivos incluem identificar se há falta de literatura especializada sobre o tema e deste modo trazer mais atenção ao tema além de servir como base de consulta para que outros estudos possam ser realizados. Para atingir tais objetivos realizamos um estudo de mapeamento sistemático com questões que, ao serem respondidas, colocam em evidência onde estão sendo colocados, ou não, os esforços da comunidade acadêmica, além das práticas ágeis que propiciam uma maior qualidade de software quando abordados em termos de dependabilidade.

1.3.2 Objetivos Específicos

As questões de pesquisa, que serão apresentadas no Capítulo 4, buscam elucidar os seguintes objetivos:

- Identificar quais são os polos de estudo sobre o tema, classificando-os entre a academia e o mercado.
- Identificar as principais práticas de métodos ágeis que auxiliam no desenvolvimento de um software confiável.
- Identificar as fases do projeto ágil mais abordadas pela literatura em que o projeto de software se encontra mais suscetível à ocorrência de falhas e defeitos.

1.4 Metodologia

O método utilizado neste trabalho foi realizado em três etapas: inicialmente foi feito um levantamento do referencial teórico sobre os assuntos de métodos ágeis e dependabilidade para a melhor compreensão do objeto de estudo. Em seguida tivemos a realização do estudo sistemático (*systematic mapping study*) para a identificação, classificação e análise dos estudos de relevância para nosso trabalho. A última etapa foi a escrita desta monografia.

1.5 Organização do trabalho

Este trabalho se divide em seis capítulos, os quais:

Capítulo 2 Traz um referencial teórico sobre métodos ágeis que compõem o domínio deste estudo, nele serão apresentadas as principais práticas utilizadas por estes métodos, as quais são introduzidas no manifesto ágil [9] e utilizadas por todas metodologias ágeis. Também serão investigadas algumas práticas que se restringem a metodologias específicas mais conhecidas, como a programação extrema, ou XP,

de Beck [2], o Scrum de Schwaber [48] e o Lean Development de Poppendieck e Poppendieck [47].

Capítulo 3 São apresentados a teoria e os principais conceitos de dependabilidade que serão tratados neste trabalho, como eles podem ser classificados e a importância dos mesmos em um software. Ao fim do capítulo o leitor deve ter uma fundamentação teórica necessária para entender todo o tema do estudo, quais são os atributos, meios e ameaças tratados em dependabilidade.

Capítulo 4 Traz uma apresentação da teoria sobre a realização de estudos de mapeamento sistemático (*mapping studies*) explicando cada passo realizado e mostrando como o mesmo foi feito em nosso trabalho além de apresentar os resultados obtidos em cada etapa. Ao fim deste capítulo teremos uma direção do rumo do trabalho visto que é aqui onde são apresentados os *gaps* encontrados quando abordamos os tópicos dependabilidade e métodos ágeis.

Capítulo 5 Traz uma análise relacionando diversos pontos sobre dependabilidade em métodos ágeis, ou seja, são analisados neste capítulo as práticas que influenciam nos atributos de dependabilidade por nós escolhidos e como a utilização da metodologia ágil impacta no software em geral. É neste capítulo que são apresentados os resultados obtidos por nossa pesquisa.

Capítulo 6 São apresentadas as limitações e ameaças à validade do trabalho, assim como são trazidas propostas de trabalhos futuros a partir dos resultados encontrados e finaliza-se com as conclusões do trabalho.

Capítulo 2

Métodos Ágeis

Em meados dos anos 1990 a popularização dos então chamados "métodos leves" (*lightweight development methods*), alavancada como uma reação aos "métodos pesados" (*heavywheight development methods*) tradicionais mais comumente utilizados - como o modelo cascata (*waterfall*) - despertou o interesse de desenvolvedores em se reunirem para discutir tais métodos. Em fevereiro de 2001, dezessete desenvolvedores se reuniram na cidade de Snowbird, Utah, e escreveram o Manifesto para Desenvolvimento Ágil de Software [13], passando a chamar os métodos leves de "métodos ágeis". Na Tabela 2.1 apresentamos brevemente alguns dos principais métodos ágeis: o XP, o Scrum e o Desenvolvimento Lean.

2.1 Manifesto para Desenvolvimento Ágil de Software

O Manifesto [9] apresenta uma abordagem mais dinâmica no processo de desenvolvimento de software, colocando iterações, adaptabilidade e colaboração com o cliente como fatores mais importantes do que documentação abrangente, seguir o planejado e negociação de contratos, conforme apresentado na Figura 2.1.

Os métodos ágeis têm como princípios, segundo Beck et al. [9]:

> Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
>
> Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
>
> Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
>
> Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
>
> Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
>
> O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa cara a cara.
>
> Software funcionando é a medida primária de progresso.

Tabela 2.1: Métodos Ágeis. Traduzido de Dybå e Dingsøyr [7].

Método Ágil	Descrição
Programação Extrema (XP)	Focado nas melhores práticas de desenvolvimento. Consiste em 12 práticas: jogo do planejamento, entregas frequentes, metáfora, projeto simples, testes, refatoração, programação em pares, propriedade coletiva, integração contínua, semanas de 40 horas, cliente presente, padrões de codificação. A revisão "XP2"consiste nas seguintes "práticas primárias": trabalhar junto, time coeso, ambiente de trabalho informativo, trabalho energizado, programação em pares, histórias, ciclo semanal, ciclo trimestral, afrouxamento, 10-minute build, integração contínua, testar primeiro, projeto incremental.
Srcum	Focado no gerenciamento de projetos em situações em que é difícil planejar à frente, com mecanismos para "controle de processo empírico"; onde ciclos de feedback consistem no elemento principal. Software é desenvolvido por um time auto-organizado em incrementos (chamados de "*sprints*"), iniciando com o planejamento e finalizando com a revisão. Funcionalidades que serão implementadas no sistema são registradas em um *backlog*. O *Product Owner* decide quais itens do *backlog* devem ser desenvolvidos no próximo *sprint*. Membros da equipe coordenam seu trabalho em uma reunião em pé diária. Um membro da equipe, o *Scrum Master*, é encarregado de resolver problemas que impedem a equipe de trabalhar efetivamente.
Desenvolvimento Lean	Uma adaptação dos princípios da produção enxuta e, em particular, do sistema de produção Toyota, para desenvolvimento de software. Consiste em sete princípios: eliminar desperdício, ampliar o aprendizado, decidir o mais tarde possível, entregar o mais rápido possível, empoderar o time, construir integridade e enxergar o todo.

Figura 2.1: Manifesto para Desenvolvimento Ágil de Software [9]

> Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
>
> Contínua atenção à excelência técnica e bom design aumenta a agilidade.
>
> Simplicidade–a arte de maximizar a quantidade de trabalho não realizado–é essencial.
>
> As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
>
> Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Metodologias ágeis são baseadas no desenvolvimento incremental, tendo por base a entrega frequente de software funcional. O desenvolvimento ágil não nega a importância da documentação do software, porém prega que para o cliente vale muito mais ver um programa em funcionamento do que vários modelos UML.

A satisfação do cliente é um dos focos dos métodos ágeis, utilizando-se de ciclos de iteração curtos para poder-se apresentar resultados efetivos do desenvolvimento. Além disso existe uma proximidade entre os desenvolvedores e o cliente, existindo uma colaboração das partes no que diz respeito à adequação dos requisitos, propiciando a rápida identificação em caso de inconformidade das funcionalidades implementadas com os requisitos do cliente.

Adaptabilidade implica aceitar alterações de requisitos mesmo durante etapas avançadas do desenvolvimento do software, buscando entregar o melhor produto para as necessidades do cliente. Os métodos ágeis entendem a imprevisibilidade existente no processo de desenvolvimento de software e, ao invés de resistir às mudanças, busca aplicá-las da forma mais simples e eficiente possível.

A arte de maximizar a quantidade de trabalho não realizado: simplicidade implica em desenvolver o essencial, com qualidade, visto que é mais fácil alterar algo simples, quando se fizer necessário, do que algo complexo, além de diminuir o custo de mudanças e minimizar a programação desnecessária.

2.2 Práticas dos Métodos Ágeis

Em seu livro *Extreme Programming Explained: Embrace Change*, Beck [3] descreve práticas comuns a metodologias ágeis que são utilizadas no desenvolvimento de software, explicando como são aplicadas e sua importância. O XP considera estas práticas e as leva a níveis extremos, na teoria de que "se algo é bom, mais é melhor".

Cliente Presente (*On-Site Customer*) - O cliente participa ativamente do projeto, sendo considerado parte da equipe de desenvolvimento. Está presente para auxiliar a equipe e disponível para esclarecer dúvidas a qualquer momento, visto que é o usuário final do sistema.

Integração Contínua (*Continuous integration*) - Todo novo incremento desenvolvido deve ser integrado imediatamente ao sistema, precisando obrigatoriamente passar em todos os testes de unidade e não pode fazer com que o sistema já existente deixe

de funcionar em qualquer aspecto. Isto garante a integridade do sistema e que o código disponível é sempre o mais atualizado possível.

Projeto Simples (*Simple Design*) - O incremento desenvolvido deverá conter somente as funcionalidades necessárias no momento, e nada além disso. Manter o projeto simples facilita a manutenção e a correção do código e evita trabalho desnecessário.

Desenvolvimento Orientado a Testes (*Test Driven Development* - TDD) - Primeiramente são escritos os testes em que o novo código deverá ser aceito e, somente então, o novo incremento é desenvolvido. Esta prática minimiza o esforço gasto com funcionalidades que não são necessárias no momento. Nem todas as metodologias ágeis utilizam o TDD, porém, em sua maioria, ainda assim realizam testes de unidade em seus incrementos para validade seu funcionamento correto.

Programação em Pares (*Pair Programming*) - Todo o código é escrito com duas pessoas utilizando uma mesma máquina, cada par contendo dois papéis: o programador que está utilizando o teclado deve se preocupar com a melhor maneira de implementar o código, enquanto o outro programador deve preocupar-se com a abordagem do problema em seu contexto, se há outros casos de teste que podem ser aplicados ou se há alguma forma de simplificar o sistema.

Refatoração (*Refactoring*) - O sistema deve possuir o projeto mais simples possível e receber novos incrementos com a maior facilidade. Refatoração é a alteração no código ou projeto do sistema de forma a atingir estes requisitos, sem deixar de executar nenhum dos testes corretamente.

Padrão de Código (*Coding Standards*) - Como todo o código pertence à equipe e vários programadores vão alterar alguma parte do código em dado momento, todos devem seguir as mesmas práticas de programação. Para ser considerado devidamente padronizado, deve ser impossível distinguir qual programador escreveu cada parte do código.

Testes de Aceitação (*Acceptance Tests*) - Como o cliente é de fácil acesso, testes de aceitação devem ser executados frequentemente para garantir o correto funcionamento do sistema. Os testes são executados pelo cliente, funcionalidade por funcionalidade, e os resultados são compartilhados com a equipe.

2.3 Etapas do Desenvolvimento Ágil

As etapas de desenvolvimento das metodologias ágeis, segundo Sommerville [51], têm como fundamentação o processo de Desenvolvimento Incremental. Este modelo de desenvolvimento de software propõe atividades bastante semelhantes com os Princípios Ágeis [9].

No Desenvolvimento Incremental (Figura 2.2) os clientes inicialmente identificam, de forma simplificada, quais os requisitos do sistema e quais são os mais e os menos importantes. Em seguida são definidas as iterações de entrega, sendo cada iteração um novo incremento que trará novas funcionalidades de acordo com a priorização dos requisitos.

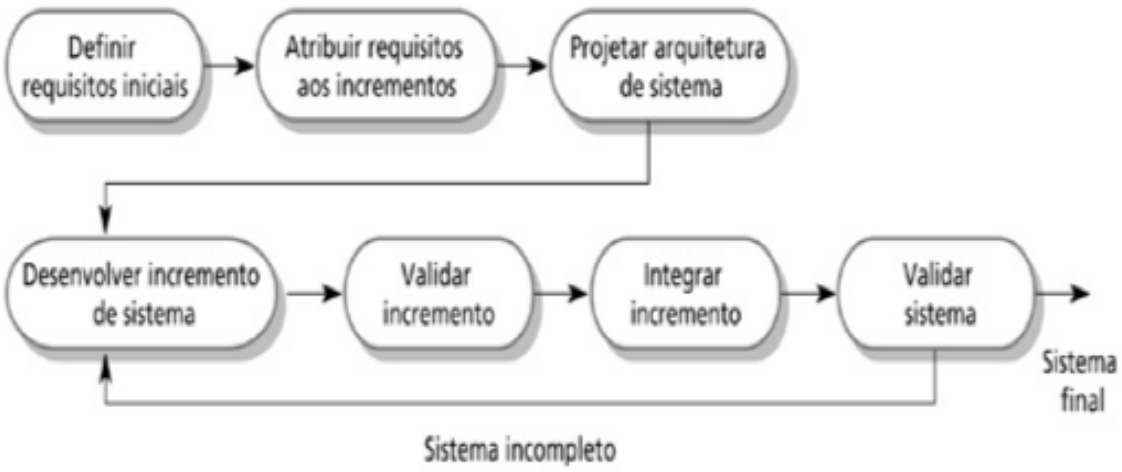

Figura 2.2: Desenvolvimento incremental [51]

Os incrementos devem ser validados e integrados ao sistema, validando-se então o sistema com o novo incremento, até que todas as iterações sejam finalizadas.

Utilizando como base o modelo de Desenvolvimento Incremental, podemos definir um modelo simplificado que representa o processo de Desenvolvimento Ágil de Software, representado na Figura 2.3. Cada ciclo iterativo é composto de etapas de requisitos, modelagem, desenvolvimento, teste, integração do incremento e validação do sistema. Assim, podemos definir cada etapa da seguinte forma:

Definir requisitos iniciais - São levantados os requisitos gerais, funcionalidades, principais entradas e saídas, restrições, que serão utilizados para modelar o sistema.

Modelar sistema - Modelagem mais simples possível do sistema para que este funcione, pois a cada novo ciclo iterativo podem ocorrer mudanças nos requisitos. Também são definidas e priorizadas as funcionalidades que serão desenvolvidas ao longo do projeto.

Definir requisitos do incremento - Definição de todos os requisitos do incremento que será desenvolvido, quais funcionalidades deve apresentar e quando deve ser entregue.

Modelar incremento - O incremento é modelado da forma mais simples possível, sendo estimadas as maneiras de se implementar as funcionalidades e a forma como será integrado ao sistema.

Desenvolver e validar incremento - As funcionalidades do incremento são desenvolvidas e testadas de acordo com a modelagem. Os testes podem ser escritos antes ou depois do código do incremento, de acordo com a metodologia utilizada.

Integrar incremento e validar sistema - O incremento entregue é integrado ao sistema, que então é testado (testes de integração e testes de aceitação). O ciclo só pode ser finalizado se o incremento cumprir todas as suas funcionalidades e o sistema esteja funcionando corretamente.

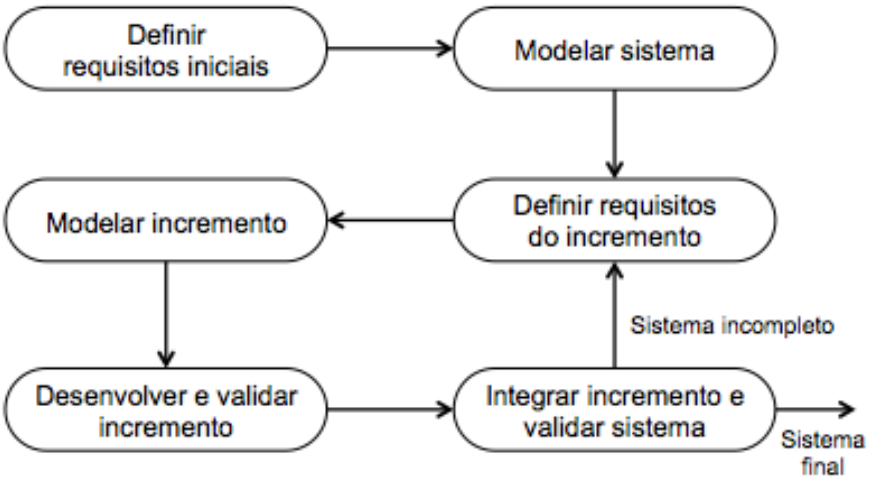

Figura 2.3: Desenvolvimento Ágil

2.4 Limitações

Apesar da grande popularização dos métodos ágeis, ainda há limitações quanto à sua adoção. Tanto o XP [3] quanto o Scrum [48] são apresentados como metodologias para equipes pequenas (usualmente variando entre sete e quinze participantes). Sommerville [51] também defende que, em casos gerais, as metodologias ágeis não são indicadas para sistemas muito complexos ou, principalmente, para sistemas críticos, pois estes carecem de documentação formalizada e extensa para garantir seu correto funcionamento.

Em seus estudos, Petersen e Wohlin [45] [46] apresentam várias dificuldades oriundas da implementação de metodologias ágeis em grandes empresas, como a coordenação de grandes equipes, realização e controle contínuos dos testes e dificuldades de integração devida à falta de foco na arquitetura.

Capítulo 3

Dependabilidade

Ainda como parte da teoria necessária para realização deste trabalho devemos entender o conceito de dependabilidade de sistemas computacionais (*dependability*) como sendo a habilidade de entregar um serviço que pode, justificadamente, ser confiado (Avizienis et al. [1]), e também três conceitos básicos necessários para um estudo aprondudado do tema, são eles: as ameaças à dependabilidade do sistema, os atributos de dependabilidade e os meios que permitem alcançá-la.

Neste capítulo serão apresentadas breves descrições dos atributos de dependabilidade existentes, focando no atributo de confiabilidade (*reliability*), para que então possamos falar mais detalhadamente sobre as ameaças e meios de se obter um sistema no qual se possa confiar.

3.1 Atributos

Atributos são qualidades do sistema, que podem definir a dependabilidade utilizando medidas qualitativas e quantitativas no funcionamento de um sistema ou em sua documentação. De acordo com a definição trazida por Avizienis et al. [1] podemos dividir o conceito de dependabilidade em seis atributos principais, sendo eles:

1. Disponibilidade (*Availability*): é a probabilidade de um sistema estar funcionando corretamente em um dado instante do tempo.
2. Confiabilidade (*Reliability*): probabilidade de um sistema estar funcionando corretamente de acordo com as especificações dadas, dentro de certas condições em um período de tempo.
3. Segurança (*Safety*): ausência de consequências catastróficas para os usuários (humanos ou outros sistemas) e/ou para o próprio ambiente em que o sistema está inserido.
4. Integridade (*Integrity*): ausência de alterações que modifiquem o comportamento do sistema, ou seja, o sistema mantém-se conforme entregue e os dados continuam válidos.
5. Manutenibilidade (*Maintainability*): habilidade de receber reparos e modificações, com o mínimo de indisponibilidade possível e com o mínimo de impacto para os

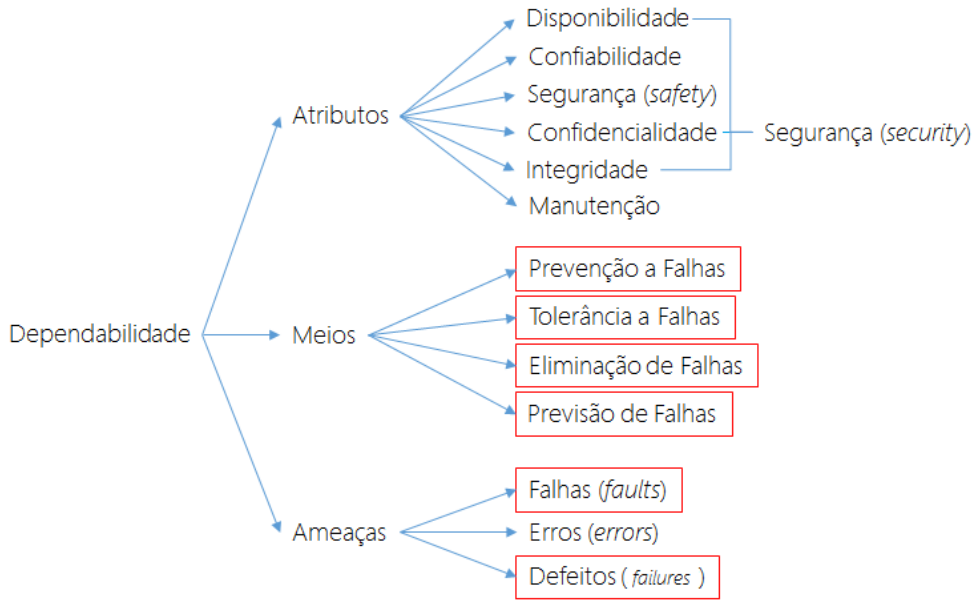

Figura 3.1: Árvore de Dependabilidade. Tradução de Avizienis et al. [1]. Os destaques em vermelho são o foco da análise em nosso trabalho.

usuários, sejam estas modificações correções de falhas ou implementação de novas funcionalidades.

6. Confidencialidade (*Confidentiality*): a ausência de divulgação não-autorizada de informações. Alguns autores não o classificam como um atributo primário e sim um atributo derivado quando se avalia a questão de segurança em nível de controle de acesso.

A combinação destes atributos nos trazem alguns outros derivados como o requisito não-funcional de segurança (*security*), que corresponde a junção dos atributos integridade, disponibilidade e confidencialidade e diz respeito à quantidade de erros originados externamente ao sistema.

Vale lembrar aqui que para sistemas específicos podem existir atributos que se sobressaem quanto a sua importância, como por exemplo, sistemas militares devem ter como prioridade o atributo de segurança (*safety*) enquanto sistemas bancários prezam pela disponibilidade (*availability*) e segurança.

Nosso trabalho terá como foco o estudo da confiabilidade (*reliability*) de sistemas computacionais. A seguir apresentaremos definições mais detalhadas para este atributo e os conceitos de ameaças e meios que nos ajudarão a definí-lo de melhor forma.

3.1.1 Confiabilidade

A confiabilidade é talvez o atributo mais importante quando falamos de dependabilidade e é a medida mais usada em sistemas em que mesmo curtos períodos de operação incorreta são inaceitáveis, ela pode ser considerada um fator chave para expressar o sucesso ou o fracasso de uma aplicação (Lyu [34]). Definimos então este atributo como a

probabilidade de que um sistema desempenhará satisfatoriamente (ou adequadamente) o seu propósito específico por um dado período até a ocorrência da primeira falha (Maciel et al.[43], Xie et al. [36], Leemis [31]). Como a confiabilidade é uma função de probabilidade então podemos realizar previsões e medições a partir de uma equação matemática na qual a função de confiabilidade R(t) é a probabilidade de que o sistema irá funcionar corretamente sem falha no intervalo de tempo de 0 a t e T é uma variável aleatória representando o tempo de falha ou tempo para falha.(Maciel et al.[43]), conforme apresentado na Equação 3.1 a seguir.

$$R(t) = P(T > t), t >= 0 \tag{3.1}$$

A probabilidade de falha, ou inverso da confiabilidade, é então representada por:

$$F(t) = 1 - R(t) = P(T < t) \tag{3.2}$$

que é conhecida como a função de distribuição de T, a qual nos diz que um componente irá funcionar corretamente (isto é, não apresentará defeitos) até o tempo t.

Devemos ressaltar ainda que a confiabilidade não deve ser confundida com disponibilidade. Um sistema pode ser altamente disponível mesmo apresentando períodos de inoperabilidade, desde que esses períodos sejam curtos e não comprometam a qualidade do serviço.

Métricas de Confiabilidade

A confiabilidade de um sistema é medida através da contagem do número de falhas em um sistema, para isso mostramos algumas medidas básicas apresentadas em Musa et al. [39] e Fenton e Littlewood [12]:

1. Probabilidade de Falha sob Demanda - *(Probability of Failure on Demand)* - PFD

   É a probabilidade de que o sistema irá falhar quando uma requisição é feita. Uma PFD de 0.001 significa que uma em cada mil requisições poderá resultar em falha. É bastante importante para sistemas críticos.

2. Taxa de ocorrência de falta - *Rate of Fault Occurrence (ROCOF)*

   É a frequência da ocorrência de falhas. Uma taxa de 0.02 significa que provavelmente existirão duas falhas a cada cem unidades de tempo. É importante para sistemas de processamento de transações.

Ainda deseja-se conhecer o tempo esperado para que ocorra a próxima falha. Esta métrica é chamada de Tempo Médio Para Falha (Mean Time To Failure - MTTF) e é definida como o valor esperado de operação do sistema antes da falha ocorrer.

No entanto a medida de tempo médio para falha não se mostra suficiente visto que ela só considera o caso em que um sistema funciona de forma correta e então sofre uma interrupção, devemos também considerar que ao sofrer uma unterrupção esse sistema passará um tempo indisponível e portanto esse tempo também deve ser medido, para isso é necessário definir a métrica de Tempo Médio Para Reparo (Mean Time To Repair - MTTR) de um componente com defeito e a partir das duas métricas apresentadas acima então teremos uma métrica que poderá nos dizer quanto tempo o sistema esteve

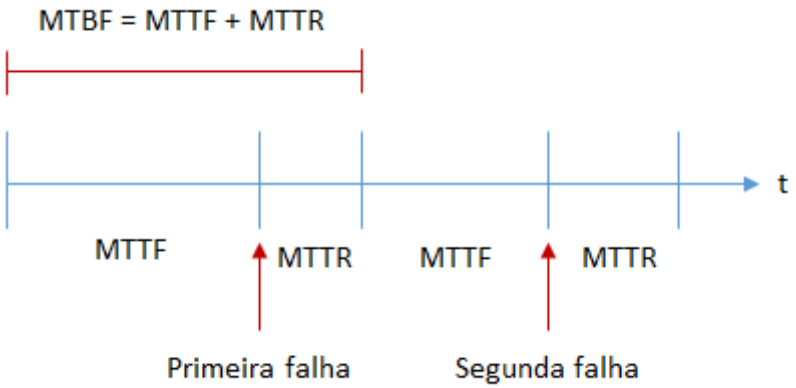

Figura 3.2: Relação entre métricas de dependabilidade.

indisponível, chamamos esta métrica de Tempo Médio Entre Falhas (Mean Time Between Failures - MTBF) e a medimos com a Equação 3.3. A relação entre estas três medidas é apresentada na Figura 3.2.

$$MTBF = MTTF + MTTR \tag{3.3}$$

3.2 Ameaças

Ao definirmos a dependabilidade e confiabilidade, são introduzidos conceitos de ameaças a dependabilidade de um software que podem ser consideradas como fatores os quais podem afetar um sistema e diminuir sua dependabilidade, podendo ser divididas, de acordo com a tradução encontrada em Dantas [5], nos três conceitos seguintes: falhas (*faults*), erros (*errors*) e defeitos (*failures*).

A falha (*fault*) ou falta, é um estado incorreto do hardware ou do software resultante de problemas associadas ao universo físico, erros ao universo da informação e defeitos ao universo do usuário. Um sistema pode apresentar uma ou mais falhas e não apresentar erros. Neste caso, a falha é conhecida como falha dormente (ou latente). Por outro lado, quando a falha efetivamente produz um erro passa a ser classificada como falha ativa. O tempo de entre o surgimento da falha e sua ativação, ou produção do erro, é chamado de latência da falha.

O erro (*error*) é definido, segundo Laprie [29], como uma manifestação da falha no sistema, correspondendo a um estado do sistema que pode levar a outras falhas ou pode não causar nenhuma interrupção de algum serviço, apenas inconsistência entre dados esperados e percebidos.

Já o defeito (*failure*) pode ser definido como a ocorrência de um desvio de comportamento do sistema do que seria o comportamento correto, ele pode ocorrer devido ao fato que o sistema não atende à sua especificação, ou devido ao fato de sua especificação não descrever a funcionalidade do sistema corretamente (Avizienis et al. [1]). A transição do serviço incorreto para o serviço correto corresponde a recuperação do serviço, e o

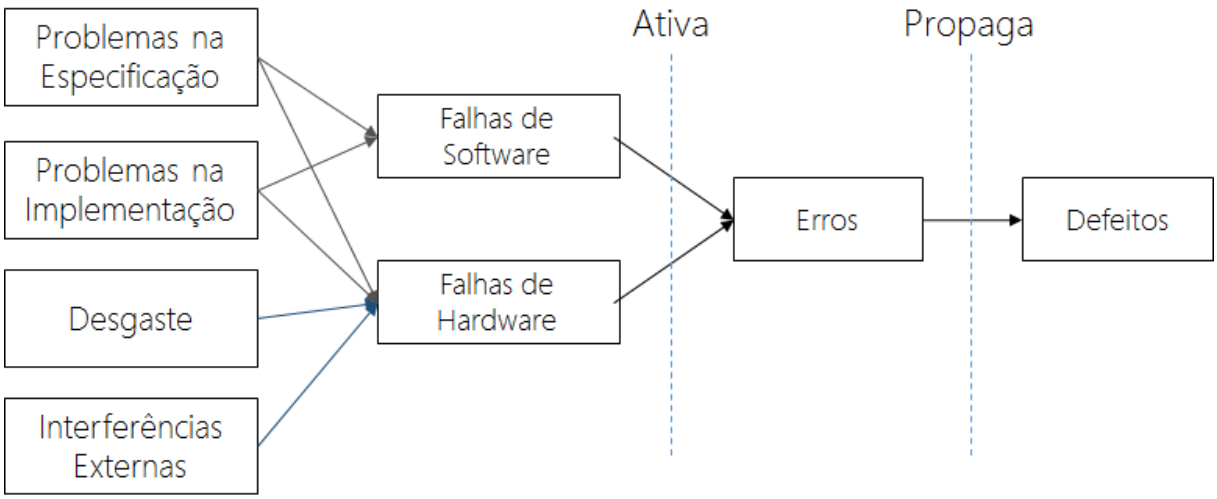

Figura 3.3: Relação de Causa-Efeito entre Falhas, Erros e Defeitos. Adaptação e Tradução de Johnson [20].

período de tempo em que o sistema foi disponibilizado de forma incorreta é chamado de interrupção de serviço.

A origem e tipo de uma falha podem ser diversos. Um curto-circuito ocorrido em uma porta lógica pode ser classificado como uma falha. A consequência pode ser a alteração do resultado da operação da porta lógica, que caracteriza erro. Este erro poderá, ou não, vir a causar um defeito nos pacotes de software que estão sendo executados no computador.

Enquanto os erros acontecem no domínio da informação do sistema, as falhas ocorrem no domínio de componentes e os defeitos no domínio externo do sistema, ou seja, são perceptíveis ao usuário. A Figura 3.3 demonstra como o fluxo entre estes três componentes apresentados.

A seguir apresentamos uma classificação mais detalhada para os conceitos de falhas e defeitos, os quais serão estudados neste trabalho.

3.2.1 Falhas (*faults*)

Avizienis et al. [1] mostra que as falhas de um sistema podem ser classificadas em oito *classes elementares* na qual cada falha pode pertencer à uma dessas classes ou à uma combinação das mesmas as quais são chamadas *classes de falhas combinadas* e que se restringem a 31 tipos de falhas, agrupadas em três grandes categorias: falhas de desenvolvimento, físicas (afetam o hardware) ou de intenção. Esta categorização está representada na Figura 3.4.

Fase de criação ou ocorrência

Aqui as falhas são classificadas de acordo com a fase do ciclo de vida na qual elas se originam e são divididas em falhas de desenvolvimento (podem ocorrer durante o período de desenvolvimento do sistema ou durante o período de uso do sistema quando são feitas manutenções) ou falhas operacionais (ocorrem na entrega do sistema).

Limites do sistema

Classificadas de acordo com a localização da falha em relação aos limites do sistema. São divididas em falhas internas (ocorrem dentro dos limites do sistema) ou falhas externas (ocorrem fora dos limites do sistema e propagam os erros no sistema).

Causas fenomenológicas

Classificadas entre falhas humanas (em relação ao uso do sistema ou ao desenvolvimento incorreto, que gerou um sistema com falhas embutidas) ou falhas físicas (ocasionadas por Hardware defeituoso, com o tempo de vida atingido ou até mesmo eventos da natureza).

As falhas humanas estão relacionadas tanto ao projeto de desenvolvimento, onde diversos fatores podem influenciar a existência de uma falha como, por exemplo, processo de desenvolvimento usado, falhas em requisitos, testes incompletos, entre outros. Já falhas físicas envolvem diversos fatores ambientais e físicos os quais o sistema está implantado. Sejam fatores relacionados a servidores ou até mesmo fatores que definem o tempo de vida útil de um componente ou tempos médios entre falhas conhecidos dos componentes eletrônicos.

Dimensão

Dividas em falhas de hardware (originadas a partir do hardware ou aquelas que afetam os mesmos) ou falhas de software (presentes na camada de informações do sistema).

Objetivo

Dividas em **falhas maliciosas**, ou seja, causada por humanos com inteções de alterar o sistema e seus dados, e **falhas não maliciosas** que ocorrem por descuidos, falta de conhecimento ou outras causas.

Intenção

Dividas em falhas deliberadas (resultado de quando humanos tomam decisões prejudiciais ao funcionamento do sistema) e falhas não deliberadas (introduzidas sem conhecimento).

Capacidade

Dividas em falhas acidentais e falhas por incompetência (resultado da falta de competência profissional do humano ou inadequação da organização desenvolvedora).

Temporal

São divididas em falhas:

- permanente descreve uma falha que é contínua e estável, em hardware falhas permanentes refletem de uma mudança física irreversível;

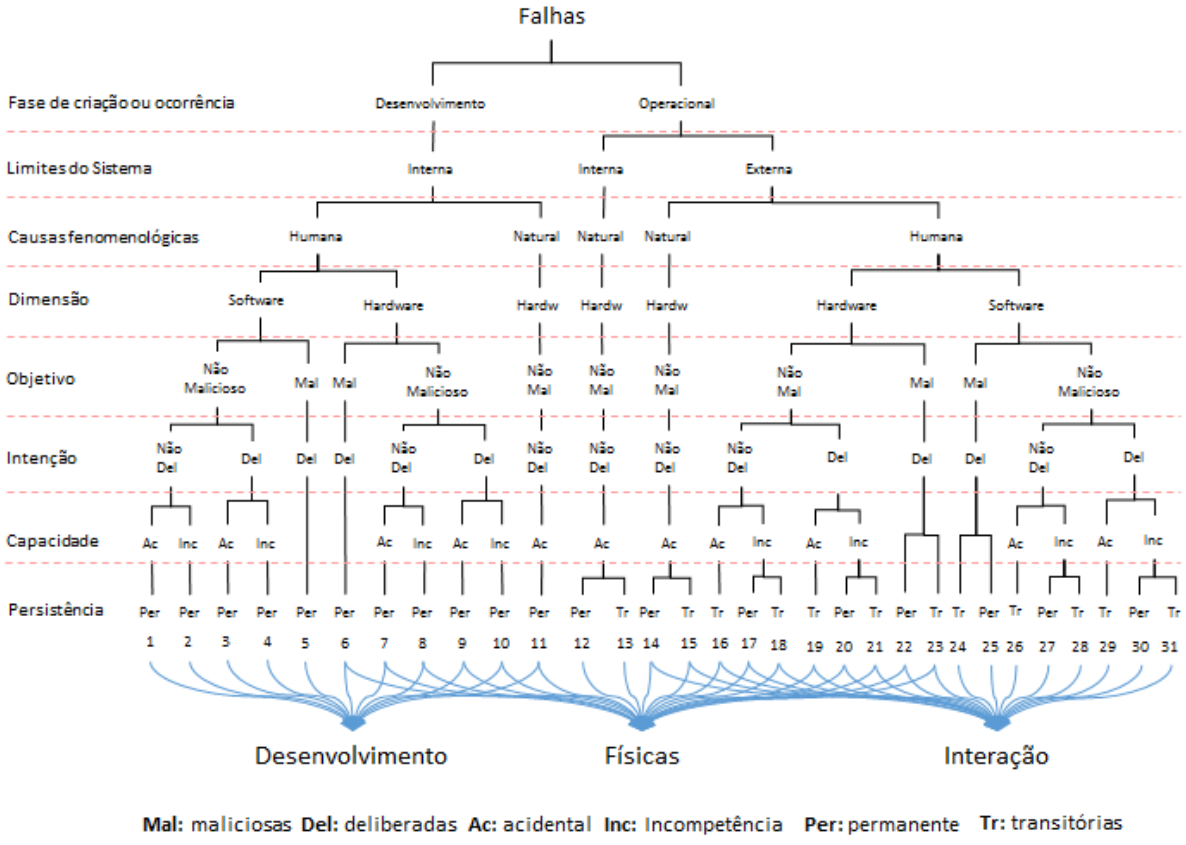

Figura 3.4: Classificação das Falhas. Tradução de Avizienis et al. [1].

- transitória, que ocorre do resultado de uma falha causada por condições temporárias do sistema. Dependendo da literatura podemos encontrar também a definição de falhas intermitentes entre as falhas transitórias, tais falhas descrevem uma falha que só é apresentada ocasionalmente, causando instabilidade devido ao hardware ou estados de software (por exemplo, em função da carga ou atividade).

É posssível então perceber o amplo escopo de falhas que podem ocorrer a um sistema.

3.2.2 Defeitos

Os defeitos são responsáveis pelos desvios na execução correta do serviço de modo que isto seja propagado para o usuário. As diferentes formas de que o defeito pode se manifestar são chamadas modos de defeitos e são classificados de acordo com quatro pontos de vista (Figura 3.6):

Domínio do defeito

Estes podem ser defeitos de conteúdo, quando o conteúdo da informação entregue a interface do serviço é diferente da especificação do sistema, ou defeitos de tempo, quando o tempo de chegada ou duração da informação entregue a interface do serviço é diferente da especificação.

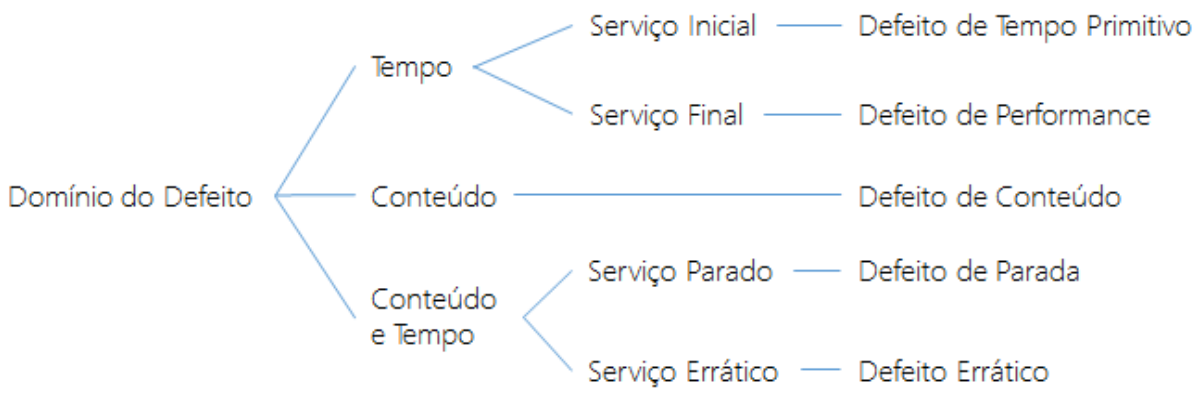

Figura 3.5: Classificação dos defeitos de domínio. Tradução de Avizienis et al. [1].

Quando possuímos defeitos de tempo e de conteúdo ocorrendo simultâneamente surgem então os defeitos de parada (as atividades do sistema não são mais perceptíveis para o usuário) e os defeitos erráticos (quando o sistema não para mas o serviço é entregue de forma errática).

A Figura 3.5 traz uma esquematização da classificação dos defeitos de domínio.

Detectabilidade do defeito

Esta categoria contempla todas aquelas que dizem respeito a sinalização da perda de funções para os usuários, sinalização que verifica a corretude do serviço prestado. Caso existam tal perdas temos como resultado então os modos reduzidos de serviço que podem variar de pequenas perdas de funcionalidade até serviços de emergência e encerramento seguro dos mesmos. Quando tais perdas são encontradas e sinalizadas então dizemos que ocorreu um defeito sinalizado, do contrário eles serão defeitos não-sinalizados.

Consistência dos defeitos

Determina a percepção dos defeitos por diferentes usuários, podendo ser divididas para dois ou mais usuários em:

- **Defeitos Consistentes** são percebidos de forma idêntica por todos os usuários do sistema.
- **Defeitos Inconsistentes** são percebidos de forma diferente por alguns ou todos os usuários do sistema.

Consequências do defeito

Os defeitos desta categoria podem ser classificados de acordo com sua gravidade e podem assumir diversos critérios diferentes para fazê-lo. No geral podemos definir dois níveis limitantes:

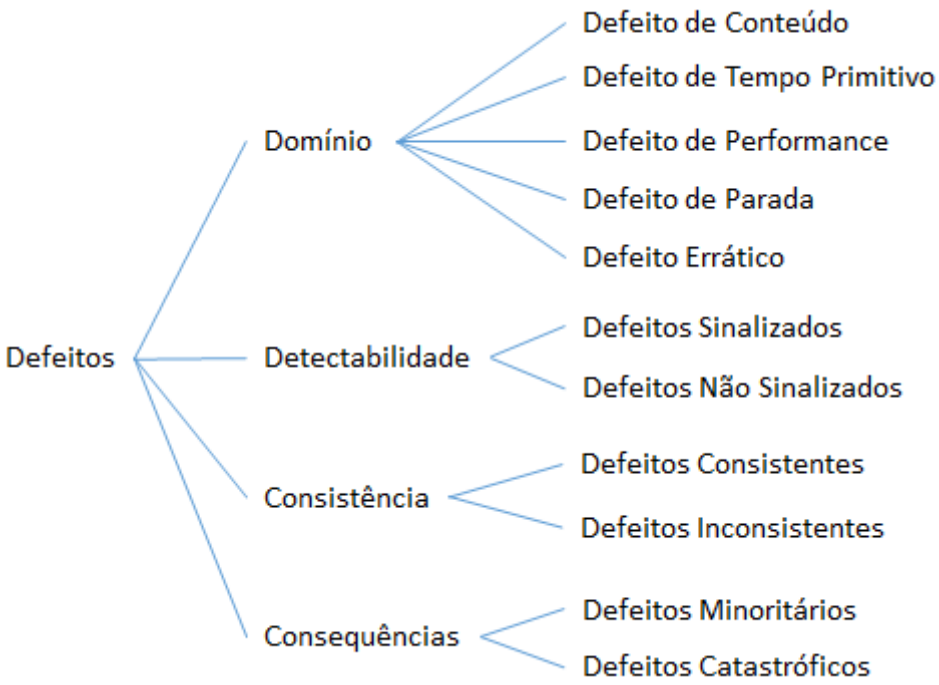

Figura 3.6: Classificação dos defeitos. Tradução de Avizienis et al. [1].

- **Falhas minoritárias**, nas quais as consequências danosas são de custo similar ao benefício pela entrega do serviço correto.
- **Falhas catastróficas**, nas quais o custo das consequências danosas é muito, ou as vezes imensuravelmente, maior do que o benefício gerado pela entrega do serviço correto.

Outros defeitos

A Figura 3.6 apresenta um resumo dos modos de defeito. No entanto existem outros defeitos que podem ser encontrados quando falamos de dependabilidade, tais defeitos não dizem respeito ao sistema em si mas sim ao processo de desenvolvimento (defeitos de desenvolvimento como defeitos no orçamento, no cronograma, na especificação e arquitetura) e a utilização do sistema (defeitos de dependabilidade, que ocorrem quando o sistema falha mais frequentemente do que é aceitável pelos usuários).

3.3 Meios

Quando um sistema pára de funcionar, normalmente aplicam-se operações de reparo para corrigir a falha. O sistema então recupera o seu estado operacional em virtude de ajustes feitos ou mesmo de substituição de componentes (Xie et al. [36]).

Os meios para se alcançar a dependabilidade são divididos em quatro categorias, sendo elas:

1. Prevenção de Falhas (*Fault Prevention*): está relacionada a técnicas de controle de qualidade aplicadas durante o desenvolvimento de software para prevenir a ocorrência ou introdução de falhas, como programação estruturada e modularização.

2. Tolerância a Falhas (*Fault Tolerance*): é definido como a capacidade de um sistema apresentar um comportamento bem definido na ocorrência de falhas. Para tal são utilizadas técnicas de modo a evitar defeitos via detecção de erros e recuperação do sistema. A detecção de erros pode ser dada por tratamento de erros que tem como objetivo a remoção de erros ou por tratamento de falhas que tem como objetivo prevenir que falhas sejam ativadas novamente.

3. Remoção de Falhas (*Fault Removal*): que pode ser feita durante o período de operação do sistema em que podemos ter a remoção corretiva, com objetivo de remover falhas que provoquem erros e a manutenção preventiva que tem como objetivo remover as falhas antes que elas provoquem erros.

4. Previsão de Falhas (*Fault Forecasting*): que se dá através de uma avaliação do comportamento do sistema em relação a ocorrência ou ativação de falhas, essa avaliação é qualitativa de modo a classificar e categorizar os modos de falhas e quantitativa que avalia o sistema de acordo com probabilidades nos quais os atributos de dependabilidade atingem níveis satisfatórios. Os dois principais meios para a previsão de falhas são, de forma complementar, a modelagem do sistema e os testes de avaliação a serem realizados a partir desta modelagem.

Capítulo 4

Estudo de Mapeamento Sistemático

A revisão sistemática da literatura, realizada através da leitura de estudos primários sobre um assunto, revisões aprofundadas e descrição de suas metodologias, é hoje parte de diversos trabalhos realizados no contexto de engenharia de software (Kitchenham et al. [24], Dyba et al. [8], Hannay et al. [15], Kampenes et al. [22]). Apesar de suas diversas vantagens apresentadas (Kitchenham et al. [24]), a realização da revisão sistemática da literatura é um processo que demanda bastante esforço e tempo para fazê-lo.

Outra forma de estudo secundário muito utilizado em áreas de medicina e que vem ganhando atenção em áreas da engenharia de software são os estudos de mapeamentos sistemáticos, uma breve pesquisa com o termo *"mapping study"* realizada na base do IEEE Xplore mostra que nos últimos dois anos (2011 e 2012) o número de estudos deste tipo foi mais que 200% maior do que a soma de todos os anos anteriores combinados.

Tais estudos sistemáticos propõem uma estrutura de modo a categorizar os tipos de pesquisa e os resultados publicados de estudos realizados sobre um tema específico e então apresentá-los, normalmente, de forma visual tornando assim mais fácil a identificação de lacunas na área do conhecimento estudada.

Antigamente estudos de mapeamento sistemáticos sobre engenharia de software eram recomendados, em sua maioria, para áreas de pesquisa nas quais faltam estudos primários relevantes e de alta qualidade (Kitchenham e Charters 2007 [24]), no entanto Petersen et al. [44] propoem uma utilização mais ampla para tal metodologia.

Neste capítulo apresentamos a teoria para realização de estudos de mapeamento sistemático (*systematic mapping studies*) e como o nosso mapeamento foi feito. As etapas realizadas foram baseadas nos estudos de Petersen et al. [44] e Kitchenham et al. [24], e uma esquematização da adaptação por nós realizada de tais abordagens encontra-se na Figura 4.1.

4.1 Necessidade do Estudo

A necessidade da realização do estudo é descrita na motivação apresentada no primeiro capítulo deste trabalho. Para verificar se trabalhos similares envolvendo os temas a serem estudados já haviam sido realizados nós fizemos uma busca manual nas bases de artigos *IEEExplore Digital Library*, *SpringerLink* e *ScienceDirect* utilizando uma *string* de busca na qual relacionamos as principais metodologias ágeis encontradas na literatura com os conceitos de dependabilidade e confiabilidade, os quais serão nossos temas de es-

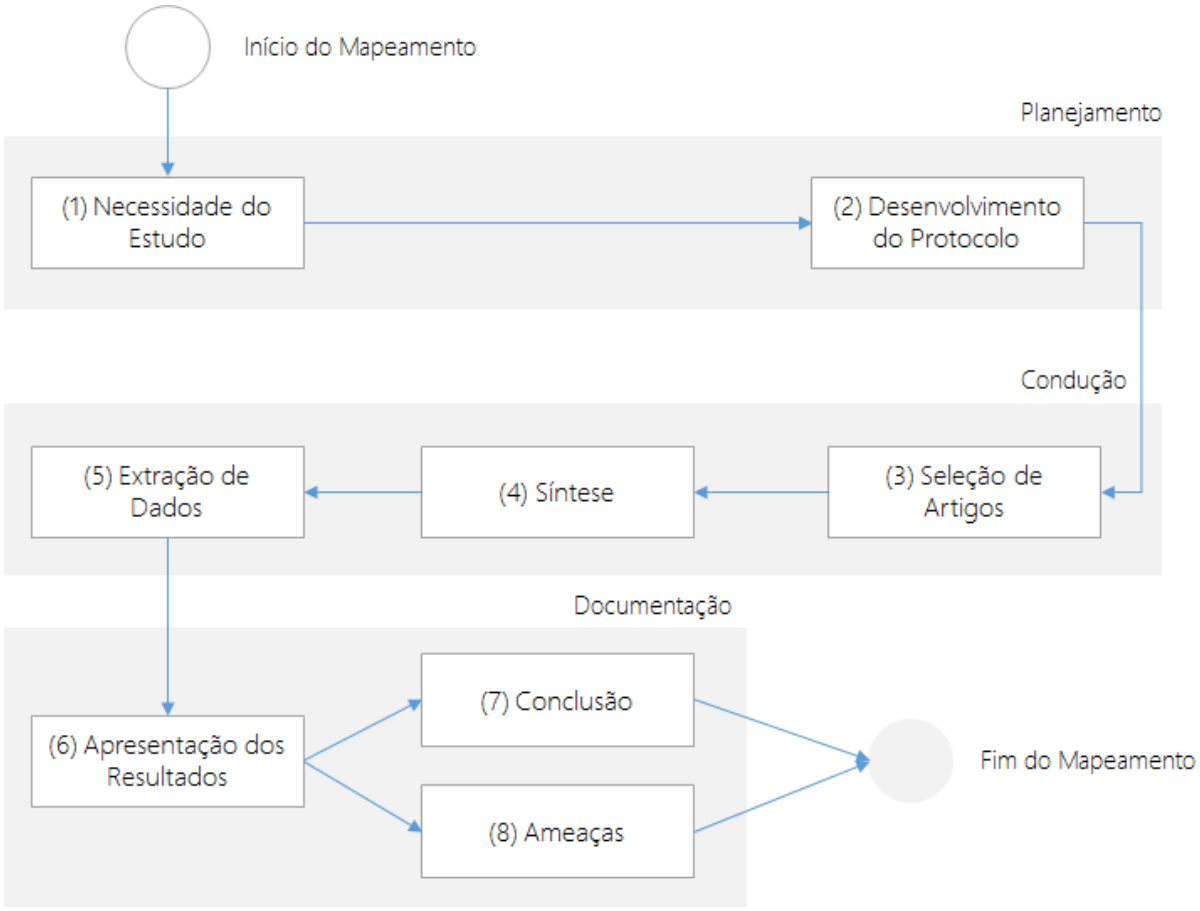

Figura 4.1: Processo para realização de Estudos de Mapeamento Sistemático. Adaptação de Petersen et al. [44] e e Kitchenham et al. [24].

Tabela 4.1: *String* de Busca da Pesquisa de Necessidade

(("systematic review"OR "mapping study"OR "research review"OR "research synthesis"OR "research integration"OR "systematic overview") AND (("Agile"OR "extreme programming"OR "scrum"OR "lean") AND ("Dependability"OR "Reliability")))

Tabela 4.2: Quantidade de resultados retornados em cada etapa da pesquisa de necessidade.

	IEEExplore	SpringerLink	ScienceDirect
String de busca	193	82	174
Títulos + Abstracts	15	6	8
Introdução + Conclusão	9	1	6

tudo. Também foram utilizados os sinônimos para revisão sistemática (*systematic review*) definidos por Biolchini et al. [6]. A *string* de busca a qual chegamos para tal verificação é descrita na tabela 4.1.

A partir desta busca inicial um total de 449 trabalhos foram encontrados já excluindo aqueles que não se encontravam no domínio da ciência da computação ou não eram escritos em inglês ou português. Uma breve leitura dos títulos e abstracts permitiu a exclusão de mais 354 artigos, e a leitura da introdução e conclusão dos restantes nos permitiu averiguar a inexistência de estudos de mapeamento sistemáticos que abordassem os dois temas simultâneamente da maneira que desejamos. Foram encontrados dois trabalhos que se assemelhavam de certo modo a nossos objetivos. A Tabela 4.2 mostra a quantidade de resultados retornados em cada etapa de nossa pesquisa e a Tabela 4.3 nos mostra como foi feita a classificação dos estudos finais (lista completa disponível no Anexo 5) entre: métodos ágeis, dependabilidade e métodos ágeis e dependabilidade.

Discutiremos aqui apenas os estudos classificados como "Dependabilidade e Métodos Ágeis", ressaltando seus objetivos, resultados e porquê se diferenciam do nosso estudo:

- A primeira revisão existente na literatura sobre dependabilidade e métodos ágeis foi feita por Mitchel e Seaman [37] em 2009. O trabalho discute e faz uma compação sobre o custo, duração e qualidade de um produto desenvolvido utilizando o modelo cascata contra o desenvolvimento iterativo e incremental. Como resultados do trabalho eles afirmam que a pequena quantidade de estudos encontrados não poderia servir de base para a se tirar uma conclusão de qual método seria melhor.
- A revisão feita por Sfetsos e Stamelos [49] em 2010 tem como foco a qualidade nos métodos ágeis, no entanto não especifica a mesma em conceitos de dependabilidade. Os resultados do trabalho realizado foi então divido em três categorias, sendo elas:

Tabela 4.3: Classificação dos resultados da pesquisa de necessidade.

	Artigos
Métodos Ágeis	SRW6, SRW7, SRW8, SRW12, SRW13, SRW14, SRW15
Dependabilidade	SRW2, SRW3, SRW4, SRW5, SRW10, SRW11, SRW16
Dependabilidade e Métodos Ágeis	SRW1, SRW9

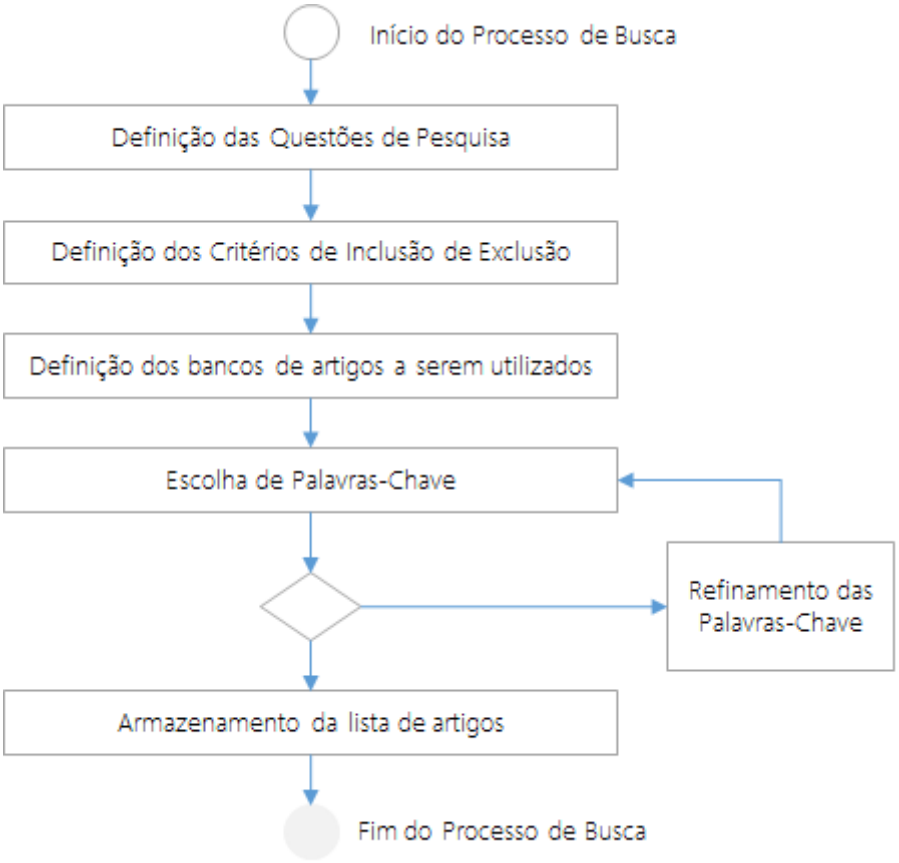

Figura 4.2: Criação do Protocolo.

> *test driven* ou *test first development*, *pair programming*, e práticas e métodos ágeis em geral. Eles afirmam que enquanto a prática de desenvolvimento orientado a testes levou a um aumento da qualidade do software, este sendo medido através de proporções de falhas, produzido na indústria, no entanto os resultados não foram os mesmos na academia. Também afirmam que a prática da programação em pares aumenta a qualidade do código e a arquitetura do software.

Nosso trabalho se diferenciará dos apresentados de modo que enquanto o foco do primeiro era realizar uma comparação entre métodos e do segundo era medir a qualidade sem especificar termos em geral, o nosso focará apenas nos aspectos de dependabilidade do software e as fases em que estes se encaixam no ciclo de desenvolvimento.

4.2 Desenvolvimento do Protocolo

O desenvolvimento de um protocolo de revisão faz com que exista uma padronização da pesquisa entre os diversos pesquisadores e também permite a replicação da mesma em experiências futuras. Uma esquematização do procotocolo por nós desenvolvido é apresentado na Figura 4.2.

4.2.1 Questões de Pesquisa

Após termos o tema definido e a necessidade de realização da pesquisa comprovada devemos, como primeiro passo do processo para realização do estudo de mapeamento sistemático, definir quais são as questões de pesquisa que servirão como base do processo. Estas questões definirão o escopo do processo, servindo para todas as outras etapas do mesmo.

Para a criação de tais questões deve se levar em conta que o intuito de um estudo de mapeamento é propiciar uma ampla visão sobre uma área, classificando tópicos contidos nela e quantificando-os em categorias específicas, além de trazer uma análise posterior sobre como, no geral, o tema abordado é trazido em tais estudos e quais são os benefícios mostrados nos estudos identificados. Nossas questões de pesquisa foram criadas de acordo com a motivação e objetivos do estudo apresentados no Capítulo 1. Foram então criadas quatros questões de pesquisa para suprir nossos objetivos. Seus enunciados e motivação encontram-se descritos a seguir:

- **QP1.** Em quais estágios do ciclo de desenvolvimento ágil de um software as falhas e defeitos estão sendo tratados?

  Esta é a questão principal do estudo, com ela buscamos identificar se as falhas e defeitos estão sendo tratados no início, meio ou fim do ciclo de desenvolvimento quando são utilizadas metodologias ágeis como Lean, XP e Scrum. Deste modo torna-se possível avaliar a extensão dos problemas encontrados ao deixar se negli-cenciar o tramento da dependabilidade em uma etapa. A definição de início, meio ou fim se mostra um pouco mais complicada do que o usual uma vez que os métodos ágeis utilizam uma abordagem incremental.

- **QP2.** Quais são as principais práticas ágeis que garantem a entrega de um produto confiável?

  Esta pergunta serve para focar a análise dos resultados, apresentando não apenas quais são os estudos da área mas também identificando, através de seus conteúdos, quais foram as práticas ágeis mais relevantes considerando o impacto causado na dependabilidade do software.

- **QP3.** Quais são pesquisadores e centros de pesquisa de maior relevância quando falamos sobre dependabilidade e métodos ágeis?

  Procuramos com esta pergunta identificar os autores e universidades destaques nas pesquisas relacionadas ao assunto de nosso trabalho.

- **QP4.** Quais das práticas apresentadas são maduras (foram amplamente testadas) e tem sido adotadas pela indústria do software?

  Com esta última questão buscamos identificar se as práticas apresentadas como favoráveis ou prejudiciais a dependabilidade do software tem alcançado um nível de maturidade por meio de avaliações em casos reais.

4.2.2 Critérios de Inclusão e Exclusão

Ao executar a string de busca nas bases de dados selecionados são retornados diversos resultados que não farão parte da pesquisa, isso porque podem ter sido utilizadas

palavras-chave que fazem parte do título ou até mesmo abstract dos artigos mas que não representam o domínio do nosso estudo. É necessária então a definição de critérios de inclusão e exclusão para que na etapa de seleção de artigos seja possível a realização de uma filtragem de resultados de forma justificada e replicável.

Os critérios por nós selecionados encontram-se listados abaixo:

- **Critérios de Inclusão:** Trabalhos (capítulos de livros e artigos) que possuam os temas dependabilidade e métodos ágeis como foco escritos até julho de 2013. Caso vários artigos se refiram ao mesmo estudo, o trabalho mais completo será considerado.
- **Critérios de Exclusão:** *Keynotes*, *White Papers*, trabalhos apresentados apenas na forma de abstract ou apresentação, trabalhos em outras línguas que não inglês e português, trabalhos cujo domínio não seja o da ciência da computação, trabalhos em que o tema principal não é dependabilidade e métodos ágeis e aqueles não disponíveis ao nosso acesso através do login institucional propiciado pela rede da Universidade de Brasília. Como nosso foco está no estudo de metodologias ágeis como um todo, os estudos que focavam em apenas uma técnica ou prática, como TDD ou programação pareada foram excluídos.

Destes critérios devemos explicar que a escolha de excluir trabalhos que avaliem uma prática ágil específica foi realizada pois deseja-se avaliar as metodologias ágeis em geral. Alguns trabalhos avaliando práticas como o desenvolvimento orientado a testes e a programação pareada foram retornados na busca inicial, no entanto é de nosso entendimento que tais estudos não avaliam o método ágil em si e sim uma prática específica que pode ser utilizada mesmo quando não se realiza um projeto ágil.

A partir destes critérios foi possível gerar uma lista para validar se um trabalho era válido para nosso estudo ou não:

- O trabalho foi escrito inglês ou português? (Sim/Não)
- O trabalho está disponível na íntegra? (Sim/Não)
- O contexto do trabalho é referente a métodos ágeis? (Sim/Não)
- O tema dependabilidade é abordado no trabalho? (Sim/Não)

Utilizamos também um método para identificar a relevância dos artigos encontrados. Este processo nos permite avaliar, após a leitura completa do artigo, se seus resultados são realmente influentes em nossa pesquisa. Elaboramos quatro questões que devem ser respondidas positivamente para que o artigo seja considerado relevante à nossa pesquisa:

1. Os objetivos da pesquisa estão claros?
2. A pesquisa foi realizada em um contexto apropriado para a observação dos resultados?
3. Responde em qual etapa do processo de desenvolvimento de software ocorrem falhas e defeitos?
4. Apresenta o impacto das práticas ágeis utilizadas na ocorrência de falhas e defeitos?

Tabela 4.4: Termos de pesquisa.

Fonte	**Termos**
Tema do Estudo	*dependable*, *dependability*, *reliability*, *reliable*
Questão 1	"*agile method*", "*agile methods*", "*agile methodology*", "*agile methodologies*", "*agile development*", *scrum*, xp, "*extreme programming*", "*Lean Development*", *failure*, *fault*
Questão 2	*correctness*, "*software quality*", *principles*, *values*
Questão 3	"*research groups*"
Questão 4	-

A partir destas questões conseguimos observar se os resultados apresentados estão de acordo com os objetivos da pesquisa e se esta foi realizada em um contexto real de desenvolvimento de software, com programadores capacitados. Além disso verificamos em quais etapas do desenvolvimento foram encontrados as falhas e defeitos bem como quais práticas ágeis impactaram sua taxa de ocorrência. Por fim, temos a análise dos resultados da utilização dos métodos ou práticas ágeis.

4.2.3 Bancos de Artigos

Para a realização da pesquisa eletrônica foram selecionadas quatro bases de dados, levando em conta a permissão de acesso atribuída ao perfil institucional da universidade e relevância da base nos campos da ciência da computação e engenharia de software.

- IEEE Explore
- ACM Digital
- Springer Link
- Science Direct

4.2.4 Definição da string de busca

O segundo passo consiste na busca por estudos primários, esta pode ser feita de forma manual pesquisando por conteúdo relevante em diversas publicações ou de forma automatizada através da utilização de *strings* de pesquisa em bancos de dados de artigos científicos.

A criação de tais *strings* de pesquisa é feita levando em conta o tema do estudo e as questões propostas anteriormente. Para uma construção correta das mesmas é necessária a utilização de conectivos lógicos "AND"e "OR"de modo a mostrar um relacionamento entre os termos. Vale ainda lembrar que deve-se traduzir os termos para a língua inglesa, visto que esta é a língua utilizada por todos os bancos de artigos em que pesquisamos.

A seguir apresentamos a construção de nossa *string* de pesquisa a partir dos passos mencionados:

A string resultante em nosso estudo foi montada da seguinte forma:

1. (dependable OR dependability OR reliability)

Tabela 4.5: *String* Final de Busca da Pesquisa de Eletrônica

(dependable OR dependability OR reliability) AND (("agile methods"OR "agile methodology"OR "agile method"OR "agile methodologies"OR scrum OR "extreme programming"OR "agile development"OR "lean development"OR "agile principles") AND (correctness OR failure OR fault))

2. ("agile methods"OR "agile methodology"OR "agile method"OR "agile methodologies"OR scrum OR xp OR "extreme programming"OR "agile development"OR "lean development"OR "agile principles")
3. (correctness OR failure OR failures OR faults OR fault OR validation OR "software quality"OR reliable)

(1) AND (2) AND (3)

Optamos por não incluir os termos de 3 pois são muito amplos e serão informações que obteremos derivadas do estudo dos artigos retornados através dos termos das perguntas 1 e 2. Os termos *values* e *principles* foram omitidos devido ao grande número de resultados errôneios, e o termo *reliable* foi removido pois sua utilização principal se dava em frases como dados, resultados confiáveis de experimentos (*reliable data*). Chegando dessa forma a string indicada na tabela 4.5.

Tendo a string de pesquisa (4.5) em mãos é possível fazer a pesquisa manual nos bancos de dados ou utilizar alguma ferramenta que permita essa busca de forma automatizada (utilizando *crawlers*) em diversos bancos diferentes. Optamos pela realização manual da pesquisa eletrônica visto que uma ferramenta para realização automática da mesma ainda está em fase de desenvolvimento na Universidade de Brasília e portanto não se adaptava a todas as nossas necessidades.

4.3 Seleção de artigos

Após a realização da busca inicial nas quatro bases de dados com os termos selecionados no protocolo obtivemos um número total de 1795 trabalhos encontrados, já excluindo aqueles artigos que não fossem escritos em inglês ou português e que não fizessem parte do domínio da ciência da computação. Foi então necessário importar todas as citações e adicionar os valores de origem da citação, status da avaliação, elegibilidade para cada um destes trabalhos em uma planilha do Excel, sendo que em cada nova etapa do processo uma nova planilha era utilizada.

A filtragem dos resultados ocorreu em três etapas. Na primeira dividimos os trabalhos entre os dois avaliadores, removemos os trabalhos duplicados, e então foram analisados apenas os títulos e, caso necessário, os abstracts dos trabalhos retornados pela *string* de busca. A segunda etapa foi a análise dos textos de introdução e conclusão dos trabalhos selecionados na etapa anterior, aqui os dois avaliadores avaliaram todos os trabalhos até que chegassem em uma conclusão única de quais trabalhos seriam selecionados, para isso foram necessárias três rodadas de classificação até que um consenso fosse encontrado. Nesta etapa também foram excluídos artigos os quais não se possuía acesso através da rede da Universidade de Brasília (aproximadamente 19% dos trabalhos selecionados). Na

Tabela 4.6: Quantidade de resultados retornados em cada etapa da busca eletrônica.

Etapa	IEEExplore	SpringerLink	ScienceDirect	ACM DL
String de busca	721	404	392	278
Títulos + Abstracts	104	69	75	59
Introdução + Conclusão	10	3	2	1
Avaliação de Qualidade + Leitura Completa	5	1	2	1

útlima etapa unimos os artigos então encontrados com aqueles retornados pela pesquisa manual (ver seção 4.4.1) e realizamos a verificação da qualidade de tais artigos (ver seção 4.5), além da leitura completa dos mesmos para a realização da classificação dos mesmos (ver seção 4.6), neste momento ainda excluímos 13 trabalhos do nosso estudo, ficando com um total de 18 estudos para utilizar em nossa pesquisa.

4.3.1 Pesquisa Manual

Para tornar mais precisa nossa análise, realizamos pesquisas manuais nas principais conferências, workshops e periódicos dos temas confiabilidade (*reliability*) e metodologias ágeis, além da utilização da técnica de *snowballing* para que nossa pesquisa ficasse mais completa.

Conferências, Workshops e Periódicos

Para a busca em eventos relativos ao tema de dependabilidade foram pesquisados os termos: *agile*, *lean*, *extreme programming* e *scrum*. Já nos eventos sobre métodos ágeis utilizamos os termos: *reliability*, *dependability* na pesquisa. Os eventos selecionados para tal foram:

1. Agile Development Conference - 3 resultados
2. IEEE International Conference on Software Reliability Engineering Workshops (ISSRE Wksp) - 3 resultados
3. International Conference on Availability, Reliability and Security (ARES) - 3 resultados
4. International Symposium on Software Reliability Engineering (ISSRE) - 4 resultados
5. Outros eventos e workshops sobre dependabilidade que não retornaram resultados: Annual Symposium Reliability and Maintainability (RAMS), IEEE International Conference on Quality and Reliability, IEEE International Workshop Integrated Reliability, International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), International Conference on Reliability, Maintainability and Safety (ICRMS), International Conference on Secure System Integration and Reliability Improvement (SSIRI), European Dependable Computing Conference (EDCC), Latin-American Symposium on Dependable Computing (LADC), Pacific Rim International Symposium on Dependable Computing, International Conference on Dependable Systems and Networks (DSN), International Conference

on Dependable Systems and Networks Workshops (DSN-W), IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC), International Conference on Secure System Integration and Reliability Improvement (SSIRI, SERE).

6. Conferências que abordam temas em geral sobre engenharia de software e que são reconhecidas: ICSE (International Conference on Software Engineering), FSE (Foundations of Software Engineering), ASE (IEEE/ACM International Conference on Automated Software Engineering), SPLASH (Systems, Programming, Languages and Applications: Software for Humanity), ECOOP (European Conference on Object-Oriented Programming), ISSTA (International Symposium on Software Testing and Analysis) e FASE (Fundamental Approaches to Software Engineering).

7. Periódicos selecionados para nossa pesquisa: IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering, Methodology, IEEE Transactions on Software Engineering, IEEE Software, Wiley Software Testing, Verification and Reliability, Springer Empirical Software Engineering, Springer Software and Systems Modeling, Wiley Journal of Software Maintenance and Evolution, IEEE Transactions on Reliability, Journal of Systems and Software, Software Practice and Experience, Information and Software Technology.

Dos treze trabalhos encontrados, lembrando que aqueles trabalhos já encontrados na busca eletrônica foram retirados, apenas um (Jonsson et al. [21]) foi considerado válido para nossa pesquisa. (Anexo B)

4.3.2 *Snowballing*

A técnica de *snowballing* é utilizada para encontrar a população oculta de trabalhos, ou seja, aqueles trabalhos os quais não foram retornados pelo nosso processo de busca mas que podem ser interessantes para a nossa pesquisa. Segundo Jalai e Wohlin [19] o snowballing, quando comparada com as buscas automatizadas em bases de dados realizados em pesquisas sistemáticas da literatura, mostra-se menos custoso e traz uma menor quantidade de "ruído"(artigos que não serão selecionados). Esta técnica possui duas abordagens distintas quanto a sua realização:

1. *Forward Snowballing* busca trabalhos que utilizem a nossa lista inicial de estudos como refência.

2. *Backward Snowballing* que busca novos trabalhos nas referências da nossa lista.

Optamos pela utilização apenas do backward snowballing, que já nos retornou mais que seiscentos trabalhos para ser filtrados. Após uma filtragem, com etapas semelhantes as etapas apresentadas na seleção dos trabalhos retornados na busca automatizada, ficamos com apenas seis novos artigos selecionados. A Figura 4.3 mostra as etapas e quantidades de trabalhos retornados em todo o processo de busca.

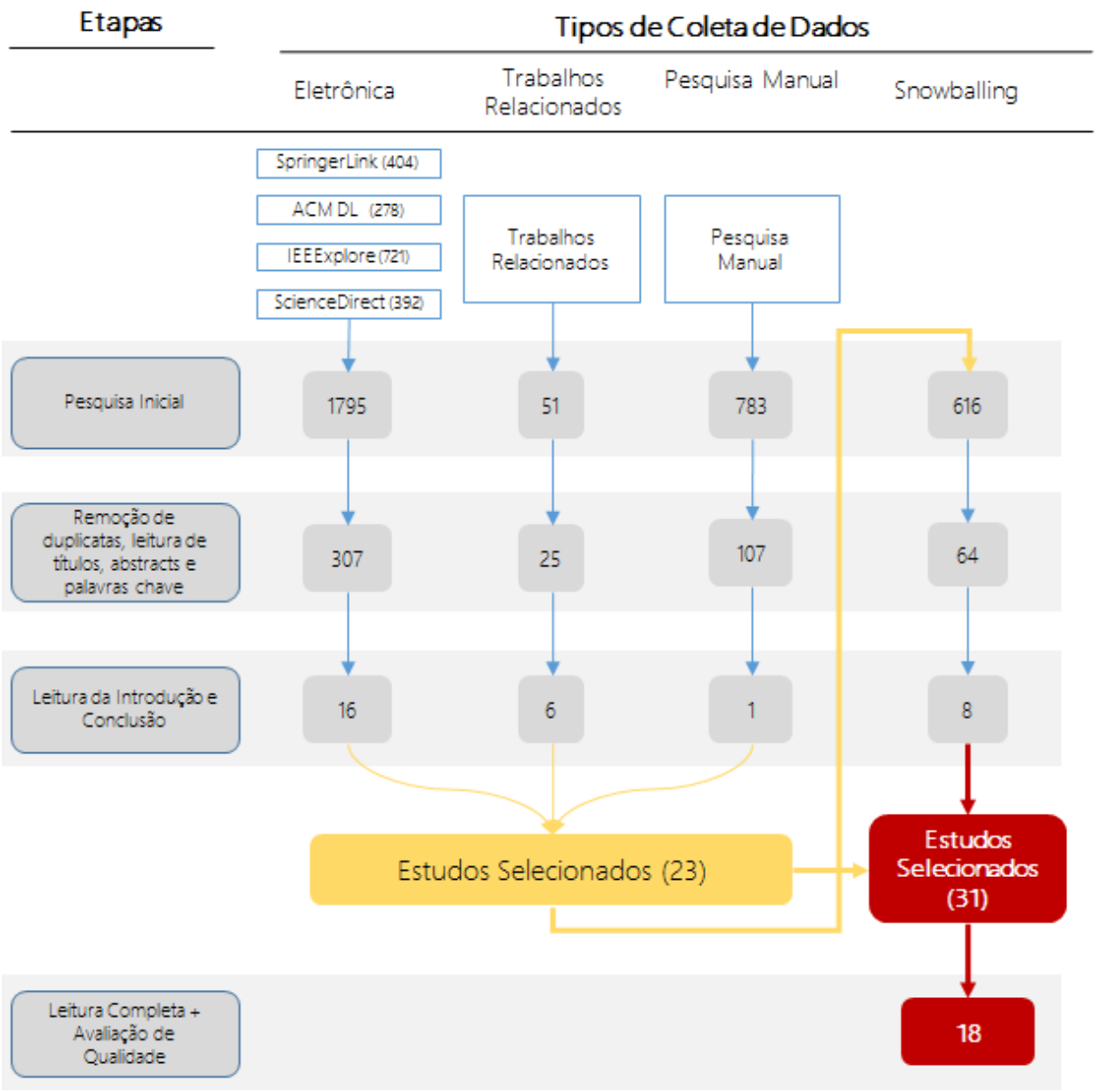

Figura 4.3: Etapas do processo de seleção e extração de dados

Tabela 4.7: Faceta de Pesquisa. Traduzido de Wieringa [56]

Tipo	Descrição
Pesquisa de Validação	As técnicas investigadas são novas e ainda não foram implementadas. As técnicas usadas são, por exemplo, experimentos realizados em laboratório.
Pesquisa de Avaliação	Técnicas são implementadas na prática e uma avaliação da técnica é conduzida. Ou seja, é mostrado como a técnica é implementada na prática (solução de implementação) e quais são as consequências da implementação em termos de benefícios e perdas (avaliação da implementação). Isso também inclui identificar problemas na indústria.
Proposta de Solução	A solução para o problema é proposta, a solução pode ser nova ou uma extensão significativa de uma técnica existente. Os potenciais benefícios e a aplicabilidade da solução são mostrados por pequenos exemplos ou uma boa linha de argumentação.
Artigos Filosóficos	Esses artigos esboçam uma nova forma de visualizar coisas existentes pela estruturação do campo de pesquisa em termos de taxonomia ou framework conceitual.
Artigos de Opinião	Esses artigos expressam uma opinião pessoal de alguém sobre alguma técnica afirmando se ela é boa ou ruim, ou como as coisas deveriam ser feitas. Eles não dependem de trabalhos relacionados ou metodologias de pesquisa.
Artigos de Experiência	Artigos de experiência explicam o que e como algo vem sendo feito na prática. Deve ser uma experiência pessoal do autor.

4.4 Classificação

O modelo de classificação por nós utilizado adotou a sugestão encontrada em Petersen et al. [44] na qual a classificação é estruturada em facetas. Foram adotadas então três facetas, sendo elas: a faceta de pesquisa, a faceta de contribuição e a faceta de contexto.

A faceta de contexto diz respeito ao tema estudado (dependabilidade) e portanto foram considerados artigos que tratavam sobre defeitos e falhas como explicados no Capítulo 3. Já na faceta de contribuição foram consideradas as melhores práticas presentes nos métodos ágeis como *client on-site*, *refactoring*, *test driven development*, entre outros descritos no Capítulo 2. Para a terceira faceta, a de tipos de pesquisa, utilizamos a definição dada em Wieringa [56] mostrada na Tabela 4.7.

Como Petersen et al. [44] explica, a utilização desta classificação quanto a tipos de pesquisa é feita facilmente sem a necessidade de avaliar cada trabalho em detalhe como é feito em revisões sistemáticas. Para padronizar as informações coletadas utilizamos uma planilha do Excel com 32 campos que envolviam informações gerais e específicas.

	Descrição do Estudo
1.	Identificador
2.	Autor(es)
3.	Ano de Publicação
4.	Título
5.	Local de Publicação
6.	País do Pesquisador
7.	Tipo Instituição (Mercado/Academia/Ambos)
8.	Tipo de Publicação (artigo em revista, artigo de conferência, artigo de workshop,...)
9.	Tipo de Artigo (Pesquisa de Validação, Pesquisa de Avaliação, Proposta de Solução, Artigo Filosófico, Artigo de Opinião, Artigo de Experiência)
10.	Objetivos do estudo (descrição)
11.	Tipo de Estudo (um estudo de caso, multiplos estudos de caso, experimento, pesquisa, mistura)
12.	Tamanho da amostra
13.	Tipo da amostra (estudantes, profissionais)
14.	Experiência da amostra (iniciante/intermediário/avançado)
	Dados analisados
15.	Métricas Utilizadas
16.	Fala de Faults (S/N)
17.	Como?
18.	Fala de Failure (S/N)
19.	Como?
20.	Trata de Defects (S/N)
21.	Como?
22.	Método Ágil Analisado
23.	Pratica ágil analisada (Whole Team, Continuous integration, TDD/Unit Tests, Simple design, Pair programming, Refactoring, Coding standards, Acceptance tests)
24.	Etapa(s) de desenvolvimento analisada(s)
	Qualidade
25.	Os objetivos da pesquisa estão claros?
26.	A pesquisa foi realizada em um contexto apropriado para a observação dos resultados?
27.	Responde em qual etapa do processo de desenvolvimento de software ocorre fault, failure ou defect?
28.	Apresenta o impacto das práticas ágeis utilizadas na ocorrência de fault, failure ou defect?
	Resultados do Estudo
29.	Dados Encontrados
30.	Conclusões
31.	Limitações/Ameaças a validade

Figura 4.4: Planilha de avaliação de artigos

Capítulo 5

Resultados Obtidos e Análise

Dividimos este capítulo da seguinte forma: na primeira seção daremos uma visão geral dos estudos encontrados; na segunda seção daremos uma breve descrição de cada um dos trabalhos retornados e como se relacionam com o nosso tema; por último responderemos as questões de pesquisa do nosso mapeamento sistemático propostas no Capítulo 4.

5.1 Síntese dos Resultados

5.1.1 Dados Gerais

Dos estudos selecionados nós percebemos, a partir da Tabela 5.1, que os estudos encontrados foram bem distribuídos sendo que aqueles apresentando práticas ágeis em geral são 44% do total, aqueles realizados com Scrum são 27% e XP tem 27% também.

A Tabela 5.2 mostra a distribuição dos resultados por veículo publicado, foram 11 trabalhos publicados em conferências (61%) e 7 publicados em periódicos (39%).

Quanto à distribuição nos anos, todos os estudos retornados foram publicados na última década como esperado, uma vez que as metodologias ágeis são bastante novas e o manifesto ágil foi publicado apenas em 2001. Percebe-se um pequeno aumento no número de estudos sobre o assunto nos últimos anos, no entanto não podemos afirmar que seja uma tendência devido a baixa quantidade de resultados.

5.1.2 Metodologias de Pesquisa e Validade

Apresentamos aqui as metodologias utilizadas por cada um dos estudos e as respectivas classificações de pesquisa de acordo com o tipo do artigo. No Capítulo 4 propusemos a

Tabela 5.1: Tipo de metodologia ágil utilizada nos estudos

Metodologia	Quantidade	Porcentagem
Geral*	8	44,4%
Scrum	5	27,7%
XP	5	27,7%
Total	18	100%

* "Geral"se refere a estudos que citam um método ágil sem especificar qual.

Tabela 5.2: Periódicos e Conferências em que os resultados foram publicados.

Veículo	Tipo	Quantidade
ESEM	Conferência	2
ICSE	Conferência	1
FIT	Conferência	1
ICCSA	Conferência	1
Journal of Systems and Software	Periódico	1
Software Quality Journal	Periódico	1
Empirical Software Engeneering	Periódico	1
Software Quality Professional	Periódico	1
ADC	Conferência	1
IEEE Software	Periódico	1
IEEE Transactions on Software Engineering	Periódico	1
Journal of Software Maintenance and Evolution	Periódico	1
Agile Conference	Conferência	1
Euromicro Conference	Conferência	1
COMPSAC	Conferência	1
ICSEA	Conferência	1
CCECE	Conferência	1
Total		18

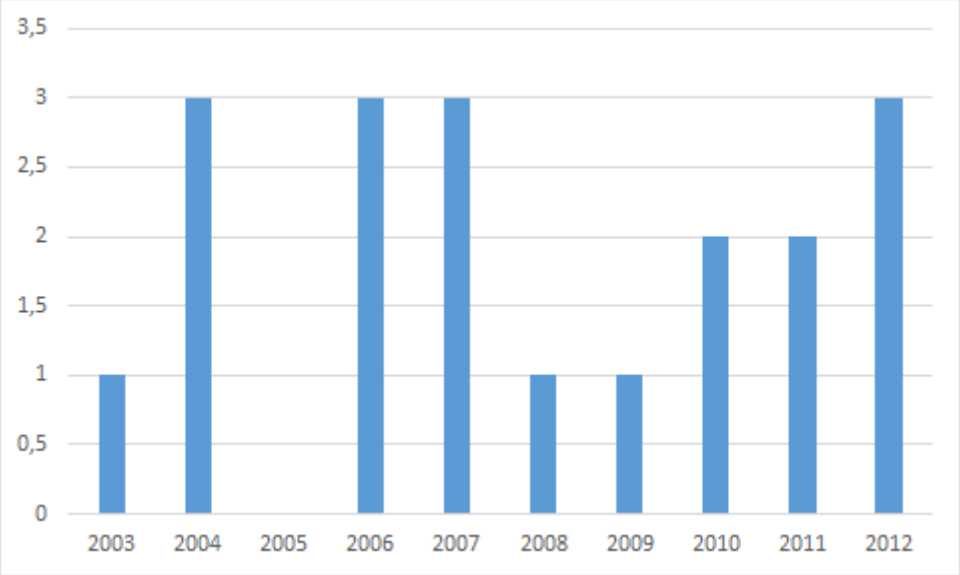

Figura 5.1: Número de resultados por ano de publicação.

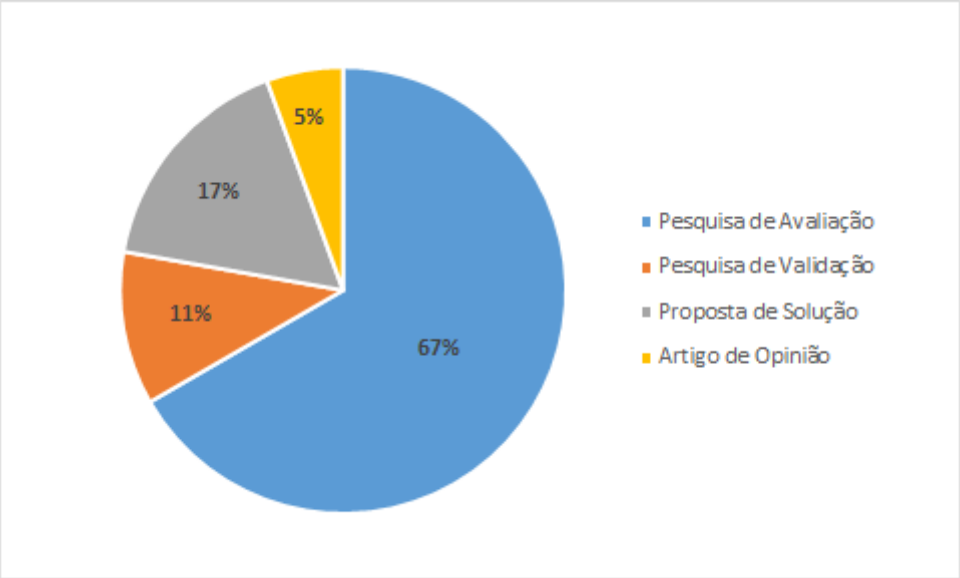

Figura 5.2: Classificação de Acordo com a Faceta de Pesquisa.

Tabela 5.3: Tipo de método de pesquisa utilizado

Metodologia	Quantidade	Porcentagem
Estudo de Caso	11	61,1%
Múltiplos Estudos de Caso	3	16,6%
Experimento	1	5,5%
Sem pesquisa	3	16,6%
Total	18	100%

utilização de uma faceta de tipo de artigos, a faceta de pesquisa, dividida em seis tipos, temos que a grande maioria dos estudos são pesquisas de avaliação (12 de 18, ou 67%), isto reflete a suposição de que muitos trabalhos sobre métodos ágeis são comprovações de práticas ou experiências sobre a adoção de tais metodologias feitas em empresas. Os outros estudos foram classificados em proposta de soluçao (3, ou 17%), pesquisa de validação (2, ou 11%) e apenas um como artigo de opinião. A Figura 5.2 reflete os resultados mencionados.

Também realizamos a classificação de acordo com os métodos de pesquisa utilizados em cada um dos estudos, os resultados encontram-se na Tabela 5.3 que nos mostra que a grande maioria dos estudos consistiam em estudos de caso em empresas de software (61% dos estudos). Outros métodos utilizados foram também os múltiplos estudos de caso e um utilizou-se da experimentação, três estudos não apresentaram pesquisa para as proposições feitas.

Além dos métodos de pesquisa analisados, verificamos também se as pesquisas eram bem estruturadas, ou seja, respondiam todos os itens de nosso questionário de classificação. Chegamos ao resultado de que apenas dois estudos respodiam a todos as nossas perguntas apresentadas no questionário de classificação e que 12 deles (66%) não apresentavam questionamentos quanto a validação de seus métodos.

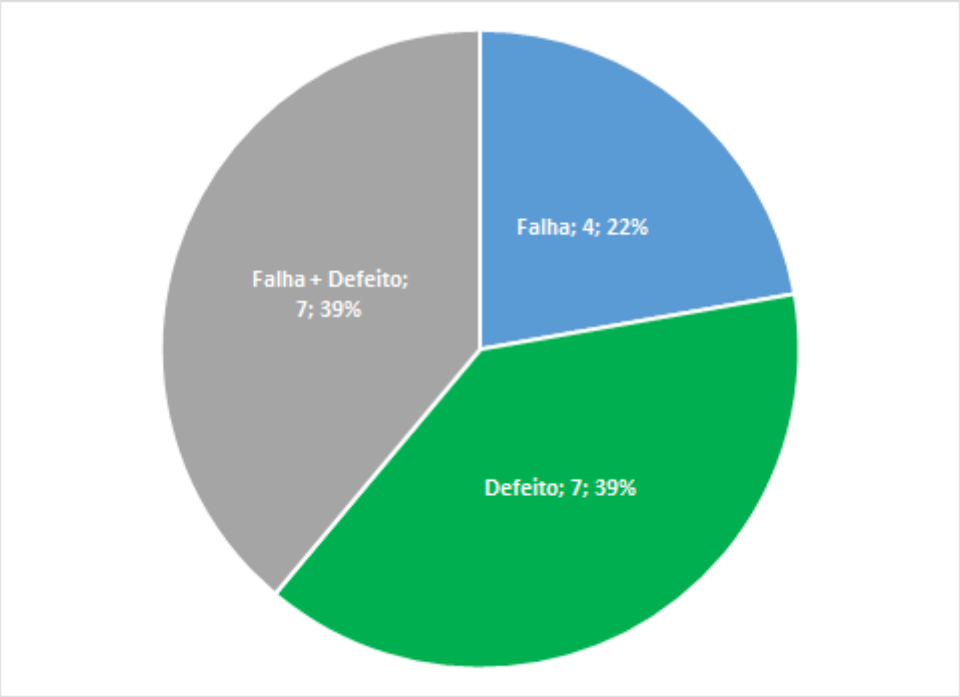

Figura 5.3: Distribuição dos termos de dependabilidade encontrados.

5.1.3 Faltas, erros e defeitos

A primeira tarefa ao analisar a dependabilidade foi identificar quais aspectos da mesma eram tratados em cada estudo, a Figura 5.3 apresenta a distribuição geral dos termos de dependabilidade encontrada. Já nesta etapa percebemos uma dificuldade do tema que foi a falta de definição generalizada para tais termos, a grande maioria dos estudos apesar de trazer tais termos não os especificava de modo que coube a nós adaptar tais termos as definições dadas nos Capítulo 3.

Após identificados quais estudos traziam cada aspecto pudemos relacioná-los com as metodologias estudadas, o resultado é apresentado na Figura 5.4. Uma análise mais aprofundada sobre as etapas em que cada metodologia analisará o tema dependabilidade e as práticas que garantem a mesma será apresentada na seção 5.3.

5.2 Resumo dos Estudos

Realizamos então a análise de todo o conteúdo de cada resultado encontrado e escolhemos quatro categorias para classificar os estudos quanto aos seus objetivos. São elas: adoção, que trata do impacto de começar a utilizar metodologias ágeis em um projeto ou empresa; comparativo, que são estudos que farão a comparação de uma metodologia tradicional com as metodologias ágeis; métricas, que trazem formas de medir faltas e falhas aplicadas a metodologias ágeis; e por último, outros, que são aqueles artigos que não se enquadram nas classificações anteriores.

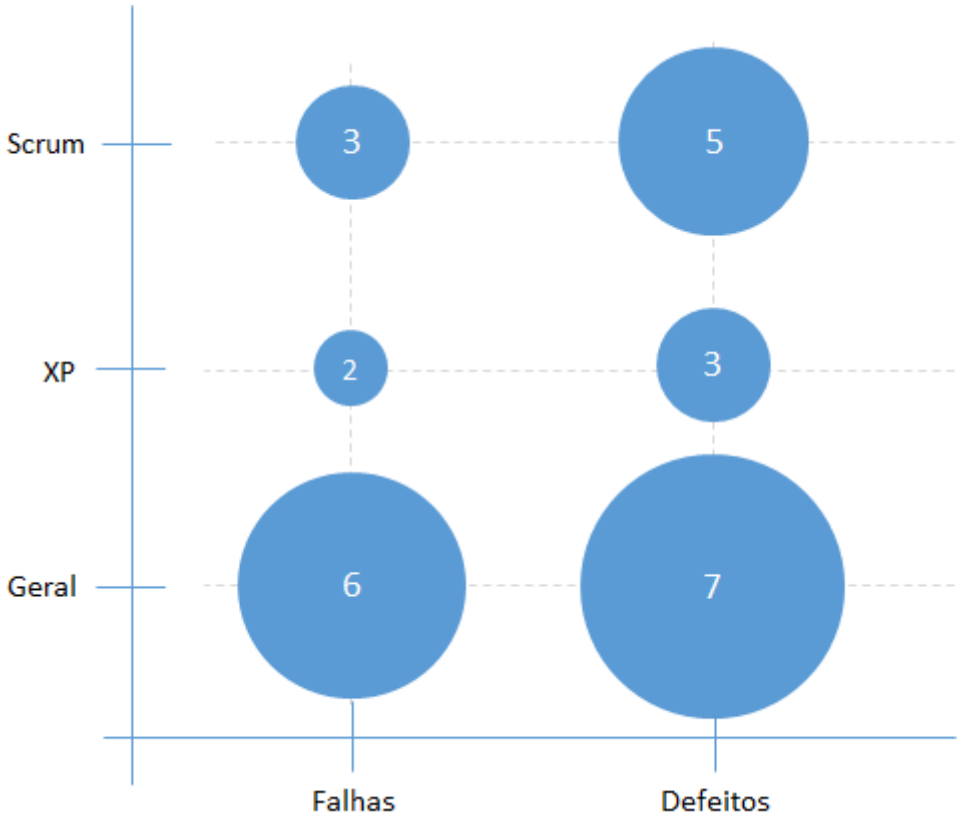

Figura 5.4: Relação entre metodologias e termos.

Adoção

Diversos trabalhos encontrados trouxeram os temas de como as metodologias ágeis são introduzidas e adotadas em empresas de diferentes portes. Um estudo de caso conduzido por Tufail e Malik [53] apresenta os resultados da utilização do Scrum no processo de desenvolvimento de software. A análise foi feita nos anos de 2008 (em que não havia processo formal de desenvolvimento) e 2010 (utilizando uma metodologia ágil). Para tal foram divididos os defeitos encontrados de acordo com a severidade em uma escala de quatro níveis sendo 4 o mais grave. Em seus resultados apresentam que a porcentagem dos mesmos diminuiu com a utilização do Scrum, nos níveis 4 (2,71% para 1,31%), 3 (28,05% para 24,5%) e 1 (44,11% para 43,7%), enquanto aumentou no nível 2 (25,33% para 30,46%). Mostram também que a taxa de testes bem sucedidos/mal sucedidos aumentou de 1,25 para 8,62.

Já os artigos de Korhonen [27] e Layman et al. [30] apresentam as melhorias proporcionadas ao se introduzir o XP em uma empresa. O primeiro apresenta um estudo de caso sobre os efeitos da introdução do XP em uma grande organização de desenvolvimento de software, a área de tecnologia da Nokia, em que foram desenvolvidos projetos utilizando o modelo Cascata e XP para efeito de comparação. O projeto utilizando XP apresentou uma taxa de 15% de defeitos contra 35% do projeto utilizando o modelo Cascata. Além disso foram realizados questionários após seis meses do início dos projetos. A partir dos questionários foram obtidas respostas de que 67% acreditavam que a flexibilidade do desenvolvimento aumentou, 53% que houve um aumento na motivação da equipe e 23% que houve um aumento na qualidade do código. Outros 23% responderam que a qualidade do código na realidade diminuiu, Korhonen atribui estas respostas negativas principalmente à reação dos profissionais à propriedade coletiva do código, que apresentaram respostas como "há muitas pessoas trabalhando no código, alguém deveria ser responsável por ele".

O segundo apresenta os resultados da metodologia em uma empresa que presta serviços de soluções em TI para empresas aéreas. O estudo de caso longitudinal foi realizado durante 18 meses com profissionais experientes e utilizou-se do Framework de Avaliação do XP (XP-EF), no qual é divido em fatores de contexto, métricas de aderência e métricas de resultados. Eles constataram que a utilização do XP permitiu a diminuição dos erros antes de lançamento em 65% e aqueles pós-lançamento em 35%, também apresentam que a produtividade aumentou em 50% após a adoção do XP.

Outros estudos como o de Petersen e Wohlin [46] investigam como a percepção de gargalos, trabalho desnecessário e retrabalho é alterada migrando de um modelo tradicional para um método ágil. Utilizando uma métrica de detecção de falhas, mostram que foram encontradas 30 falhas no modelo tradicional, com 31% tendo passado despercebidas por estágios anteriores de testes, enquanto foram encontrados 20 falhas no método ágil, com 19% não sendo percebidas em estágios anteriores. Além disso foram percebidas as seguintes melhorias: requisitos mais estáveis levam a menos retrabalho; tudo que é iniciado é implementado; estimativas são mais precisas; detecção de falhas e feedback dos testes são recebidos rapidamente; redução no tempo de testes; comunicação direta diminui a necessidade de documentação. E o de Korhonen [] que busca explorar o impacto do desenvolvimento ágil nos dados relativos a defeitos e as ferramentas de comunicação de defeitos utilizadas ao longo da adoção do Scrum em uma empresa. Através de uma análise do número de defeitos abertos e o tempo de vida que um defeito fica sem ser corrigido os autores puderam chegar a conclusão que os métodos ágeis aumentaram a detecção de

defeitos mais cedo, os defeitos foram corrigidos mais rapidamente e um maior número de defeitos foi corrigido, aumentando assim a qualidade do software. Como conclusão do trabalho eles sugerem que empresas que estão adotando uma metodologia ágil utilizem maneiras de administrar a comunicação de defeitos e correção, uma vez que esta prática permite, segundo eles, a identificação de gargalos no processo.

Comparativo

Huo et al. [17] faz em 2004 um estudo comparativo entre a tradicional metodologia cascata e os processos ágeis para mostrar como métodos ágeis podem garantir qualidade de software quando os requisitos são instáveis e o tempo é curto. O estudo traz exemplos bastante úteis quanto a como os métodos tradicional e ágil tratam da garantia de qualidade, em sua conclusão os autores apontam que métodos ágeis realizam atividades de garantia com uma maior frequência, no entanto ressaltam também que a comparação entre tais metodologias é muitas vezes não realista devido as suas condições de desenvolvimento inicial.

A pesquisa realizada por Li et al. [33] tem como motivação a mesma que o nosso trabalho, ou seja, a inexistência de literatura especializada investigando os efeitos do Scrum na qualidade do software em termos de defeitos, densidade de defeitos e processos de garantia de qualidade. O estudo realizado durante três anos analisa um mesmo projeto que por 17 meses foi feito com metodologia guiada pelo planejamento, para que então adotasse o Scrum pelos próximos 20 meses. Como resultados da pesquisa os autores afirmam que a análise de defeitos não apresentou uma redução significativa na densidade dos mesmos ou que o perfil dos defeitos tenha mudado após a introdução da metodologia ágil.

Tal resultado contrasta com o encontrado por Hashmi e Baik [16] em que propõem uma comparação entre o modelo Espiral e o XP, utilizando as práticas de programação em pares, TDD e refatoração. A comparação é baseada na taxa de falhas por mil linhas de código em dois projetos, que apresentou um resultado de 6,91 no modelo Espiral e 1,43 no XP. Os autores defendem ainda que as práticas do XP analisadas no estudo se preocupam com a qualidade do código, e por causa da natureza iterativa do XP a frequência de atividades relacionadas à garantia de qualidade é maior que no modelo Espiral.

Ainda quanto a estudos comparativos temos o trabalho de Petersen e Wohlin [45] que realizam uma comparação entre os problemas e as vantagens de se utilizar métodos ágeis em desenvolvimento em grande escala. O estudo concorda com resultados apresentados em outras pesquisas sobre os as vantagens dos métodos ágeis, como: 1) requisitos mais precisos e fáceis de estimar, 2) comunicação direta diminui a necessidade de documentação, 3) feedback rápido devido a entregas frequentes, 4) redução de retrabalho, 5) testes são utilizados de forma mais eficiente e 6) maior transparência nas responsabilidades incentiva a entregar maior qualidade. Porém, apresentam problemas que não foram registrados em pesquisas anteriores quando se trata de grande escala, sendo eles: 1) grande duração da engenharia de requisitos devido a decisões complexas, 2) dificuldade em criar e manter listas de prioridades de requisitos, 3) períodos de espera no processo, especificamente a modelagem aguardando os requisitos, 4) redução na cobertura de testes devido a diminuição de projetos e falta de testes independentes e 5) aumento no esforço para manter o sistema.

E o trabalho de Lemos et al. [32] que apresenta um estudo empírico da aplicação de programação em pares e programação com testes a priori no desenvolvimento de funções auxiliares. O estudo parte da premissa que funções auxiliares são geralmente simples e costumam ser delegadas para programadores menos experientes, mas seu funcionamento incorreto pode resultar em falhas ou defeitos ao sistema. Apresentam hipóteses de que, caso as práticas ágeis sejam utilizadas, aumentariam a corretude do software, a quantidade de casos de teste e a cobertura dos testes. O estudo, conduzido com 85 programadores iniciantes, mostra que as hipóteses eram corretas e realmente há um aumento na corretude do software, quantidade e cobertura dos casos de teste. Porém, apresentam também que ambas as práticas resultam em um maior esforço, demandando maior tempo de desenvolvimento. O teste foi replicado com programadores experientes e apresentou resultados similares.

Métricas

Estes trabalhos tem como objetivo principal apresentar métricas que possam ajudar na verificação de aspectos da dependabilidade de um software desenvolvido com metodologia ágil, no entanto o foco da análise é quanto a orientação a objetos e não as práticas utilizadas. Percebe-se que os dois trabalhos (Olague et al. [42] e Olague et al. [41]) foram realizados pelos mesmos pesquisadores nos anos de 2007 e 2008 fazendo a análise de um mesmo software de código aberto da Mozilla Foundation. Os trabalhos se distinguem da seguinte forma:

O primeiro (Olague et al. [42]) busca avaliar três conjuntos de métricas diferentes em relação a propensão a existir falhas. As métricas analisadas são as de Chidamber and Kemerer (CK), Abreu's Metrics for Object-Oriented Design (MOOD) e Bansiya and Davis' Quality Metrics for Object-Oriented Design (QMOOD). Entre as conclusões os autores afirmam que enquanto as métricas CK e QMOOD são benéficas para a previsão de defeitos tanto em processos tradicionais como ágeis sendo utilizadas tanto na primeira entrega quanto nas seguintes, já métrica MOOD não é eficaz na previsão de defeitos.

O segundo estudo (Olague et al. [41]) identifica métricas que avaliam se métricas de complexidade aplicadas a classes orientadas a objetos são boas para identificar classes que possuem propensão a erro. Os estudo foi realizado com diversas métricas pouco estudadas que foram comparadas com a mais conhecida e validada métrica de métodos ponderados em uma classe (WMC) de Chidamber e Kemerer [4]. Em seus resultados os autores mostram que métricas menos conhecidas como a avaliação da complexidade de métodos utilizando desvio padrão proposta Michura et al. [35] e as de média de complexidade de métodos propostas por Etzkorn et al. [10] são mais consistentes que as métricas WMC. Também afirmam que estas métricas são úteis em encontrar classes propensas a erros em projetos desenvolvidos com metodologias ágeis.

Apesar de ambos tratarem de faltas e defeitos, não existe uma definição clara de tais termos, ambos são utilizados como erros reportados nos logs de correção de bugs.

A principal contribuição de tais trabalhos para nosso estudo é mostrar que existem métricas que, mesmo que com foco em orientação a objetos, podem ser utilizadas para avaliar a dependabilidade de métodos ágeis.

Outros

O estudo de Williams et al. [57] apresenta os resultados da utilização do Scrum com práticas ágeis como integração contínua e TDD em três equipes da Microsoft. As equipes eram formadas por profissionais com níveis de experiência médio e avançado, cada uma escrevendo uma média de 20 mil linhas de código-fonte e 16 mil linhas de teste. Os resultados mostram que uma das equipes apresentou um aumento de 250% na produtividade, enquanto outra equipe que teve baixa taxa de linhas de código teste/fonte (53%) apresentou a maior densidade de defeitos por linha de código (20%), reforçando a necessidade de se testar extensivamente o código.

Talby et al. [52] faz um estudo com informações adquiridas de um sistema crítico desenvolvido para a força aérea israelense de modo a estudar sua confiabilidade. Em seu estudo analisando diversas práticas ágeis eles chegam a conclusão de que a correção de defeitos contínua é realizada com antecedência proporcionada pelos métodos ágeis possui grandes benefícios, entre eles a redução do tempo necessário para corrigir os defeitos, a duração de tais defeitos e os custos de administração dos mesmos. Ainda afirmam que a utilização dos testes em métodos ágeis permite um aumento bastante grande na produtividade da equipe e qualidade do sistema.

O estudo de Korkala et al. [28] realizado com quatro projetos diferentes em que a quantidade e forma de comunicação com o cliente variava bastante analisa o impacto da existência do cliente junto à equipe de desenvolvimento ou em constante contato com a mesma tem na dependabilidade de um software quando analisada através da quantidade de defeitos existentes. Em seus resultados eles apresentam a seguinte conclusão: projetos que possuem uma menor quantidade de comunicação com o cliente tendem a possuir uma maior taxa de defeitos. Associada a essa conclusão os autores ainda propõem algumas diretrizes para a seleção de métodos apropriados para a comunicação no projeto.

Mnkandla e Dwolatzky [38] nos traz uma visão de como definir e garatir a qualidade do software quando utilizamos métodos ágeis. O trabalho propõe uma técnica de avaliação de metodologias ágeis para identificar quais fatores da qualidade eles aumentam. Como principal contribuição do trabalho os autores citam a utilização da técnica proposta para identificar novos focos de estudo da área.

Talvez o trabalho que mais se aproxima com o nosso é o realizado por Far [11], nele o autor mapeia a engenharia de software voltada para confiabilidade com as metodologias ágeis. São discutidas as maneiras de adaptar os processos ágeis para a engenharia de software confiável e também existe uma reflexão quanto aos problemas encontrados. A conclusão do autor sugere um maior estudo na área e cita possíveis tópicos para tal como o relacionamento entre a cobertura de testes e a métrica de tempo mínimo para o defeito, e também a classificação de defeitos para garantir que defeitos potenciais sejam testados.

Por fim, um outro trabalho analisado propõe uma metodologia baseada em XP. A nova metodologia chamada eXPERT tem como objetivo aumentar a produtividade em 30%, reduzir os defeitos em 30% e reduzir os excedentes do projeto em até 15%. O estudo de caso realizado na empresa Rila Solutions apresentou um aumento significativo na produtividade, diminuiu a taxa de defeitos e os esforços realizados. Sugere-se a utilização de tal metodologia em projetos com pequenas equipes e que não possuam uma definição detalhada dos requisitos.

Tabela 5.4: Estágios do ciclo de desenvolvimento

Estágio	Quantidade
Definir requisitos do incremento	2
Modelar incremento	0
Desenvolver e validar incremento	10
Integrar incremento e validar sistema	7

5.3 Análise dos Resultados

Nesta seção apresentamos a análise dos resultados encontrados em cada estudo de modo a responder cada uma das perguntas propostas no nosso estudo de mapeamento sistemático. Por fim, fazemos uma conclusão quanto aos resultados apresentados.

5.3.1 Respostas as questões de pesquisa

QP1 - Em quais estágios do ciclo de desenvolvimento ágil de um software as falhas e defeitos estão sendo tratados?

Considerando as quatro etapas do ciclo iterativo de desenvolvimento ágil de software propostas no Capítulo 2, representadas na Figura 2.3, pudemos encontrar referências aos estágios analisados por cada artigo, apresentados na Tabela 5.4. A partir deste resultado observamos que 55% dos artigos ([32] [53] [57] [16] [45] [27] [46] [25] [30] [18]) analisaram o desenvolvimento e o teste dos incrementos, e 38% ([30] [33] [52] [42] [41] [28] [18]) consideram a etapa de integração do incremento e validação do sistema em sua pesquisa. Dois dos artigos ([45] [46]) defendem que requisitos mais estáveis levam a menos retrabalho, o que diminui o tempo total de desenvolvimento e a chance de surgirem novas falhas ou defeitos ao alterar código já implementado. Embora nenhum dos artigos mencione explicitamente a modelagem dos incrementos, podemos inferir que a etapa não foi o foco dos estudos ou foi tratada implicitamente junto às etapas de requisitos ou desenvolvimento. A falta de referências à modelagem também pode ser observada como fator agravante às limitações apresentadas no Capítulo 2, principalmente sobre as argumentações contra a utilização de métodos ágeis no desenvolvimento de sistemas críticos.

QP2 - Quais são as principais práticas ágeis que garantem a entrega de um produto confiável?

Após a leitura aprofundada dos artigos selecionados pudemos identificar oito práticas ágeis que impactavam diretamente na dependabilidade de um software. Dos 18 estudos selecionados 9 traziam resultados relativos a redução de defeitos, 5 sobre redução de falhas e outros quatro não relacionavam o impacto na dependabilidade do software com a redução de falhas ou defeitos mas sim com outros aspectos. A Figura 5.5 mostra a distribuição das práticas de acordo com os termos de dependabilidade apresentados.

O gráfico de bolhas apresentado, apesar de ser um bom visualizador de quantidade de menções de uma prática, não identifica explicitamente o impacto causado pelas mesmas apenas apresenta que tais termos foram abordados no trabalho. Dos resultados apresentados, os estudos [S1, S5, S14, S30] mencionam o TDD como principal responsável

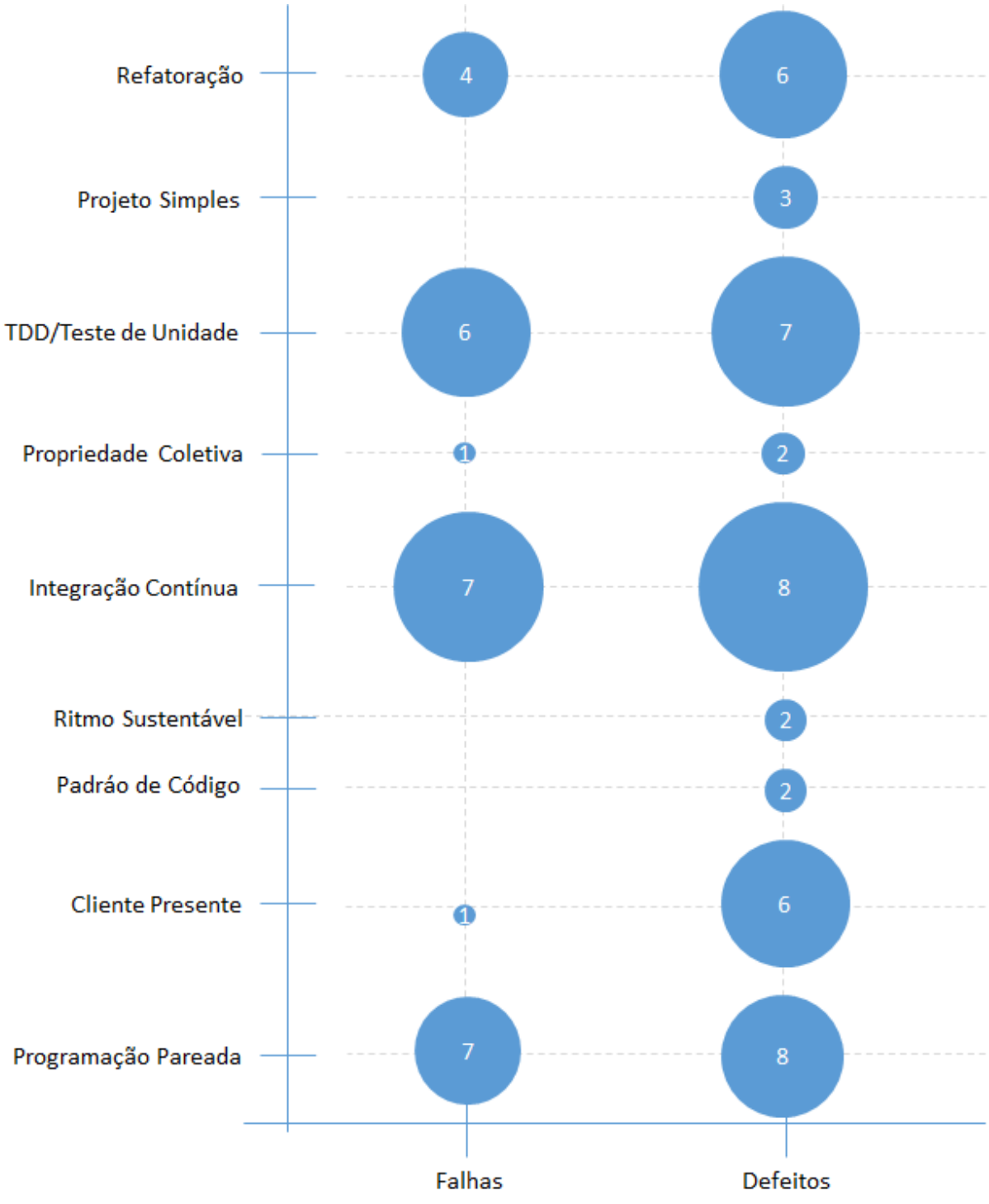

Figura 5.5: Distribuição de práticas ágeis

pela redução de defeitos e falhas de um projeto, com o número de defeitos reduzidos chegando a até 90% (Tufail e Malik [53]), outras práticas mencionadas como responsáveis pela redução de tais aspectos foram a presença do cliente na equipe ([S30]) de modo que aproximadamente 60% dos defeitos poderiam ter sido evitados com a utilização desta técnica no projeto, também formam mencionadas a integração contínua quando feita desde o início do projeto ([S13]) e a programação pareada que apesar de ser amplamente divulgada como uma das principais práticas benéficas a qualidade só apareceu em evidência em um estudo ([S1]), ainda assim esse mesmo estudo nos diz que a programação pareada e o TDD aumentam significativamente o tempo de produção do produto.

Apesar de não possuir resultados empíricos o artigo [S7] traz uma reflexão interessante quanto a como os métodos ágeis trabalham com a questão da dependabilidade. A análise da dependabilidade aplicada as práticas ágeis foi pensada considerando técnicas de remoção e prevenção de falhas. Devemos primeiro ter em mente que a prevenção de falhas é normalmente feita nos métodos tradicionais utilizando-se da engenharia de requisitos, comunicação com os envolvidos, utilização de documentação e padronização, deste modo a prevenção de falhas tem um foco em validação mais do que verificação. Já a remoção de falhas é feita através da revisão do código e teste do mesmo após ser escrito.

No entanto ao adotarmos as metodologias ágeis as práticas que eram de remoção de falhas são passadas para a categoria de prevenção uma vez que o TDD leva os testes de unidade e aceitação para antes da escrita do código, deste modo tornado-os em práticas preventivas, além disso a combinação do TDD com a integração contínua permite a realização de testes de regressão que também servirão para este propósito. Já a prática da programação pareada garante uma inspeção contínua do código durante o desenvolvimento de modo a se torna uma prática de prevenção de falhas também.

Com a utilização de todas estas práticas para a prevenção do aparecimento de falhas cabe apenas a equipe de desenvolvimento criar e priorizar, de acordo com a gravidade da falha, novas histórias de usuário de modo que esta prática foi a única identificada como sendo utilizada diretamente para a remoção de falhas. Uma esquematização do que foi dito encontra-se na Figura 5.6.

QP3 - Quais são pesquisadores e centros de pesquisa de maior relevância quando falamos sobre dependabilidade e métodos ágeis

Um total de 43 autores escreveram os 18 artigos selecionados. Escolhemos os 6 principais deles com base nos critérios apresentados a seguir: para fazer a seleção dos autores que considerávamos mais importantes fez-se uma pesquisa no Google Scholar e no próprio Google para identificar a quantidade de citações do autor e a página pessoal para que então pudessemos inferir a linha de pesquisa do mesmo. Foram então escolhidos todos aqueles que possuíam mais que uma citação ou aqueles que trabalhavam com dependabilidade e tinham diversos artigos citados no Google Scholar. Uma lista completa com os autores, quantidade de citações, suas instituições de ensino/trabalho, site pessoal e linha de pesquisa encontra-se no anexo C. Vale lembrar que os autores com mais de um resultado em nossa pesquisa ocorreram devido à prática do *snowballing* visto que é comum que um autor faça citações a trabalhos próprios passados que seguem uma mesma linha de pesquisa, logo, uma maior quantidade de resultados não indica necessariamente uma maior relevância na área, por isso investigamos cada um dos autores e suas publicações.

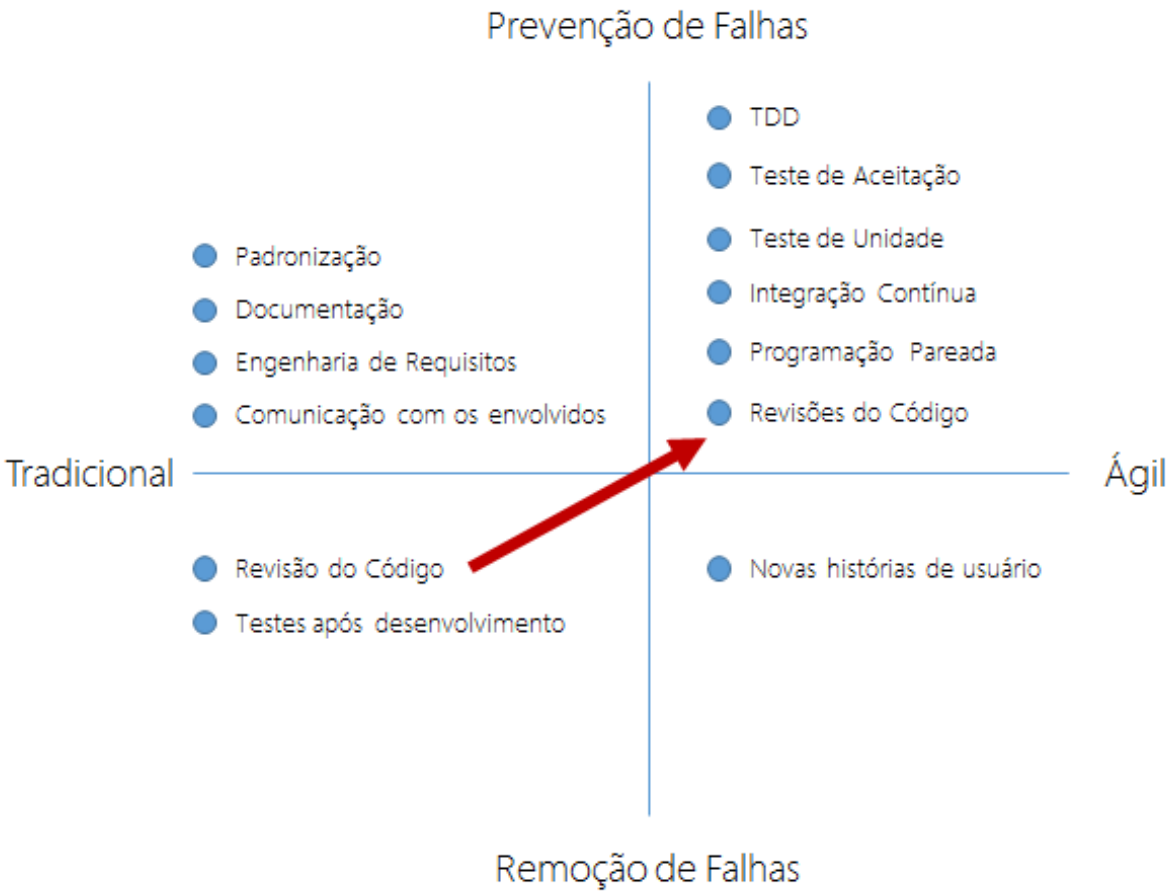

Figura 5.6: Comparação das práticas em métodos tradicionais e ágeis.

Claes Wohlin

É um pesquisador sueco do Instituto de Tecnologia de Bleking, possui mais que 7000 citações de acordo com o Google Scholar e tem como foco as seguintes áreas: engenharia de software empírica qualidade de software e processos de software. Seu trabalho envolve dependabilidade e métodos ágeis, tendo publicado livros e artigos sobre os dois assuntos. Os únicos trabalhos encontrados sobre os dois assuntos ao mesmo tempo são os mencionados neste estudo ([S10] e [S14]).

Kai Petersen

Foi orientando do Claes Wohlin, seus trabalhos na área são os mencionados neste mapeamento ([S10] e [S14]). Possui um foco mais em métodos ágeis e assim como seu orientador é adepto também da prática de estudos de mapeamento sistemático.

Kirsi Korhonen

Apesar de não se encontrar muita informação sobre a autora finlandesa ela possui mais estudos sobre o assunto dos que os mencionados em nossos resultados ([S13] e [S19]), isso porque eles tiveram que ser excluídos pois não eram disponíveis para nós. Sua tese de doutorado [26] é de grande valia para a área pois junta diversos artigos sobre o assunto.

Laurie Williams

A pesquisadora americana é quem nós encontramos que pode ser considerada referência na área devido a quantidade de estudos específicos voltados para os temas e as referências feitas aos mesmos. Seus temas de estudo são exatamente os mesmos que abordamos, dependabilidade e métodos ágeis e possui diversos artigos publicados sobre o assunto, muitos deles não foram considerados em nosso estudo pois são, por exemplo, avaliações de dependabilidade em uma prática ágil específica. Deste modo, incluímos apenas o trabalho [S5] neste estudo.

Letha H. Etzkorn

Uma pesquisadora com diversos trabalhos publicados mas que trabalha voltada para a parte de métricas. Seus estudos sobre o nosso tema foram contemplados no mapeamento ([S28] e [S29]).

Lucas Layman North

Foi aluno da Laurie Willians, também possui como foco os temas de dependabilidade e métodos ágeis, já tendo publicado alguns artigos nas duas áreas ([S20]).

QP4 - Quais das práticas apresentadas são maduras (foram amplamente testadas) e tem sido adotadas pela indústria do software?

Inicialmente dividimos os estudos de acordo com a faceta de pesquisa, a distribuição é apresentada na Figura 5.7. Dos estudos encontrados 10 deles foram realizados em empresas, entre elas empresas bastante consolidadas no mercado como Nokia ([S13] e [S19]),

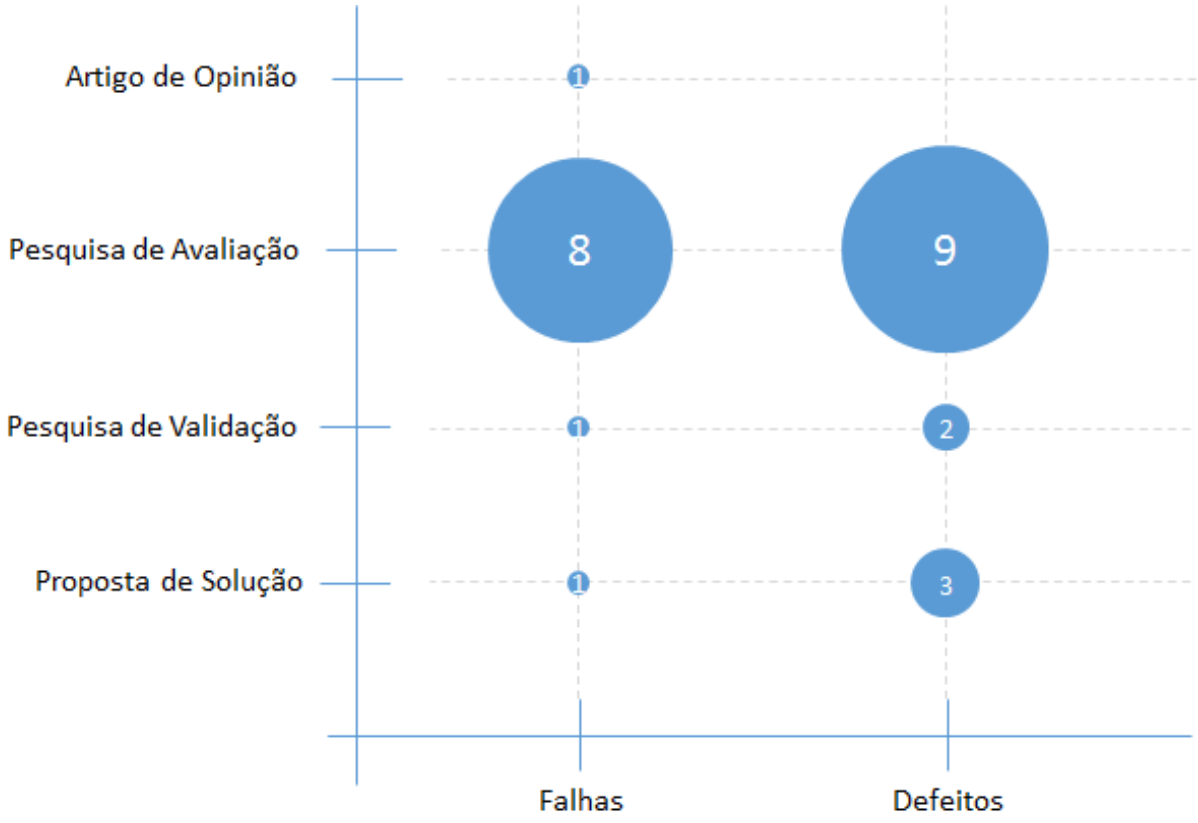

Figura 5.7: Distribuição da faceta de pesquisa.

Tabela 5.5: Práticas e princípios ágeis maduros

Prática ou Princípio	Quantidade
Integração Contínua	3
Propriedade Coletiva	2
Programação Pareada	2
Refatoração	2
TDD	2
Padrões de Código	1
Metáfora	1
Arquitetura Simplificada	1
Ritmo Sustentável	1

Ericsson ([S10] e [S14]), Microsoft ([S20]) e a Força Áerea Israelense ([S24]). Tivemos um estudo feito com base em outros artigos, um estudo feito na academia, um estudo que não apresentava onde foi realizado, além de dois estudos que analisaram um produto open-source da Mozilla Foundation ([S28] e [S29]).

Destes estudos, 8 (44%) foram realizados com profissionais, enquanto tivemos apenas um estudo com estudantes e outro que misturava estudantes e profissionais, sendo que pelo menos 220 pessoas foram utilizadas para tais trabalhos no total. Os outros trabalhos retornados não apresentavam esta informação. Quanto a experiência com as metodologia ágeis muitos dos trabalhos não apresentavam dados sobre a amostra, outros consistiam em estudos de caso logitudinais em que no início do projeto a amostra era iniciante e no fim já tinha conhecimento avançado em práticas ágeis.

Com base nessas informações selecionamos, para esta parte da análise, apenas estudos realizados em empresas com profissionais que possuíssem experiência intermediária ou avançada em práticas ágeis, de modo que apenas três estudos foram inclusos considerando tais critérios [S5,S19,S20]. Destes estudos dois abordavam o Scrum e um o XP. As práticas e princípios mencionadas em tais trabalhos foram a integração continuada (*continuous integration*) mencionada nos três trabalhos, pair programming, refactoring, collective ownership e test-driven development, mencionados em dois trabalhos e por último padrões de codificação, metáforas, arquitetura simplificada (*simple design*) e ritmo sustentável (*sustainable pace*) que foram mencionados em apenas um trabalho. A Tabela 5.5 abaixo faz um resumo do que foi mencionado.

5.3.2 Conclusões da análise

Agora que possuímos todos os dados coletados e a análise deles apresentada neste trabalho podemos fazer uma análise final sobre a dependabilidade quando utilizamos métodos ágeis. Para tal dividimos esta conclusão preliminar em três itens sendo eles as metodologias ágeis encontradas, ou seja, XP, Scrum e outras metodologias.

Quando falamos de dependabilidade no Scrum quatro dos cincos estudos mencionados nos apresentam dados que indicam que a utilização do mesmo nos leva a uma melhor qualidade do projeto e redução de defeitos, são citados como principais resultados da utilização do Scrum e práticas ágeis a possível redução de defeitos em até 90% e a diminuição de defeitos classificados como críticos, no entanto a quantidade de trabalhos selecionados

é muito baixa para tirarmos conclusões difinitivas. Algumas ressalvas são feitas como a de que a melhora de qualidade apresentada não necessáriamente está ligada ao código do projeto mas sim ao processo utilizado, isso se dá pois esta metodologia foca mais em aspectos gerenciais do desenvolvimento de software do que técnicos. O único estudo que não traz como conclusão que o Scrum aumenta a qualidade do projeto reduzindo defeitos, e portanto aumenta a dependabilidade, diz que quando comparado com metodologias guiadas pelo planejamento o Scrum não apresenta diferença significativa na redução de defeitos.

Os resultados apresentados quanto ao XP não diferem daqueles encontrados com o Scrum, apenas um estudo que compara o XP com o modelo em espiral diz que não existem diferenças quanto a qualidade do projeto, os outros três estudos apresentados mostram que o XP pode prover um ambiente em que 65% dos defeitos encontrados antes do lançamento de um produto podem ser evitados e 35% após sê-lo feito. Também são mencionados como benefícios a diminuição do tempo para reparo de um defeito apresentado em um projeto utilizando XP e a diminuição de código e funcionalidades não utilizadas.

Por último temos a análise dos estudos que não trouxeram uma metodologia especificada e resolveram focar em métodos ágeis em geral e as práticas por eles utilizadas. Seus resultados relativos a dependabilidade nos mostram que a programação pareada tende a aumentar a confiabilidade em termos de corretude enquanto o TDD tende a aumentar a confiabilidade em termos da cobertura de testes no software. São mencionados ainda a comunicação constante com o cliente como forma de diminuir defeitos e que apesar do aumento de qualidade a utilização de métodos ágeis impacta negativamente no esforço dos funcionários e causa um maior estresse aos mesmos.

Capítulo 6

Considerações Finais

Apresentamos neste capítulo as considerações finais deste trabalho. Para uma melhor compreensão por parte do leitor, optamos por dividir o conteúdo em três seções: Seção 5.1 descrevendo as ameaças a validade do nosso trabalho e limitações para o desenvolvimento do mesmo; Seção 5.2, contendo as conclusões encontradas; Seção 5.3 com as sugestões de trabalhos futuros.

6.1 Ameaças a Validade e Limitações

Nessa seção, são descritas as ameaças e limitações baseadas nas quatro categorias quanto a validade de um estudo: interna, externa, de conclusão e de contrução. Explicitamos tais questões que envolveram o desenvolvimento deste trabalho sem classificá-las especificamente pois podem ser aplicáveis para mais de uma categoria:

- Metodologia utilizada: a metodologia utilizada foi testada por autores que são referências na área e bastante utilizados em estudos de mapeamento sistemáticos e revisões sistemáticas. Ao utilizar este processo maturado diminuimos então a possibilidade da existência de erros no processo.
- Questões de Pesquisa: o conjunto de questões definidas neste trabalho não cobre toda a área compreendida por dependabilidade e métodos ágeis. Deste modo, algumas respostas sobre a abordagem, possivelmente, não serão encontradas nesse trabalho. Apesar disso as questões escolhidas para este trabalho foram bastante discutidas entre os autores e sua orientadora de modo que foram selecionadas aquelas consideradas mais relevantes para cumprir nossos objetivos.
- *String* de Busca e seleção de estudos: A quantia de 3245 títulos, obtida através de um processo exaustivo de pesquisa que buscou suprir as possíveis falhas na criação da *string* de busca utilizada em nossa pesquisa automática nas bases digitais com a pesquisa manual em periódicos e conferências, além da realização do snowballing posteriormente, nos possibilitou condições necessárias para responder satisfatoriamente todas as questões.
- Fontes de Dados: nós fizemos as buscas em apenas quatro grandes bases de dados, deste modo é possível que trabalhos relevantes não presentes nas mesmas tenham ficado fora de nossa pesquisa. Além disso as ferramentas de busca executam as

procuras de maneiras diferentes, deste modo, foi necessário adaptar nossa *string* e critérios de filtragem para cada uma delas.

- Extração dos Dados: Durante a extração dos dados, os estudos foram classificados de acordo com o julgamento do autor. A fim de mitigar o problema, todos os resultados da classificação de um autor foram revisados por outro.
- Período: Os dados analisados neste trabalho registraram contribuições apenas até julho de 2013 e sem restrição quanto a data inicial de modo que para obter resultados mais relevantes e atuais sugere-se a replicação da pesquisa e metodologia aqui apresentada.

6.2 Conclusão

Ao final deste trabalho indentificamos na literatura 3245 estudos, dos quais 18 julgamos que possuíam uma qualidade mínima estipulada e foram considerados relevantes para a nossa pesquisa. O tema de dependabilidade aplicada a metodologias ágeis ainda se mostra bastante inexplorado de modo que nossa análise não pode ser aprofundada e conclusões definitivas sobre quais práticas ágeis são mais benéficas a dependabilidade de um software ou quanto tais aspectos são tratados no ciclo de desenvolvimento não puderam ser dadas. Acreditamos que este trabalho desempenha uma grande importância em explicitar a premissa dada no primeiro capítulo de que faltam estudos sobre o assunto, e encorajamos pesquisadores a utilizarem os resultados aqui encontrados para se basearem em estudos futuros.

Ainda como resultados deste trabalho destacamos que há a necessidade de mais estudos que apresentem uma melhor definição do uso do termo qualidade em pesquisas realizadas sobre métodos ágeis e dependabilidade visto que a definição para o mesmo varia bastante dependendo do autor. Um trabalho nesse sentido já pode foi feito por Mnkandla e Dwolatzky [38] mas ainda tem sido pouco utilizado. Também ressaltamos que mesmo com uma clara definição dada por Avizienis et al. [54] há um claro déficit na utilização dos termos falta e defeito, sendo eles muitas vezes tratados pelos autores simplesmente como o erro reportado.

Por último fazemos um resumo dos principais resultados encontrados neste trabalho e que podem servir de guia para estudos futuros:

- Existe um déficit visível na quantidade de estudos envolvendo os temas dependabilidade e métodos ágeis representado aqui pela quantidade de trabalhos retornados em nosso estudo.
- Existe uma necessidade de melhor definição e utilização dos termos qualidade, faltas e defeitos quando tratamos de dependabilidade visto que cada autor apresenta os mesmos de forma diferente.
- A partir dos resultados encontrados podemos concluir que metodologias ágeis tendem a ser benéficas a dependabilidade do software, no entanto são necessários mais contribuições para uma conclusão definitiva.
- Existe a necessidade de criação de métricas específicas para avaliar a dependabilidade de um método ágil.

- Este trabalho identificou alguns dos nomes que devem servir de referência para estudos na área como a pesquisadora Laurie Williams devido a quantidade de artigos e relevância dos mesmos para a área.

6.3 Trabalhos Futuros

Durante a execução desse trabalho, percebemos diversos problemas quanto a literatura relativa aos dois temas estudados, deste modo, foram identificados possíveis continuações e novos assuntos que podem ser aprofundados por meio de novos trabalhos:

- É necessária a realização de mais estudos relevantes para definir "qualidade do software"quando utilizamos termos de dependabilidade.
- É possível repetir o estudo periódicamente para avaliar se novos trabalhos tem sido publicados sobre o assunto.
- Pode-se explorar mais o assunto através de um novo estudo de mapeamento sistemático utilizando a mesma metodologia e string de busca (ou alguma bastante semelhante), para averiguar estudos específicos sobre práticas ágeis que tratem do tema dependabilidade. Muitos estudos foram retornados sobre práticas ágeis como o test-driven development e pair programming, no entanto foram removidos de acordo com nossos critérios de exclusão.
- Após a realização deste estudo percebeu-se uma grande deficiência na quantidade de estudos tratando os dois temas, para isso sugere-se os sguintes passos para um novo estudo:

  Estudo da literatura sobre métricas de qualidade aplicadas a métodos ágeis. Um trabalho semelhante ao apresentado por Kitchenham [23], porém focado em metodologias ágeis.

  Compilação das metodologias encontradas e criação um conjunto de métricas que viabilize a avaliação de um projeto ágil durante todo o seu ciclo de desenvolvimento e não apenas avaliando alguma prática específica.

  Por último, a realização de casos de estudo se possível na academia e na indústria utilizando o conjunto de métricas criado de modo a validá-lo.

Apêndice A

Listas de Artigos

A.1 *Necessidade do estudo*

No.	Artigo	Categoria*
SRW1	A comparison of software cost, duration, and quality for waterfall vs. iterative and incremental development: A systematic review	MAD
SRW2	A Systematic Literature Review on Fault Prediction Performance in Software Engineering	D
SRW3	A systematic review of security requirements engineering	D
SRW4	A systematic review of software maintainability prediction and metrics	D
SRW5	A systematic review of software robustness	D
SRW6	Agile Practices in Global Software Engineering - A Systematic Map	MA
SRW7	Developers Motivation in Agile Teams	MA
SRW8	Empirical studies of agile software development: A systematic review	MA
SRW9	Empirical Studies on Quality in Agile Practices: A Systematic Literature Review	MAD
SRW10	Evaluation and Measurement of Software Process Improvement - A Systematic Literature Review	D
SRW11	Software fault prediction metrics: A systematic literature review	D
SRW12	The evolution of agile software development in Brazil	MA
SRW13	The lean gap: A review of lean approaches to large-scale software systems development	MA
SRW14	User-Centered Design and Agile Methods: A Systematic Review	MA
SRW15	What Does Research Say about Agile and Architecture?	MA
SRW16	What's up with software metrics? – A preliminary mapping study	D

* Categorias: MAD - Métodos Ágeis e Dependabilidade; A - Método Ágil; D - Dependabilidade

A.2 *Busca Eletrônica*

No.	Artigo
SSQ1	Development of auxiliary functions: should you be agile? an empirical assessment of pair programming and test-first programming
SSQ2	A Case Study Analyzing the Impact of Software Process Adoption on Software Quality
SSQ3	A fault-driven lightweight process improvement approach
SSQ4	Determining the Applicability of Agile Practices to Mission and Life-Critical Systems
SSQ5	Scrum + Engineering Practices: Experiences of Three Microsoft Teams
SSQ6	Software Quality Assurance in XP and Spiral - A Comparative Study
SSQ7	Software Reliability Engineering for Agile Software Development
SSQ8	Towards quantitative software reliability assessment in incremental development processes
SSQ9	Defining Agile Software Quality Assurance
SSQ10	A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case
SSQ11	Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation
SSQ12	Multi-sprint planning and smooth replanning: An optimization model
SSQ13	Evaluating the impact of an agile transformation: a longitudinal case study in a distributed context
SSQ14	The effect of moving from a plan-driven to an incremental software development approach with agile practices
SSQ15	Strengths and barriers behind the successful agile deployment insights from the three software intensive companies in Finland
SSQ16	Investigating the extreme programming system - An empirical study

A.3 *Snowballing*

No.	Artigo
SSB1	Migrating defect management from waterfall to agile software development in a large-scale multi-site organization: A case study
SSB2	Exploring defect data, quality and engagement during agile transformation at a large multisite organization
SSB3	Evaluating the impact of agile adoption on the software defect management practices
SSB4	Exploring extreme programming in context: an industrial case study
SSB5	Transition from a plan-driven process to Scrum: A longitudinal case study on software quality
SSB6	How does agility ensure quality?
SSB7	Lean software development: two case studies
SSB8	Agile Software Testing in a Large-Scale Project

Apêndice B

Estudos Selecionados

ID	Referência	Título
[S1]	Lemos et al. [32]	Development of auxiliary functions: should you be agile? an empirical assessment of pair programming and test-first programming
[S2]	Tufail e Malik [53]	A Case Study Analyzing the Impact of Software Process Adoption on Software Quality
[S5]	Williams et al. [57]	Scrum + Engineering Practices: Experiences of Three Microsoft Teams
[S6]	Hashmi e Baik [16]	Software Quality Assurance in XP and Spiral - A Comparative Study
[S7]	Far [11]	Software Reliability Engineering for Agile Software Development
[S9]	Mnkandla e Dwolatzky [38]	Defining Agile Software Quality Assurance
[S10]	Petersen e Wohlin [45]	A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case
[S13]	Korhonen [27]	Evaluating the impact of an agile transformation: a longitudinal case study in a distributed context
[S14]	Petersen e Wohlin [46]	The effect of moving from a plan-driven to an incremental software development approach with agile practices
[S19]	Korhonen [25]	Evaluating the impact of agile adoption on the software defect management practices
[S20]	Layman et al. [30]	Exploring extreme programming in context: an industrial case study
[S21]	Li et al. [33]	Transition from a plan-driven process to Scrum: A longitudinal case study on software quality
[S22]	Huo et al. [17]	How does agility ensure quality?
[S24]	Talby et al. [52]	Agile Software Testing in a Large-Scale Project
[S28]	Olague et al. [42]	Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes

[S29]	Olague et al. [41]	An empirical validation of object-oriented class complexity metrics and their ability to predict error-prone classes in highly iterative, or agile, software: a case study
[S30]	Korkala et al. [28]	A case study on the impact of customer communication on defects in agile software development
[S31]	Ilieva et al. [18]	Analyses of an agile methodology implementation

Tabela B.1: Artigos Selecionados no Mapping Study

Apêndice C

Autores

Nome	Qtd	GS*	Instituição	Página	Categ.**
Claes Wohlin	2	7041	Blekinge Institute of Technology	[1]	AD
Hector M. Olague	2	x			
Kai Petersen	2	491	Blekinge Institute of Technology		AD
Kirsi Korhonen	2	x			AD
Laurie Williams	2	6984	North Carolina State University	[2]	AD
Letha H. Etzkorn	2	x	U. of Alabama in Huntsville	[3]	A
Adam Meltzer	1	x			
Alessandro Garcia	1	3965	PUC-Rio	[4]	D
Ali Afzal Malik	1	x			
Barry Dwolatzky	1	x	University of the Witwatersrand		
Behrouz Far	1	x	University of Calgary	[5]	
Dubinsky, Y.	1	x			
Eliza Stefanova	1	x			
Ernest Mnkandla	1	60	University of South Africa	[6]	A
Fabiano C. Ferrari	1	521	UFSCAR		
Fábio F. Silveira	1	x			
Gabe Brown	1	x			
Harry S. Delugach	1	x			
Jingyue Li	1	x			
Jongmoon Baik	1	433	KAIST	[7]	AD
June Verner	1	x	Drexel University	[8]	
Keren, A.	1	x			
Liming Zhu	1	x	University of New South Wales	[9]	
Lucas Layman North	1	778	Fraunhofer Center for Experimental Software Engineering	[10]	AD
Lynn Cunningham	1	x			
Mikko Korkala	1	x			
Ming Huo	1	x			
Muhammad Ali Babar	1	2354	Lancaster University	[11]	
Nachiappan Nagappan	1	3616	Microsoft Research	[12]	D
Nils B. Moe	1	697	SINTEF ICT		A
Orit Hazzan	1	1789	Israel Institute of Technology	[13]	A
Otávio A. L. Lemos	1	315	UNIFESP	[14]	A
Pekka Abrahamsson	1	3234	Free University of Bozen-Bolzano		A

Pekka Kyllönen	1	x			
Penko Ivanov	1	x			
Reham Tufail	1	x			
Sajid Ibrahim Hashmi	1	49	University of Limerick		
Sampson Gholston	1	x	U. of Alabama in Huntsville		
Sherri L. Messimer	1	x			
Stephen Quattlebaum	1	x			
Sylvia Ilieva	1	x	University of Sofia		A
David Talby	1	x			A
Tore Dybå	1	3800	University of Oslo	[15]	A

Tabela C.1: Estudo dos Autores

* Número de citações na página do autor do Google Scholar (scholar.google.com).
** Categorias: A - Ágil; D - Dependabilidade; AD - Ágil + Dependabilidade
[1] http://www.wohlin.eu/
[2] http://collaboration.csc.ncsu.edu/laurie/
[3] http://www.cs.uah.edu/~letzkorn/
[4] http://www.inf.puc-rio.br/~afgarcia
[5] http://www.enel.ucalgary.ca/People/far/
[6] http://www.unisa.ac.za/
[7] http://spiral.kaist.ac.kr/wp/?page_id=113
[8] http://www.pages.drexel.edu/~jmv23/
[9] http://cgi.cse.unsw.edu.au/~limingz/home/
[10] http://lucas.ezzoterik.com/
[11] http://malibabar.wordpress.com/
[12] http://research.microsoft.com/en-us/people/nachin/
[13] http://edu.technion.ac.il/Faculty/OritH/HomePage/
[14] http://www.sjc.unifesp.br/docente/otavio/
[15] http://www.mn.uio.no/ifi/english/people/aca/toredy/

Referências

[1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, January 2004. x, 11, 12, 14, 15, 17, 18, 19

[2] Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, October 1999. 3

[3] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004. 7, 10

[4] S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, 1994. 41

[5] Mario Dantas. *Computação Distribuída de Alto Desempenho*. Axcel Books, Brasília, 2005. 14

[6] Jorge Calmon de Almeida Biolchini, Paula Gomes Mian, Ana Candida Cruz Natali, Tayana Uchôa Conte, and Guilherme Horta Travassos. Scientific research ontology to support systematic review in software engineering. *Adv. Eng. Inform.*, 21(2):133–151, April 2007. 23

[7] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9–10):833 – 859, 2008. xi, 1, 5

[8] Tore Dybå, Vigdis By Kampenes, and Dag Sjøberg. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, 48(8):745–755, August 2006. 21

[9] Kent Beck et al. *Manifesto for Agile Software Development*. Disponível online em: http://agilemanifesto.org/, 2001. x, 2, 4, 6, 8

[10] Davis C Etzkorn LH, Bansiya J. Design and code complexity metrics for oo classes. *Journal of Object-oriented Programming*, 1999. 41

[11] B. Far. Software reliability engineering for agile software development. In *Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on*, pages 694–697, 2007. 42, 56

[12] Norman Fenton and Bev Littlewood. *Software Reliability and Metrics*. Springer, New York, NY, USA, 1991. 13

[13] Martin Fowler and Jim Highsmith. The agile manifesto. *Software Development Magazine*, 9(8):29–30, 2001. 4

[14] A. Ghazarian. Reliability in agile software engineering: A dilemma. Technical report, IEEE Reliability Society, East Lansing, Michigan, 2011. 1

[15] Jo Hannay, Dag Sjøberg, and Tore Dybå. A systematic review of theory use in software engineering experiments. *IEEE Trans. Softw. Eng.*, 33(2):87–107, February 2007. 21

[16] S.I. Hashmi and Jongmoon Baik. Software quality assurance in xp and spiral - a comparative study. In *Computational Science and its Applications, 2007. ICCSA 2007. International Conference on*, pages 367–374, 2007. 40, 43, 56

[17] Ming Huo, J. Verner, Liming Zhu, and M.A. Babar. Software quality and agile methods. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, pages 520–525 vol.1, 2004. 40, 56

[18] Sylvia Ilieva, Penko Ivanov, and Eliza Stefanova. Analyses of an agile methodology implementation. In *Proceedings of the 30th EUROMICRO Conference*, EUROMICRO '04, pages 326–333, Washington, DC, USA, 2004. IEEE Computer Society. 43, 57

[19] Samireh Jalali and Claes Wohlin. Systematic literature studies: database searches vs. backward snowballing. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '12, pages 29–38, New York, NY, USA, 2012. ACM. 30

[20] Barry Johnson, editor. *Design & analysis of fault tolerant digital systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988. x, 15

[21] H. Jonsson, S. Larsson, and S. Punnekkat. Agile practices in regulated railway software development. In *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on*, pages 355–360, 2012. 30

[22] Vigdis By Kampenes, Tore Dybå, Jo Hannay, and Dag Sjøberg. Systematic review: A systematic review of effect size in software engineering experiments. *Inf. Softw. Technol.*, 49(11-12):1073–1086, November 2007. 21

[23] Barbara Kitchenham. What's up with software metrics? - a preliminary mapping study. *J. Syst. Softw.*, 83(1):37–51, January 2010. 53

[24] Barbara Kitchenham and Stuart Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007. x, 21, 22

[25] Kirsi Korhonen. Evaluating the impact of agile adoption on the software defect management practices. *Software Quality Professional*, 14(1):599–624, 2011. 43, 56

[26] Kirsi Korhonen. *Supporting Agile Transformation with Defect Management in Large Distributed Software Development Organisation*. PhD thesis, Tampere University of Technology, 2012. 47

[27] Kirsi Korhonen. Evaluating the impact of an agile transformation: a longitudinal case study in a distributed context. *Software Quality Journal*, 21(4):599–624, 2013. 39, 43, 56

[28] M. Korkala, P. Abrahamsson, and P. Kyllonen. A case study on the impact of customer communication on defects in agile software development. In *Agile Conference, 2006*, pages 11 pp.–88, 2006. 42, 43, 57

[29] Jean-Claude Laprie. *Dependability: basic concepts and terminology in English, French, German, Italian, and Japanese*. Dependable computing and fault-tolerant systems. Springer-Verlag, 1992. 14

[30] Lucas Layman, Laurie Williams, and Lynn Cunningham. Exploring extreme programming in context: An industrial case study. In *Proceedings of the Agile Development Conference*, ADC '04, pages 32–41, Washington, DC, USA, 2004. IEEE Computer Society. 39, 43, 56

[31] Lawrence Leemis. *Probabilistic Models and Statistical Methods*. Prentice Hall, 1995. 13

[32] O.A.L. Lemos, F.C. Ferrari, F.F. Silveira, and A. Garcia. Development of auxiliary functions: Should you be agile? an empirical assessment of pair programming and test-first programming. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 529–539, 2012. 41, 43, 56

[33] Jingyue Li, Nils B. Moe, and Tore Dybå. Transition from a plan-driven process to scrum: a longitudinal case study on software quality. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 13:1–13:10, New York, NY, USA, 2010. ACM. 40, 43, 56

[34] Michael R. Lyu, editor. *Handbook of software reliability engineering*. McGraw-Hill, Inc., Hightstown, NJ, USA, 1996. 12

[35] J. Michura and L.F. Capretz. Metrics suite for class complexity. In *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, volume 2, pages 404–409 Vol. 2, 2005. 41

[36] Yuan-Shun Dai Min Xie, Kim-Leng Poh. *Computing System Reliability: Models and Analysis*. Springer, US, 2004. 13, 19

[37] S.M. Mitchell and C.B. Seaman. A comparison of software cost, duration, and quality for waterfall vs. iterative and incremental development: A systematic review. In *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, pages 511–515, 2009. 23

[38] E. Mnkandla and B. Dwolatzky. Defining agile software quality assurance. In *Software Engineering Advances, International Conference on*, pages 36–36, 2006. 42, 52, 56

[39] John D. Musa, Anthony Iannino, and Kazuhira Okumoto. *Software reliability: measurement, prediction, application*. McGraw-Hill, Inc., New York, NY, USA, 1987. 13

[40] Claudia O. Melo, Viviane Santos, Eduardo Katayama, Hugo Corbucci, Rafael Prikladnicki, Alfredo Goldman, and Fabio Kon. The evolution of agile software development in brazil. *Journal of the Brazilian Computer Society*, pages 1–30, 2013. 1

[41] Hector M. Olague, Letha H. Etzkorn, Sherri L. Messimer, and Harry S. Delugach. An empirical validation of object-oriented class complexity metrics and their ability to predict error-prone classes in highly iterative, or agile, software: a case study. *J. Softw. Maint. Evol.*, 20(3):171–197, May 2008. 41, 43, 57

[42] H.M. Olague, L.H. Etzkorn, S. Gholston, and S. Quattlebaum. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *Software Engineering, IEEE Transactions on*, 33(6):402–419, 2007. 41, 43, 56

[43] Rivalino Mathias Paulo Maciel, Kishor Trivedi and Dong Kim. *Dependability Modeling In: Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions.* Ed. Hershey: IGI Global, Pennsylvania, USA, 2010. 13

[44] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, Swinton, UK, UK, 2008. British Computer Society. x, 21, 22, 32

[45] Kai Petersen and Claes Wohlin. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *J. Syst. Softw.*, 82(9):1479–1490, September 2009. 10, 40, 43, 56

[46] Kai Petersen and Claes Wohlin. The effect of moving from a plan-driven to an incremental software development approach with agile practices. *Empirical Softw. Engg.*, 15(6):654–693, December 2010. 10, 39, 43, 56

[47] Tom Poppendieck and Mary Poppendieck. *Lean Software Development: An Agile Toolkit for Software Development Managers.* Addison-Wesley Professional, 1 edition, May 2003. 3

[48] Ken Schwaber. *Agile Project Management With Scrum.* Microsoft Press, Redmond, WA, USA, 2004. 3, 10

[49] Panagiotis Sfetsos and I. Stamelos. Empirical studies on quality in agile practices: A systematic literature review. In *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, pages 44–53, 2010. 23

[50] Forrest Shull, Vic Basili, Barry Boehm, A. Winsor Brown, Patricia Costa, Mikael Lindvall, Dan Port, Ioana Rus, Roseanne Tesoriero, and Marvin Zelkowitz. What we have learned about fighting defects. In *Proceedings of the 8th International Symposium on Software Metrics*, METRICS '02, pages 249–, Washington, DC, USA, 2002. IEEE Computer Society. 1

[51] Ian Sommerville. *Software engineering (8th ed.).* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2007. x, 8, 9, 10

[52] D. Talby, A. Keren, O. Hazzan, and Y. Dubinsky. Agile software testing in a large-scale project. *Software, IEEE*, 23(4):30–37, 2006. 42, 43, 56

[53] R. Tufail and A.A. Malik. A case study analyzing the impact of software process adoption on software quality. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 254–256, 2012. 39, 43, 45, 56

[54] Algirdas Avizienis Ucla, Algirdas Avizienis, Jean claude Laprie, and Brian Randell. Fundamental concepts of dependability, 2001. 52

[55] VersionOne. 7th annual state of agile dev survey. Disponível online em: http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf, 2013. 1

[56] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requir. Eng.*, 11(1):102–107, December 2005. xi, 32

[57] Laurie Williams, Gabe Brown, Adam Meltzer, and Nachiappan Nagappan. Scrum + engineering practices: Experiences of three microsoft teams. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, ESEM '11, pages 463–471, Washington, DC, USA, 2011. IEEE Computer Society. 42, 43, 56