
UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
PRÓ-REITORIA DE ENSINO DE GRADUAÇÃO
DEPARTAMENTO DE ESTATÍSTICA E INFORMÁTICA (DEINFO)
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DENNYS DE SOUZA BARROS

# QUALIDADE DE SOFTWARE EM MÉTODOS ÁGEIS: UMA REVISÃO SISTEMÁTICA

TRABALHO DE CONCLUSÃO DE CURSO

Recife
2018

DENNYS DE SOUZA BARROS

# QUALIDADE DE SOFTWARE EM MÉTODOS ÁGEIS: UMA REVISÃO SISTEMÁTICA

Monografia apresentado ao Bacharelado em Ciência da Computação da Universidade Federal Rural de Pernambuco, como parte dos requisitos necessários à obtenção do título de Bacharel em Ciência da Computação.

Orientador: Suzana Cândido de Barros Sampaio

Recife

2018

MINISTÉRIO DA EDUCAÇÃO E DO DESPORTO
UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO (UFRPE)
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

http://www.bcc.ufrpe.br

## FICHA DE APROVAÇÃO DO TRABALHO DE CONCLUSÃO DE CURSO

Trabalho defendido por Dennys de Souza Barros às 8 horas do dia 06 de fevereiro de 2018, no Auditório do CEAGRI-02 – Sala 07, como requisito para conclusão do curso de Bacharelado em Ciência da Computação da Universidade Federal Rural de Pernambuco, intitulado **Qualidade de Software em Métodos Ágeis: uma Revisão Sistemática**, orientado por Suzana Cândido de Barros Sampaio e aprovado pela seguinte banca examinadora:

Suzana Cândido de Barros Sampaio
DEINFO/UFRPE

Marcelo Luiz Monteiro Marinho
DEINFO/UFRPE

Marcos José de Menezes Cardoso Junior
DEINFO/UFRPE

# Agradecimentos

Agradeço a todos que tiveram participação no desenvolvimento deste trabalho, tanto de forma direta quanto indireta. Dentre os que mais se destacaram, posso citar Suzana Sampaio, Marcelo Marinho, Caroline Hortêncio, Virgínia Lira, Danielle Barros, Deane Barros, Marco Segundo, Thomás Leal e Maria Isabel.

# Resumo

Em décadas recentes, produtos de software eram produzidos de maneira desorganizada e sem um planejamento adequado, resultando assim em produtos de má qualidade ou até mesmo não gerando produto algum. Por conta disso, foram criados modelos ou guias que auxiliam o desenvolvimento de software. As metodologias chamadas tradicionais são orientadas à planejamento, resultando em problemas por falta de flexibilidade. Surgiu então o conceito de métodos ágeis. A noção de que o desenvolvimento de software utilizando métodos ágeis está orientado para à geração de resultados de melhor qualidade se comparados à métodos tradicionais é encontrada em uma série de literaturas sobre metodologias ágeis. Esta monografia tem o objetivo, por meio de uma revisão sistemática, de investigar quais práticas presentes em metodologias ágeis auxiliam o atingimento da qualidade de software. Técnicas como TDD, programação em pares e constante interação do cliente foram vistas como bastante positivas, contribuindo bastante para o atingimento da qualidade. Áreas tais como segurança e usabilidade, e a execução incorreta do TDD, porém, foram vistas como áreas que precisam de melhoria.

Palavras-chave: métodos ágeis; qualidade de software; revisão sistemática da literatura.

# Abstract

Few decades ago, software products were built in an unorganized way and without a proper planning, resulting in products without quality or even not resulting in products at all. Due to that, methods or guidelines have been created to assist the software development. The methodologies called traditionals are driven by plan, resulting in flexibility problems. So, it has been arisen the concept of agile methods of software development. The notion that software development using agile methods generates products with better quality if compared with traditional methods can be found in several articles. This monography has the goal, by performing a systematic literature review, of investigate what practices present on agile methods assist the aim of provide products with quality. Techniques as TDD, pair programming and constant client involvement have been considered good in order to reach quality. Areas as security, usability and incorrect execution of TDD, though, have been seen as areas that need improvement.



Keywords: agile methods; software quality; systematic literature review.

# Lista de ilustrações

# Lista de tabelas

# Lista de abreviaturas e siglas

| | |
|---|---|
| CMMI | Capability Maturity Model Integration |
| IEC | International Eletrotechnical Commision |
| ISO | International Organization for Standardization |
| MPS.BR | Melhoria de Processo do Software Brasileiro |
| PP | Programação em Pares |
| TCC | Trabalho de Conclusão de Curso |
| TDD | Test-Driven Development |
| TI | Tecnologia da Informação |
| XP | eXtreme Programming |

Sumário

# 1 INTRODUÇÃO

O desenvolvimento de software deve ser organizado para entregar soluções ótimas, rápidas e baratas (DYBÅ; DINGSØYR, 2007). Desenvolver um software não é uma tarefa simples. Uma organização que trabalha com desenvolvimento de soluções tecnológicas precisa lidar com habilidades, tecnologias, requisitos e ainda gerenciar pessoas (DEVI, 2013). Por volta de 1970, os softwares eram produzidos de maneira desorganizada e sem um planejamento adequado, resultado assim em produtos de má qualidade ou até mesmo não gerando produto algum (LIMA, 2015). Por conta disso, foram criados modelos ou guias que auxiliam o desenvolvimento de software.

As metodologias chamadas tradicionais são orientadas à planejamento ou documentação. Um exemplo de modelo tradicional é o Modelo Cascata (SOMMERVILLE, 2011), onde somente após a conclusão da documentação é que o software poderia ser implementado.

Uma vantagem de se utilizar metodologias tradicionais está na grande quantidade de documentação gerada, aumentando assim a quantidade de informação. Além disso, um planejamento bem traçado e robusto pode ser de ajuda no sucesso do projeto. Porém, é assumido que os requisitos do usuário são bem entendidos desde o início do planejamento e estes não sofrem muita alteração com o passar do tempo, algo que raramente acontece (DEVI, 2013).

Processos orientados a documentação podem ser considerados fatores limitadores aos desenvolvedores e muitas organizações não possuem recursos para processos pesados de produção de software. Por conta disso, organizações pequenas poderiam por decidir a não usar nenhum processo. Isto poderia levar a efeitos desastrosos na qualidade do produto final, além de dificultar a entrega do software nos prazos e custos predefinidos (SOARES, 2004). Além disso, utilizando-se métodos tradicionais, o que pode acontecer é o fato de o usuário não ter muito contato com o time de desenvolvimento, nem receber versões parciais do sistema para validar se o resultado está suprindo suas expectativas ou não (DEVI, 2013).

Por conta disso, em 2001, engenheiros de software experientes decidiram se reunir e pensar em alguma alternativa para os métodos tradicionais de desenvolvimento de software, focados em extensa documentação e densos processos de software (HIGHSMITH et al., 2001). Surgiu então o conceito de métodos ágeis. Estão inclusos nos métodos ágeis o *eXtreme Programming* (XP) (BECK, 2004) e o Scrum (SUTHERLAND; SCHWABER, 2017).

Métodos ágeis são amplamente usados na indústria. Existe uma importante

necessidade de desenvolvedores estarem atentos à qualidade do software produzido e a como adequar os processos com o objetivo de atingir a qualidade (HUO et al., 2004).

De acordo com Huo, Verner, Zhu (2004) e Bhasin (2012) a utilização de métodos ágeis produzem software de boa qualidade em um curto espaço de tempo. A palavra “qualidade” pode ter vários significados. Mas, de acordo com a *International Organization of Standardization* (ISO), a definição de qualidade é o grau no qual as funcionalidades e características de um produto ou serviço satisfaz seus requisitos (ISO, 2012). A satisfação do cliente gera um impacto positivo no custo da organização, nos lucros e no aumento de vendas (JUNG; KIM; CHUNG, 2004).

Nesse contexto, este trabalho visa investigar e analisar quais características presentes em métodos ágeis colaboram para o atingimento da qualidade. Além disso, encontrar lacunas nesses métodos e apontar possíveis soluções para preenchê-las.

## 1.1 JUSTIFICATIVA

Desenvolvimento de software utilizando métodos ágeis é reconhecido por conseguir gerenciar mudanças de requisitos no decorrer do ciclo de desenvolvimento, entregar produtos em espaço de tempo reduzido e restrições de orçamento se comparado com métodos tradicionais de desenvolvimento (HUO et al., 2004).

A noção de que software o desenvolvimento de software utilizando métodos ágeis está orientado a para o desenvolvimento de software de melhor qualidade se comparado com os métodos tradicionais é encontrada em uma série de literaturas sobre metodologias ágeis (MNKANDLA; DWOLATZKY, 2006; SFETSOS; STAMELOS, 2010; BHASIN, 2012). O que podemos observar é que, de fato, o modo como qualidade é tratado em métodos ágeis é diferente de como é tratado nos métodos tradicionais.

Para alguns, se atinge qualidade em métodos ágeis quando o desenvolvimento de software consegue responder a mudanças requisitadas pelo cliente (MCBREEN, 2003). Isso implica que a entrega frequente de software funcional, testado e aprovado pelo cliente no final de cada iteração é um importante fator de garantia de qualidade (MNKANDLA; DWOLATZKY, 2006).

Certos comentários críticos tais como o de Rakitin (2001), advocam que os valores que guiam métodos ágeis são a maneira de legitimizar o comportamento de somente produzir software que supostamente está funcionando, mas sem nenhuma preocupação com planejamento. Mas, é interessante verificar a garantia de que métodos ágeis de desenvolvimento de software realmente produzem software de qualidade.

Com base nisso, foi levantado o seguinte questionamento:

- Como métodos ágeis de desenvolvimento de software atingem a qualidade de

software?

Para servir de auxilio a responder a pergunta acima, as seguintes sub-perguntas foram propostas:

- Quais práticas, estratégias, ferramentas e métodos utilizados no desenvolvimento de software utilizando métodos ágeis auxiliam o atingimento da qualidade de software e em quais áreas?
- Em quais áreas são observadas lacunas nos processos ou atividades de desenvolvimento de software utilizando métodos ágeis que comprometem a qualidade de software?

Este trabalho busca identificar quais práticas, estratégias, ferramentas e métodos utilizados no desenvolvimento de software utilizando métodos ágeis auxiliam o atingimento da qualidade de software e em quais áreas. Além disso, tenta observar em quais áreas são observadas lacunas nos processos ou atividades de desenvolvimento de software utilizando métodos ágeis que comprometem a qualidade de software.

## 1.2 OBJETIVOS

Esta pesquisa tem como objetivo realizar uma revisão sistemática, para entender como métodos ágeis de desenvolvimento de software atingem a qualidade de software.

Os objetivos específicos deste trabalho consistem em investigar e analisar práticas e processos usados por métodos ágeis de desenvolvimento de software que auxiliam o atingimento da qualidade. Além disso, pretende-se investigar e examinar possíveis lacunas nesses métodos que prejudicam o alcançamento da qualidade, analisando razões dos possíveis problemas e propondo melhorias.

## 1.3 ESTRUTURA DO TRABALHO

Quanto a estrutura do trabalho, logo após a introdução, o capítulo dois mostra os métodos ágeis desenvolvidos para superar as limitações dos modelos tradicionais de desenvolvimento de software.

O capítulo três apresenta a metodologia seguida por este trabalho. Apresenta os fundamentos de uma revisão sistemática, bem como introduz o protocolo utilizado para a realização deste trabalho.

O quarto capítulo expõe o resultado da revisão sistemática. Apresenta técnicas e processos de métodos ágeis que auxiliam o atingimento da qualidade, bem como la-

cunas em alguns processos e os motivos desses problemas, além de mostrar possíveis soluções.

Por fim, o quinto capítulo, discussão e considerações finais, apresenta uma discussão sobre os achados da revisão e também limitações e trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma pesquisa exploratória sobre Métodos Ágeis de desenvolvimento de software e Qualidade de Software que serviu de apoio para este trabalho.

## 2.1 MÉTODOS ÁGEIS

Para ser considerado ágil, o método deve seguir os princípios e valores presentes no Manifesto para Desenvolvimento Ágil de Software. Entre os valores do manifesto ágil, podemos citar que são valorizados indivíduos e interações mais que processos e ferramentas, software em funcionamento mais que documentação abrangente, colaboração com o cliente mais que negociação de contratos, responder a mudanças mais que seguir um plano (HIGHSMITH et al., 2001). Isso não quer dizer que os itens menos valorizados devem ser esquecidos.

A tabela 1 mostra os 12 princípios por trás do Manifesto Ágil.

**Tabela 1 – Doze princípios por trás do Manifesto Ágil**

| | |
|---|---|
| 1 | A maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado. |
| 2 | Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente. |
| 3 | Entregar frequentemente software funcionando, de poucas semanas poucos meses, com preferência à menor escala de tempo. |
| 4 | Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto. |

| | |
|---|---|
| 5 | Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho. |
| 6 | O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face. |
| 7 | Software funcionando é a medida primária de progresso. |
| 8 | Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente. |
| 9 | Contínua atenção à excelência técnica e bom design aumenta a agilidade. |
| 10 | Simplicidade - a arte de maximizar a quantidade de trabalho não realizado - é essencial. |
| 11 | As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis. |
| 12 | Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo. |

Fonte: J. Highsmith et al. (2001)

Métodos ágeis são reconhecidos por prover satisfação do cliente, rápida resposta a mudanças e entregar produtos em pouco tempo (JEON et al., 2011). Além disso, promovem mudanças evolucionais dentro do sistema de software e possuem um conjunto de práticas que são consideradas por garantir a qualidade e controle (SFETSOS; STAMELOS, 2010).

Muitas metodologias surgiram e são hoje caracterizadas como ágeis, tais como: Kanban (ANDERSON; CARMICHAE, 2016), Scrum (SUTHERLAND; SCHWABER, 2017), Extreme Programming (BECK, 2004) e Lean (POPPENDIECK; POPPENDIECK, 2003).

Nesta seção, vamos dar uma visão geral do Scrum, Extreme Programming e Kanban.

### 2.1.1 SCRUM

Scrum é uma metodologia permite que a equipe de desenvolvimento de software opere adaptativamente dentro de um ambiente complexo debaixo de rápidas mudanças de requisitos. Scrum assume que a análise, o projeto e o desenvolvimento de software são imprevisíveis (SCHWABER, 2004). O sucesso do Scrum depende de pessoas

se tornarem mais proeficientes em serem comprometidas, corajosas, focadas, mente abertas e respeitosas (SUTHERLAND; SCHWABER, 2017).

O Scrum possui três papéis principais (ALLIANCE, 2016):

- Product Owner: Aquele que possui a visão do produto. Determina o que precisa ser feito e define a prioridade das funcionalidades de maior valor.
- Scrum Master: Aquele que ajuda o time de desenvolvimento a usar Scrum da melhor maneira possível para construir o produto.
- Time de desenvolvimento: Aqueles quem constroem o produto.

A figura 1 faz um resumo dos papéis principais do Scrum.

**Figura 1 – Papéis principais do Scrum**


Fonte: Scrum Alliance (2017)

O fluxo do Scrum começa com uma visão do sistema a ser desenvolvido. A visão pode ser vaga inicialmente, mas ficando mais clara enquanto o projeto move adiante. O Product Owner formula um plano para maximizar o Retorno de Investimento, no qual inclui o *backlog* de produto (uma lista de requisitos funcionais e não-funcionais que, quando transformados em funcionalidades, vão dar forma à visão) (SCHWABER, 2004).

O time monitora todas as atividades identificadas do backlog de produto. O backlog direciona as atividades do time. Antes de cada sprint, o time, em conjunto com o Product Owner, escolhem as atividades que serão trabalhadas e criam o backlog da sprint (atividades advindas do *backlog* de produto que serão implementadas na sprint) (SCHWABER, 2004). Cada membro da equipe concorda em completar atividades que eles acreditam que são atingíveis durante a sprint. Durante uma sprint, nenhuma mudança feita por membros fora do time é permitida (RISING; JANOFF, 2000);(SUTHERLAND; SCHWABER, 2017).

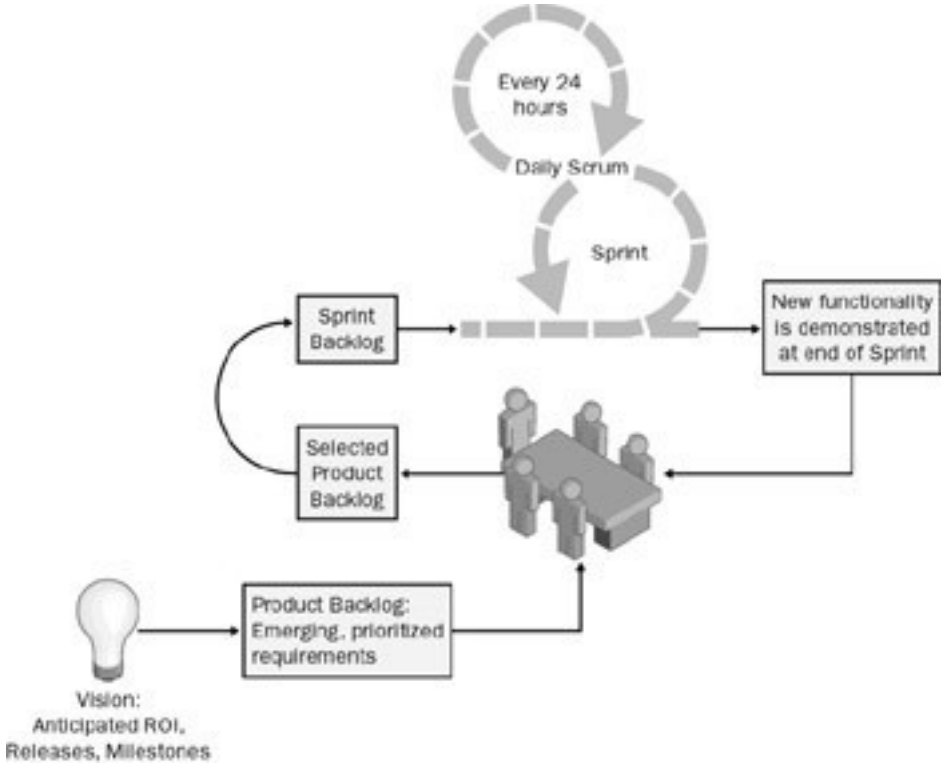
Uma sprint produz um produto visível e utilizável que implementa uma ou mais interações de usuário com o sistema. O fator chave por trás de cada sprint é entregar funcionalidades que possuam valor para o cliente. A sprint possui um tempo de duração fixo, a data final para o fim da sprint não muda. Portanto, o objetivo é completar as atividades até o fim de cada sprint (RISING; JANOFF, 2000).

Durante a sprint, as chamadas *Daily Scrums* são realizadas. *Daily Scrums* consistem em reuniões de 15 minutos realizadas todos os dias da Sprint. Essas reuniões tem o objetivo de dar visibilidade a cada um dos membros da equipe com respeito às suas atividades e resolver possíveis problemas que o desenvolvedor está tendo com sua atividade. Somente o Product Owner tem a autoridade de cancelar a sprint - caso um objetivo da sprint se torne obsoleto ou não faz mais sentido continuá-la (SUTHERLAND; SCHWABER, 2017).

No final de cada sprint, todos os membros do projetos, incluindo interessados externos e clientes, se reúnem. É a chamda revisão da sprint. Possui o objetivo de incrementar ou adaptar o backlog de produto se necessário (SUTHERLAND; SCHWABER, 2017). Toda a informação obtida na sprint é reportada. Nessa reunião, qualquer coisa pode ser modificada - funcionalidade pode ser adicionada, eliminada ou priorizada novamente (RISING; JANOFF, 2000).

A figura 2 ilustra o fluxo de processos do Scrum.

**Figura 2 – Fluxo de processos do Scrum**




Autor: Ken Schwaber (2004)

Existe também a chamada retrospectiva da sprint. É uma oportunidade dada ao time para uma auto avaliação e criar um plano para melhorias a serem feitas durante a próxima sprint. Ela ocorre após a revisão da sprint e antes do planejamento da próxima sprint (SUTHERLAND; SCHWABER, 2017).

Scrum foi projetado para ser flexível ao longo do projeto de software. Dá liberdade aos desenvolvedores de elaborar soluções criativas durante o projeto. Além disso, aumenta o compartilhamento de conhecimento entre todos os membros da equipe, o que acaba sendo um ambiente de aprendizado para todos (SCHWABER, 1994).

Como outras metodologias de desenvolvimento ágil, Scrum pode ser implementado através de uma ampla gama de ferramentas. Muitas empresas usam ferramentas de software universais, como planilhas para criar e manter artefatos como o backlog. Há também pacotes de software abertos e proprietários dedicados a gestão de produtos no âmbito do processo Scrum. Outras organizações implementam o Scrum sem o uso de ferramentas de software e mantém seus artefatos em formas impressas como papel, quadros brancos e notas adesivas (LINA; DAN, 2012).

### 2.1.2 EXTREME PROGRAMMING

Extreme Programming, ou XP, é uma maneira leve, de baixo risco, eficiente e flexível de desenvolver software. XP foi criado para projetos que podem ser construídos por times de 2 a 10 programadores. Algumas características do XP são as seguintes (BECK, 2004):

- Possui feedback inicial, concreto e contínuo de ciclos curtos;
- Possui uma abordagem de planejamento incremental, seguido de um plano geral que deverá evoluir ao longo do ciclo de vida do projeto;
- Possui capacidade de organizar de forma flexível a implementação de funcionalidades, respondendo às necessidades comerciais em constante mudança;
- Confia em testes automatizados escritos por programadores e nos clientes que acompanham o progresso do desenvolvimento para permitir a evolução do sistema e detectar defeitos antecipadamente;
- Possui uma estreita colaboração entre programadores.

Algumas práticas do XP originalmente concebidas são as seguintes (BECK, 1999):

- *planning game:* Clientes decidem o escopo e datas de entrega baseados na estimativa dada pelos desenvolvedores. Os desenvolvedores implementam somente a funcionalidade demandadas pelas histórias na atual iteração;
- pequenas versões: O sistema é colocado em produção em poucos meses, mesmo antes do sistema completo estar concluído. Novas versões são entregues com o tempo;
- uso de metáforas: O formato do sistema é definido através de metáforas ou um conjunto de metáforas compartilhados entre os clientes e os desenvolvedores;
- simplicidade de projeto: A todo o momento, todos os testes são rodados, os programadores comunicam o que precisam comunicar, nenhum código é duplicado, e o projeto possui o mínimo de classes e métodos possíveis;
- testes: Programadores escrevem testes de unidade a cada minuto. Esses testes precisam ser rodados corretamente. Clientes escrevem testes funcionais na iteração;

- refatoração: O sistema evolui através de transformações do projeto existente e mantém todos os testes passando;
- programação em pares: Todo código é escrito por dois programadores em somente uma máquina;
- integração contínua: Novos códigos são integrados no sistema existente em não mais do que algumas horas. Quando integrado, todo o sistema é construído desde o início e todos os testes são rodados e não devem produzir nenhuma falha;
- propriedade coletiva de código: Todo programador melhora o código em qualquer parte do sistema assim que for vista uma oportunidade para fazer isso;
- o cliente está sempre presente: O cliente está sempre presente e disponível para o time. Se possível, estando no mesmo ambiente; e
- 40 horas semanais: Ninguém pode fazer horas extras. Mesmo horas extras isoladas podem identificar um problema que precisa ser endereçado.

Porém, algumas práticas foram refinadas com base na experiência de usuários do XP. Algumas novas práticas incrementadas foram (ALLIANCE, 2018):

- sentar junto: o time deve sentar junto em locais sem barreiras para a comunicação;
- time completo: Um grupo de pessoas com diferentes funções com os papéis necessários para um produto formam um único time. Pessoas com necessidade e todas as outras pessoas que trabalham para satisfazer essa necessidade precisam trabalhar em conjunto diariamente para atingir um resultado específico;
- área de trabalho informativa: organizar um espaço de trabalho que facilita a comunicação face a face, permitir que pessoas tenham privacidade quando necessário e fazer com que o trabalho do time seja transparente para todos.
- trabalho energizado: significa dar passos para assegurar que o time está fisicamente e mentalmente focado. Isso significa não trabalhar demais nem deixar que outros trabalhem demais. Envolve ter um time sadio e respeitoso;
- histórias: descrever o que o produto deve ser em termos que tem significado para usuários e clientes. Essas histórias são descrições curtas de coisas que usuários desejam fazer com o produto. Podem ser usadas para planejar e servem como lembretes para conversas mais detalhadas entre membros da equipe;
- ciclo semanal: sinônimo de iteração. O time se reúne no primeiro dia de semana para refletir no progresso feito, o cliente seleciona as histórias que gostariam de

ser entregues nesta semana e o time determina como vai ser abordagem para as histórias. O alvo no final da semana é de funcionalidades funcionando e testadas que implementam as histórias selecionadas. A intenção é dar um feedback ao usuário;

- ciclo trimestral: sinônimo de entrega. O propósito é de manter um trabalho detalhado de cada ciclo semanal no contexto geral do projeto. O cliente estabelece o plano geral para o time em termos de funcionalidades desejadas dentro de um determinado trimestre, que provê ao time uma visão geral do projeto e ajuda o cliente a trabalhar com outros interessados que talvez precisem de uma ideia de quando as funcionalidades estarão disponíveis;
- sobra: a ideia é adicionar atividades de baixa prioridade nos ciclos semanais e trimestrais que podem ser excluídas caso o time atrase nas atividades ou histórias mais importantes; e
- *build* de 10 minutos: o objetivo é de construir o sistema automaticamente e rodar todos os testes em 10 minutos. Esse tempo é sugerido porque se um time tem uma *build* que dura mais de 10 minutos para ser construída, provavelmente essa construção não será feita com muita frequência, aumentando assim o tempo entre erros.

Para atingir seus objetivos, o XP é baseado em quatro valores: simplicidade, vai ser feito somente o que foi pedido para ser feito; comunicação, todos são parte do time e se comunicam diariamente; respeito, todos são considerados membros valiosos no time; e coragem, a verdade vai ser dita no que diz respeito a progresso e estimativa (BOUGROUN; ZEAARAOUI; BOUCHENTOUF, 2014).

#### 2.1.2.1 TEST DRIVEN DEVELOPMENT

Apesar de ser citado como uma metodologia ágil, TDD é encontrado dentro das práticas do XP. TDD é citado como uma prática chave do XP (BUCHAN; LI; MACDONELL, 2011). Por conta disso, esta seção foi separada para dar uma visão geral do TDD.

TDD começa com o pensamento sobre como testar um pequeno pedaço da funcionalidade escolhida para desenvolver. A primeira atividade de codificação é o planejamento e a escrita do teste unitário automático que testará se o requisito funcional é atingido. Desenvolvedores escrevem alguns testes para cada parte da funcionalidade antes de começar a codificar a funcionalidade propriamente dita. Os testes quebram o sistema e cada falha direciona a atividade de escrever um código somente o suficiente

para fazer com o que o teste que estava falhando, passe. Após isso, deve-se eliminar toda a duplicação criada no passo anterior (BECK, 2002).

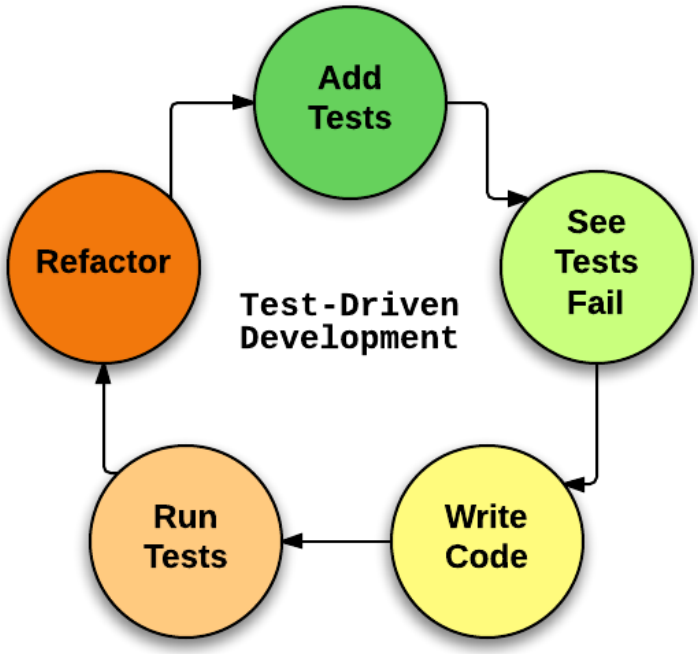
Outra característica do TDD é a natureza incremental e iterativa do processo. Testes são adicionados gradualmente durante o processo de desenvolvimento. Usando TDD, o desenvolvedor (BUCHAN; LI; MACDONELL, 2011):

1) rapidamente escreve um novo teste para uma pequena parte da funcionalidade que é parte do requisito de usuário;
2) executa todos os testes para ver o novo teste falhar;
3) faz pequenas mudanças no código de produção com o objetivo de passar o teste que estava falhando;
4) executa todos os testes novamente para ver todos os testes passarem;
5) refatora o código de produção e de testes com o objetivo de aprimorar a estrutura e remover redundâncias, se necessário;
6) integra os códigos de teste e de produção no final do dia.

Este ciclo é repetido e as funcionalidades e o projeto do produto evoluem incrementalmente (BUCHAN; LI; MACDONELL, 2011).

A figura 3 ilustra o processo do TDD.

**Figura 3 – Processo do TDD**




Fonte: Josh Davis (2013)

### 2.1.3 KANBAN

Kanban é uma palavra japonesa que significa “letreiro” (AHMAD; MARKKULA; OVIO, 2013). Kanban é um método para definir, gerenciar e melhorar serviços que entregam trabalho, tais como projetos de software. Pode ser caracterizado como um método “comece pelo o que você faz agora” - um catalisador para mudanças rápidas e focadas dentro da organização - que reduz a resistência a mudanças benéficas de acordo com os objetivos da organização (ANDERSON; CARMICHAE, 2016).

Entre os valores que estão por trás do Kanban, podemos citar transparência, equilíbrio, colaboração, foco no cliente, fluxo, liderança, entendimento, acordo e respeito (ANDERSON; CARMICHAE, 2016).

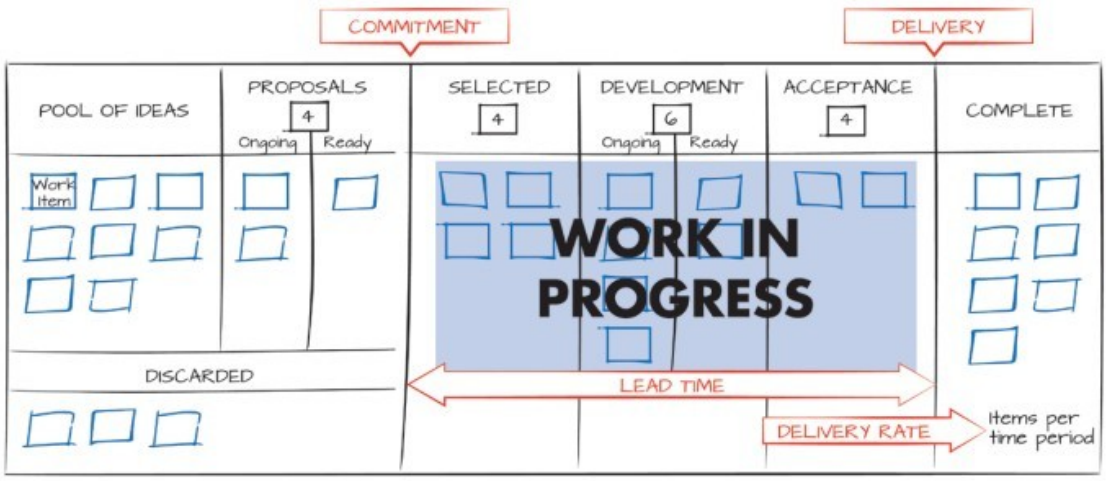
Kanban utiliza cartões e um quadro como símbolos visuais para controlar o fluxo através de um processo de produção. O objetivo é de ter um fluxo de trabalho visual, mostrando todos os passos do processo. Ações desnecessárias são eliminadas e ênfase é dada em maximizar valor para o cliente por manter um fluxo uniforme de trabalho (BOUGROUN; ZEAARAOUI; BOUCHENTOUF, 2014).

A visibilidade torna a identificação de falhas mais fácil. O trabalho em andamento ajuda a manter a velocidade necessária de produção (BOUGROUN; ZEAARAOUI; BOUCHENTOUF, 2014).

Algumas condições precisam existir para o fluxo de sistema seja um sistema Kanban. Primeiro, sinais visuais devem existir para limitar o trabalho em progresso. Sistemas Kanban também precisam identificar o compromisso (acordo entre o cliente e o time de que o cliente precisa de um item e o time irá entregar este ítem) e os pontos de entrega (local onde os ítens são considerados como finalizados). O tempo de espera é o tempo que o ítem está em processo entre o compromisso e o ponto de entrega. O ritmo no qual ítens são entregues é conhecido como taxa de entrega (ANDERSON; CARMICHAE, 2016).

A figura 4 mostra um exemplo de quadro Kanban.

**Figura 4 – Exemplo de quadro Kanban**




Fonte: David J. Anderson and Andy Carmichael (2016)

### 2.1.4 LEAN

A Toyota manufaturava carros para o Japão, mas tinham um problema: como muitas pessoas não tinham muito dinheiro, os carros tinham que ser baratos. Produção em massa era a maneira mais barata de produzir carros, mas significava produzir milhares de unidades do mesmo modelo. O mercado japonês não era grande o suficiente para precisar desses carros. A questão era, como a Toyota poderia produzir carros em pequena quantidade mas mantendo a manufatura barata como produção em massa? O Sistema de Produção Toyota nasceu para ser a base de uma maneira totalmente nova de pensar com respeito a manufatura, logística e desenvolvimento de produtos. A Honda também teve uma maneira similar de solucionar problemas no setor automotivo. Essas abordagens são tipicamente chamadas de Desenvolvimento Lean (POPPENDIECK; POPPENDIECK, 2003).

Os princípios do desenvolvimento Lean foram bem sucedidos na indústria automotiva, que possui um ambiente de projeto tão complexo quanto a maioria dos ambientes de desenvolvimento de software (POPPENDIECK; POPPENDIECK, 2003).

Poppendieck (2003) cita 7 princípios que guiam o Lean:

- eliminar desperdício: desperdício é qualquer coisa que não adiciona valor percebido pelo cliente ao produto. No pensamento Lean, o conceito de desperdício é um grande obstáculo. Se um componente não é usado, é um desperdício. Se desenvolvedores codificam mais funcionalidades do que é necessário, é desperdício. O ideal é descobrir o que o cliente deseja imediatamente;
- aprender constantemente: Assim como chefes de cozinha não conseguem fazer uma receita perfeita na primeira tentativa, eles precisam ficar sempre tentando,

o desenvolvimento de software deve ter um sistema de aprendizado similar. A melhor abordagem para melhorar um ambiente de desenvolvimento de software é ampliando o aprendizado;

- tomar decisões o mais tarde possível: práticas de desenvolvimento que possuem uma tomada de decisão tardia são efetivas em domínios que envolvem incertezas, porque eles possuem uma abordagem baseada em opções. Atrasar decisões possui valor porque melhores decisões podem ser tomadas quando elas são baseadas em fatos, não em especulações;
- entregar o mais rápido possível: sem velocidade, o atraso de decisões não é possível. Sem velocidade, um feedback confiável não é obtido. No desenvolvimento, o ciclo de descoberta é crítico para o aprendizado. Quanto mais rápido esses ciclos são, mas pode ser aprendido. Velocidade assegura que o cliente terão o que eles precisam agora, não o que eles precisavam ontem;
- motivar a todos: envolver desenvolvedores nos detalhes de decisões técnicas é fundamental para alcançar a excelência. No desenvolvimento de software Lean são feitos acordos para entregar incrementalmente versões refinadas do software em intervalos regulares. Além disso, são usados cartazes, reuniões diárias, integração contínua e testes;
- construir qualidade: um sistema é considerado tendo integridade quando um usuário pensa: é exatamente isso que eu preciso. Software precisa manter sua utilidade ao longo do tempo. Geralmente é esperado que o software evolua graciosamente com o passar do tempo. Software com integridade tem uma arquitetura coerente, possui uma boa usabilidade e propósito, é sustentável, adaptável e extensível; e
- otimizar o todo: um dos problemas presentes no desenvolvimento de produto é que especialistas em uma área têm a tendência de maximizar o desempenho da parte do produto que representa sua especialidade em vez de focar no desempenho geral do produto. Normalmente, o bem comum sofre se pessoas focarem primeiro em suas áreas de especialidade.

De modo geral, os princípios do Lean possuem uma ênfase em valor e considera toda atividade ou processo como desperdício, a não ser que adicione algum valor para a organização ou aos clientes (ALAHYARI; SVENSSON; GORSCHEK, 2016).

## 2.2 QUALIDADE DE SOFTWARE

Qualidade é definida de várias maneiras por vários autores, dependendo do contexto. Pode representar coisas diferentes, dependendo da área de conhecimento.

No que diz respeito à qualidade de software, o IEEE define o termo “qualidade” como o grau no qual um produto ou processo alcança os requisitos estabelecidos; no entanto, qualidade depende do grau em que esses requisitos representam as necessidades, desejos e expectativas dos stakeholders (SOCIETY, 2014).

Já segundo o ISO 9001:2005, qualidade é o grau no qual um conjunto de características inerentes a um objetivo cumpra seus requisitos (ISO, 2015).

O SWEBOK 3.0 (BOURQUE; FAIRLEY, 2004) cita que qualidade de software pode se referir a características desejáveis de produtos de software, na medida em que um produto de software específico possui essas características. E com respeito a processos, qualidade pode se referir a ferramentas e técnicas utilizadas para atingir essas características.

Qualidade de software também é definido como o que mede o quão bem o software foi projetado e quão bem ele está de acordo com o uso pretendido e o se está satisfazendo todas as características de qualidade e de acordo com a satisfação do usuário (DAVULURU; MEDIDA; REDDY, 2014).

Podemos separar a qualidade de software em duas áreas: qualidade de produto e qualidade de processo.

### 2.2.1 QUALIDADE DE PRODUTO

Alguns modelos foram criados com o objetivo de auxiliar o atingimento da qualidade de produto.

Um deles é o modelo de Jim McCall (MCCALL; RICHARDS; WALTERS, 1977). Este modelo é direcionado aos desenvolvedores e aos processos de desenvolvimento. Ele tenta fechar a lacuna que existe entre usuários e desenvolvedores por propos um conjunto de fatores de qualidade que refletem tanto as visões do usuário quanto as prioridades dos desenvolvedores.

No modelo de McCall, três perspectivas principais definem e identificam a qualidade do produto de software (MCCALL; RICHARDS; WALTERS, 1977):

- revisão do produto: inclui a manutenção (esforço necessário para localizar e corrigir uma falha no programa dentro do ambiente operacional), flexibilidade (a facilidade em fazer mudanças requisitadas pelo ambiente operacional) e a testabilidade (a facilidade em testar o programa para garantir que atinge sua especificação);
- transição do produto: se refere à portabilidade (o esforço necessário para transferir o programa de um ambiente para outro), reusabilidade (a facilidade em reusar o

programa em um contexto diferente) e à interoperabilidade (o esforço necessário para unir o sistema em um outro sistema); e

- operações do produto: dependa da corretude (a extensão na qual o produto atende à seus requisitos), confiabilidade (a capacidade do sistema não falhar), eficiência (está relacionada ao uso dos recursos, tais como tempo de processamento e armazenante), integridade (proteção do programa com respeito ao acesso não autoriado) e usabilidade (facilidade em usar o software).

Barry Boehm também criou um modelo de qualidade (BOEHM; BROWN; LIPOW, 1976). O modelo de Boehm tenta definir qualidade de software em um conjunto de atributos e métricas. O modelo apresenta sete fatores de qualidade que em conjunto representam as qualidades esperadas em um produto de software: portabilidade, confiabilidade, eficiência, usabilidade, testabilidade, compreensibilidade e flexibilidade.

Robert Grady também criou um modelo chamado de FURPS (ROBERT GRADY, 1992), que se baseia em:

- Funcionalidade: que pode incluir conjunto de funcionalidades, capacidades e segurança;
- Usabilidade: inclui fatores humanos, estéticos, consistência da interface com o usuário, ajuda ao usuário, documentação do usuário e matérias de treinamento;
- Confiabilidade: inclui frequência e severidade de falhas, capacidade de recuperação e eficácia;
- Performance: impõe condições a requisitos funcionais tais como velocidade, eficiência, disponibilidade, acurácia, tempo de resposta e uso de recursos;
- Suportabilidade: inclui testabilidade, adaptabilidade, manutenção, compatibilidade e localizabilidade.

O ISO 25010:2011 (ISO/IEC, 2011) define as características de qualidade de produto. Entre essas, podemos citar: adequação funcional, eficiência de desempenho, compatibilidade, usabilidade, confiabilidade, segurança, manutenção e portabilidade. Além disso, essa norma define a qualidade em uso - o grau no qual um produto ou sistema pode ser usado por usuários específicos para atender suas necessidades em atingir seus objetivos com efetividade, eficiência, liberdade de risco e satisfação em contextos específicos de uso.

### 2.2.2 QUALIDADE DE PROCESSO

Não é possível distinguir completamente a qualidade do processo da qualidade do produto porque os resultados do processo incluem produtos. O gerenciamento da qualidade do software e a qualidade do processo de engenharia de software têm uma influência direta na qualidade do produto do software. Os modelos e os critérios que avaliam as capacidades das organizações de software estão incluidos na qualidade de processo. Determinar se um processo tem a capacidade de produzir consistentemente produtos de qualidade desejada não é simples. O processo de engenharia de software influencia as características de qualidade dos produtos de software, que, por sua vez, afetam a qualidade percebida pelos stakeholders (BOURQUE; FAIRLEY, 2004).

## 2.3 MODELOS E NORMAS

Alguns modelos e normas foram criados para auxiliar no atingimento da qualidade de processo na área da engenharia de software.

### 2.3.1 CMMI

O CMMI é um modelo de referência que consiste num conjunto de boas práticas para uma ampla gama de atividades de engenharia cobrindo todo o ciclo de vida do produto de software: desde a definição dos requisitos até a entrega e manutenção. CMMI é adequado para organizações que buscam quantificar suas capacidades dentro do escopo de software (LINA; DAN, 2012). CMMI tem a finalidade de guiar o que a organização precisa fazer para melhorar seus processos e não definir os processos para a organização (CMMI, 2018).

O principal objetivo do CMMI é de reduzir o custo de implementação de melhorias nos processos por eliminar inconsistências e estabelecer diretrizes para auxiliar a organização em todos os estágios de um projeto de software. Porém, em certos projetos, CMMI pode precisar ser adaptado para facilitar o desenvolvimento, especialmente em projetos ágeis (SILVA et al., 2015).

CMMI possui duas representações: uma de estágio e outra contínua. Na representação de estágio, a maturidade de uma organização pode ir de 1 a 5. Esta representação é mais adequada para uma organização que não sabe quais processos precisam de melhoria primeiro, porque essa representação oferece áreas de processo aplicáveis para cada nível de maturidade (YOO et al., 2004). Os 5 níveis de maturidade são os seguintes (BOUGROUN; ZEAARAOUI; BOUCHENTOUF, 2014):

1) Inicial - o processo é informal e sob demanda;

2) Repetível - foco em se obter um gerenciamento de projeto básico;

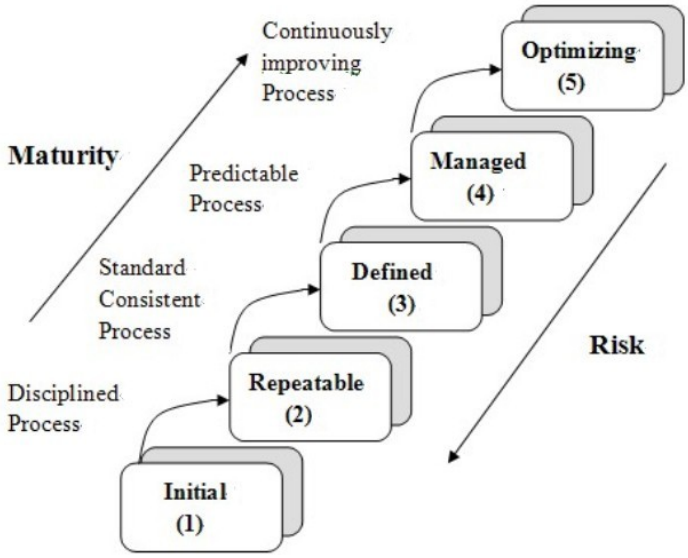
3) Definido - foco na uniformidade e consistência dos processos;

4) Gerenciado - foco na gestão quantitativa; e

5) De otimização - foco na contínua melhora dos processos.

A figura 5 ilustra os níveis do CMMI.

**Figura 5 – Os cinco níveis do CMMI**




Fonte: Zhang Lina e Shao Dan (2011)

### 2.3.2 ISO

ISO (International Organization for Standardization) é uma organização internacional que reúne especialistas em várias áreas do conhecimento, não só relacionados à computação, com o objetivo de desenvolver normas para garantir que produtos são seguros, confiáveis e de boa qualidade (ISO, 2018).

Em 1987, ISO em conjunto com o IEC (International Eletrotechnical Commision), foi criado um um comitê técnico nos quais criaram várias normas referentes à área de software e sistemas em geral (LAPORTE; APRIL; RENAULT, 2006). Entre essas normas, podemos citar:

- ISO/IEC/IEEE 12207:2017, que provê processos que podem ser empregados para definir, controlar e melhorar os processos de ciclo de vida do software dentro de uma organização ou projeto;

- ISO/IEC 90003:2014, que provê um guia para o desenvolvimento, operação e manutenção de software e serviços de suporte relacionados; e
- ISO/IEC 25010:2011, que define um modelo de qualidade em uso e qualidade de produto.

### 2.3.3 MPS.BR

No Brasil, a maioria das indústria de software são constituídas de pequenas e médias empresas. Estudos feitos no início dos anos 2000 no mercado brasileiro de desenvolvimento de software mostraram necessidade de aumentar a maturidade dos processos (VIEIRA et al., 2015).

Já que os ambientes tecnológico e de negócios vivem em constante mudança, para que se tenha sucesso na indústria de software, é importante que os empreendedores do setor coloquem foco na qualidade e eficiência dos seus processos (SOFTEX, 2018).

Esse cenário motivou a criação do programa MPS.BR (Melhoria de Processo do Software Brasileiro). O principal objetivo deste programa é de melhorar o processo de software brasileiro, com o alvo de estabelecer um caminho economicamente viável para as organizações alcançarem os benefícios da melhoria dos processos e da utilização de boas práticas da engenharia de software em um intervalo de tempo razoável. São propostos 3 modelos: MPS-SW (modelo voltado para software), MPS-SV (focado em serviços), MPS-RH (destinado à gestão de pessoas) (SOFTEX, 2016).

Falando, por exemplo, sobre o MPS-SW, esse modelo define sete níveis de maturidade dos processos para empresas de desenvolvimento de software. Com base nisso, são estabelecidos novos níveis de evolução de processos, caracterizando os estágios da melhoria na implementação do processo na organização. A capacidade do processo é representada por um conjunto de atributos descritos em termos de resultados esperados e expressa o nível de refinamento onde o processo é executado na organização (VIEIRA et al., 2015).

## 2.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou referências que servem como base conceitual e auxiliam o desenvolvimento deste trabalho. Foram apresentados definições, processos e uma visão geral de métodos ágeis, tais como Scrum, XP, Lean e Kanban e fluxos de algumas dessas metodologias. Além disso, foi mostrado conceitos de qualidade e alguns modelos tais como CMMI, MPS e normas ISO.

# 3 METODOLOGIA

Este capítulo apresenta a metodologia adotada neste trabalho. Foi realizada uma pesquisa exploratória da literatura sobre métodos ágeis de desenvolvimento de software e qualidade de software na qual serviu de motivação para ser feita uma revisão sistemática da literatura sobre o estado da arte em torno do tema. A seção 3.1 irá apresentar definições dos passos envolvidos nesta pesquisa: pesquisa exploratória e revisão sistemática. A partir da seção 3.2, o modo como esses passos foram implementados, incluindo o protocolo da revisão sistemática, serão expostos.

## 3.1 MÉTODO DE PESQUISA

### 3.1.1 PESQUISA EXPLORATÓRIA

Segundo Piovesan e Temporini (1995), um pesquisa exploratória é o estudo preliminar feito a fim de melhor adequar o instrumento de medida à realidade que se pretende conhecer. Tem por objetivo conhecer o objeto de estudo, seu significado e contexto onde ele se insere. De acordo com Gil (2008), as pesquisas exploratórias têm como principal finalidade desenvolver e elucidar conceitos, tendo em vista a formulação de problemas mais precisos ou hipóteses para estudos posteriores.

Pesquisas exploratórias geralmente envolvem levantamento bibliográfico e documental, entrevistas não padronizadas e estudos de caso (GIL, 2008).

### 3.1.2 ENGENHARIA BASEADA EM EVIDÊNCIAS

A engenharia de software baseada em evidências tem a finalidade de gerar meios que façam a integração das melhores evidências advindas das pesquisas com as experiências práticas e valores humanos no processo de tomada de decisão, levando em consideração o desenvolvimento e a manutenção de software (MAFRA; TRAVASSOS, 2006).

Estudos primários e secundários são tipos de estudos que a engenharia de software baseada em evidências faz uso para se atingir um nível adequado de evidência com respeito a uma determinada tecnologia. Quando é falado em estudos primários, entende-se estudos que tem por objetivo caracterizar uma tecnologia dentro de um contexto específico. No que se refere a estudos secundários, entende-se a realização de estudos que visem identificar, avaliar e interpretar todos os resultados relevantes a um determinado tópico de pesquisa. Revisões sistemáticas são tipos de estudos secundários (MAFRA; TRAVASSOS, 2006).

### 3.1.3 REVISÃO SISTEMÁTICA DA LITERATURA

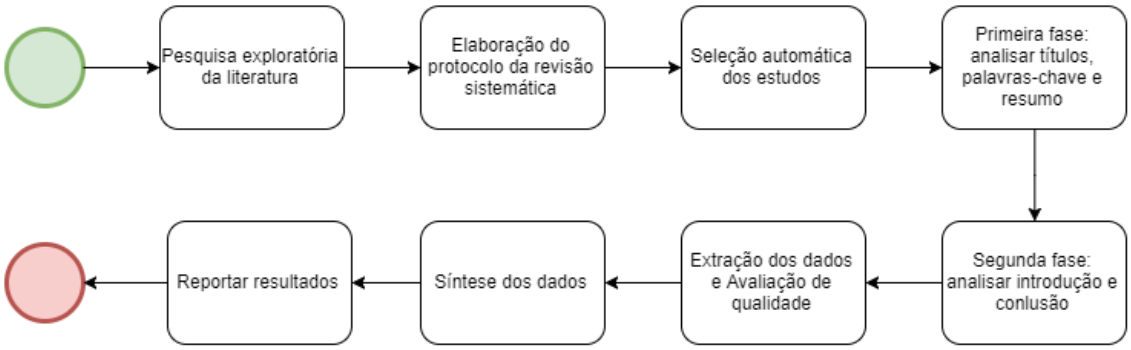
A revisão sistemática trata-se de um tipo de investigação focada em questão bem definida, que visa identificar, selecionar, avaliar e sintetizar as evidências relevantes disponíveis a um contexto específico (GALVÃO; PEREIRA, 2014). O processo de revisão sistemática deve seguir uma estrita e bem definida sequência de etapas, de acordo com um previamente definido. Os passos, as estratégias para conseguir as informações e o foco das questões são explicitamente definidas, assim outros pesquisadores podem reproduzir o mesmo protocolo e serem capazes de julgar o adequação dos padrões escolhidos (BIOLCHINI et al., 2005).

A revisão sistemática da literatura envolve algumas atividades definidas, que podem ser resumidas em: planejar a revisão, na qual envolve identificar a necessidade de uma revisão; conduzir a revisão, que inclui selecionar e qualificar os estudos; e reportar a revisão (KITCHENHAM, 2007).

## 3.2 METODOLOGIA DE DESENVOLVIMENTO

A figura 6 apresenta as etapas da metodologia exercida neste trabalho.

**Figura 6 – Metodologia adotada no trabalho**




Fonte: Autor

O primeiro passo deste trabalho foi realizar uma pesquisa exploratória da literatura. Pesquisas exploratórias tem o objetivo de conhecer com maior profundidade um assunto, de modo a torná-lo mais claro ou elaborar questões importantes para a condução da pesquisa (GIL, 2008). Foi feito um levantamento bibliográfico com o objetivo de ter uma visão abrangente a respeito de métodos ágeis e qualidade de software e também serviu como fundamentação teórica deste trabalho.

Com base nos resultados da pesquisa exploratória, não foi achado nenhum artigo onde a qualidade de software fosse analisada de uma maneira consolidada dentro

de métodos ágeis. Portanto, foi vista a necessidade de uma revisão sistemática, com o objetivo de analisar práticas presentes em métodos ágeis que auxiliam o atingimento da qualidade de software.

## 3.3 PROCESSO DE PESQUISA EXPLORATÓRIA

Como essas pesquisas possui uma menor rigidez no planejamento, a forma de coleta de dados é bastante flexível (GIL, 2008).

Durante a pesquisa foi realizada uma busca *ad hoc* por artigos e estudos sobre o tema trabalho. O Google e o Google Scholar foram utilizados como ferramentas e através deles foi possível acessar trabalhos disponíveis em diversos sites. Para a pesquisa, as seguintes palavras-chave foram utilizadas: extreme programming, xp, scrum, agility, agile methods, quality, quality assurance, quality in agile methods, lean, kanban, agile quality, entre outras combinações e variações de idioma.

Assim, foi obtida uma visão abrangente a respeito de métodos ágeis e qualidade de software e na qual serviu como base para a revisão sistemática da literatura a respeito desse tema.

## 3.4 PROCESSO DE REVISÃO SISTEMÁTICA

Essa seção irá mostrar os passos da metodologia utilizada para realizar esta revisão sistemática. Este trabalho inicialmente com a participação de dois pesquisadores (Marco Segundo e Dennys Barros), mas essa revisão foi concluída somente por Dennys Barros. Antes de iniciar a revisão sistemática, um protocolo foi desenvolvido. Esse protocolo contém os objetivos da revisão, fontes de busca, critérios de inclusão e exclusão, e demais assuntos que dão suporte à revisão sistemática (ver Apêndice A).

Para auxiliar o processo de revisão foi utilizado o software StArt (DC/UFSCAR, 2013), que é um programa voltado especificamente para revisões sistemáticas. Com ele é possível aceitar ou rejeitar artigos, ler resumos diretamente do software, gerar tabelas e gráficos.

### 3.4.1 QUESTÃO DE PESQUISA

Uma questão de pesquisa foi criada para guiar a revisão e dividida em duas sub-perguntas.

Como métodos ágeis de desenvolvimento de software atingem a qualidade de software?

- Sub-pergunta 1: Quais práticas, estratégias, ferramentas e métodos utilizados no

desenvolvimento de software utilizando métodos ágeis auxiliam o atingimento da qualidade de software e em quais áreas?

- Sub-pergunta 2: Em quais áreas são observadas lacunas nos processos ou atividades de desenvolvimento de software utilizando métodos ágeis que comprometem a qualidade de software?

A sub-pergunta 1 foi criada com o objetivo de investigar práticas, estratégias e métodos utilizados em metodologias ágeis de desenvolvimento que contribuem para a qualidade do software. A sub-pergunta 2 foi desenvolvida com a finalidade de investigar quais lacunas existem nos processos e atividades de metodologias ágeis.

### 3.4.2 TERMOS CHAVE DA PESQUISA

A partir das perguntas formuladas na seção anterior, os principais termos foram identificados. Como as bases de dados pesquisadas possuem a maioria dos resultados na língua inglesa, os termos usados foram traduzidos para o inglês.

Além disso, sinônimos são identificados e os termos chaves são pesquisados no plural e no singular (o caractere asterisco é aceito na maioria das bibliotecas virtuais e permite essa variação singular/plural). Os termos e sinônimos são descritos abaixo:

- Agile method;
- Scrum;
- Extreme Programming;
- XP;
- Lean software;
- TDD
- Quality
- Quality assurance;
- Agile quality; e
- QA.

### 3.4.3 STRINGS DE BUSCA

Kitchenham (2007) afirma que as strings de busca são derivadas das estruturas das questões e as vezes adaptações são necessárias de acordo com as necessidades especificas de cada base de dados. As strings de busca foram geradas a partir da combinação dos termos chaves e sinônimos usando AND (e) e OR (ou), e possíveis mudanças de acordo com o engenho de busca. A string completa é a seguinte:

((“agile method*” OR “scrum” OR “extreme programming” OR “XP” OR “lean software” OR “TDD”) AND (“quality” OR “quality assurance” OR “agile quality” OR “qa”))

### 3.4.4 ENGENHOS DE BUSCA

As pesquisas iniciais dos estudos primários podem ser realizadas em bibliotecas digitais, mas pesquisadores da área de pesquisa também podem ser consultados para a indicação de fontes de material mais adequado, além de listas de referências e a internet (KITCHENHAM, 2007). As fontes de pesquisa utilizadas para a busca dos estudos são listadas a seguir:

- IEEExplore[1]
- ACM[2]
- Elsevier[3]

Outras fontes foram inicialmente consideradas, tais como Google Scholar e SpringerLink. Em uma futura pesquisa os engenhos citados serão acrescidos.

Todos os resultados foram obtidos das fontes de busca selecionadas. Foram identificadas 1335 publicações. Em seguida foi extraído o arquivo Bibtex de cada um dos artigos, e esses arquivos foram adicionados no software Start . Com isso, os artigos foram organizados em formas de tabelas. Assim, a fase de seleção dos estudos foi iniciada.

### 3.4.5 SELEÇÃO DOS ESTUDOS

Os estudos que fazem parte dessa pesquisa são: artigos de periódicos, revistas, conferências e congressos. Uma vez que estudos potencialmente candidatos a se tornarem estudos primários tenham sido obtidos, eles precisam ser analisados para que a sua relevância seja confirmada e trabalhos com pouca relevância sejam descartados

[1] http://ieeexplore.ieee.org
[2] http://portal.acm.org/dl.cfm
[3] http://www.sciencedirect.com

(KITCHENHAM, 2007). Foram escolhidos apenas artigos da década atual com o objetivo de analisar os métodos ágeis com mais maturidade, contemplando assim necessidades, estratégias e práticas mais atuais.

A fase de seleção de dados foi feita em conjunto com o pesquisador Marco Segundo. Os critérios de inclusão foram os seguintes:

1) artigos que respondam à questão de pesquisa;
2) estudos que apresentem primária ou secundariamente práticas, estratégias, técnicas ou métodos presentes no desenvolvimento de software utilizando métodos ágeis que auxiliem no atingimento da qualidade de software;
3) estudos que apresentem avaliação de qualidade de métodos ágeis de desenvolvimento de software; e
4) estudos que apresentem primária ou secundariamente lacunas ou práticas deficientes no processo de desenvolvimento de software utilizando métodos ágeis, nas quais após preenchidas, melhoraria a qualidade de software.

Os critérios de exclusão são descritos abaixo:

1) estudos claramente irrelevantes à pesquisa, de acordo com as questões de investigação levantadas;
2) estudos que não respondam à nenhuma das questões de pesquisa;
3) estudos duplicados;
4) artigos que não estejam escritos na língua inglesa;
5) artigos que não estejam no período de Janeiro de 2010 a Outubro de 2017;
6) estudos que não estejam livremente disponíveis para a consulta na web;
7) artigos de livros e resumos; e
8) artigos incompletos.

O processo foi dividido em duas fases. A primeira fase selecionou os artigos que possivelmente satisfaziam os critérios de inclusão com base na leitura do título, resumo e palavras chaves. Os estudos selecionados por cada um dos pesquisadores nesta fase foram organizados em tabelas de artigos aceitos no StArt (DC/UFSCAR, 2013). Cada pesquisador possuía diferentes resultados.

Para reduzir o risco de inclinação tendenciosa, as tabelas de cada pesquisador foram comparadas entre eles, onde se discutiram os resultados divergentes. Um consenso foi alcançado, e os artigos que passariam para a segunda fase foram definidos.

Na segunda fase, a leitura da introdução e da conclusão foi feita e os mesmos critérios de inclusão e exclusão observados na primeira fase foram utilizados para avaliar os artigos. Após isso, uma avaliação de qualidade dos artigos selecionados foi feita.

### 3.4.6 AVALIAÇÃO DE QUALIDADE

Em adição aos critérios gerais de inclusão e exclusão, é considerado crítico avaliar a qualidade dos estudos primários. Apesar de não existir uma definição universal do que seja qualidade de estudo, esta análise visa minimizar viés e maximizar validade interna e externa. Validade interna é a extensão na qual o projeto e a condução do estudo evita o viés. Validade externa é a extensão na qual os efeitos observados no estudo são aplicáveis externamente (KITCHENHAM, 2007).

Foi avaliada a qualidade de cada publicação. A garantia de qualidade é definida conforme o rigor, credibilidade e relevância os artigos. Os artigos foram selecionados com base nas seguintes perguntas:

1) O artigo expõe claramente seus objetivos e resultados?
2) O artigo expõe os métodos utilizados para a análise dos dados?
3) As conclusões são explícitas?
4) O artigo contribui ou responde parcialmente às perguntas de pesquisa?

Para cada pergunta, uma resposta “SIM” ou “NÃO” era dada a cada artigo. Se o artigo em análise responde “SIM” a todas as perguntas, ele era considerado relevante para a revisão. Com base nessas perguntas, 3 artigos foram excluídos, restando assim 37 artigos que foram para a fase de extração dos dados. Os artigos excluídos foram os seguintes:

1) QA to AQ Part Two: Shifting from Quality Assurance to Agile Quality: “Measuring and Monitoring Quality” (YODER; WIRFS-BROCK, 2014);
2) QA to AQ Part Four: Shifting from Quality Assurance to Agile Quality: “Prioritizing Qualities and Making Them Visible” (YODER; WIRFS-BROCK; WASHIZAKI, 2015);

3) The Theory of Relative Dependency: Higher Coupling Concentration in Smaller Modules (KORU; EMAM, 2010);

Os artigos de Yoder, Wirfs-Brock e Washizaki, apesar de mostrarem várias definições e questões relacionadas à qualidade em métodos ágeis, não fizeram nenhuma validação das informações em projetos reais, o que acaba não mostrando se os dados realmente trazem resultados positivos ou não. O artigo de Koru e El Emam apresenta um estudo no qual afirma que módulos menores possui um maior acoplamento (dependência de outros módulos) se comparado com módulos maiores. O artigo cita como a refatoração afeta o acoplamento, mas acaba não sendo muito claro com respeito à qualidade.

### 3.4.7 EXTRAÇÃO DOS DADOS

O objetivo desta etapa é criar formas de extração dos dados para registrar com precisão as informações obtidas a partir dos estudos primários (KITCHENHAM, 2007). Para apoiar a extração e registro dos dados e posterior análise, será utilizada a ferramenta StArt (DC/UFSCAR, 2013), que é voltada especificamente para revisões sistemáticas.

Na fase de extração dos dados, foi feita a leitura completa dos artigos selecionados para a obtenção das informações. Cada frase que responde uma ou duas questões é considerada uma quota. Foram obtidas 131 quotas de 37 artigos.

### 3.4.8 ANÁLISE E SÍNTESE DOS DADOS

A síntese dos dados envolve agrupar e sumarizar os resultados dos resultados primários incluídos. Tabelas devem ser estruturadas de forma a destacar as semelhanças e diferenças entre os resultados do estudo (KITCHENHAM, 2007).

Acabada a etapa de extração, foi feita uma análise e síntese dos dados para gerar a combinação das quotas com as perguntas de pesquisa. A ferramenta Google Sheets em conjunto com a ferramenta StArt foram utilizadas para auxiliar neste processo.

## 3.5 CONSIDERAÇÕES FINAIS

O objetivo deste capítulo foi apresentar a metodologia utilizada neste trabalho. Foram mostradas as definições de uma pesquisa exploratória, bem como de uma revisão sistemática. A importância de seguir um protocolo bem definido para a realização de uma revisão foi apresentada.

O protocolo (ver Apêndice A) que guia este trabalho foi feito inicialmente em conjunto com o pesquisador Marco Segundo e seus pontos essenciais foram apresentados: strings e engenhos de busca, critérios de inclusão e exclusão, avaliação de qualidade, extração e síntese dos dados.

# 4 RESULTADOS DA REVISÃO SISTEMÁTICA

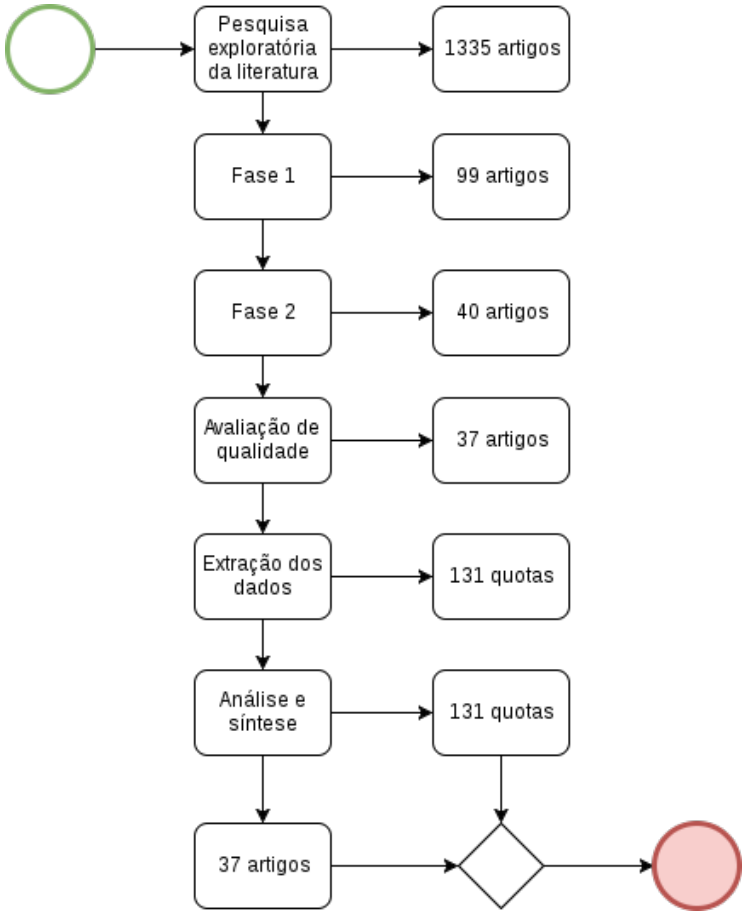
Esta seção discorre sobre a análise dos dados extraídos pelos estudos selecionados. O processo de revisão sistemática foi dividido em três etapas: Seleção de dados, Extração de dados e Avaliação de qualidade, e Síntese dos dados.

## 4.1 DADOS DE PESQUISA

Nesta fase, os dados de pesquisa foram extraídos de três fontes de busca. Foram encontrados 1335 estudos entre os anos de 2010 à 2017. Esse período foi escolhido com o objetivo de identificar os estudos feitos na década atual.

A figura 7 mostra os resultados obtidos em cada estágio do processo de revisão sistemática.

**Figura 7 – Resultados obtidos em cada etapa do processo de revisão sistemática**




Fonte: Autor

## 4.2 SELEÇÃO DOS DADOS

A seleção foi dividida em duas fases:

- Fase 1: leitura do título, palavras chaves e resumo;
- Fase 2: análise da introdução e da conclusão.

Inicialmente, 1335 artigos foram encontrados de forma automática. Na fase 1, depois de os títulos, resumos e palavras chave terem sido analisados, 99 estudos foram selecionados para a próxima fase. Um total de 1236 artigos foram excluídos.

Na fase 2, depois de ler a introdução e conclusão, foram selecionados 40 artigos para a fase de extração de dados. Na segunda fase, foram eliminados 59 artigos e 40 foram selecionados para uma tabela final de estudos e posteriormente foram lidos em sua inteireza na fase de extração dos dados. Entre os critérios utilizados, foi observado que 0% dos estudos não estavam livremente para consulta na web; 3% não estavam escritos na língua inglesa; 20% eram claramente irrelevantes à pesquisa, 77% dos estudos não responderam à nenhuma das questões de pesquisa. A tabela 2 mostra a lista de *engines* e a quantidade de artigos selecionados em cada uma das etapas.

**Tabela 2 – Lista de *engines* e suas contribuições**

| ***Engine*** | **Seleção automática** | **Primeira fase** | **Segunda fase** | **Extração** |
|---|---|---|---|---|
| IEEE | 328 | 41 | 18 | 18 |
| ACM | 340 | 45 | 17 | 14 |
| Science Direct | 667 | 13 | 5 | 5 |
| **Total** | 1335 | 99 | 40 | 37 |

Fonte: autor

## 4.3 AVALIAÇÃO DE QUALIDADE

A avaliação de qualidade foi realizada e apenas três artigos foram excluídos. A tabela 3 contém os artigos presentes da fase de extração de dados:

**Tabela 3 – Lista de artigos presentes na fase de extração de dados**

| ID | Título |
|---|---|
| 01 | The projection of the specific pratices of the third level of CMMI model in agile methods: Scrum, XP and Kanban |
| 02 | The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis |
| 03 | Adaptation of the Scrum Adherent to the level G of the MPS.BR based on the experience of the implementers, evaluators and evaluated companies |
| 04 | Dynamic quality control in agile methodology for improving the quality |
| 05 | Test-Driven Development - still a promising appoach? |
| 06 | Most common mistakes in Test-Driven Development practice: results from an online survey with developers |
| 07 | Quality of testing in Test Driven Development |
| 08 | Applying usability testing to improve Scrum methodology in develop assistant information system |
| 09 | Software quality assurance in Scrum: The need for concrete guidance on SQA strategies in meeting user expectations |
| 10 | Investigation of the capability of XP to support the requirements of ISO 9001 software process certification |
| 11 | Effectiveness of Test-Driven Development and Continuous Integration - a case study |
| 12 | Towards a Secure Agile Software Development Process |
| 13 | Quality Attribute Driven Agile Development |

| ID | Título |
|---|---|
| 14 | Casual factors, benefits and challenges of Test-Driven Development - practitioner perceptions |
| 15 | Integrating testing into Agile software development processes |
| 16 | Quality assurance in agile: a study towards achieving excellence |
| 17 | Impact and challenges of requirements elicitation prioritization in quality to agile process: Scrum as a case scenario |
| 18 | Comparative analysis of agile methods and iterative enhancement model in assessment of software maintenance |
| 19 | An industry experience report on managing product quality requirements in a large organization |
| 20 | Systematic analyses and comparison of development performance and product quality of Incremental process and Agile process |
| 21 | The effects of test-driven development on internal quality, external quality and productivity: A systematic review |
| 22 | Using CMMI together with agile software development: A systematic review |
| 23 | A compliance analysis of Agile methodologies with ISO/IEC 29110 project management process |
| 24 | Evaluating the effect of agile methods on software defect data and defect reporting practices - a case study |
| 25 | A comparison of security assurance support of Agile Software Development methods |
| 26 | Transition from a plan-driven process to Scrum: A longitudinal case study on software quality |
| 27 | Empirical studies on quality in Agile practices: A systematic literature review |
| 28 | ScrumS: A model for safe agile development |
| 29 | A study to support agile methods more effectively through traceability |

| ID | Título |
|---|---|
| 30 | Procedural assessment process of software quality models using agility |
| 31 | The effectiveness of Test-Driven Development: an industrial case study |
| 32 | Research on combining Scrum with CMMI in small and medium organizations |
| 33 | Using assurance cases to develop iteratively security features using Scrum |
| 34 | Usability evaluation practices with agile development |
| 35 | The usage of usability techniques in Scrum projects |
| 36 | Pair Programming and Software Defects - A large, industrial case study |
| 37 | Busting a myth: review of agile security engineering methods |

Fonte: Autor

Os artigos excluídos foram os seguintes:

**Tabela 4 – Lista de artigos excluídos após a avaliação de qualidade**

| ID | Título |
|---|---|
| 1 | QA to AQ Part Two: Shifting from Quality Assurance to Agile Quality: “Measuring and Monitoring Quality” |
| 2 | QA to AQ Part Four: Shifting from Quality Assurance to Agile Quality: “Prioritizing Qualities and Making Them Visible” |
| 3 | The Theory of Relative Dependency: Higher Coupling Concentration in Smaller Modules |

Autor

Os artigos 1 e 2 apesar de mostrarem várias definições e questões relacionadas à qualidade em métodos ágeis, não fizeram nenhuma validação das informações em projetos reais, o que acaba não mostrando se os dados realmente trazem resultados positivos ou não. O artigo 3 apresenta um estudo no qual afirma que módulos menores possui um maior acoplamento (dependência de outros módulos) se comparado com módulos maiores. O artigo cita como a refatoração afeta o acoplamento, mas acaba não sendo muito claro com respeito à qualidade.

## 4.4 EXTRAÇÃO, ANÁLISE E SÍNTESE DOS DADOS

A partir da lista dos estudos selecionados, foi iniciado o processo de extração de dados. O processo consiste nos pontos abaixo:

1) Todos os artigos selecionados foram lidos na íntegra para a extração dos dados e avaliação de qualidade;
2) Para cada artigo incluído, seus dados foram extraídos através das quotas. As quotas eram registradas numa planilha do Google Sheets.

Nesta fase, 131 quotas foram analisadas, onde 62 responderam à primeira questão de pesquisa e 69 responderam à segunda.

As quotas foram organizadas em grupos e por relevância quanto às perguntas de pesquisa. Os grupos de quotas foram: “segurança” (se referindo a todas as quotas que citavam técnicas de segurança de software usadas em métodos ágeis ou lacunas); “TDD” (quotas que mostravam benefícios e problemas com respeito ao TDD); “MPS.BR” (quotas que analisavam conformidade de métodos ágeis com o MPS.BR); “CMMI” (quotas que analisavam conformidade de métodos ágeis com o CMMI); “ISO” (quotas que analisavam conformidade de métodos ágeis com o ISO); “programação em pares” (quotas relacionadas à vantagens e problemas da programação em pares); “elicitação de requisitos” (quotas mostrando técnicas de qualidade utilizadas para elicitar requisitos); “usabilidade” (problemas de usablidade em métodos ágeis); e “rastreabilidade” (problemas de rastreabilidade em métodos ágeis. Um grupo chamado de “outros” incluía quotas que não estavam relacionados a nenhum dos grupos citados anteriormente.

Um exemplo de quota que responde à primeira sub-pergunta de pesquisa (Quais práticas, estratégias, ferramentas e métodos utilizados no desenvolvimento de software utilizando métodos ágeis auxiliam o atingimento da qualidade de software e em quais áreas?) com respeito ao TDD, é observada no artigo de Buchan, Li e MacDonell (2011):

> Participants were unanimous in their perception that the use of TDD improved the quality of code compared to their experiences with traditional TL software development. They claimed that the use of TDD encouraged the development of simple, clean and meaningful code, more so than TL encouraged these outcomes. There was the perception that the discipline of following TDD would naturally develop habits that lead to better code as part of developers’ everyday practice. The implication was that it was easier to be “lazy” and get away with developing messy or untested code using a TL approach. TDD was also viewed as guarding against the pressure of management to deliver code more quickly and compromising code quality, since this was not an option with TDD.

Já um exemplo de quota que responde à segunda sub-pergunta de pesquisa (Em quais áreas são observadas lacunas nos processos ou atividades de desenvolvimento

de software utilizando métodos ágeis que comprometem a qualidade de software?) com respeito à segurança de software, é observada no artigo de Rindell, Hyrynsalmi e Leppanen (2017):

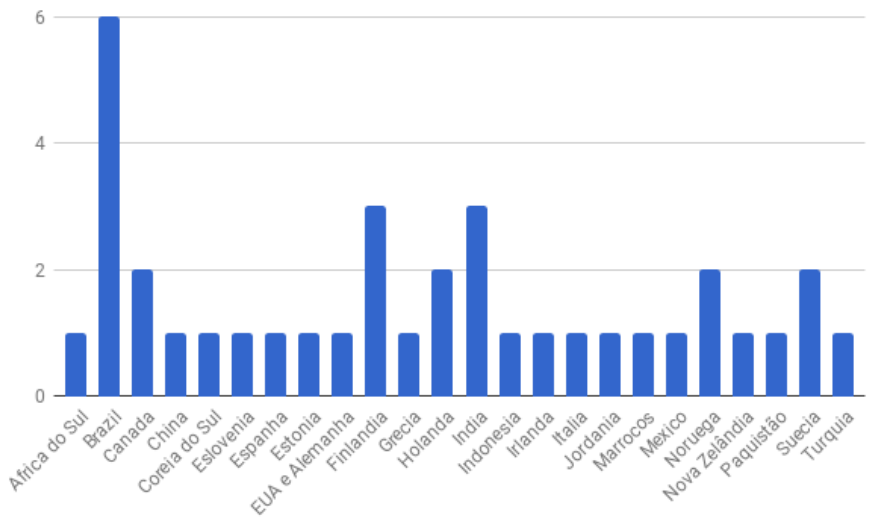
> Certain security activities are noted to be, if not incompatible with XP, at least missing from it: need for security engineer(s), static code review, security policies, formal security reviews and security documentation are mentioned. The suggested solution is as simple as one would assume: integrate the security engineering activities into the agile method

A distribuição geográfica dos estudos selecionados para a fase de síntese dos dados foi a seguinte:

- Brasil com 6 artigos;
- Finlândia e Índia com 3 artigos cada;
- Canadá, Holanda, Suécia e Noruega com 2 artigos cada;
- África do Sul, China, Coreia do Sul, Eslovênia, Espanha, Estônia, Estados Unidos, Grécia, Indonésia, Irlanda, Itália, Jordânia, Marrocos, México, Nova Zelândia, Paquistão e Turquia com 1 artigo cada.

A figura 8 apresenta a distribuição de estudos por país. O fato do Brasil ter sido o país com mais artigos nessa pesquisa pode ter sido devido ao fato do país estar querendo identificar melhorias no processo de métodos ágeis. É digno de nota que a Índia também aparece com 3 artigos. A Índia é um país que também está crescendo na área de tecnologia e está aparecendo como um local que aparenta buscar ter mais qualidade em seus produtos e processos.

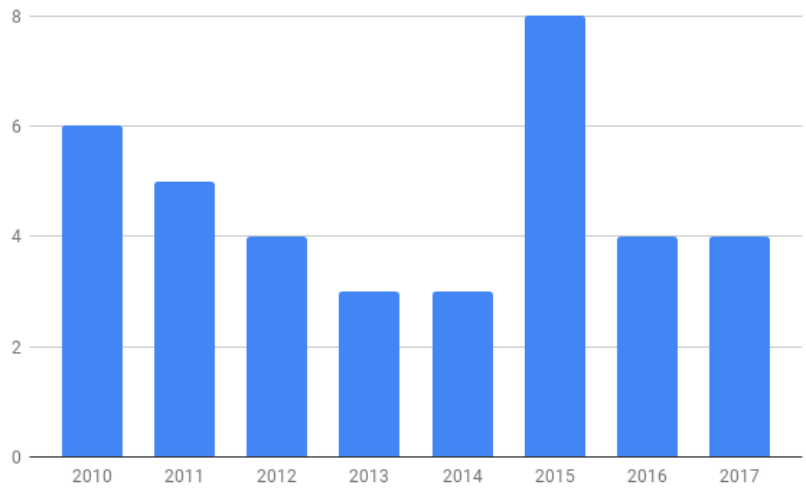
**Figura 8 – Distribuição de estudos por país**




Fonte: autor

A figura 9 mostra a distribuição de estudos selecionados através de seu ano de publicação. Foi percebido que o ano de 2015 teve o maior número de publicações incluídas neste trabalho. O ano de 2010 é um ano que mostra aumento de estudos provavelmente pelo marco de 10 anos da criaçao do Manifesto Agil (MELO et al., 2012). Apesar do número ter caído nos anos de 2016 e 2017, qualidade e métodos ágeis em conjunto com qualidade continuam a ser estudados pelos pesquisadores.

**Figura 9 – Distribuição dos artigos por ano**




Fonte: Autor

Os artigos selecionados tiveram o maior foco em Scrum e XP. Esses dois métodos foram amplamente citados nos artigos. As práticas mais estudadas entre os artigos selecionados foram TDD e programação em pares, enquanto CMMI foi o modelo mais citado.

A tabela 4 mostra as práticas e modelos mais citados nos artigos selecionados.

**Tabela 5 – Práticas e modelos mais citados nos artigos selecionados**

| **Práticas e Modelos** | **Artigos** |
|---|---|
| TDD | 12 |
| Programação em pares | 9 |

| **Práticas e Modelos** | **Artigos** |
|---|---|
| CMMI | 3 |
| ISO | 3 |

Fonte: Autor

As próximas seções vão apresentar os achados da revisão sistemática. A seção 4.5 mostra as práticas presentes em métodos ágeis que auxiliam o atingimento da qualidade, a conformidade desses métodos em relação a modelos e normas de qualidade, relação de métodos agéis com usabilidade e expectativas dos usuários e aspectos gerais de qualidade atingidos por outras práticas. A seção 4.6 apresenta problemas e lacunas de métodos ágeis que podem prejudicar a qualidade.

A tabela 5 dá uma visão das práticas e lacunas presentes em métodos ágeis em relação à qualidade.

**Tabela 6 – Práticas e problemas encontrados em métodos ágeis em relação à qualidade**

| **Práticas citadas que contribuem para a qualidade** | **Problemas citados que podem minar a qualidade** |
|---|---|
| TDD | |
| programação em pares | |
| reuniões diárias | problemas em usabilidade |
| presença e feedback constante do cliente | métodos ágeis por si só não se adéquam completamente a modelos e normas |
| ciclos de desenvolvimento de curta duração | lacunas em segurança de software |
| desenvolvimento de protótipos de alta e baixa fidelidade | dificuldade na rastreabilidade e priorização de requisitos |
| propriedade coletiva de código | |
| entregas constantes | |

Fonte: Autor

## 4.5 COMO MÉTODOS ÁGEIS ATINGEM A QUALIDADE

Esta seção apresenta as práticas presentes em métodos ágeis que auxiliam o atingimento da qualidade, a conformidade desses métodos em relação a modelos e normas de qualidade, relação de métodos agéis com usabilidade e expectativas dos usuários e aspectos gerais de qualidade atingidos por outras práticas.

### 4.5.1 PRÁTICAS MAIS CITADAS PELOS ARTIGOS QUE AUXILIAM O ATINGIMENTO DA QUALIDADE EM MÉTODOS ÁGEIS

Os métodos que mais foram citados pelos artigos encontrados são a programação em pares e o TDD. A programação em pares foi citada em 9 artigos enquanto o TDD foi estudado em 12 artigos.

#### 4.5.1.1 PROGRAMAÇÃO EM PARES

Programação em pares é a prática na qual dois desenvolvedores trabalham juntos na mesma atividade utilizando somente um computador. Enquanto um desenvolvedor codifica, o outro se concentra em atividades mais estratégicas, tais como revisão de código, encontrar melhores soluções ou brainstorming (BELLA et al., 2013). Faz parte das técnicas do XP (WELLS, 1999).

A eficácia da Programação em Pares (PP) foi investigada no artigo do autor di Bella no contexto de correções de defeitos e implementação de histórias de usuário. A análise mostrou que a introdução de novos defeitos tende a diminuir quando PP é praticado. PP aparentemente provê uma perceptível, mas pequeno efeito na redução de defeitos. Os resultados obtidos mostram que PP ajuda a reduzir a densidade de defeitos (defeitos por linhas de código) (BELLA et al., 2013).

Programação em pares é também citado por Sfetsos e Stamelos (2010) como responsável por aumentar a qualidade do código. Adicionalmente, melhora a qualidade do trabalho em equipe, melhora a comunicação entre membros da equipe, melhora o entendimento e o compartilhamento do conhecimento. Programação em pares, quando combinada com TDD, se torna uma prática chave para se atingir a qualidade.

#### 4.5.1.2 TDD

Proponentes de TDD afirmam que essa prática resulta em melhorias na qualidade de código, qualidade de testes, qualidade da aplicação se comparado com técnicas tradicionais de testes. Além disso, eles afirmam que o uso de TDD aumenta a produtividade geral de desenvolvimento, auxilia no entendimento do escopo do projeto e provê satisfação ao desenvolvedor com respeito ao trabalho e à confiança (BUCHAN; LI; MACDONELL, 2011).

A análise feita por Yahya Rafique e Vojislav B. Misic sugere que TDD resulta numa pequena melhora na qualidade mas obtiveram resultados inconclusivos no que se refere à produtividade. Os benefícios advindos da adoção do TDD pode não ser imediatamente observados e podem requerer um pouco de tempo antes de progressivamente serem visíveis (RAFIQUE; MISIC, 2013).

A pesquisa feita por Sami Kollanus, aponta que com respeito à qualidade externa (número de defeitos encontrados antes da entrega, ou defeitos encontrados pelos clientes), as evidências provêm pouco suporte com respeito à qualidade externa quando TDD é usado. Segundo ele, vários estudos de caso apontam para uma melhor qualidade externa, porém, quando são feitos experimentos controlados, os resultados acabam sendo contraditórios, deixando algumas dúvidas sobre os fatores de melhoria. Relacionado à qualidade interna (cobertura de testes, número de casos de teste, complexidade ciclomática, coesão) também não há evidência de melhoria utilizando TDD. Com relação à produtividade (esforço total de desenvolvimento, linhas de código por hora, número de user stories implementados), os artigos sugerem que TDD pode aumentar o esforço de desenvolvimento. Em resumo, ele afirma que há poucas evidências de melhor qualidade externa e interna utilizando TDD. E há uma evidência moderada de declínio de produtividade utilizando TDD (KOLLANUS, 2010).

Já de acordo com Aniche e Gerosa, TDD aumenta a qualidade de código, reduz a densidade de defeitos e provê uma melhor manutenção do código. Porém, todas essas vantagens de TDD podem ser aumentadas caso os desenvolvedores cometam menos erros durante o processo (ANICHE; GEROSA, 2010).

Causevic, Punekka e Sundmark (2012) fizeram uma comparação com respeito à qualidade dos testes e de código entre equipes que praticam TDD e equipes que testam depois do software pronto. Foi visto que a qualidade dos casos de testes entre as duas equipes era quase inexistente. Porém, foi visto que o código dos que praticam TDD tinha mais qualidade dos que o que não praticam.

Buchan, Li e MacDonell fizeram um estudo no qual eles buscam obter uma percepção dos que praticam TDD na indústria . Para obter esses dados, eles fizeram entrevistas com 5 integrantes chave de uma empresa de desenvolvimento de software. Quando perguntados sobre quais, na opinião deles, eram os principais benefícios em utilizar TDD, as respostas deles incluíram (BUCHAN; LI; MACDONELL, 2011):

- ao longo do processo de desenvolvimento os engenheiros tem um pensamento mais amplo;
- código de maior qualidade - o código faz o que deveria fazer;
- cobertura de testes aumentada;
- confiança na qualidade do software entregue ao cliente;
- mais confiança do desenvolvedor e do cliente de que o código irá fazer o que foi dito que ele iria fazer.

Os participantes foram unânimes ao dizer que o uso de TDD melhora a qualidade do software se comparada com técnicas tradicionais de testes. Eles defendem que o uso de TDD encoraja o desenvolvimento de código simples, limpo e significativo. A implicação foi de que é mais fácil ser “preguiçoso” e deixar um código bagunçado ou sem testar se técnicas tradicionais de desenvolvimento forem aplicadas. Além disso, foi visto que TDD é uma maneira de proteger a equipe de desenvolvimento da pressão da gerência de entregar código rapidamente comprometendo a qualidade, já que isso não é uma opção ao usar TDD. Os entrevistados notaram que a equipe de desenvolvimento aparentava ter mais confiança ao realizar uma nova versão do software ao cliente se comparado a quando eles usavam técnicas tradicionais de testes. Além disso, eles afirmaram que adicionar e testar novas funcionalidades foi mais rápido e requereu menos retrabalho comparado com técnicas tradicionais de testes (BUCHAN; LI; MACDONELL, 2011).

Uma revisão feita por Sfetsos e Stamelos sobre estudos empíricos de Qualidade de Software em práticas de desenvolvimento ágil de software apontou que TDD proporciona uma extensiva melhora da qualidade externa do software (SFETSOS; STAMELOS, 2010). A qualidade externa foi medida geralmente pelo número de testes de aceitação que passaram ou pela densidade de defeitos.

Para Tomaž Dogša e David Batic, TDD aumenta a qualidade de código externa e a qualidade em uso. Produtividade, porém, é prejudicada quando TDD é aplicado. Segundo eles, manutenção foi a principal vantagem apontada por TDD, porque (DOGšA; BATič, 2011):

- TDD cria uma completa rede de segurança de testes de unidade que provê código com baixo índice de regressão;
- TDD pode também prover código de baixa complexidade ciclomática e, na opinião deles, melhor projeto, já que o desenvolvedor codifica somente o que é realmente necessário.

TDD também auxilia na diminuição do tempo para corrigir defeitos (AMRIT; MEIJBERG, 2017).

### 4.5.2 MÉTODOS ÁGEIS E USABILIDADE

Métodos ágeis de desenvolvimento estão forçando profissionais da área de usabilidade a desenvolver novas maneiras de aplicar avaliações de usabilidade. Algumas técnicas de avaliação de usabilidade observadas em desenvolvimento de software utilizando métodos ágeis são revisão por pares com analistas de negócio, product owners e designers de experiência de usuário. Essas técnicas podem ser aplicadas para

diversos artefatos, tais como protótipos em papel, protótipos de baixa e alta fidelidade e software em funcionamento da sprint anterior (SILVA; SILVEIRA; MAURER, 2015).

De acordo com o estudo realizado com profissionais de TI da Suécia feito por Jia, Larusdottir e Cajander, os resultados obtidos indicam que workshop é a técnica utilizada mais utilizada em projetos que utilizam Scrum como metodologia de desenvolvimento, seguido por protótipos de baixa fidelidade, entrevistas, reuniões com usuários e cenários (JIA; LARUSDOTTIR; CAJANDER, 2012).

### 4.5.3 CONFORMIDADE DE MÉTODOS ÁGEIS COM MODELOS E NORMAS

Alguns artigos se propuseram a fazer uma investigação com respeito à conformidade de alguns métodos ágeis em relação à alguns modelos e normas tais como CMMI, MPS e ISO.

O uso de CMMI e metodologias ágeis contribuem para a institucionalização do processo de desenvolvimento, definindo um quadro otimizado e melhorando o processo de design de artefatos. Alguns outros benefícios são a colaboração com as partes interessadas e a forte inclusão do cliente, no que diz respeito à identificação e melhoria do relacionamento problemas (SILVA et al., 2015).

Scrum e CMMI podem coexistir. Scrum e CMMI podem complementar um ao outro criando sinergias que beneficiam a organização. O Scrum é recomendado como um ponto inicial para organizações com times pequenos e sem processos definidos. Por conta disso, a maioria dos fundamentos necessários para a institucionalização das áreas do processo de gerenciamento de projetos CMMI relacionados ao nível de maturidade 2 são criados sem comprometer a agilidade que essas organizações precisam (LINA; DAN, 2012).

O artigo desenvolvido por Bougron, Zeaaraoui e Bouchentouf busca fazer uma projeção do quanto Scrum, XP e Kanban se adequam ao terceiro nível do CMMI. O resultado obtido foi que Scrum cobre 44% do terceiro nível do CMMI, XP cobre 45%, enquanto Kanban cobre somente 6%. Juntando os 3 métodos ágeis, somente 58% das práticas de CMMI são atingidas pelas práticas do XP, Kanban e Scrum (BOUGROUN; ZEAARAOUI; BOUCHENTOUF, 2014). O artigo continua mostrando quais práticas dessas metodologias ágeis cobrem as práticas de CMMI.

Na área de Desenvolvimento de Requisitos, a prática do Scrum de revisar o backlog com o Product Owner cobre essa área. A cooperação diária entre pessoas de negócio e desenvolvedores presente no XP cobre a área de desenvolvimento de requisitos. Kanban não possui nenhuma prática que auxilie nessa prática do CMMI. Na área de Solução Técnica, que se refere à arquitetura de software, projeto e codificação, o projeto simples, pequenas versões e programação em pares são as características

de XP que englobam essa área. Scrum e Kanban não possuem práticas nesse sentido (BOUGROUN; ZEAARAOUI; BOUCHENTOUF, 2014).

No que se refere à Integração de Produto, que consiste na habilidade de integrar múltiplos componentes para formar um produto completo, Scrum cobre essa área por ter um desenvolvimento organizados por sprints e por ter reuniões ao final de cada uma, enquanto o XP cobre essa área por possuir uma integração contínua, pequenas versões e cooperação diária entre pessoas de negócio e desenvolvedores. Kanban novamente não possui nenhuma prática que cubra essa área. No que concerne a área de Verificação, que visa assegurar que o produto está sendo feito como o cliente previu, as atividades de testes, programação em pares e a cooperação diária entre pessoas de negócio e desenvolvedores formam o conjunto de práticas do XP que atingem essa área. Scrum e Kanban não possuem práticas nesse sentido (BOUGROUN; ZEAARAOUI; BOUCHENTOUF, 2014). É interessante notar que os autores não encararam TDD como sendo uma prática do XP. Kanban não possui práticas que englobe essa área.

Na área de Validação, que envolve testes de aceitação, a reunião no final de cada sprint da equipe em conjunto com o Product Owner é a prática presente no Scrum que atinge essa área. A cooperação diária entre pessoas de negócio e desenvolvedores é a prática do XP neste sentido. O Foco em Processos Organizacionais, que consiste em processos que afetam a habilidade da organização, é coberto pelo Scrum com os feedbacks da revisão da sprint. XP e Kanban não possuem práticas que atinjam essa área (BOUGROUN; ZEAARAOUI; BOUCHENTOUF, 2014).

No que diz respeito ao Gerenciamento de Riscos, a reunião no final de cada Sprint é a prática do Scrum que atinge essa área. A cooperação diária entre pessoas de negócio e desenvolvedores é a prática do XP neste sentido. Kanban atinge essa área por ter a possibilidade de se visualizar o fluxo de trabalho. Finalmente, a área de Análise de Decisões e Resoluções é coberta somente pelo Kanban devido ao fato da visualização do fluxo de trabalho e da limitação de trabalho em progresso (BOUGROUN; ZEAARAOUI; BOUCHENTOUF, 2014).

Sergio Galvan, Manuel Mora, Rory V. O'Connor, Francisco Acosta e Francisco Alvarez, escreveram uma análise de conformidade de métodos ágeis com o ISO/IEC 29110 na área de gerência de projetos (GALVAN et al., 2015).

No que diz respeito aos cargos dos membros da equipe, foi observado que Scrum possui uma alta conformidade com o ISO/IEC 29110. XP possui uma conformidade moderada. Foi visto que Scrum tem uma boa organização no que diz respeito aos cargos e satisfaz muitas declarações presentes no padrão ISO/IEC 29110 (GALVAN et al., 2015).

Com respeito à atividades Scrum tem uma boa conformidade mas o fato de

atualizar o repositório de projeto e formalizar documentos para finalizar um processo são detalhes que não estão presentes nas atividades do Scrum. XP possui alguns elementos que precisam ser realçados para atingir a conformidade com o ISO/IEC 29110 (GALVAN et al., 2015).

Scrum pode ser considerado ter uma alta conformidade com o ISO/IEC 29110 no que diz respeito a processos de gerência de projetos, enquanto XP possui um nível moderado de conformidade (GALVAN et al., 2015).

Ahmad Al-Elaimat e Abdel-Rahman Al-Ghuwairiw fizeram um mapeamento das práticas de XP com o ISO 12207. As práticas de XP que englobam atividades previstas no ISO 12207 são as seguintes (AL-ELAIMAT; AL-GHUWAIRI, 2015),

- planejamento incremental;
- pequenas versões;
- projeto simples;
- TDD;
- programação em pares; e
- presença do cliente.

Vieira, Farias Junior, Furtado e Silva (2015) fizeram um estudo com o objetivo de verificar a conformidade de Scrum com o nível G do MPS.BR. Os resultados mostraram que as áreas de definição do escopo, definição do ciclo de vida, viabilidade de atingir metas, revisão e comprometimento com o plano de projeto, envolvimento de stakeholders, revisões dos marcos do projeto, entendimento dos requisitos, revisões no planejamento e produtos e gerenciamento de mudança de requisitos são totalmente implementadas no Scrum. Essas atividades correspondem a 37% do esperado para o nível G do MPS.BR.

### 4.5.4 COMO MÉTODOS ÁGEIS ATINGEM AS EXPECTATIVAS DO USUÁRIO

Desenvolvimento iterativo, constante feedback, constante colaboração entre desenvolvedores, retrospectivas, reuniões diárias, revisões de código, integração contínua, entregas constantes de curto prazo e compartilhamento do conhecimento são práticas do Scrum que auxiliam na qualidade do software, ajudando assim a atingir expectativa do usuário. Foi também reportado que o fato do Scrum não permitir mudanças até o fim da sprint evita problemas no escopo (KHALANE; TANNER, 2013).

Com respeito ao XP, propriedade coletiva do código e padrões de codificação foram citadas como técnicas que aumentam a qualidade técnica (KHALANE; TANNER, 2013).

Sonali Bhasin (2012) afirma que envolvimento na fase inicial e flexibilidade de se ajustar a mudanças frequentes são a chave para se atingir a qualidade no ambiente de desenvolvimento ágil.

Métodos ágeis de desenvolvimento de software entregam protótipos constantemente, fazendo assim com que o cliente possa monitorar o desenvolvimento e prover feedback continuamente. O cliente tem uma constante participação no processo do Scrum e o produto acaba ficando mais flexível (CHUG; MALHOTRA, 2016).

### 4.5.5 CICLOS DE DESENVOLVIMENTO DE CURTA DURAÇÃO

As sprints de curta duração acabam com o risco de não resolver problemas prontamente. Defeitos são descobertos e corrigidos muito antes se comparado com processos tradicionais, o que dá ao time um melhor controle sobre o projeto e melhora a eficiência em corrigir defeitos (LI; MOE; DYBÅ, 2010).

O fato de métodos ágeis, como Scrum e XP, possuírem curtos ciclos de desenvolvimento torna frequente a interação do usuário final e constante o feedback recebido, o que é extremamente de ajuda ao implementar procedimentos de manutenção corretiva (CHUG; MALHOTRA, 2016).

### 4.5.6 REUNIÕES DIÁRIAS

Reuniões diárias facilitam o compartilhamento de conhecimento entre os membros da equipe. Os rápidos feedbacks ajudam desenvolvedores a entender melhor o sistema e a aprender rapidamente com os erros, o que também auxilia na eficiência em corrigir defeitos (LI; MOE; DYBÅ, 2010).

Reuniões diárias é uma prática do Scrum que auxilia na qualidade do software no quesito de atingir expectativa do usuário (KHALANE; TANNER, 2013), além de ser um papel importante de conformidade de modelos como o CMMI (BOUGROUN; ZEAARAOUI; BOUCHENTOUF, 2014).

### 4.5.7 OUTRAS PRÁTICAS OBSERVADAS QUE CONTRIBUEM PARA A QUALIDADE

Além da programação em pares e TDD, refatoração, planejamentos feitos por jogos e cliente no ambiente de desenvolvimento são algumas práticas do XP que contribuem para a melhoria da qualidade (SFETSOS; STAMELOS, 2010).

Apesar do papel do testador em métodos ágeis ter outros desafios se comparado a métodos tradicionais, tais como se adaptar a mudanças e ter uma boa comunicação (BROEK et al., 2014), foi observado que atividades de testes em métodos ágeis são mais regulares, consistentes, automatizadas e efetivas se comparadas com o método incremental de desenvolvimento. Foi também mostrado que métodos ágeis possui uma melhor produtividade, densidade de defeitos e taxa de esforço em corrigir defeitos (TARHAN; YILMAZ, 2013).

De acordo com a revisão feita por Hyrynsalmi, Rindell e Leppanen , onde eles quiseram mostrar como métodos ágeis e segurança andam juntos, muitas das propriedades inerentes do XP são vistas como benéficas à segurança de software, tais como simplicidade de projeto, programação em pares, TDD, refatoração, código coletivo e padrões de codificação (RINDELL; HYRYNSALMI; LEPPANEN, 2017).

Métodos ágeis de desenvolvimento de software também possuem um impacto positivo na manutenção devido à rápida detecção de defeitos (CHUG; MALHOTRA, 2016).

## 4.6 LACUNAS ENCONTRADAS QUE COMPROMETEM O ATINGIMENTO DA QUALIDADE

Esta seção apresenta problemas e lacunas encontrados em métodos ágeis que podem minar o atingimento da qualidade.

### 4.6.1 PROBLEMAS EM USABILIDADE

Scrum é visto como uma metodologia que provê qualidade de software, porém alguns pesquisadores mostram alguns problemas no uso desse método no que se refere à qualidade.

Scrum não oferece nenhum guia concreto de como se atingir requisitos de qualidade. Devido a isso, o time de desenvolvimento precisa elaborar inovações, nas quais pode ser incluído adotar atividades de outras metodologias (KHALANE; TANNER, 2013).

Scrum tem sido criticado por não envolver usuários finais reais no processo de desenvolvimento e por não cuidar adequadamente de suas necessidades de usabilidade. Foi-se concluído que necessidades de usuários finais não são suficientemente incluídas no processo ágil de desenvolvimento de software (JIA; LARUSDOTTIR; CAJANDER, 2012).

O fato de não haver feedback dos usuários após o fim da sprint pode acarretar em possíveis mudanças no final do estágio de desenvolvimento o que gera uma carga

alta de retrabalho. Muitas vezes, os usuários finais não estão envolvidos em atividades de feedbacks, o que pode acarretar na falta de entendimento do time de que eles estão desenvolvendo o produto correto ou não. Integrar testes de usabilidade em sprints de curta duração é um problema notável no Scrum (SILVA; SILVEIRA; MAURER, 2015).

Com respeito à testes, existe uma falta de documentação de testes em Scrum. Os testadores não tem informação suficiente para testar. Especialmente quando desenvolvedores e testadores estão em diferentes localizações, os testadores confiam nas suas próprias suposições para testar o sistema em desenvolvimento. Além disso, testar no final da sprint envolve o time somente o time de desenvolvimento e os principais stakeholders, sem intervenção de usuários finais reais do produto (RAHAYU et al., 2016).

Integrar testes de usabilidade em Scrum poderá melhorar o desenvolvimento de software no que diz respeito a feedback, testes e qualidade do produto. Integrar testes de usabilidade pode melhorar o processo de revisão de sprint. Além disso, feedbacks de usuário serão obtidos mais cedo no processo de desenvolvimento. No final da sprint, o time pode garantir a qualidade do software por entregar um produto que foi avaliado usando técnicas de testes de usabilidade (RAHAYU et al., 2016).

### 4.6.2 LACUNAS NA ELICITAÇÃO E RASTREABILIDADE DE REQUISITOS

A falta de requisitos de qualidade ou requisitos mal especificados podem levar projetos à falhas ou grande perdas. Elicitar requisitos efetivamente é uma tarefa difícil, especialmente em Scrum onde uma pessoa, geralmente o Product Owner, é responsável de preparar a lista de todos os requisitos a serem incluídos no projeto. O sucesso ou fracasso de sistemas depende do quão bem os requisitos de qualidade são implementados (ASGHAR et al., 2017).

O Scum permite que os engenheiros possam lidar com mudanças de requisitos enquanto elas aparecem; porém, é ainda um desafio compreender quais requisitos são vitais a ponto de ter uma alta necessidade e serem incorporados nas primeiras sprints. Organizar requisitos em prioridades ajuda o time a compreender quais requisitos são mais essenciais e mais urgentes para serem implementados e executados. Priorização é uma atividade que auxilia a tomada de decisões em outras fases da engenharia do software (ASGHAR et al., 2017).

Portanto, uma técnica de priorização de requisitos bem gerenciada deveria ser incluída no Scrum para aumentar sua qualidade. Requisitos não seriam priorizados somente com base em decisões humanas, mas por analisar criticamente os fatores que podem causar o sucesso ou fracasso do produto (ASGHAR et al., 2017).

Em métodos ágeis, a rastreabilidade de requisitos não pode ser executada como

é estabelecida em gerenciamento de requisitos tradicionais como ISO/IEC 12207:2008 ou SWEBOK. Isso se dá porque métodos ágeis frequentemente omitem um documento formal de especificação de requisitos. Essa situação, se não gerenciada corretamente, pode criar sérios problemas a outros processos, tais como gerenciamento de mudanças, análise de impactos e estimativas (ESPINOZA; GARBAJOSA, 2011).

Um dos problemas do Scrum, por exemplo, com respeito à rastreabilidade, é o fato dos backlogs serem focados em implementar as funcionalidades sem uma análise de rastreabilidade dos seus relacionamentos. Por isso, manter projetos de software se torna difícil quando os requisitos mudam, porque os efeitos colaterais que podem surgir devido a essas mudanças são desconhecidos (JEON et al., 2011) .

É altamente recomendado o desenvolvimento de práticas de rastreabilidade para evitar problemas devido à lacunas da especificação dos requisitos. Porém, as abordagens tradicionais de rastreabilidade são baseadas precisamente em documentos completos de especificação de requisitos, artefato que métodos ágeis não produzem. Portanto, é fundamental desenvolver abordagens de rastreabilidade que ataquem os requisitos em métodos ágeis, já que isso é fundamental para desenvolver software com qualidade (ESPINOZA; GARBAJOSA, 2011).

### 4.6.3 PROBLEMAS AO EXECUTAR TDD

Críticos defendem que mudanças frequentes a testes em TDD são mais propensos do que a abordagem de testar o software no final do ciclo de desenvolvimento de causar quebras do sistema, acarretando assim em retrabalho e perda de produtividade. Boehm e Turner também notam com o uso de TDD, as consequências de desenvolvedores terem habilidades de teste inadequadas podem ser amplificadas, se comparadas com as consequência com métodos tradicionais de testes. Outros já afirmam que TDD pode não ser apropriado para todos os domínios de aplicação (BUCHAN; LI; MACDONELL, 2011).

Na pesquisa feita por Bunchan, Li e MacDonell, os entrevistados notaram que tiveram uma grande sobrecarga ao aprender como implementar práticas de TDD, principalmente quando o desenvolvedor possui vários anos de experiência sem o uso de TDD. Foi afirmado que os benefícios do TDD estão fortemente atrelados à capacidade dos desenvolvedores de refatorar e em escrever testes de alta qualidade. Desenvolver essas habilidades é citado como um dos maiores desafios do TDD (BUCHAN; LI; MACDONELL, 2011).

TDD deve ser seguido estritamente para produzir benefícios (BUCHAN; LI; MACDONELL, 2011). Porém, foi visto que aplicar TDD estritamente é visto como uma pressão em cima dos desenvolvedores e devido ao curto tempo que eles tem para

desenvolver, atividades como refatoração fiquem em segundo plano (LI; MOE; DYBÅ, 2010). Existem muitos mal entendidos em implementar TDD como uma prática diária. É mais do que apenas escrever testes antes de codificar. A implementação incorreta do TDD pode significar que os benefícios defendidos pela literatura não serão atingidos. Treinamento básico de como implementar TDD e reuniões entre membros da equipe pode ajudar a assegurar que TDD está sendo corretamente praticado (BUCHAN; LI; MACDONELL, 2011).

Um estudo mostrou os principais erros durante a prática do TDD (ANICHE; GEROSA, 2010):

- não assistir o teste falhar;
- esquecer o passo de refatoração;
- refatorar uma parte do código enquanto trabalha em um teste;
- usar nomes ruins de teste;
- não começar pelo teste mais simples;
- rodar somente o teste atual que está falhando;
- escrever um cenário de teste complexo;
- não refatorar o código de teste;
- não implementar o código mais simples que faz o teste passar.

Erros em TDD podem diminuir a qualidade do código e do software (ANICHE; GEROSA, 2010).

A maioria dos resultados obtidos quando se avalia produtividade em TDD mostra que ela tem um menor desempenho se comparada à praticas tradicionais (BISSI et al., 2016) . Além disso, a gerência deve estar ciente das características do desenvolvimento de software utilizando TDD. Isso se dá porque a funcionalidade pode demorar mais tempo do que o esperado para ser implementada, se comparada com o desenvolvimento sem o uso de TDD (BUCHAN; LI; MACDONELL, 2011).

### 4.6.4 ASPECTOS DE MÉTODOS ÁGEIS QUE NÃO SE ADÉQUAM A MODELOS E NORMAS

Métodos ágeis possuem um foco reduzido no que se refere a processos de garantia de qualidade. Isso acontece devido aos princípios ágeis que preferem indivíduos e interações à processos e ferramentas (TOMMY et al., 2015). Além disso, é citado

de que ciclos curtos de desenvolvimento e contínua entrega de funcionalidades pode levar a negligenciar ou não enfatizar o suficiente a questão da qualidade do produto (MOHAGHEGHIA; APARICIO, 2017).

O foco do CMMI é na organização. A maioria dos benefícios do CMMI são obtidos quando é implementado no nível organizacional. Assim como em outros métodos ágeis, Scrum não menciona o nível organizacional em seu processo (LINA; DAN, 2012).

Scrum não cobre todas as práticas específicas da área de gerenciamento de processos, mas pode ser adaptado para ser mais compatível com o CMMI. Por outro lado, processos tradicionais baseados no modelo do CMMI podem ser melhorados por adicionar práticas ágeis em suas atividades (LINA; DAN, 2012),.

O propósito de Gerenciamento de riscos é de identificar potenciais problemas antes deles ocorrerem. Assim, atividades preventivas e corretivas podem ser planejadas e executadas se necessário. Em Scrum, riscos são identificados, mas nada é mencionado com respeito à práticas que definam origens, parâmetros ou categorias para analisar e controlar o esforço do gerenciamento de riscos. Em Scrum, não há estratégias de respostas ao risco ou um plano de mitigação para riscos críticos baseados no histórico ou algo similar. Com respeito à gerência de configuração ou garantia de qualidade, Scrum não possui nenhuma prática que ataquem essas áreas (LINA; DAN, 2012),.

Métodos ágeis sozinhos, de acordo com estudos, não são suficientes a obter níveis 2, 3 ou 5 do CMMI. É necessário recorrer a práticas adicionais. Organizações interessadas em combinar CMMI e desenvolvimento ágil não deve negligenciar a documentação. A documentação deve demonstrar que o processo está sendo seguido, documentar decisões e atividades específicas. Treinamentos também são um desafio. A combinação de diferentes metodologias ágeis, como XP usado para a área operacional, Scrum para a gerencial e Lean para a estratégica parece ser um caminho interessante (SILVA et al., 2015) .

A maior técnica de documentação de requisitos do XP é a história de usuário. No entanto, histórias de usuário possuem menos detalhes do que é especificado pelo ISO 9001. Por exemplo, história de usuário possui uma descrição de alto nível do requisito e deixa os detalhes para a comunicação face a face com o usuário durante as iterações. Além do mais, a história do usuário não leva em consideração os requisitos do sistema e nenhum detalhe técnico necessários durante o desenvolvimento (QASAIMEH; ABRAN, 2010).

Histórias de usuário são geralmente escritas em linguagem naturais e especificações formais não são providas por histórias de usuário. Por isso, requisitos são validados por protótipos e por clientes no ambiente de desenvolvimento. Avaliações

formais não são suportadas pelo XP (QASAIMEH; ABRAN, 2010).

O ISO 9001 requer que os desenvolvedores selecionem métodos de revisão durante o plano e o processo de desenvolvimento. O nível de formalidade do método de revisão deve ser relevante para a complexidade do projeto. As revisões em XP são basicamente baseadas em programação por pares, onde modificações são feitas baseados nas decisões dos programadores e nenhuma documentação é fornecida. As atividades de revisão em XP não possuem nenhum método formal, tais como inspeção (QASAIMEH; ABRAN, 2010).

XP não armazena os passos necessários para resolver problemas durante a fase de testes de unidade, testes de integração e testes de aceitação. As atividades de testes em XP são baseadas principalmente em casos de teste e não fornecem nenhuma evidência documentada para auditores do ISO 9001 de como essas atividades de testes foram planejadas e executadas ao longo do ciclo de vida do software (QASAIMEH; ABRAN, 2010).

### 4.6.5 PROBLEMAS EM SEGURANÇA DE SOFTWARE

Segurança deve ser considerada no início do processo de desenvolvimento de software. Porém, Scrum é criticado por negligenciar esse tipo de análise no início do processo de desenvolvimento (MARIA; RODRIGUES JUNIOR; PINTO, 2015). Usar métodos iterativos, tais como Scrum, para desenvolver funcionalidades de segurança geralmente leva ao desenvolvimento de funcionalidades inefetivas por causa da incompletude das cenários de testes e do conflito de requisitos entre as iterações (OTHMANE; ANGIN; BHARGAVA, 2014). Certas atividades de segurança são notadas como incompatíveis ou inexistentes em métodos ágeis: necessidade de engenheiros de segurança, revisões de código estáticas, revisões formais de segurança e documentação de segurança (RINDELL; HYRYNSALMI; LEPPANEN, 2017).

De acordo com Hyrynsalmi, Leppanen e Rindell (2015), métodos ágeis promovem desenvolvimento por iterações e interações informais e possui um foco mínimo a processos rigorosos. Introduzir requisitos rigorosos de segurança ao processo de desenvolvimento de software geralmente resulta na criação de uma arquitetura formal de segurança. Integrar requisitos de segurança tais como revisões, testes de segurança, processos e documentação, irá fazer com que o esforço de desenvolvimento aumente. Toda a sobrecarga de gerenciamento devido a isso vai de encontro com a filosofia ágil de simplicidade e informalidade. Integrar processos de segurança em métodos ágeis tem o potencial de transformar esses métodos, por definição, em algo que não nem Lean, nem ágil.

Com respeito à segurança relacionada atividades de desenvolvedores e usuá-

rios, um software seguro deve verificar a identidade dos desenvolvedores e usuários baseados nos seus privilégios e responsabilidades. Em XP, a frequente mudanças no software feitas por diferentes pares torna difícil a identificação dos desenvolvedores. Para segurança, controlar quem executa mudanças no software é primordial. Esse problema é relacionado às atividades de programação em pares e integração contínua (ADELYAR; NORTA, 2016).

O objetivo de desenvolver software seguros é de prover somente os privilégios necessários para usuários e desenvolvedores. Aplicando esse princípio nas atividades dos usuários e desenvolvedores, se um problema de mal uso de um privilégio for reportado, o número de entidades que precisam ser inspecionadas é minimizado. Diferentes pares com diferentes níveis de conhecimento executando diferentes mudanças ao software é o maior desafio nessa questão de limitação de privilégios. Frequentes mudanças dos pares e de membros dos pares é um problema para a limitação dos privilégios dos desenvolvedores (ADELYAR; NORTA, 2016).

Certificar-se de que o software está realmente implementado como o esperado precisa de atenção e cautela por parte dos desenvolvedores. Feedbacks inconstantes, ideias e priorização de atividades pelos usuários podem afetar negativamente a atenção dos desenvolvedores, aumentando a possibilidade de introduzir falhas ou vulnerabilidades no software, além de causar retrabalho. Esse retrabalho afeta negativamente a atenção dos desenvolvedores, pois ideias inconstantes dos clientes requerem mais mudanças e retrabalho pode causar pressão aos desenvolvedores. O projeto de software precisa ser simples e pequeno para que técnicas de segurança tais como inspeções por linha de código possam ser executadas. Embora métodos ágeis encorajem isso, não há garantia que os desenvolvedores e os clientes irão aderir a isso. Requisitos instáveis e mudanças frequentes podem aumentar a complexidade do software. O planejamento por meio de jogo é a prática ágil responsável por essa possível situação (ADELYAR; NORTA, 2016).

Métodos ágeis foram vistos como tendo certos problemas com adaptabilidade de atividades de segurança. Documentação repetitiva e atividades de revisão são incompatíveis com esses métodos. Algo que pode ser feito, por exemplo, com o Scrum, seria incluir requisitos de segurança nas histórias e no backlog. Porém, incorporar revisões de segurança e auditorias no processo de desenvolvimento se torna mais difícil em processos iterativos, retendo assim a 'agilidade' do método (HYRYNSALMI; LEPPÄNEN; RINDELL, 2015).

Modelos de segurança, tais como SDL, definem vários papéis de segurança para o time de desenvolvimento. Isso promove uma rigorosa separação de responsabilidades. Porém, métodos ágeis especificam somente um pequeno grupo de papéis ou até mesmo nenhum. No Scrum, por exemplo, é definido somente os papéis de

Product Owner, Desenvolvedores e Scrum Master. Desses, o desenvolvedor seria o mais apropriado em assumir as responsabilidades de um especialista em segurança. Porém, isso é uma clara violação da regra de 'separação de responsabilidades': o desenvolvedor raramente é a melhor pessoa para quebrar seu próprio código. Além disso, essa abordagem é contra a agilidade em dois sentidos: times não compartilhando informações é uma clara violação da filosofia ágil e ter diferentes times trabalhando em paralelo reduzem a velocidade de desenvolvimento e aumenta o custo. A falta de papéis de segurança também é presente em XP e Kanban (HYRYNSALMI; LEPPÄNEN; RINDELL, 2015).

## 4.7 CONSIDERAÇÕES FINAIS

Este capítulo mostrou os resultados da revisão sistemática que teve por objetivo identificar práticas, técnicas e métodos utilizados em métodos ágeis que servem de auxílio para o atingimento da qualidade de software, além de investigar lacunas presentes em métodos ágeis que podem prejudicar a conformidade com modelos, normas áreas de qualidade.

Foi apresentado primeiramente os dados da revisão. Com base nos 1335 artigos encontrados de forma automática pelos engenhos de busca, foi iniciada a primeira fase da revisão, onde foram lidos título, resumo e palavras-chave de todos os estudos. Após a primeira fase, 99 artigos passaram para a segunda fase (leitura da introdução e conclusão), dos quais 40 artigos foram selecionados para seguirem adiante. Dos 40, 3 artigos foram excluídos após a qualidade ter sido avaliada após a leitura completa de todos. Portanto, 37 artigos foram selecionados com o objetivo de responder às perguntas de pesquisa.

Vimos que o Brasil foi o país com o maior número de estudos, enquanto o ano de 2015 obteve a maior quantidade de estudos selecionados. Observamos que as práticas do TDD e programação em pares foram as mais citadas. No que se refere à modelos e normas, CMMI e ISO foram as mais estudadas em conjunto com métodos ágeis. Scrum e XP foram as metodologias mais avaliadas.

Após isso, o capítulo focou em apresentar os achados da revisão sistemática mostrando primeiramente as técnicas utilizadas em métodos ágeis que servem de auxílio para se alcançar qualidade. Práticas tais como TDD, programação em pares, reuniões diárias, uso de protótipos, propriedade coletiva de código são algumas das práticas que auxiliam para o atingimento da qualidade interna, externa, usabilidade e atingir expectativas do usuário. Porém, a falta de documentação detalhada, falta de técnicas de rastreabilidade e erros cometidos ao executar TDD são algumas das lacunas encontradas em metodologias ágeis.

O resultado dessa revisão mostra que várias práticas são defendidas como essenciais para atingir a qualidade. Porém, algumas lacunas são presentes.

# 5 DISCUSSÃO E CONSIDERAÇÕES FINAIS

Este trabalho teve por objetivo identificar práticas, técnicas e métodos utilizados em métodos ágeis que servem de auxílio para o atingimento da qualidade de software, além de investigar lacunas presentes em métodos ágeis que podem prejudicar a conformidade com modelos, normas áreas de qualidade. Uma pesquisa exploratória foi feita para se obter mais informações a respeito de métodos ágeis e qualidade de software e uma revisão sistemática foi feita com o intuito de se verificar o estado da arte deste assunto.

Foi observado que a primeira pergunta de pesquisa - quais práticas, estratégias, ferramentas e métodos utilizados no desenvolvimento de software utilizando métodos ágeis auxiliam o atingimento da qualidade de software e em quais áreas? - foi bem respondida, mostrando várias práticas inerentes à métodos ágeis que são citadas como importantes para se ter uma boa qualidade de produto e também ajudam na satisfação do cliente. Scrum e XP foram as práticas mais estudadas nos artigos selecionados, enquanto Kanban também foi citado mas com menos frequência. Com isso, podemos notar que esses métodos, por mais que não sejam novos, são ainda de grande valor ao desenvolvimento de software e ainda merecem estudos para continuar a atestar seu bom desempenho.

Analisando os resultados, podemos notar que métodos ágeis realmente possuem práticas que contribuem para a qualidade do produto e processo de software. Test Driven Development foi vista como uma prática excelente bastante utilizada. No entanto, os resultados se mostraram contraditórios. Enquanto alguns defendem a prática do TDD como aumentando a qualidade interna e externa de software (ANICHE; GEROSA, 2010; DOGšA; BATIč, 2011; AMRIT; MEIJBERG, 2017), outros já afirmam que existem poucas evidências com respeito aos benefícios do TDD e que a produtividade do time é prejudicada por isso (KOLLANUS, 2010; BUCHAN; LI; MACDONELL, 2011; RAFIQUE; MISIC, 2013). Além disso, foi observado que a prática do TDD só produz realmente um bom resultado se for executado corretamente (ANICHE; GEROSA, 2010; BUCHAN; LI; MACDONELL, 2011).

Programação em pares, reuniões com usuários, presença do cliente, propriedade coletiva de código, constante entrega de protótipos, refatoração, clientes presentes no ambiente de desenvolvimento, flexibilidade em se ajustar a mudanças, feedback constante, curtos ciclos de desenvolvimento são exemplos de técnicas consideradas importantes para alcançar a satisfação do usuário, usabilidade e outros aspectos de qualidade (SFETSOS; STAMELOS, 2010; LI; MOE; DYBÅ, 2010; BHASIN, 2012; KHALANE; TANNER, 2013; BROEK et al., 2014; CHUG; MALHOTRA, 2016).

Métodos ágeis também possuem práticas que ajudam à conformidade com respeito a modelose e normas. CMMI foi tema de 3 artigos. Programação em pares, sprints e reuniões ao final do ciclo e visualização do fluxo do trabalho são práticas que são vantajosas para se conseguir conformidade com o CMMI. Outros artigos falaram a respeito de normas ISO. O ISO 29110, que é focado para entidades consideradas muito pequenas, é uma boa ajuda para empresas melhorarem a qualidade do software e desempenho, possui uma boa relação com Scrum e XP devido às atividades de gerência e papéis definidos. Pequenas versões, TDD e presença do cliente são algumas práticas citadas para o auxílio para atingir normas tais como o ISO 12207 (QASAIMEH; ABRAN, 2010; LINA; DAN, 2012; BOUGROUN; ZEAARAOUI; BOUCHENTOUF, 2014; GALVAN et al., 2015; SILVA et al., 2015).

A tabela a seguir apresenta algumas técnicas e seus respectivos benefícios.

**Tabela 7 – Conjunto de técnicas e seus respectivos benefícios**

| **Técnica** | **Benefícios** |
|---|---|
| TDD | maior cobertura de testes; código de melhor qualidade; confiança do desenvolvedor; menor tempo para corrigir defeitos. |
| Programação em pares | melhor qualidade de código, trabalho em equipe e compartilhamento do conhecimento; menor introdução de defeitos. |
| Reuniões diárias | compartilhamento do conhecimento, aprender com erros; atingir expectiativa do usuário. |
| Presença do cliente | atingir expectativa do usuário; facilita manuntenção corretiva e entendimento dos requisitos |
| Sprints | defeitos corrigidos mais rapidamente |
| Protótipos | usabilidade |
| Propriedade coletiva de código | atingir expectativa do usuário; compartilhamento do conhecimento |
| Entregas constantes | atingir expectativa do usuário |

Fonte: Autor

Porém, foram encontradas algumas lacunas em métodos ágeis no que diz respeito à áreas como usabilidade, elicitação e rastreabilidade de requisitos, segurança de software e até mesmo práticas como o TDD.

Quanto à segunda pergunta de pesquisa - em quais áreas são observadas lacunas nos processos ou atividades de desenvolvimento de software utilizando métodos

ágeis que comprometem a qualidade de software? - podemos considerá-la respondida, já que problemas no TDD, segurança de software, gerenciamento de riscos e requisitos são áreas importantes para se obter qualidade em um projeto de software. Devido à proposta agilidade desses métodos, a equipe acaba ganhando em alguns aspectos, mas acaba perdendo em outros como estes citados acima.

O fato do Scrum não possuir um guia concreto, falta de testes de usabilidade e não possuir um envolvimento do cliente no ambiente de desenvolvimento (assim como propõe o XP) é um dos motivos citados como problemas de usabilidade (JIA; LARUSDOTTIR; CAJANDER, 2012; KHALANE; TANNER, 2013; SILVA; SILVEIRA; MAURER, 2015; RAHAYU et al., 2016). Técnicas de priorização de requisitos, falta de documentos de rastreabilidade e dificuldade em rastrear requisitos baseados no backlog são outros problemas citados (ESPINOZA; GARBAJOSA, 2011; JEON et al., 2011; ASGHAR et al., 2017).

Apesar de TDD ser citado como uma técnica que traz bastante benefícios, também é reportado por trazer alguns problemas e possuir algumas lacunas. Mudanças frequentes em testes são reportados como mais propensos a causar regressões e perda e produtividade. É defendido também de que o TDD, para produzir benefícios, deve ser seguido estritamente. Porém, isso acaba gerando uma pressão em cima dos desenvolvedores devido ao curto tempo de desenvolvimento. Por conta disso, atividades do TDD como refatoração ficam para depois. Devido a isso, erros no TDD diminuem a qualidade do código e do software. Além disso, TDD é reportado como sendo uma prática que diminui a produtividade da equipe, já que uma funcionalidade pode demorar mais tempo para ser implementada se comparada com o desenvolvimento sem o uso do TDD (LI; MOE; DYBÅ, 2010; ANICHE; GEROSA, 2010; BUCHAN; LI; MACDONELL, 2011).

Falta de documentação detalhada e falta de gerenciamento de riscos são algumas práticas citadas que diminuem a conformidade de métodos ágeis como XP e Scrum em relação à modelos e normas de qualidade. Métodos ágeis sozinhos, sem a ajuda de nenhuma prática adicional não são suficientes para obter níveis 2, 3 e 5 do CMMI, por exemplo(QASAIMEH; ABRAN, 2010; LINA; DAN, 2012; TOMMY et al., 2015; SILVA et al., 2015; MOHAGHEGHIA; APARICIO, 2017). Em certo sentido, isso era de se esperar, já que métodos ágeis fazem o *trade-off*de obter agilidade e flexibilidade, deixando de lado documentação excessiva (o que acaba sendo essencial em algumas adequações a modelos).

Na questão da segurança de software, atividades de segurança são citadas como incompatíveis ou inexistentes em métodos ágeis. A falta de análise de segurança no início do processo de desenvolvimento do Scrum, por exemplo, é visto como um problema. A prática de programação em pares, praticada pelo XP, é visto também

como uma atividade que contribui para a lacuna de segurança, já que vários pares de desenvolvedores trabalham no mesmo código, dificultando assim a identificação da pessoa que mudou o código (OTHMANE; ANGIN; BHARGAVA, 2014; MARIA; RODRIGUES JUNIOR; PINTO, 2015; HYRYNSALMI; LEPPÄNEN; RINDELL, 2015; RINDELL; HYRYNSALMI; LEPPANEN, 2017). O fato da segurança ser um problema em métodos ágeis, tais como Scrum e XP, estão ligados com a agilidade dos processos, já que documentação extensiva e papéis de segurança podem ter uma visão de que o desenvolvimento do sistema será comprometido caso a equipe foque em tais atividades.

Alguns autores, porém, defendem visões que podem ter um diferente ponto de vista. Por exemplo, Hyrynsalmi, Leppanen e Rindell (2015) afirmam que métodos ágeis promovem desenvolvimento por iterações e interações informais e possui um foco mínimo a processos rigorosos. Eles continuam dizendo que integrar processos de segurança em métodos ágeis tem o potencial de transformar esses métodos, por definição, em algo que não nem Lean, nem ágil. Porém, se bem planejado, o projeto poderia se adaptar a esse tipo de formalidade sem desconsiderar a agilidade dos processos.

Lina e Dan (2012) afirmam que o Scrum não cobre todas as práticas específicas da área de gerenciamento de processos, mas pode ser adaptado para ser mais compatível com o CMMI, mas processos tradicionais baseados no modelo do CMMI podem ser melhorados por adicionar práticas ágeis em suas atividades. É digno de nota que o artigo de não considera o Scaled Agile Framework (INC, 2018), um framework que contempla times ágeis de vários tamanhos e engloba todas as áreas, inclusive o gerenciamento.

Mohagheghia e Aparicio (2017) dizem que ciclos curtos de desenvolvimento e entrega contínua de funcionalidades podem levar a negligenciar ou não enfatizar o suficiente a qualidade do produto. Porém, isso é relativo. Dependendo da organização do projeto, das pessoas envolvidas na gerência e clientes e até mesmo do custo do projeto, ciclos curtos e entrega contínua podem continuar a serem feitas sem minar a qualidade.

Larusdottir, Jia e Cajander (2012) concordam que necessidades de usuários finais não são suficientemente incluídas no processo ágil. Porém, isso também é relativo ao projeto. Não se pode afirmar isso em todos os projetos que métodos ágeis são utilizados.

Com base em tudo o que foi apresentado, o efeito dessa pesquisa contribui para identificarmos o estado da arte na área de métodos ágeis em conjunto com qualidade, mostrando as práticas positivas e processos que possuem algumas desvantagens. Isso pode ajudar equipes de desenvolvimento de software a terem uma pequena base da validade em usar métodos ágeis, tais como Scrum e XP, para seus projetos ou se vão

preferir utilizar outros métodos que se adéquam melhor às suas necessidades.

## 5.1 LIMITAÇÕES E TRABALHOS FUTUROS

A maioria dos artigos observados focaram somente em três métodos ágeis: Scrum, Kanban e XP. Mais metodologias ágeis poderiam ter sido investigadas.

Uma outra limitação consiste no fato de que a segunda fase da revisão (leitura da introdução e conclusão de 99 artigos), a avaliação de qualidade dos artigos, a extração, a análise e a sintese dos dados desta revisão foi feita somente pelo pesquisador Dennys Barros. Caso mais de um pesquisador tivesse participado em todas as fases da revisão sistemática, isso poderia dar mais força à exposição das ideias.

Além disso, o fato de o trabalho ter sido limitado a apenas 3 engenhos de busca pode ter sido um fator para não se obter trabalhos de países mais expressivos em desenvolvimento de software, como os EUA.

As possibilidades de trabalhos futuros são:

- Pesquisar sobre outros métodos ágeis além de Scrum, XP e Kanban e verificar a conformidade deles com a qualidade de software;
- Pesquisar sobre ferramentas ou métodos que auxiliem a rastreabilidade nos métodos ágeis.

# Referências


ADELYAR, S. H.; NORTA, A. Towards Secure Agile Software Development Process. 2016.

AHMAD, M. O.; MARKKULA, J.; OVIO, M. Kanban in software development: A systematic literature review. 2013.

AL-ELAIMAT, A.; AL-GHUWAIRI, A. ProceduralAssessment Process of Software Quality Models Using Agility. 2015.

ALAHYARI, H.; SVENSSON, R. B.; GORSCHEK, T. A study of value in agile software development organizations. 2016.

ALLIANCE, A. *What is Extreme Programming (XP)*. 2018. Disponível em: <https://www.agilealliance.org/glossary/xp>. Acesso em: 16/01/2018.

ALLIANCE, S. *Scrum Roles Demystified*. 2016. Disponível em: <https://www.scrumalliance.org/agile-resources/scrum-roles-demystified>. Acesso em: 04/01/2018.

AMRIT, C.; MEIJBERG, Y. Effectiveness of Test Driven Development and Continuous Integration – A Case Study. 2017.

ANDERSON, D. J.; CARMICHAE, A. *Essential Kanban Condensed*. [S.l.]: Lean Kanban University Press, 2016.

ANICHE, M. F.; GEROSA, M. A. Most Common Mistakes in Test-Driven Development Practice: Results from an Online Survey with Developers. *Third International Conference on Software Testing, Verification, and Validation Workshops*, 2010.

ASGHAR, A. R. et al. Impact and Challenges of Requirements Elicitation & Prioritization in Quality to Agile Process: Scrum as a Case Scenario. 2017.

BECK, K. Embracing Change with Extreme Programming. 1999.

BECK, K. Test-Driven Development By Example. 2002.

BECK, K. *Extreme Programming Explained*: Embrace Change. 2. ed. [S.l.]: Addison-Wesley, 2004.

BELLA, E. di et al. Pair Programming and Software Defects– a large, industrial case study. 2013.

BHASIN, S. Quality Assurance in Agile –A study towards achieving excellence. 2012.

BIOLCHINI, J. et al. Systematic Review in Software Engineering. 2005.

BISSI, W. et al. The Effects of Test Driven Development on Internal Quality, External Quality and Productivity: A systematic review. 2016.

BOEHM, B. W.; BROWN, J. R.; LIPOW, M. Quantitative Evaluation of Software Quality. 1976.

BOUGROUN, Z.; ZEAARAOUI, A.; BOUCHENTOUF, T. The projection of the specific practices of the third level of CMMI model in agile methods: Scrum, XP and Kanban. 2014.

BOURQUE, P.; FAIRLEY, R. E. Guide to the Software Engineering Body of Knowledge. 2004.

BROEK, R. van den et al. Integrating Testing into Agile Software Development Processes. 2014.

BUCHAN, J.; LI, L.; MACDONELL, S. G. Causal Factors, Benefits and Challenges of Test-Driven Development: Practitioner Perceptions. *18th Asia-Pacific Software Engineering Conference*, 2011.

CHUG, A.; MALHOTRA, R. Comparative analysis of agile methods and iterative enhancement model in assessment of software maintenance. 2016.

CMMI. *About CMMI® Institute*. 2018. Disponível em: <http://cmmiinstitute.com/about-cmmi-institute>. Acesso em: 06/01/2018.

DAVULURU, T.; MEDIDA, J.; REDDY, D. V. A Study of Software Quality Models. 2014.

DC/UFSCAR. *StArt*. 2013. Disponível em: <http://lapes.dc.ufscar.br/tools/start_tool>. Acesso em: 16/01/2018.

DEVI, V. Traditional and Agile Methods: An Interpretation. 2013. Disponível em: <https://www.scrumalliance.org/community/articles/2013/january/traditional-and-agile-methods-an-interpretation>. Acesso em: 04/01/2018.

DOGšA, T.; BATč, D. The effectiveness of test-driven development: an industrial case study. 2011.

DYBÅ, T.; DINGSØYR, T. Empirical studies of agile software development: A systematic review. 2007.

ESPINOZA, A.; GARBAJOSA, J. A study to support agile methods more effectively through traceability. 2011.

GALVAN, S. et al. A Compliance Analysis of Agile Methodologies with the ISO/IEC 29110 Project Management Process. 2015.

GALVÃO, T. F.; PEREIRA, M. G. Revisões sistemáticas da literatura: passos para sua elaboração. *Epidemiologia e Serviços de Saúde*, p. 183 – 184, 2014.

GIL, A. C. Como Elaborar Projetos de Pesquisa. In: ____. 6. ed. [S.l.]: Atlas, 2008. cap. 3, p. 27 – 27.

HIGHSMITH, J. et al. Agile Manifesto. 2001.

HUO, M. et al. Software Quality and Agile Methods. In: PROCEEDINGS OF THE 28TH ANNUAL INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, IEEE„ 2004. [S.l.], 2004.

HYRYNSALMI, S.; LEPPÄNEN, V.; RINDELL, K. A Comparison of Security Assurance Support of Agile Software Development Methods. 2015.

INC, S. A. *What's new in SAFe 4.5*. 2018. Disponível em: <http: //www.scaledagileframework.com/whats-new-in-safe-45/>. Acesso em: 03/02/2018.

ISO. *ISO 9000:2005*. 2012. Disponível em: <https://www.iso.org/obp/ui/%23iso%3Astd%3Aiso%3Ats%3A19158%3Aed-1%3Av1%3Aen>. Acesso em: 15/09/2017.

ISO. ISO 9000:2015. 2015.

ISO. *About ISO*. 2018. Disponível em: <https://www.iso.org/about-us.html>. Acesso em: 06/01/2018.

ISO/IEC. ISO/IEC 25010:2011. 2011.

JEON, S. et al. Quality Attribute driven Agile Development. 2011.

JIA, Y.; LARUSDOTTIR, M. K.; CAJANDER, Å. The Usage of Usability Techniques in Scrum Projects. 2012.

JUNG, H.; KIM, S.; CHUNG, C. Measuring Software Product Quality: A Survey of ISO/IEC 9126. 2004.

KHALANE, T.; TANNER, M. Software Quality Assurance in Scrum: The need for concrete guidance on SQA strategies in meeting user expectations. 2013.

KITCHENHAM. Guidelines for performing Systematic Literature Reviews in Software Engineering. 2007.

KOLLANUS, S. Test-Driven Development - Still a Promising Approach? *Seventh International Conference on the Quality of Information and Communications Technology*, 2010.

KORU, A. G. ; EMAM, K. E. The Theory of Relative Dependency: Higher Coupling Concentration in Smaller Modules. 2010.

LAPORTE, C. Y.; APRIL, A.; RENAULT, A. Applying ISO/IEC Software Engineering Standards in Small Settings: Historical Perspectives and Initial Achievements. 2006.

LI, J.; MOE, N. B.; DYBÅ, T. Transition from a Plan-Driven Process to Scrum – A Longitudinal Case Study on Software Quality. 2010.

LIMA, G. R. de C. Benefícios das metodologias ágeis no gerenciamento de projetos de Tecnologia da Informação. 2015.

LINA, Z.; DAN, S. Research on Combining Scrum with CMMI in Small and Medium Organizations. 2012.

MAFRA, S.; TRAVASSOS, G. Estudos Primários e Secundários apoiando a busca por evidência em Engenharia de Software. 2006.

MARIA, R. E.; RODRIGUES JUNIOR, L. A.; PINTO, N. A. ScrumS - A Model for Safe Agile Development. 2015.

MCBREEN, P. *Quality Assurance and Testing in Agile Project*. 2003. Disponível em: <http://www.mcbreen.ab.ca/talks/CAMUG.pdf>. Acesso em: 26/12/2017.

MCCALL, J. A.; RICHARDS, P. K.; WALTERS, G. F. *Factors in Software Quality*. [S.l.: s.n.], 1977.

MELO, C. de O. et al. *Métodos ágeis no Brasil*: Estado da prática em times e organizações. 2012. Disponível em: <http://www.agilcoop.org.br/files/metodos_ageis_brasil_estado_da_pratica_em_times_e_organizacoes.pdf>. Acesso em: 27/02/2018.

MNKANDLA, E.; DWOLATZKY, B. Defining Agile Software Quality Assurance. 2006.

MOHAGHEGHIA, P.; APARICIO, M. E. An industry experience report on managing product quality requirements in a large organization. 2017.

OTHMANE, L. ben; ANGIN, P.; BHARGAVA, B. Using Assurance Cases to Develop IterativelySecurity Features Using Scrum. 2014.

POPPENDIECK, M.; POPPENDIECK, T. *Lean Software Development*: An Agile Toolkit. [S.l.]: Addison Wesley, 2003.

QASAIMEH, M.; ABRAN, A. Investigation of the Capability of XP to Support the Requirements of ISO 9001 Software Process Certification. 2010.

RAFIQUE, Y.; MISIC, V. B. The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis. 2013.

RAHAYU, P. et al. Applying Usability Testing to Improving Scrum Methodology in Develop Assistant Information System. 2016.

RINDELL, K.; HYRYNSALMI, S.; LEPPANEN, V. Busting a Myth: Review of Agile Security Engineering Methods. In: . [S.l.: s.n.], 2017.

RISING, L.; JANOFF, N. S. The Scrum Software Development Process for Small Teams. 2000.

ROBERT GRADY. *Practicalsoftware metrics for project management and process improvement*. [S.l.]: Prentice Hal, 1992.

SCHWABER, K. SCRUM Development Process. 1994.

SCHWABER, K. *Agile Project Management with Scrum*. [S.l.]: Microsoft Press, 2004.

SFETSOS, P.; STAMELOS, I. Empirical Studies on Quality in Agile Practices: A Systematic Literature Review. *Seventh International Conference on the Quality of Information and Communications Technology*, 2010.

SILVA, F. S. et al. Using CMMI together with agile software development: A systematic review. 2015.

SILVA, T.; SILVEIRA, M.; MAURER, F. Usability Evaluation Practices within Agile Development. 2015.

SOARES, M. D. S. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software. 2004.

SOCIETY, I. C. IEEE Standard for Software Quality Assurance Processes. 2014.

SOFTEX. *MPS.BR - Melhoria de Processo de Software Brasileiro - Guia Geral.* [S.l.], 2016.

SOFTEX. *Modelos de Referência*. 2018. Disponível em: <https://www.softex.br/mpsbr/modelos/>. Acesso em: 25/01/2018.

SOMMERVILLE, I. Software engineering, 9th. In:____. [S.l.]: Addison-Wesley, 2011.

SUTHERLAND, J.; SCHWABER, K. The Scrum Guide™. 2017.

TARHAN, A.; YILMAZ, S. G. Systematic analyses and comparison of development performance and product quality of Incremental Process and Agile Process. 2013.

TOMMY, R. et al. Dynamic Quality Control in Agile Methodology for Improving the Quality. 2015.

VIEIRA, J. K. M. et al. Adaptation of the Scrum Adherent to the Level G of the MPS.BR Based on the Experience of the Implementers, Evaluators and Evaluated Companies. *6th Brazilian Workshop on Agile Methods*, 2015.

WELLS, D. *Pair Programming*. 1999. Disponível em: <http://www.extremeprogramming.org/rules/pair.html>. Acesso em: 26/12/2017.

YODER, J. W.; WIRFS-BROCK, R. QA to AQ Part Two: Shifting from Quality Assurance to Agile Quality: “Measuring and Monitoring Quality”. 2014.

YODER, J. W.; WIRFS-BROCK, R.; WASHIZAKI, H. QA to AQ Part Four: Shifting from Quality Assurance to Agile Quality: “Prioritizing Qualities and Making Them Visible”. 2015.

YOO, C. et al. An Integrated Model of ISO 9001:2000 and CMMI for ISO Registered Organizations. 2004.

# Apêndices

# Protocolo da Revisão Sistemática

## Introdução

A revisão sistemática trata-se de um tipo de investigação focada em questão bem definida, que visa identificar, selecionar, avaliar e sintetizar as evidências relevantes disponíveis a um contexto específico (GALVÃO; PEREIRA, 2014). O processo de revisão sistemática deve seguir uma estrita e bem definida sequência de etapas, de acordo com um previamente definido. Os passos, as estratégias para conseguir as informações e o foco das questões são explicitamente definidas, assim outros pesquisadores podem reproduzir o mesmo protocolo e serem capazes de julgar a adequação dos padrões escolhidos (BIOLCHINI et al., 2005).

Algumas razões de realizar uma revisão sistemática são: sumarizar evidências sobre uma tecnologia; identificar lacunas em determinada área; e prover uma base para posicionamento em novas atividades de pesquisa (KITCHENHAM, 2007).

Um protocolo especifica os métodos que serão usados para empreender uma revisão sistemática. Segundo Kitchenham (2007), além das razões e objetivos da pesquisa, devem fazer parte do protocolo: as questões de investigação que a pesquisa pretende responder; as estratégias usadas para as pesquisas dos estudos primários, incluindo os termos usados, bibliotecas digitais, jornais e conferências; critérios de inclusão e exclusão dos estudos primários; procedimentos de avaliação da qualidade dos estudos selecionados; e estratégia de extração dos dados e síntese dos dados extraídos.

Assim, este documento apresenta o protocolo de uma revisão sistemática, cujo objetivo principal é investigar as práticas presentes em métodos ágeis que auxiliam o atingimento da qualidade de software e suas possíveis lacunas.

## Questão de pesquisa

Com o objetivo de investigar “investigar as práticas presentes em métodos ágeis que auxiliam o atingimento da qualidade de software e suas possíveis lacunas” uma questão de pesquisa foi criada para guiar a revisão e dividida em duas sub-perguntas.

- Como métodos ágeis de desenvolvimento de software atingem a qualidade de software?

**Sub-pergunta 1**: Quais práticas, estratégias, ferramentas e métodos utilizados no desenvolvimento de software utilizando métodos ágeis auxiliam o atingimento da qualidade de software e em quais áreas?

**Sub-pergunta 2**: Em quais áreas são observadas lacunas nos processos ou atividades de desenvolvimento de software utilizando métodos ágeis que comprometem a qualidade de software?

**Termos Chaves da Pesquisa**

A partir das perguntas formuladas anteriormente, os principais termos foram identificados. Como as bases de dados pesquisadas possuem a maioria dos resultados na língua inglesa, os termos usados foram traduzidos para o inglês.

Além disso, sinônimos são identificados e os termos chaves são pesquisados no plural e no singular (o caractere asterisco é aceito na maioria das bibliotecas virtuais e permite essa variação singular/plural). Os termos e sinônimos são descritos abaixo:

- Agile method;
- Scrum;
- Extreme Programming;
- XP;
- Lean software;
- TDD
- Quality
- Quality assurance;
- Agile quality;
- QA.

**Strings de busca**

Segundo Kitchenham (2007), as strings de busca são derivadas das estruturas das questões e as vezes adaptações são necessárias de acordo com as necessidades especificas de cada base de dados. As strings de busca foram geradas a partir da combinação dos termos chaves e sinônimos usando AND (e) e OR (ou), e possíveis mudanças de acordo com o engenho de busca. A string completa é a seguinte:

((“agile method*” OR “scrum” OR “extreme programming” OR “XP” OR “lean software” OR “TDD”) AND (“quality” OR “quality assurance” OR “agile quality” OR “qa”))

**Engenhos de busca**

As pesquisas iniciais dos estudos primários podem ser realizadas em bibliotecas digitais, mas pesquisadores da área de pesquisa também podem ser consultados para

a indicação de fontes de material mais adequado, além de listas de referências e a internet (KITCHENHAM, 2007). As fontes de pesquisa utilizadas para a busca dos estudos são listadas a seguinte:

- IEEExplore: <http://ieeexplore.ieee.org>
- ACM: <http://portal.acm.org/dl.cfm>
- Elsevier: <http://www.sciencedirect.com>

Outras fontes foram inicialmente consideradas, tais como Google Scholar e SpringerLink, mas foram excluídas da lista final de fontes de busca por não estarem presentes em outras revisões sistemáticas ou não terem sido recomendadas por especialistas.

Quando os estudos primários são obtidos, uma análise precisa ser feita para que sua relevância seja confirmada e trabalhos com pouca relevância sejam descartados. Tendo em vista isto, nas próximas seções, critérios de inclusão e exclusão são definidos para ajudar na análise desses trabalhos.

**Seleção dos Estudos**

Os estudos que fazem parte dessa pesquisa são: artigos de periódicos, revistas, conferências e congressos. Uma vez que estudos potencialmente candidatos a se tornarem estudos primários tenham sido obtidos, eles precisam ser analisados para que a sua relevância seja confirmada e trabalhos com pouca relevância sejam descartados (KITCHENHAM, 2007). Alguns critérios de inclusão e exclusão são definidos nas próximas seções:

**Critérios de inclusão**

A inclusão de um trabalho é determinada pela relevância em relação às questões de investigação, determinada pela análise do titulo, palavra-chave, resumo e conclusão. Os seguintes critérios de inclusão foram definidos:

- Artigos que respondam à questão de pesquisa;
- Estudos que apresentem primária ou secundariamente práticas, estratégias, técnicas ou métodos presentes no desenvolvimento de software utlizando métodos ágeis que auxiliem no atingimento da qualidade de software;
- Estudos que apresentem avaliação de qualidade de métodos ágeis de desenvolvimento de software;

- Estudos que apresentem primária ou secundariamente lacunas ou práticas deficientes no processo de desenvolvimento de software utilizando métodos ágeis, nas quais após preenchidas, melhoraria a qualidade de software.

**Critérios de exclusão**

A partir análise do titulo, palavras-chave, resumo e conclusão serão excluídos os estudos que se adequem às situações abaixo:

- Estudos claramente irrelevantes à pesquisa, de acordo com as questões de investigação levantadas;
- Estudos que não respondam à nenhuma das questões de pesquisa;
- Estudos duplicados;
- Artigos que não estejam escritos na língua inglesa;
- Artigos que não estejam no período de Janeiro de 2010 a Outubro de 2017;
- Estudos que não estejam livremente disponíveis para a consulta na web;
- Artigos de livros e resumos;
- Artigos incompletos.

**Processo de Seleção dos Estudos Primários**

Após a definição das questões de pesquisa, da estratégia usada para a busca dos estudos primários e dos critérios de inclusão e exclusão, o processo de seleção dos estudos primários é descrito abaixo:

- Dois pesquisadores inicialmente realizam a busca de acordo com a estratégia definida anteriormente para identificar os estudos primários em potencial baseados na leitura dos títulos e palavras-chave que a pesquisa retorna e excluem trabalhos claramente irrelevantes para as questões de pesquisa. Estudos irrelevantes serão descartados no início e nenhuma lista de estudos excluídos é mantida;
- Cada pesquisador chega então a uma lista de potenciais estudos primários. As duas listas são então comparadas e os pesquisadores chegam a uma única lista de potenciais candidatos. Se houver qualquer discordância na inclusão ou exclusão de um estudo, o mesmo deve ser incluído;

- A partir da lista unificada com os resultados da pesquisa de potenciais candidatos a estudos primários, todos os trabalhos são avaliados, através da leitura do resumo e conclusão, considerando-se os critérios de inclusão e exclusão, para então se chegar a uma lista final de estudos primários.

**Avaliação da Qualidade dos Estudos**

Em adição aos critérios gerais de inclusão e exclusão, é considerado crítico avaliar a qualidade dos estudos primários. Apesar de não existir uma definição universal do que seja qualidade de estudo, esta análise visa minimizar viés e maximizar validade interna e externa. Validade interna é a extensão na qual o projeto e a condução do estudo evita o viés. Validade externa é a extensão na qual os efeitos observados no estudo são aplicáveis externamente (KITCHENHAM, 2007).

Na fase de extração de dados, a qualidade da metodologia de cada publicação foi avaliada. As seguintes perguntas auxiliaram nessa avaliação:

1) O artigo expõe claramente seus objetivos e resultados?
2) O artigo expõe os métodos utilizados para a análise dos dados?
3) As conclusões são explícitas?
4) O artigo contribui ou responde parcialmente às perguntas de pesquisa?
5) Será que a publicação menciona possíveis ameaças à validade interna?
6) Será que a publicação menciona possíveis ameaças à validade externa?

**Estratégia de Extração dos Dados**

O objetivo desta etapa é criar formas de extração dos dados para registrar com precisão as informações obtidas a partir dos estudos primários (KITCHENHAM, 2007). Para apoiar a extração e registro dos dados e posterior análise, será utilizada a ferramenta StArt (http://lapes.dc.ufscar.br/tools/sta rt_tool), que é voltada especificamente para revisões sistemáticas. Com ela é possível aceitar ou rejeitar artigos, ler resumos diretamente do software, gerar tabelas e gráficos. A ferramenta será bastante útil para o gerenciamento das referências bibliográficas e síntese dos dados.

**Síntese dos Dados Coletados**

A síntese dos dados envolve agrupar e sumarizar os resultados dos resultados primários incluídos. Tabelas devem ser estruturadas de forma a destacar as semelhanças e diferenças entre os resultados do estudo (KITCHENHAM, 2007). Os dados extraídos dos estudos são organizados em tabelas através da ferramenta StArt, que permite a visualização de cada informação extraída em relação as demais.

### Documentação e Apresentação dos Resultados

A fase final de uma revisão sistemática envolve escrever os resultados da análise e divulgar os resultados aos interessados. Alguns tópicos necessários para a apresentação de uma revisão sistemática são: título; autores; resumo; background (justificativa da necessidade da revisão); questões da pesquisa; métodos da revisão; estudos incluídos e excluídos; resultados; discussão; e conclusões (KITCHENHAM, 2007).