Centro Universitário de Brasília
Instituto CEUB de Pesquisa e Desenvolvimento - ICPD

BRUNO LOPES RIBEIRO SILVA

QUALIDADE DE SOFTWARE NO DESENVOLVIMENTO UTILIZANDO METODOLOGIAS ÁGEIS

Brasília
2017

BRUNO LOPES RIBEIRO SILVA

QUALIDADE DE SOFTWARE NO DESENVOLVIMENTO UTILIZANDO METODOLOGIAS ÁGEIS

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB/ICPD) como pré-requisito para obtenção de Certificado de Conclusão de Curso de Pós-graduação *Lato Sensu* em Governança de Tecnologia da Informação

Orientador: Prof. Dr. Paulo Foina

Brasília
2017

BRUNO LOPES RIBEIRO SILVA

QUALIDADE DE SOFTWARE NO DESENVOLVIMENTO UTILIZANDO METODOLOGIAS ÁGEIS

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB/ICPD) como pré-requisito para obtenção de Certificado de Conclusão de Curso de Pós-graduação *Lato Sensu* em Governança de Tecnologia da Informação

Orientador: Prof. Dr. Paulo Foina

.

Brasília, 07 de dezembro de 2017.

Banca Examinadora

_____

Prof. Dr. Maurício Lyra

_____

Prof. Dr. Gilson Ciarallo

Dedico a minha esposa, Nathalia, por todo o carinho, apoio e conselhos e principalmente por acreditar na minha capacidade de superar os desafios de cada dia.

RESUMO

Observando a engenharia de software sobre o prisma histórico, percebemos como o desenvolvimento de software evoluiu ao longo dos anos. O uso adequado da metodologia de desenvolvimento com as corretas configurações sempre foi o desafio. Além disso, sempre foi necessário a observação prática sobre o que funciona e o que não funciona, e sobre quais fatores afetam as diferentes variáveis do desenvolvimento de software. A mudança mais notável nos últimos 15 anos foi a introdução do paradigma ágil. Como em qualquer área é necessário o entendimento de seus componentes e relacionamentos, como um estudo empírico que evidencie o funcionamento do método em situações reais. Esse trabalho tem como objetivo investigar o tema qualidade de software sobre o contexto do desenvolvimento ágil, ou seja, estudar quais métricas e ferramentas de controle e garantia de qualidade são empregadas nesse contexto. Para tanto, realizou-se um estudo empírico, tipo *survey*, sobre as atividades de controle e garantia de qualidade utilizadas no processo de desenvolvimento. Assim sendo, foi possível observar que a qualidade de software é tema importante durante todo o ciclo de desenvolvimento ágil por meio de atividades essenciais de garantia de qualidade, como revisão e refatoração. O ambiente ágil é motivador e fortalece o engajamento das equipes de desenvolvimento, fatores que influenciam diretamente na qualidade final dos produtos.

Palavras-chave: Àgil. Qualidade. Desenvolvimento. Software.

ABSTRACT

Looking at software engineering over the historical prism, realize how software development has evolved over the years. The correct use of the development methodology with the correct settings has always been the challenge. In addition, practical observation has always been needed on what works and what does not, and on what factors affect the different variables of software development. The most notable change over the last 15 years has been the introduction of the agile paradigm. As in any area, it is necessary the understanding of its components and relationships, as an empirical study that evidences the operation of the method in real situations. The purpose of this paper is to investigate the quality of software on the context of agile development, that is, to study which metrics and tools of quality assurance and control are employed in this context. Therefore, it was conducted an empirical study, a survey, about the activities of control and assurance of quality used in the development process. Therefore, it was possible to observe that software quality is an important theme throughout the agile development cycle through essential quality assurance activities such as revision and refactoring. The agile environment is motivating and strengthens the engagement of development teams, factors that directly influence the final quality of the products.

Key words: Agile. Quality. Development. Software.

LISTA DE FIGURAS

LISTA DE SIGLAS

ASD	*Adaptative Software Development*
CMM	*Capability Maturity Model*
CMMI	*Capability Maturity Model Integration*
DSDM	*Dynamics Systems Development Methods*
FDD	*Feature-Driven Development*
IBM	*Internacional Business Machines*
IEC	*Internacional Electrotechnical Commission*
IEEE	*Institute of Electrical and Electronics Engineers*
ISO	*Internacional Organization for Stadardization*
GQM	*Gol,Question and Metric*
MPS.BR	*Melhoria do Processo de Software Brasileiro*
RTF	*Running Testing Features Metric*
RUP	*Rational Unified Process*
TDD	*Test-driven development*
TI	*Tecnologia da Informação*
XP	*Extreme Programming*

SUMÁRIO

INTRODUÇÃO

Apresentação

O foco da Engenharia de Software é aumentar a produtividade no processo de desenvolvimento de software e propiciar qualidade ao produto resultante desse processo. Pressman (2010) afirma que o principal objetivo da matéria é a redução dos custos e dos esforços geralmente exigidos nas várias fases do desenvolvimento e da manutenção do software. Nesse sentido, metodologias têm pontos em comum e tendem para a reutilização de mecanismos, experiências e técnicas bem-sucedidas.

A indústria vem adotando e definindo diferentes processos para o desenvolvimento de software, contudo essas metodologias, no desenvolvimento de software, costumam lançar mão de etapas repetitivas. Somerville (2013) cita que o domínio das aplicações desenvolvidas e as características das empresas influenciarão na escolha do método. Não existe um modelo padrão que atenda a todas as aplicações e tipos de negócio.

Tendo em vista essa heterogeneidade no processo de desenvolvimento de software, despertou-se, tanto em ambientes de pesquisas, como em ambientes corporativos, o interesse pela utilização de métodos ágeis. Tais métodos focam-se na contínua interação entre os desenvolvedores e os clientes, tentando alavancar a probabilidade de que um software atenderá às necessidades e às mudanças de requisitos do sistema, ao mesmo tempo em que a entrega do software será rápida. De acordo com Fowler (2005), prazos de entregas cada vez mais curtos, somados com a complexidade das metodologias tradicionais de desenvolvimento, reforçam o resultado como a principal motivação para o surgimento dos modelos ágeis.

Qualidade é um tema de crescente importância na indústria de software. Existem várias perspectivas para qualidade de software, que focam em diferentes aspectos. A literatura mostra que além de diversas definições, qualidade possui significados diferentes para diferentes pessoas, tais como clientes, desenvolvedores, gestores. Gillies (2002) afirma que “qualidade é transparente quando presente, porém, facilmente percebida em sua ausência”. Crosby (1992) afirma que “qualidade é produzir conforme foi especificado, atendendo o que o cliente deseja”.

Confirmando as visões acima, a ISO 9126-1, que fala sobre as características da qualidade de software, afirma que a qualidade de um sistema de software pode ser entendida de diversas formas e utilizando diferentes abordagem. A norma estabelece um modelo de qualidade com os seguintes componentes:

- Processo de desenvolvimento, onde a sua qualidade afeta a qualidade do produto de software gerado e é influenciado pela natureza do produto desenvolvido.
- Produto, compreendendo os atributos de qualidade do produto (sistema) de software. Estes atributos de qualidade podem ser divididos entre atributos internos e externos. Estes se diferenciam pela forma como são aferidos - interna ou externamente ao produto de software – e em conjunto compõem a qualidade do produto de software em si.
- Qualidade em uso, que consiste na aferição da qualidade do software em cada contexto específico de usuário. Esta é, também, a qualidade percebida pelo usuário.

Como mencionado, qualidade é algo subjetivo que não pode ser facilmente mensurado, buscando quantificar a qualidade dos processos de softwares, foram criados alguns modelos de qualidade, que recomendam processos, atividades e métricas para medir a qualidade de um software. Um do mais conhecido é o CMM - Capability Maturity Model que tem seu foco nos processos de software por entender que a qualidade de um sistema de software é fortemente influenciada pela qualidade do processo utilizado para desenvolvê-lo e mantê-lo. O CMM desenvolveu-se e evoluiu para CMMI - Maturity Model Integration que é um processo gradual, levando as corporações ao aprimoramento constante, buscando incentivar o desenvolvimento de soluções próprias para os problemas de desenvolvimento de software.

Carosia (2004) cita que atividades como garantia de qualidade e métricas como confiabilidade, eficiência, segurança e manutenibilidade são rotinas diárias do desenvolvimento de um projeto em grandes empresas. A garantia da qualidade é definida por Pressman (2010) como uma estrutura composta por responsabilidades, procedimentos, processos e recursos para implementar a gestão da qualidade no produto e/ou processo. Tais sistemas têm como meta fornecer apoio às organizações

para garantia da satisfação de seus clientes. Conforme Pressman (2010) esses sistemas executam uma série de atividades que passam por todo o ciclo de vida do produto.

Justificativa

Chamadas de pesadas, ou orientadas ao planejamento, as metodologias tradicionais surgiram em um contexto completamente distinto do modelo atual, no qual o desenvolvimento de software era feito para mainframes e terminais burros. Assim o custo de alteração era muito alto, logo o planejamento e documentação do software era feito antes da implementação.

Buscando se adaptar ao novo contexto do mundo, as metodologias ágeis, propõe uma nova abordagem para o desenvolvimento, eliminando gastos excessivos com documentação e burocracia, enfatizando a comunicação, colaboração com o cliente e as atividades que trazem valor imediato na produção de software com qualidade.

Sempre existiu a necessidade de se introduzir qualidade no processo de produção de Software o que levou ao estabelecimento de normas e modelos de qualidade que são guias para aumentar, medir e garantir a qualidade de software. Dentre os quais podemos citar o CMM, CMMI, ISO 9000 ISO/IEC 12207.

Uma revisão junto a teoria atual nos mostra que os modelos atuais de qualidade de software foram desenvolvidos utilizando os modelos tradicionais de desenvolvimento de software. Sendo assim, a compatibilidade desses modelos com a metodologia ágil é muito difícil. Dessa forma, é necessário o aprofundamento da matéria, tanto na Academia, quanto em ambientes corporativos.

Objetivos

Objetivo Geral

Identificar fatores que contribuem para a qualidade de software no desenvolvimento ágil

Objetivos Específicos

- Mapear trabalhos relativos às atividades de garantia de qualidade;
- Pesquisar os fatores que influenciam a baixa aderência aos modelos tradicionais de qualidade no desenvolvimento de software ágil;
- Analisar os fatores que contribuem para a qualidade de software em desenvolvimento ágil;
- Pesquisar quais são as métricas, processos e ferramentas utilizadas no desenvolvimento ágil que contribuem para qualidade;

Metodologia

O mapeamento dos fatores que contribuem para a qualidade de software no desenvolvimento ágil na perspectiva desse trabalho foi realizado a partir da análise dos dados coletados por questionário aplicado em pessoas envolvidas no desenvolvimento de software utilizando metodologias ágeis. Para a construção deste instrumento de pesquisa, utilizou-se de revisão bibliográfica com foco na identificação de atividades, ferramentas e métricas utilizadas na metodologia ágil de desenvolvimento de software que de fato contribuem para melhoria ou garantia da qualidade. A análise e discussão do conteúdo levantado foi confrontado com essa mesma referência e as respectivas conclusões encontram-se nos próximos tópicos.

Estrutura da Monografia

O trabalho foi estruturado da seguinte forma: No capitulo um são apresentados os conceitos de qualidade, qualidade de software e as metodologias desenvolvimento de software tradicionais e ágeis. Nesse capítulo também se apresenta uma síntese dos principais métodos ágeis. O capítulo 2 mostra a situação dos modelos ágeis e qualidade. No capítulo 3 é apresentado o estudo de campo e a coleta dos dados, feita com o auxílio de um questionário, com profissionais de TI que utilizam métodos ágeis. A aplicação do questionário tem por objetivo demonstrar a relação de desenvolvedores que utilizam métodos ágeis com as atividades de controle e garantia de qualidade.

1 REFERENCIAL TEÓRICO

1.1 Qualidade

O termo qualidade vem do latim *qualitate,* e pode significar o grau de utilidade esperado ou adquirido de um produto ou serviço. O conceito de qualidade é subjetivo e está conectado com a percepção, a necessidades e resultados esperados por cada indivíduo. A percepção de qualidade é influenciada por diversos fatores, como cultura, modelos mentais, tipo de produto ou serviço prestado. As necessidades e expectativas influenciam diretamente a percepção da qualidade.

Na literatura existem diferentes definições para qualidade, cada especialista irá definir de acordo com sua área de atuação. Somado a isso, qualidade tem significados diversos para diferentes pessoas como usuários, consumidores e gerentes. O dicionário on-line Aurélio define qualidade como: a) maneira de ser boa ou má de uma coisa; b) Atributo, virtude, valor.

Os autores Philip Crosby, Joseph Juran e Edward Deming (Gilles, 2002) nos trazem importantes perspectivas sobre qualidade. Segundo Crosby (1980) qualidade é a conformidade com os requisitos, que devem estar bem definidos e serem medidos constantemente. Juran define qualidade como aptidão para o uso (JURAN et al., 1988), já Deming (1982) conceitua como um grau previsível de uniformidade e confiabilidade a um custo baixo e adequado ao mercado.

A definição de qualidade segundo a norma ISO/IEC 8402 é “a totalidade das características de uma entidade que lhe confere a capacidade de satisfazer necessidades explícitas e implícitas”. Podemos conceituar como necessidades explícitas as definições expressas de requisitos propostas pelo fabricante e as necessidades implícitas como as que mesmo não expressas como requisitos, são necessárias ao usuário.

Já para o consumidor, qualidade é a percepção de valor que determinado produto irá trazer. Guaspari (1994) cita que o consumidor não compra o produto, mas a garantia que o produto irá atender suas expectativas.

1.2 Qualidade de Software

A IEEE (1990) define qualidade de software como: a) o grau que cada sistema, componente ou processo atende aos requisitos específicos e; b) o grau em que cada sistema, componente ou processo atende as expectativas ou necessidades do usuário ou consumidor.

Tal definição vai ao encontro do que foi definido por Crosby – conformidade com os requisitos e Juran – aptidão para o uso.

Assim como a definição de qualidade varia de acordo com o contexto ou pessoa que avalia, cada *Stakeholder* também definirá sua visão sobre o assunto: O usuário verá a qualidade como o atendimento as suas necessidades; o software responde de forma rápida e eficiente com baixo custo de operação; o designer de software avalia qualidade se o software foi bem especificado, se está tecnicamente correto ou se foi bem documentado (GILLIES, 2002).

A qualidade de software pode ser definida em dois níveis, de acordo com Kan (2002): com “q minúsculo” e com “Q maiúsculo”. O primeiro representa a qualidade do produto em nível de atributos como taxa de defeito e a sua confiabilidade. O segundo nível se refere a qualidade do processo e a satisfação do cliente. Nesse contexto, mesmo que o produto final atenda os requisitos especificados, pode ser o que o cliente fique satisfeito, logo qualidade deve ser vinculada aos requisitos definidos pelo cliente.

Outro ponto de vista vem de Gillies (2002) ao afirmar que qualidade são pessoas. Ele explica que isso ocorre porque qualidade é determinada por pessoas que enfrentam o problema a ser resolvido pelo software, pessoas irão definir o problema, pessoas encontrarão a solução, pessoas irão usar o sistema e pessoas farão o julgamento sobre a qualidade.

Gilles afirma que é mais difícil se analisar a qualidade de um software quando comparada com outras áreas pois: a) o software não tem uma existência física, é uma entidade conceitual; b) normalmente, o cliente não sabe exatamente o que ele precisa; c) os pedidos de mudança, pelo cliente, ocorrem a todo tempo; d) a mudança de software e hardware ocorrem frequentemente e rapidamente e; e) as expectativas dos clientes são sempre muito altas. Entretanto, para auxiliar nas

medições, devido à complexidade de se avaliar a qualidade de software, existem métricas feitas por programas de computadores.

É possível, quando o ponto a ser avaliado é tangível, definir dois tipos de qualidade. Aquela que opera em conformidade com as especificações e a qualidade de projeto, que se refere as características que os projetistas planejam para um item, como, indicadores de carga e tolerância. Outra definição é expressa por Pressman (2010) como o nível com que um produto atende às suas especificações.

Como dito anteriormente, é possível perceber que a definição de qualidade pode ser modificada ou ampliada. Ainda segundo Pressman (2010), a qualidade deve focar-se em três pontos.

1. Requisitos de software: são a base para se realizar medidas de qualidade. A falta de conformidade com as especificações é um problema de falta de qualidade.
2. Padrões especificados: definem um conjunto de critérios de desenvolvimento que guiam a forma como o software será definido. A não conformidade com as definições desse padrão resultará em falta de qualidade.
3. O software deve obedecer requisitos que muitas vezes estão implícitos, mas são esperados de um software profissional, como manutenibilidade. A ausência desses itens torna a qualidade de um software contestável.

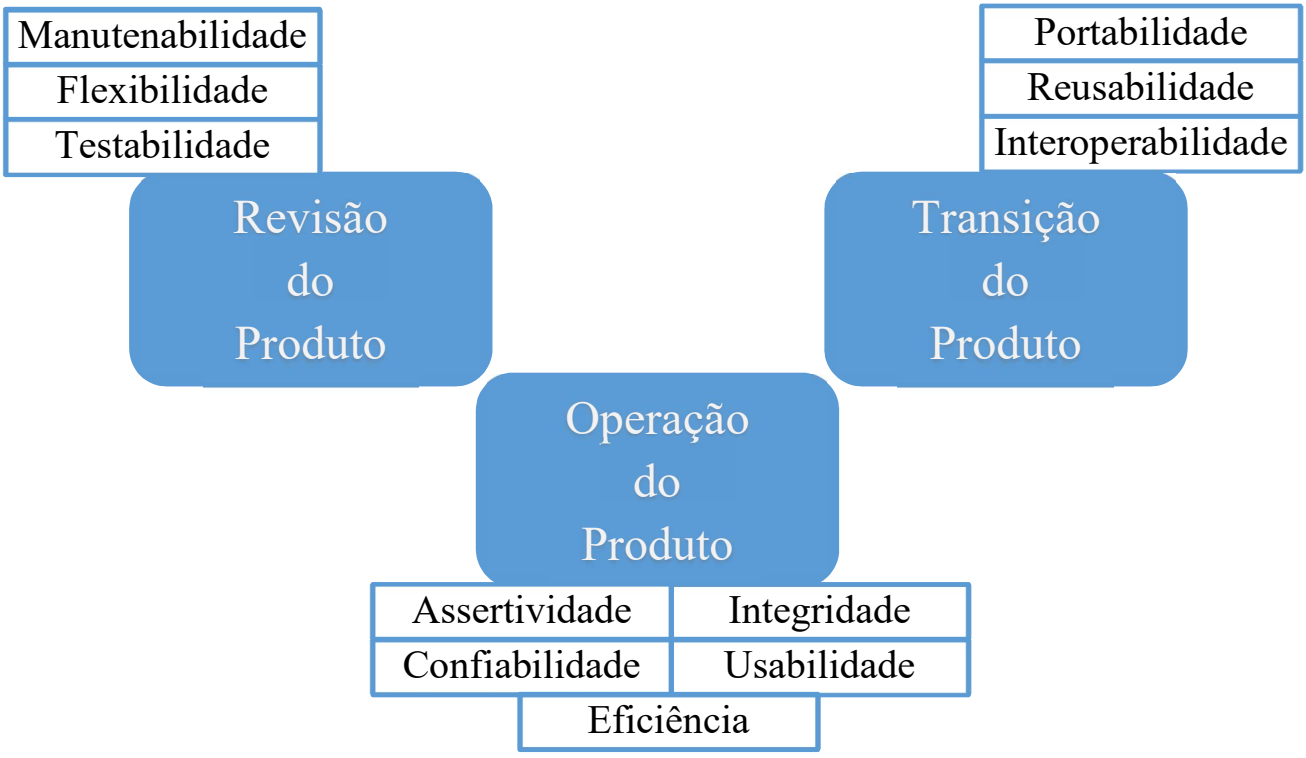
O processo de desenvolvimento influência diretamente diversas características. Podemos detalhar em vários níveis essas características, criando, assim, um conjunto de atributos que irão descrever a qualidade de um produto de software (ROCHA; MALDONADO; WEBER, 2001).

1.3 Modelos de Qualidade de Software.

Os modelos de qualidade de software são utilizados quando é necessário avaliar a qualidade em diferentes situações. Existem diversos modelos para a avaliação da qualidade, mas que em sua maioria trazem um conceito hierárquico, em que as características, que contribuem para a qualidade do software, são apresentadas de forma hierárquica. Como os apresentados por McCall(1977) e Boehm (1978).

O modelo de qualidade apresentado por McCall (1977) foca seu uso durante o processo de desenvolvimento. Esse esquema identifica três áreas que o software atua: Operação do Produto, Revisão do Produto e Transição do Produto. A figura 1 mostra os critérios adotados por McCall em cada área. A Tabela 1 mostra a descrição de cada critério que está na figura 1. Nessa representação, cada critério foi escolhido para refletir as visões dos usuários e desenvolvedores, com a intenção de criar um elo entre as duas visões.

Figura 1: Critérios de Qualidade de McCall



Fonte – McCall (1997)

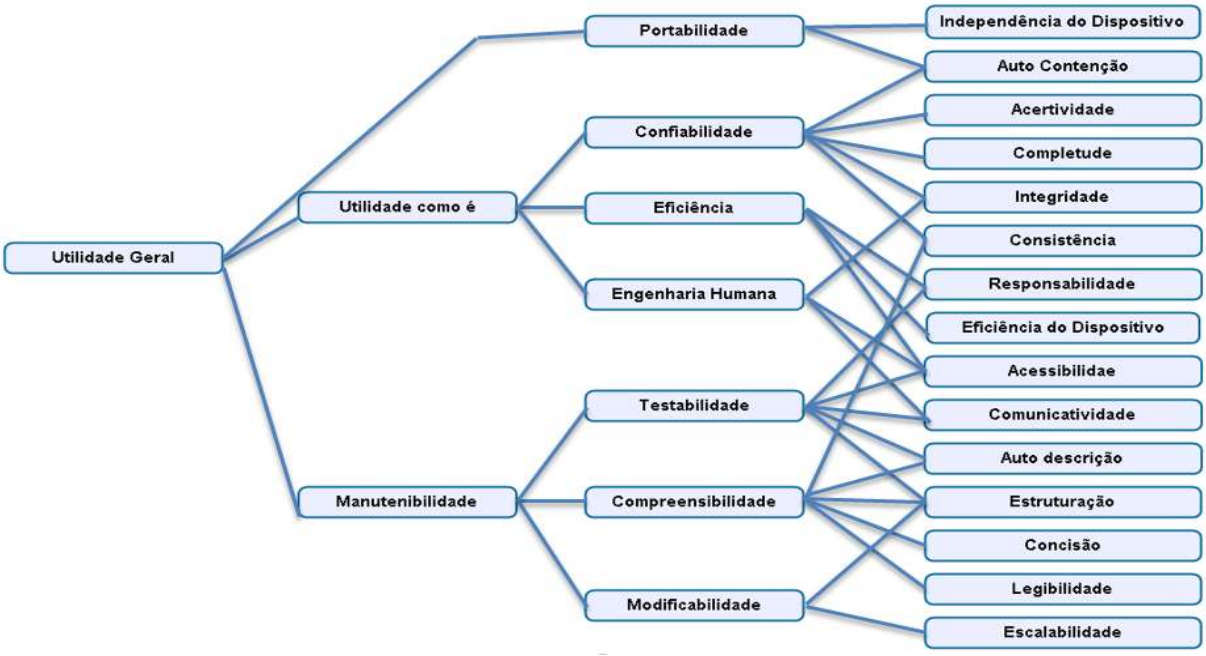
Tabela 1: Critérios de Qualidade de McCall

Fatores de Qualidade de Software	**Descrição**
Assertividade	Medida que afere o nível com que um software satisfaz suas especificações
Confiabilidade	Medida que afere se o sistema realiza suas atividades sem falhas
Eficiência	Quantidade de recursos computacionais utilizados por um sistema para executa suas atividades
Integridade	Medida que indica se os dados estão coerentes e precisos ao longo do sistema
Usabilidade	A facilidade em se utilizar o software
Manutenibilidade	O esforço necessário para se encontrar e corrigir um problema após o software estar em uso
Flexibilidade	Esforço necessário para se modificar o sistema
Testabilidade	A facilidade em se testar o programa e garantir que ele está livre de erros
Portabilidade	O esforço necessário para portar o sistema de uma configuração de hardware ou software para outra
Reusabilidade	A facilidade de se reutilizar o software em diferentes contextos
Interoperabilidade	O esforço necessário para integrar o sistema com outros sistemas

FONTE: MacCall. (1977)

Em 1976 Boehm sugeriu outro modelo hierárquico para a aferição da qualidade. O modelo proposto estende o proposto por McCall e acrescenta novos critérios de qualidade. Como pode ser visto na figura 2.2, o modelo possui três níveis de hierarquia: O nível mais alto da estrutura representa as utilizações feitas pelo sistema. O nível mais baixo estabelece uma série de características que combinadas formam o nível intermediário de características que definem os atributos necessários para o alcance da qualidade (BOEHM et al., 1978).

Figura 2: Modelo de qualidade de Boehm



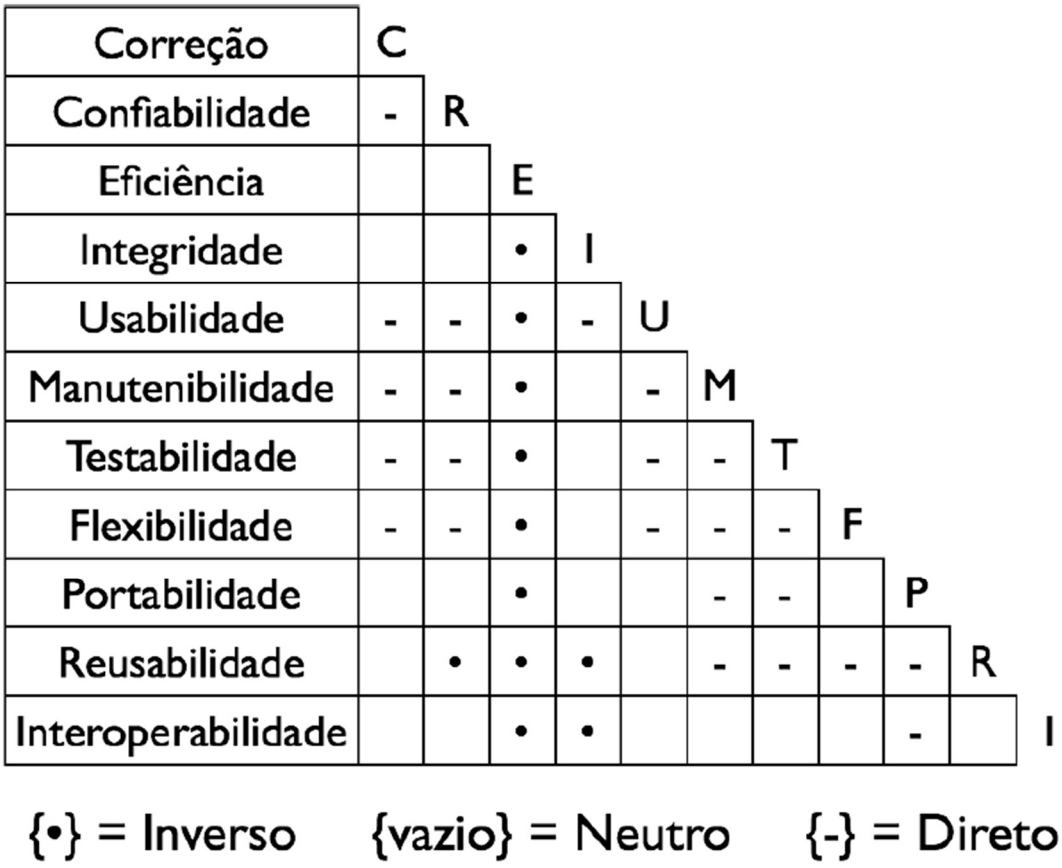
Fonte – Boehm et al (1978)

Os modelos de qualidade de MaCall (1979) e Boehm et al (1978) especificaram a qualidade até o nível de métricas, mas indicadores individuais não fornecem uma visão geral da qualidade do produto e portando esses indicadores devem ser analisados. Em 1987 Perry apresentou um relacionamento entre esses indicadores, como mostrado na figura 2.3. Gilles (2002a) criticou a proposta de Perry (1987) pelo fato dele assumir que os relacionamentos entre os indicadores são cumulativo, o que nem sempre é verdadeiro.

A ISO/IEC 9126, publicada pela ISO (*Internacional Organization for Standardization)* e o IEC (*Internacional Electrotecnical Comission*), é o padrão que define modelos de qualidade para produtos de software, características de qualidade

‘de software e métricas relacionadas. Esse conjunto permite avaliar e criar metas para a qualidade de um produto de software.

Figura 3: Relações entre critérios



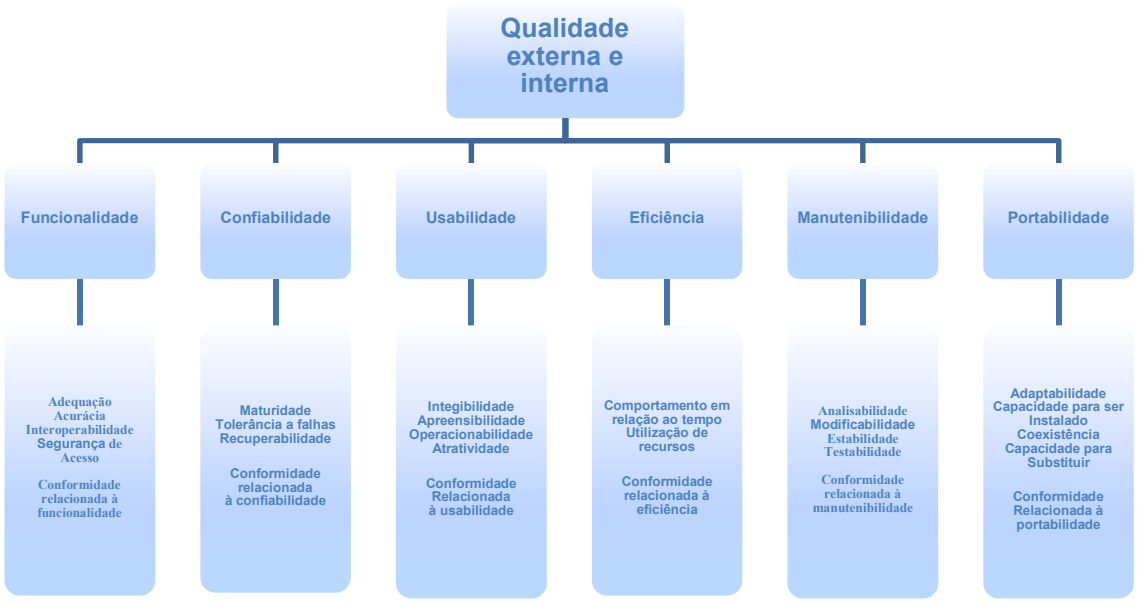
Fonte – Perry (1987)

O padrão ISO/IEC 9126 é dividido em duas partes. A primeira é aplicável para se modelar a qualidade interna e externa de um produto de software, enquanto a segunda se foca em modelar a qualidade em uso de um produto de software.

Normalmente, através de revisões de documentos de especificação, verificação de modelos ou pela análise estática do código-fonte se consegue avaliar a qualidade interna. A qualidade externa serão as propriedades de software que interagem com o meio ambiente. A qualidade em uso se refere a qualidade que será percebida pelo usuário final.

A norma ISO/IEC 9126 define um modelo para caracterizar a qualidade interna e externa. O modelo é baseado em seis características conforme a figura 2.4 mostra.

Figura 4: Modelo de qualidade interna e externa da ISO/IEC 9126



Fonte – ISO/IEC 9126

Esse modelo é baseado nas seguintes características: efetividade, produtividade, segurança, satisfação. São definidas métricas para aferir características ao longo da ISO/IEC 9126. Essas métricas são abstratas tornando-as aplicáveis para vários tipos de produtos de software, entretanto é necessário que sejam adequados de acordo com o tipo de aplicação.

1.4 Controle e Garantia de Qualidade.

O controle de qualidade é um conjunto de inspeções, revisões e testes utilizados no decorrer do processo de desenvolvimento de software, buscando garantir que o produto final cumpra os requisitos especificados. Ele fornece feedbacks ao processo de desenvolvimento. O controle de garantia une medições e feedbacks que trazem informações que podem auxiliar na melhoria do processo de desenvolvimento de software quando necessário. As atividades do controle de qualidade podem ser automatizadas, manuais ou ambas. A base do controle de qualidade é que todos os produtos devem possuir especificações definidas e mensuráveis e, portanto, que podem ser comparadas com a saída de cada atividade do processo de desenvolvimento.

O dicionário IEEE (IEEE,1990) define que a garantia da qualidade é resultado do planejamento sistemático de todas as ações necessárias para adequar a confiança de que um item ou produto está conforme foi estabelecido pelos requisitos técnicos funcionais. Ou seja, um conjunto de atividades desenhadas para avaliar o processo em que o produto é desenvolvido ou manufaturado. O objetivo da garantia de qualidade é abastecer a gestão de um projeto com os dados sobre a qualidade de um produto.

As atividades de garantia de qualidade serão feitas, de forma integrada, pela equipe de desenvolvimento que deve realizar o trabalho técnico e pela equipe de garantia de qualidade responsável pelo planejamento de qualidade, por manter ativos de projetos para análise, realizar análises de qualidade e reportar os resultados. O papel da equipe de garantia de qualidade é de fornecer suporte a equipe de desenvolvimento para que se possa alcançar um produto de alta qualidade. No quadro 1 temos as principais atividades desenvolvidas pela equipe de garantia de qualidade.

Quadro 1: Atividades da equipe de garantia de qualidade

Atividade	**Descrição**
Preparar plano de qualidade	Identificar as avaliações a serem realizadas, auditorias e revisões, padrões de projeto a serem seguidos, procedimentos para reportar erros, documentos a serem produzidos como provas das avaliações de qualidade e seus resultados, e forma e data de realização de feedbacks para o time de desenvolvimento. O plano deve ser elaborado em conjunto com o plano do projeto e revisado por todos os *Stakeholders*.

Participar no projeto de descrição do processo de software	Revisar o processo, verificar sua compatibilidade com as políticas organizacionais, padrões internos de desenvolvimento, padrões externos de desenvolvimento e outros impactos no planejamento do projeto. Identificar e documentar os desvios do processo e verificar se os ajustes necessários foram realizados.
Auditar produtos para verificar conformidade com o projeto	Revisar o produto criado, identificando e documentando desvios e as correções realizadas para os mesmos. Periodicamente devem ser entregues relatórios com os resultados das avaliações aos gerentes.
Garantir que os desvios sejam documentados e geridos de acordo com o processo utilizado	Todo desvio encontrado no plano de projeto, na descrição do processo, na aplicação de padrões devem ser reportados, e acompanhado até a resolução do problema.
Controlar Mudanças	Coordenar as mudanças realizadas no projeto. Realizar coleta e análise e ajuste das métricas de software quando necessário.

Fonte – O autor (2017)

As atividades desenvolvidas pela equipe de garantia de qualidade são de extrema importância, por isso as subseções abaixo abordam com detalhes algumas atividades consideradas importantes, como revisão, refatoração e testes.

1.4.1 Revisões

De acordo com o IEEE (IEEE,1991) revisão é o processo ou reunião na qual um produto ou um conjunto de produtos é apresentado à equipe de projeto, gerentes, usuários, consumidores ou partes interessadas para comentários e aprovações. São divididas em: revisão de código, planejamento, formal, de requisitos ou de conclusão dos testes.

Abran e Moore (2004) definem cinco tipos de revisão ou auditorias que são descritas no IEEE1028-97(padrão de revisão de software), e apresentados no Quadro 2.

Quadro 2: Padrões para revisão

Tipos de Revisão	**Descrição**
Revisão de Gestão	Monitora o progresso dos projetos, determina o estado dos planos e cronogramas, verifica os requisitos e sua alocação no sistema, válida a efetividade da abordagem de gestão utilizada para atingir o objetivo.
Revisões Técnicas	Para avaliar o produto de software de acordo com o seu propósito de uso, o resultado deve prover o gerenciamento do projeto com evidências que confirmem que o produto está de acordo com as especificações e aderente aos padrões e que existe o controle de mudanças.
Inspeções	Técnica de análise estática que requer exame visual do desenvolvimento do produto para detectar erros, violações dos padrões de desenvolvimento e outros problemas. Divide-se em: inspeção do código e inspeção do projeto. Tem como principal objetivo identificar e detectar produtos de software com anomalias.

Walk – Throughs	Técnica de análise estática em que o projetista líder ou programador líder do time de desenvolvimento mostra partes do código ou documentação para os desenvolvedores e/ou partes interessadas no projeto. Os participantes discutem sobre possíveis erros, violações dos padrões de desenvolvimento, etc. Tem como principal objetivo avaliar o produto de software. É menos formal que a inspeção.
Auditorias	Fornece uma avaliação independente da conformidade dos produtos de software e processos, a aplicação de padrões regulamentários, planos e procedimentos.

Fonte – O autor (2017)

1.4.2 Refatoração

Refatorar é o processo de alterar o software, melhorando sua estrutura interna, sem alterar o seu comportamento externo. Pode ser aplicada em qualquer parte do produto de software ou de sua documentação. A refatoração, diferentemente de outras atividades da garantia de qualidade, é uma atividade executada pela equipe do projeto.

Fowler (2005) define refatoração com uma mudança na estrutura interna de um software que o torna mais legível e barato de modificar. Refatoração não é sinônimo de correção de problemas de um produto de software, não é o seu objetivo principal, mas ainda assim é uma atividade de extrema importância e bem vista por muitos gerentes e equipes de projetos (ABBAS, 2009).

1.4.3 Testes

As atividades de teste têm influência direta na qualidade um produto. O teste em sí não aumenta a qualidade de um sistema, mas serve de mecanismo de controle e aferição do alcance dos requisitos do software. Pressman (2010) afirma que as atividades de teste servem de evidência para confirmar que um software desempenha as funções de acordo com suas especificações. Testes contribuem para o aumento da confiança, já que não existe a certeza que um produto de software está livre de erros. Os indicadores, por sua vez, demonstrarão que o software executa as

funcionalidades esperadas de forma aceitável e com o mínimo de qualidade (HARROLD, 2000)

As atividades de teste podem ser divididas em (DELAMARO, MALDONADO, JINO. 2007):

- **Teste de Unidade:** Busca defeito de lógica e implementação, presentes na menor unidade de software, foca em garantir o bom funcionamento do software (MYERS et al., 2004)
- **Teste de Integração:** Executado na fase de integração de módulos, foca em revelar defeitos de comunicação e interfaces (MALDONADO, 1991)
- **Teste de Sistema:** Realizado após a fase de integração, tem o objetivo de avaliar o sistema como um todo, verifica se as funcionalidades descritas nas especificações do sistema foram corretamente implementadas (PRESSMAN, 2010).

Existe ainda um teste utilizado durante a fase de manutenção do software, comumente conhecido com teste de regressão (DELAMARO, MALDONADO, JINO. 2007)

Devido ao seu alto custo computacional e financeiro, o teste completo é impraticável (Myers et al., 2004). Logo, buscar selecionar um subconjunto de casos de testes, que possuam alta probabilidade de revelar defeitos, é uma tarefa de extrema relevância (Myers et al., 2004) para o sucesso do produto final.

Regras, conhecidas como critérios de testes, são estabelecidas para cada seleção desse subconjunto. Para cada subdomínio, são selecionados casos de teste por meio dos critérios de teste (DELAMARO, MALDONADO, JINO. 2007). De modo geral são definidos requisitos de teste que, quando satisfeitos, determinaram a inclusão de casos de teste em um subdomínio.

As técnicas de teste, aplicando os conceitos acima, podem ser divididas em três classes (DELAMARO, MALDONADO, JINO. 2007). A diferença entre cada técnica é o tipo de informação coletada par se classificar os subdomínios.

- **Teste Funcional:** Baseado na especificação funcional do sistema. São considerados apenas as entradas, saídas e estado do programa, não se

exige do testador o conhecimento do código fonte do software. Tem como meta encontrar diferenças entre o comportamento atual de parte do sistema e sua especificação. Conhecido como teste de caixa-preta.

- **Teste Estrutural:** Os casos de teste são derivados a partir da estrutura interna do sistema (PRESSMAN, 2010). Os critérios podem ser divididos ainda em fluxo de controle e em fluxo de dados. Para o fluxo de controle são utilizadas informações referentes a execução do programa, como comandos e desvios. Quando utilizamos fluxo de dados verificam-se os caminhos que envolvam as definições de uso de variáveis. Tais critérios visam investigar os modos nos quais os valores são associados as variáveis do programa e como essas associações afetam a sua execução (ZHU; HALL; MAY, 1997). Conhecido com teste de caixa-branca.

- **Teste Baseado em Defeitos:** tem como principal fonte de informação os erros mais frequentemente cometidos por programadores no processo de desenvolvimento de software. O critério mais conhecido desta técnica é a análise de mutantes (BUDD, 1981; DEMILLO; LIPTON; SAYWARD, 1978).

1.5 Avaliando a Qualidade do Software

Avaliar a produtividade e qualidade do software, por muitos anos, foi tão difícil que somente corporações de grande porte, como a IBM, se arriscavam a fazer. Tal problema hoje sofre um viés cultural, existe a resistência humana em acreditar que medidas sempre irão depor contra (JONES, 1991).

Jones define que para se obter informações sofre a qualidade do software é necessário a coleta de três tipos de dados:

1. **Dados Brutos:** Geralmente possuem pouca ou nenhuma subjetividade. Os elementos chave dos dados brutos são: quantidade de membros da equipe, esforço gasto na tarefa, duração do cronograma, documentos, códigos, volume dos casos de teste, número de defeitos. Apesar de poder ser mensurada com alta precisão, na teoria, organizações geralmente não possuem competência necessária para a coleta.

2. **Dados Subjetivos:** Principal fonte de informação que explica variações na produtividade e qualidade. Os elementos chave para esse tipo de dados são: habilidades e experiência da equipe, compressão ou restrições ao cronograma, estabilidade dos requisitos, satisfação do usuário, a *expertise* e cooperação dos usuários, adequação ao uso de ferramentas e métodos, estrutura organizacional, adequação ao espaço do escritório, valor percebido do projeto. Dados subjetivos são o tipo de informação, mas útil que pode ser coletado, apesar da dificuldade em sua coleta, pois envolve atividade intelectual associada com programas de medição.
3. **Dados Normalizados:** Métricas padronizadas são utilizadas com o objetivo de determinar quais projetos estão acima ou abaixo em termos de qualidade e produtividades. Linhas de código e pontos de função são alguns exemplos desses tipos de dados.

1.6 Métricas de Qualidade de Software

Gillles (1992) afirma que uma métrica é uma propriedade mensurável que indica se um ou mais critérios de qualidade estão sendo alcançados. Métricas podem ajudar a prever ou a descrever, dependendo da fase do ciclo de vida do desenvolvimento, o estado de um produto de software.

Kan (2002) classificou as métricas de software em três tipos: métricas do produto, métricas do processo e métricas do projeto. Além disso, Kan definiu que métricas de qualidade de software como um subconjunto de métricas de software baseadas na definição ampla de qualidade de software. As métricas sugeridas por ele são o tempo para falha, densidade do defeito, problemas dos consumidores e satisfação do cliente.

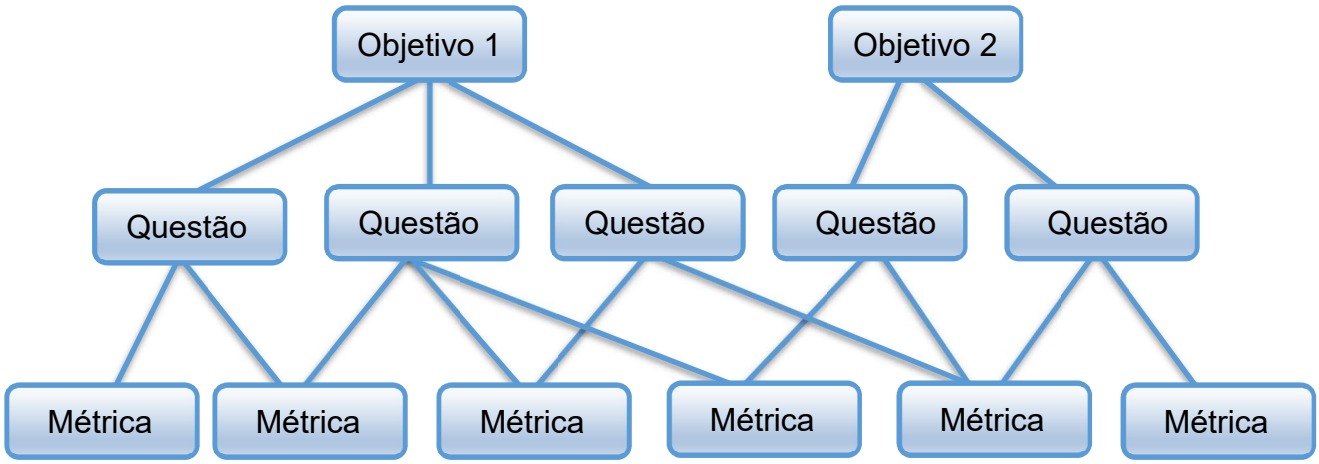
Watts (1987) enumerou 40 métricas que estão disponíveis na literatura. Apesar de parecer um bom número, a distribuição das métricas através dos critérios de qualidade é tendenciosa. Alguns critérios, como eficiência, adaptabilidade, interoperabilidade e reusabilidade, não possuem métricas, enquanto outras, como manutenibilidade e confiabilidade, possuem 18 e 12 respectivamente (Gilles, 1992).

Gilles (1992) definiu que a métricas disponíveis na literatura são limitadas pois:

- Muitas não podem ser validadas;
- Normalmente não são objetivas;
- Qualidade é relativa, não absoluta;
- Métricas não aferem a qualidade em sua totalidade.

Proposto por Basili, Caldeira e Rombach (1994) o modelo conhecido como GQM(*Gol/Question/Metric)* tenta relacionar de forma mais simples e direta os fatores de qualidade às métricas disponíveis. O modelo baseia-se na premissa que a empresa precisa definir suas metas e as metas de seus projetos, assim podendo mensurar a qualidade de seus produtos. A ideia base é refinar os objetivos por meio de perguntas. Essas perguntas devem ser refinadas, por meio de outras perguntas, até que se tornem objetivas para que se possa extrair uma métrica direta - objetiva ou subjetiva (ABBAS, 2009). Conforme visto na figura 5

Figura 5: Estrutura do modelo hierárquico GQM



Fonte – Basili, Caldeira e Rombach (1994)

1.7 Quem é o responsável pela garantia da qualidade do software

É importante saber quais são as atividades necessárias para se alcançar o mais alto nível de qualidade de software, como também é importante saber quem é o responsável por essas atividades e como alcançar os seus objetivos. A teoria sobre garantia de qualidade lista os atores em um típico framework de garantia de qualidade (GALIN, 2003):

- **Gerentes:** Alto escalão do gerenciamento executivo, gerentes de desenvolvimento de software, gerentes de testes de software, gerentes de projetos, líderes de equipe, líderes da equipe de teste.
- **Testadores:** Membros da equipe de teste.
- **Profissionais e praticantes da garantia da qualidade de software:** Esse grupo se refere às pessoas que se dedicam exclusivamente às atividades de garantia de qualidade, conhecida como unidade de garantia de qualidade. Podemos acrescentar a esse grupo os administradores da garantia de qualidade, membros do comitê de qualidade e membros de fórum de qualidade.

Apesar do que foi descrito por Galin (2003), esses não são todos os atores. Espera-se de toda a equipe da organização a contribuição para o compartilhamento da qualidade organizacional. A garantia da qualidade deve ser integrada dentro do processo de desenvolvimento e deve fazer parte, por padrão, do trabalho diário de cada um. Logo, todo o time é responsável pela qualidade, não somente os gerentes, testadores e profissionais em garantia de qualidade.

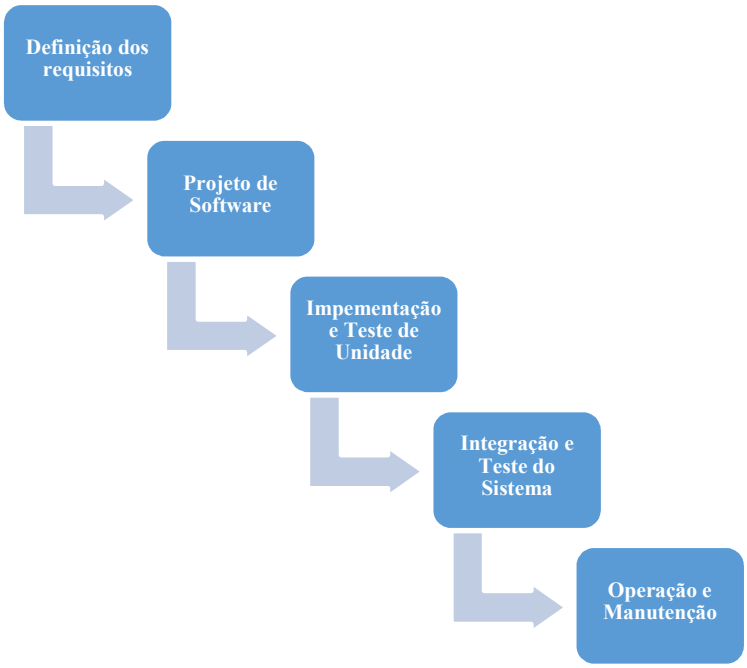
1.8 Metodologias Tradicionais de Desenvolvimento de Software.

Também conhecidas como pesadas, ou orientadas a documentação. O contexto original do desenvolvimento de software era muito diferente do contexto atual, pois era baseado focado no desenvolvimento de softwares para ambiente de *mainframe* e terminais burros (ROYCE,1970). Esse ambiente que levava a incrementar os custos de fazer alterações e correções, por isso o software era todo planejado e documento antes de sua implementação.

1.8.1 Modelo Cascata

Modelo Cascata, ou Clássico (PRESSMAN, 2001), fornece uma sequência de etapas que devem ser seguidas uma após a outra. Ao final de cada etapa, uma documentação padrão deve ser entregue e aprovada para que se inicie a próxima etapa. Basicamente, possuem as etapas de definição de requisitos, projeto de software, implementação e teste unitário, integração e teste do sistema, operação e manutenção. Os problemas básicos do modelo em cascata são a sua inflexibilidade na divisão do projeto em fases distintas e baixa tolerância a risco e mudança. Normalmente usa-se em projetos onde os requisitos são bem conhecidos. A figura 6 ilustra o modelo Cascata.

Figura 6: Modelo Cascata

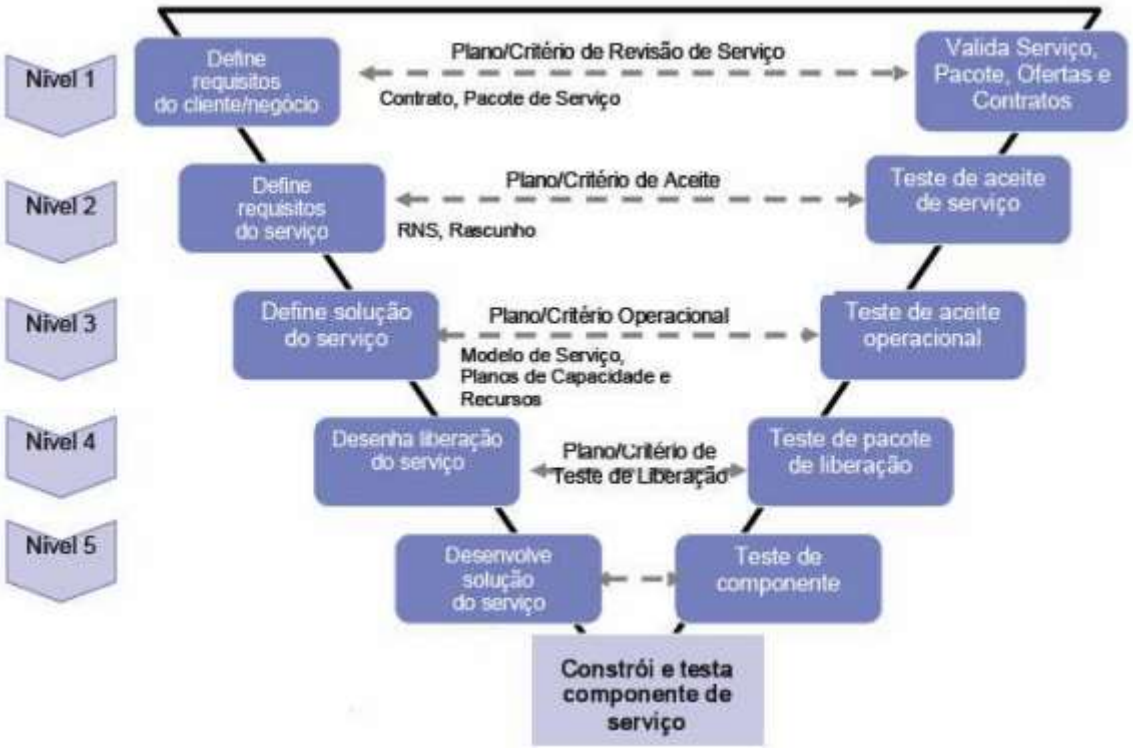


Fonte – Pressman (2001)

1.8.2 O modelo em V

Extensão do modelo Cascata proposto por Boehm (1988), no qual cada fase passa por uma verificação e validação das atividades realizadas. O que o diferencia do modelo Cascata é que em cada fase de requerimento e projeto existirá uma fase de testes para garantir a verificação e validação do sistema. A figura 7 ilustra o modelo em V.

Figura 7: Modelo em V



Fonte – Pressman (2010)

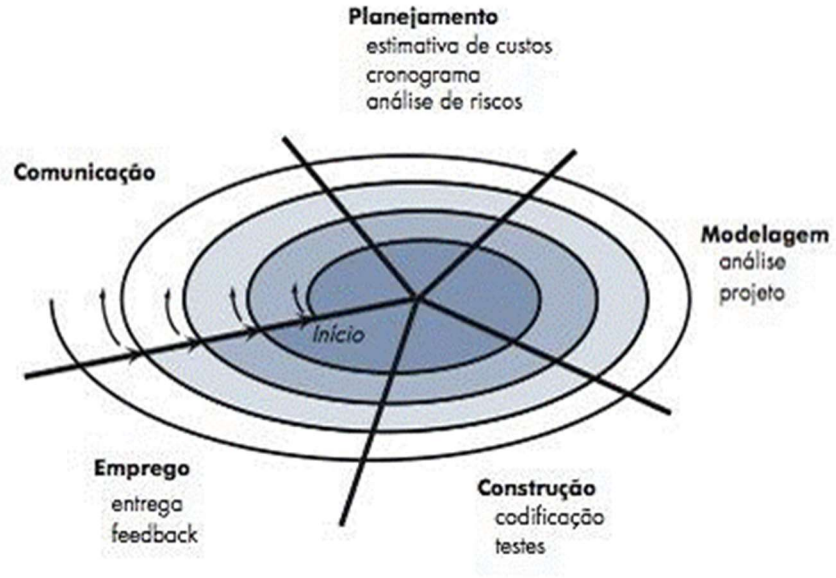
1.8.3 Modelo Espiral

O modelo espiral foi proposto originalmente por Boehm em 1988 (Figura 8). É um modelo direcionado ao risco. Nesse modelo o software é desenvolvido em uma série de versões incrementais. O processo é representado como uma espiral. O esquema combina aspectos de desenvolvimento com aspectos gerenciais (planejamento, tomada de decisão, Análise de Risco, etc). Cada loop da espiral representa uma fase do processo. Dessa forma, o loop mais interno está relacionado à viabilidade do sistema, o próximo loop, à definição de requisitos; o próximo, ao projeto do sistema e assim por diante. Conforme Boehm (1988), cada ciclo da espiral se propõe a:

1. Determinar objetivos, alternativas e restrições;
2. Avaliar alternativas, identificar e resolver riscos;
3. Desenvolver e testar;
4. Planejar as próximas fases.

Em outras palavras, o modelo começa explorando e determinando os riscos da parcela corrente do produto em desenvolvimento. O próximo passo será a avaliação do risco e a busca de estratégias para sua resolução. Tal estratégia pode ser a prototipagem ou simulação. Após a avaliação, os riscos restantes serão determinados no próximo passo.

Figura 8: Modelo em Espiral

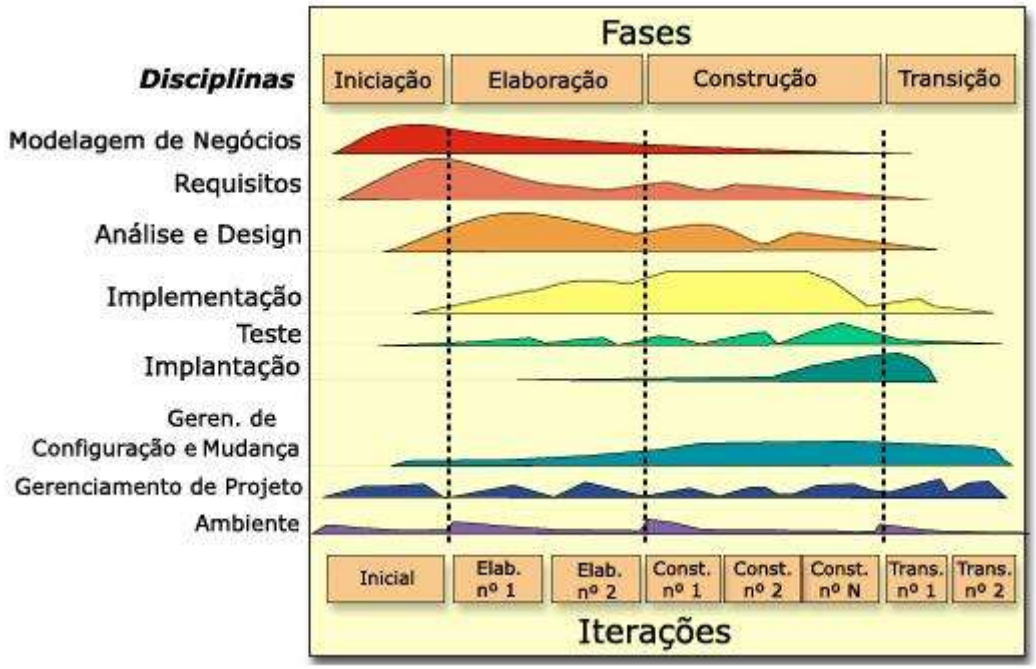


Fonte – Pressman (2010)

1.8.4 *Rational Unified Process*

O *Rational Unified Process* (RUP) foi lançado em 1998. Atualmente, o RUP é mais que um ciclo de vida, é um *framework* de processos e pode ser adaptado para acomodar as necessidades organizacionais. A arquitetura RUP (Figura 2.9) é dividida em duas dimensões onde a horizontal representa o tempo e mostra o aspecto dinâmico do processo que descreve fases e iterações. A dimensão vertical representa o fluxo de trabalho dos processos e mostra o aspecto estático do processo que descreve atividades e papéis.

Figura 9: Arquitetura RUP



Fonte – Rational (1998)

Na arquitetura RUP cada fase pode ser iniciada de forma iterativa e o seu resultado irá desenvolver um incremento, como, todo o conjunto de fases também pode ser iniciado de forma iterativa. Ao final de cada ciclo de desenvolvimento, como resultado, teremos um novo release do sistema.

1.9 Metodologia Ágil

O termo “ágil”, conforme citado do Fowler (2005), refere-se a uma filosofia de desenvolvimento de software. O termo foi definido, em 2001, em um *workshop* onde se buscavam métodos mais leves para o desenvolvimento de software. Antes do *workshop*, diferentes métodos independentes de desenvolvimento e práticas de resposta a mudanças estavam sendo experimentados no desenvolvimento (COHEN; LINDVALL; COSTA, 2004; Fowler, 2005).

O foco do *workshop* era a participação de pessoas que compartilhavam a mesma linha de pensamento em relação ao desenvolvimento de software para juntos definirem uma linha de atuação e definirem um nome que abarcasse as várias

abordagens. Como resultado, foi construído o Manifesto Ágil (HIGHSMITH et al. 2001).

1.9.1 Manifesto Ágil

O manifesto ágil possui integrantes dos adeptos ao *Extreme Programming* (XP), *Scrum, Dynamics Systems Development Methods (*DSDM*), Adaptative Software Development (*ASD), *Crystal Methods, Feature-Driven Development (*FDD), *Pragmatic Programming,* entre outros, que são alternativas as metodologias orientadas a documentação com processos pesados de desenvolvimento.

O manifesto ágil para o desenvolvimento de software declara:

> "*Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:"*
>
> - ***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.*

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à Esquerda"

Os quatros valores, mostrados acima, se desdobram em doze princípios listados por Cockburn et al. (2001)

1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.

Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.

2. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
3. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
4. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para o trabalho.
5. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
6. Software funcionando é a medida primária de progresso.
7. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
8. A contínua atenção à excelência técnica e bom projeto aumenta a agilidade.
9. Simplicidade – a arte de maximizar a quantidade de trabalho não realizado – é essencial.
10. As melhores arquiteturas, requisitos e projetos emergem de equipes auto organizáveis.
11. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

A ideologia ágil não é uma “*anti-metodologia*”. Ela preconiza que se adote a modelagem e que se evite a construção de diagramas que nunca serão usados. Deve-se documentar, mas não escrever de forma excessiva páginas que nunca serão lidas ou mantidas. Deve-se planejar, mas deve-se entender que mudanças ambientais precisam estar limitadas no escopo. (BOEHM; TURNER, 2003).

Mas o que significa ser ágil?

O entendimento do termo “ágil” varia na prática. Somado a isso, é difícil definir os métodos ágeis que também variam na prática. De acordo com o dicionário Houaiss (2017), o termo tem três significados:

- Que se move com facilidade; ligeiro, veloz;
- Desembaraçado, vivo, rápido
- Eficiente, rápido no trabalho; diligente, trabalhador.

Embora as definições acima apresentadas se adaptem bem a resposta à mudança que o método ágil possui, ele não mostra totalmente seu real significado.

Pesquisadores normalmente definem “ágil” como uma filosofia. Segundo a definição proposta por Cockburn (2002, p. 121)

> ágil implica em ser efetivo e manobrável. Um processo ágil é leve e suficiente. A leveza significa que ele permanece manobrável. A suficiência é importante para permanecer no jogo”. Boehm et al (2003) descreve os métodos ágeis como: “O desenvolvimento da prototipagem rápida e de experiências de desenvolvimento rápido bem como o ressurgimento da filosofia que programação e mais artesanal do que um processo industrial.

Outra forma de descrever processos ágeis é declarando as práticas básicas que vários métodos compartilham. Larman (2004. p. 25) disse:

> Não é possível, exatamente, definir métodos ágeis, as práticas específicas variam. Entretanto, o tempo curto entre as iterações com adaptabilidade, refinamento evolucionário dos planos e metas são práticas básicas que vários métodos compartilham.

Boehm et al.(2003, p. 95) definiu outras práticas,

> Em geral, métodos ágeis possui processos muito leves. Ciclo de iterações curtas; participação ativa dos usuários na definição, priorização e verificação dos requisitos; e confiar no conhecimento tácito de uma equipe em oposição a documentação.

Logo definiremos método ágeis como iterativos, incrementais e auto-organizáveis. Acrescentando, os métodos ágeis devem seguir os quatro valores e doze princípios descritos no manifesto ágil (COHEN; LINDVALL; COSTA, 2004).

1.9.2 Reação das Abordagens Tradicionais

Percebemos que o modelo sequencial possui baixíssima tolerância a risco e a mudanças. Apesar dos modelos: em V, o modelo em espiral e o RUP tentarem melhoram isso, eles ainda permanecem sendo metodologias pesadas e orientadas a documentação e ao planejamento. Fowler (2005) cita que essas abordagens assim como as metodologias de engenharia podem funcionar perfeitamente para a construção de uma ponte, mas não para a construção de um software. Como a construção de um software é um tipo diferente de atividade necessitará de diferentes processos. Outra razão para o desenvolvimento ágil é a crescente mudança no ambiente de negócio. Os métodos ágeis foram propostos em uma perspectiva turbulenta nos negócios e de mudança tecnológica (COCKBURN; HIGHSMITH. 2001a). As metodologias tradicionais não conseguiram lidar com essa mudança pois assumiram que é possível antecipar um conjunto completo de requerimentos no início do ciclo de vida do projeto. Mas na realidade, a maioria das mudanças nos requisitos e da tecnologia acorre com o projeto em andamento.

1.9.3 Reuso de ideias

Apesar de a metodologia ágil parecer completamente nova, muitas das ideias presentes no padrão não são novas (ABBAS, 2009). Além do mais, muitas pessoas acreditaram que essa era a forma mais bem-sucedida de se desenvolver softwares. Todavia, essas ideias nunca foram levadas a sério, e nunca tinham sido apresentadas em sua totalidade, como uma metodologia completa. (LARMAN, 2004).

Existem em outros métodos de desenvolvimento ideias como entrega rápida, iterações curtas e abordagens diferentes para adequação a mudanças que também estão presentes em padrões de desenvolvimento ágil. O RAD (do inglês *Rapid Application Development)*, apresentado por James Martin, possui muitas ideias presentes em metodologias ágeis. O RAD sugere entregas rápidas, iteratividade no desenvolvimento, pequenas equipes especializadas e constante envolvimento, em cada estágio, do cliente (MARTIN, 1991).

1.9.4 SCRUM

SCRUM foi desenvolvido por Ken Schwaber e Jeff Sutherland quando perceberam que a atividade de desenvolvimento de software é imprevisível. O Scrum é um framework de gestão, que é conduzido pelo *Scrum Master* (BOEHM; TUNER, 2003). Ele apresenta uma abordagem empírica que aplica algumas ideias da teoria de controle de processos industriais para o desenvolvimento de software, reintroduzindo ideias de flexibilidade, adaptabilidade e produtividade.

O SCRUM tem como base o fato do desenvolvimento de software estar cercado de variáveis técnicas e de ambiente, como requisitos, recursos e tecnologias, que podem mudar no decorrer do processo. Isso faz com que o processo de desenvolvimento seja imprevisível e complexo, necessitando de alta mobilidade para se adaptar as mudanças. O resultado final do processo deverá ser um software que realmente atenda às necessidades do cliente (SCHWABER, 1995).

O desenvolvimento nessa metodologia é dividido em iterações de 30 dias conhecidas como *sprints*. Cada *sprint* será precedido por um planejamento pré-sprint e será seguido de uma reunião pós-sprint. Todos os integrantes da equipe trabalham nos requisitos que foram definidos no início de cada sprint (ABRAHAMSSON et al., 2002).

1.9.4.1 Práticas

Abrahamsson et al.(2002) define que o SCRUM não possui práticas especificas de desenvolvimento, ele concentra seu esforços no processo de gestão do desenvolvimento. As práticas de gestão são:

Lista de *Backlog* de Produto: Priorização dos requisitos técnicos e de negócio. Podem ser funcionalidades, correções, atualizações de tecnologia, entre outras. A lista é controlada pela inserção, remoção e atualização de itens no decorrer do processo de desenvolvimento.

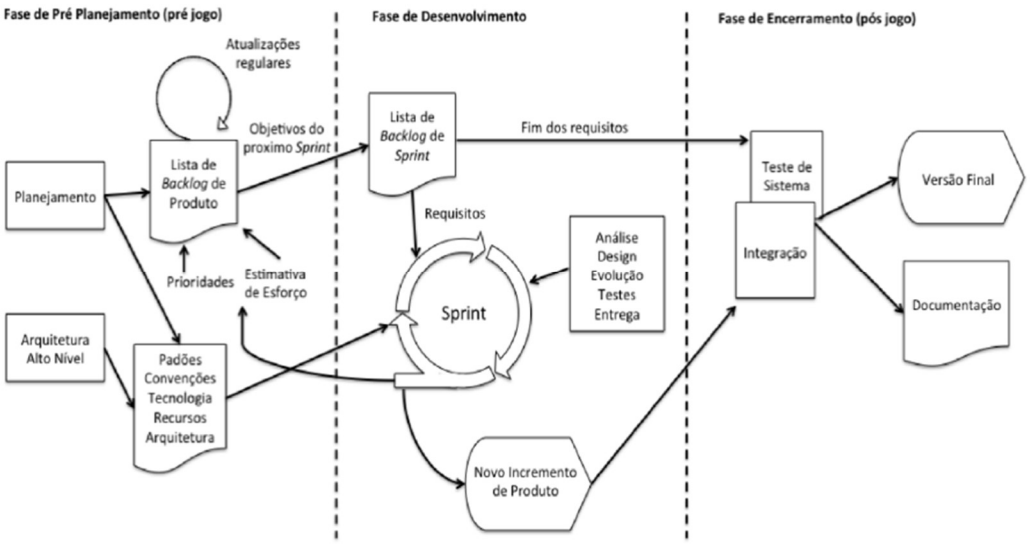
- **Estimativa de esforço:** Estimativa de esforço necessário para cumprir cada atividade dos itens do *backlog.*

- ***Sprint:*** Iterações realizadas durante o desenvolvimento. Cada iteração irá corresponder a um ciclo composto por variáveis de ambiente as quais são modificadas a cada iteração. A equipe, auto organizável, deve produzir um incremento de software executável a cada iteração. Cada *sprint* dura aproximadamente 30 dias.
- **Reunião de Planejamento da *Sprint:*** Possui duas fases, na primeira o cliente seleciona os itens do *backlog* e os prioriza. Na segunda, a equipe do projeto planeja como o incremento será produzido.
- ***Backlog de Sprint:*** Lista de itens selecionados do b*acklog* que serão desenvolvidos em uma *sprint.*
- **Reunião diária:** Verificação diária do andamento do projeto. O foco deve ser sempre as atividades que realizadas no dia e quais atividades serão realizadas até a próxima reunião. Tem duração rígida de no máximo 15 minutos.
- **Reunião de revisão de *sprint:*** Ocorre ao final da iteração. São apresentados, pela equipe, os resultados finais da *Sprint*. Os resultados podem gerar mais itens de *backlog* do produto e esses devem ser considerados no planejamento do próximo *Sprint.*

1.9.4.2 Processo

No método SCRUM o ciclo de vida possui três fases: pré-jogo, desenvolvimento e pós-jogo (ABRAHAMSSON et al., 2002). As três fases junto com a atividades de cada fase são apresentadas na figura 10.

Figura 10: Processo de Desenvolvimento SCRUM



Fonte – Abrahamsson et al (2002)

Pré-Planejamento (pré-jogo)

Essa fase define o sistema que está sendo desenvolvimento. Cria-se a lista de *Backlog*, que contém todos os requisitos atuais e informações sobre o planejamento do projeto. Cria-se também a arquitetura de alto nível.

A fase de pré-jogo é composta por duas subfases: planejamento e desenho da arquitetura de alto nível.

Fase de desenvolvimento

Abrahamsson et al. (2002) afirma que, a fase de desenvolvimento é a parte ágil do SCRUM. Nessa fase o sistema é desenvolvimento em *sprints*, por meio de uma abordagem iterativa. A cada s*print*, novas funcionalidades são adicionadas de modo tradicional, isso é, requisitos, análise, projeto, desenvolvimento, evolução e entrega. A arquitetura e o planejamento do sistema evoluem durante o desenvolvimento de um *sprint,* que deve durar de uma a quatro semanas.

O SCRUM foca no controle de variáveis técnicas e do ambiente, como tempo, qualidade, requisitos, recursos e ferramentas durante o processo de desenvolvimento, tornando assim o processo mais flexível a mudanças (ABRAHAMSSON et al., 2002)

Pós-Planejamento (pós-jogo)

Nesta fase serão feitas reuniões para analisar o progresso do projeto e demostrar o software atual para os clientes. Serão executadas as etapas de integração, testes finais e documentação. Em outras palavras, contém as atividades de encerramento de uma versão.

1.9.4.3 Papéis e responsabilidades

O SCRUM possui papéis bem definidos. Um time de desenvolvimento SCRUM é formado por: Product Owner, Scrum Master, Time de Desenvolvimento, Cliente e Gerente. O Quadro 3 mostra as principais características de cada papel.

Quadro 3: Papéis e Responsabilidade no SCRUM

Papéis	**Características**
SCRUM Master	• Responsável pela gestão de pessoas e do processo. • Garante o entendimento e aplicação das práticas, valores e regras do SCRUM. • Responsável pela comunicação entre o cliente, equipe e administrador.

Product Owner	• Responsável pela macrogestão e pela gestão do produto. • Responsável por maximizar o valor do produto e do trabalho da equipe de desenvolvimento. • Único responsável pelo gerenciamento da Lista de *Backlog*. • Responsável pelo Retorno sobre Investimento. • Expressa claramente os itens da Lista de *Backlog*. • Garante que o *Backlog* do produto é visível, transparente, claro para todos, e mostra o que a Time de Desenvolvimento vai trabalhar. • Garante o entendimento pelo Time de Desenvolvimento entenda os itens da Lista do *Backlog* no nível necessário
Time de Desenvolvimento	• Responsável pela micro gestão e pela criação do produto. • São auto organizados, multifuncionais
Cliente	• Participa das tarefas relacionadas aos itens de *backlog* de produto, para que o sistema seja desenvolvido ou melhorado.
Gerente	• Toma decisões finais sobre contratos, padrões e convenções a serem seguidas no projeto. Ele também participa das definições dos objetivos e dos requisitos.

Fonte – O autor (2017)

1.9.5 *Extreme Programming (XP)*

Desenvolvida por Kent Beck, as práticas do XP foram desenvolvidas, originalmente, para pequenos e médios times e surgiram de problemas decorrentes de processos de desenvolvimento que utilizavam abordagens tradicionais de

desenvolvimento de software. (BECK, 1999a). O XP possui quatro valores básicos (TELES, 2005):

1. ***Feedback:*** Os clientes não podem prever corretamente as funcionalidades que serão necessárias para o software que os atenderá. Logo, uma grande interação entre o cliente e os desenvolvedores ao logo do projeto é de extrema importância. A correta compreensão das necessidades dos usuários é um aprendizado contínuo no qual os desenvolvedores aprendem sobre problemas de negócio e os usuários tomam conhecimento das dificuldades e de limitações técnicas (TELES, 2005).

   O XP é organizado em pequenos ciclos de desenvolvimento, isso faz com que exista um *feedback* constante. O foco passa a ser apresentar a funcionalidade ao usuário rapidamente, agilizando o processo de detecção de falhas. Dessa forma o XP permite que os erros das pessoas sejam descobertos relativamente cedo e reparados de forma metódica e a um menor custo (COCKBURN, 2002)

2. **Comunicação:** Teles (2005) afirma que um dos grandes problemas em processos de desenvolvimento de software é a transmissão do conhecimento tácito. O projeto de software geralmente é composto por um grupo de pessoas heterogêneas como por exemplo o desenvolvedor e o usuário.

   Para sanar esse tipo de problema, o XP procura envolver ativamente os usuários, tornando membros da equipe de desenvolvimento e fazendo-os compartilhar a mesma sala física de projetos. Facilitando assim a comunicação que passa a ser constante e direta (COCKBURN, 2005).

3. **Simplicidade:** Os desenvolvedores irão implementar as funcionalidades de acordo com suas prioridades para cada iteração com foco apenas no essencial. Generalizações que não estejam explicitamente descritas como necessárias não são implementadas, tentando manter a simplicidade das funcionalidades (JEFFRIES et al., 2001).

4. **Coragem:** Frequentemente, nos processos de desenvolvimento de software alguns temores dos clientes que podem atrapalhar o desenvolvimento. Beck e Folwer (2001) afirmaram que o cliente teme não obter o foi solicitado, ou em fazer uma solicitação que leva a uma implementação incorreta.

Equipes XP reconhecem e enfrentam esses medos. É necessário se confiar nos mecanismos oferecidos pela metodologia. Mesmo que seja descrita uma funcionalidade de forma errada ou que a equipe não implemente as funcionalidades esperadas, devido as iterações curtas, o tempo para *feedback* é curto, fazendo com que os problemas sejam descobertos com rapidez e não se espalhem pelo do projeto (TELES, 2005).

1.9.5.1 Práticas

O XP busca o sucesso do desenvolvimento em um ambiente em constante mudança ou com requisitos vagos. Iterações curtas com pequenos lançamento e *feedback* rápido, inclusão do cliente, comunicação e coordenação, integração continua e testes, documentação limitada e programação pareada, estão entre as principais características do XP (ABRAHAMSSON et al., 2002). Suas principais práticas são:

- **Planejamento Incremental:** Os requisitos do sistema são registrados em cartões de histórias e as histórias a serem incluídas em um release são determinadas pelo tempo disponível e a sua prioridade. A equipe de desenvolvimento divide as histórias em tarefas.
- **Jogo do Planejamento:** O planejamento de uma entrega e das iterações são elaborados com base nas histórias e conta com a colaboração de toda equipe de desenvolvimento, inclusive o cliente, divididos em papeis: negócio e técnico. Os clientes irão priorizar e os desenvolvedores avaliam e estimam.
- **Entregas frequentes:** Os pequenos releases são o conjunto mínimo útil de funcionalidade que agrega valor ao negócio que serão desenvolvidos primeiro. As entregas serão frequentes e adicionaram funcionalidade incrementalmente ao primeiro release.
- **Projeto Simples:** O projeto será suficientemente simples de modo que atenda aos requisitos atuais e não se preocupe com os requisitos futuros.
- **Testes:** Os testes para uma nova parte da funcionalidade devem ser escritos antes que esta seja implementada.

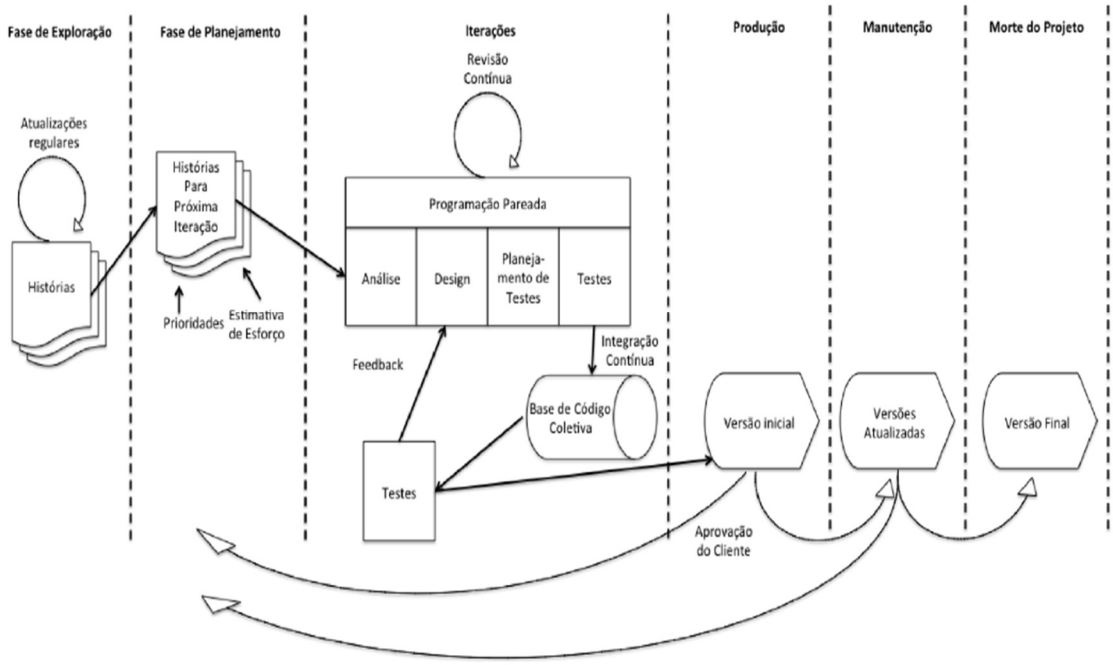
- **Programação em pares:** Os desenvolvedores trabalharam em pares, de modo que um verifique o trabalho do outro e fornecendo apoio para o aumento da qualidade do código. Eles utilizarão o mesmo teclado, mouse e monitor.
- **Refatoração:** Contribui para torna o código mais simples de entender e fácil de manter. Todos devem recriar o código de forma continua tão logo seja possível seu aprimoramento.
- **Propriedade coletiva do código:** O código pertence a todos os membros do projeto. Isto significa que qualquer pessoa, pertencente a equipe, pode adicionar valor ao código, desde que faça a bateria de testes necessárias.
- **Integração contínua:** Assim que o trabalho em uma tarefa esteja concluído, esta deverá ser integrada ao sistema como um todo. Depois de qualquer integração, todos os testes unitários do sistema devem ser realizados.
- **40 horas de trabalho semanal:** Na metodologia XP, o trabalho por longos período de tempos é considerado contraproducente, pois, no médio prazo, existe uma redução na qualidade do software e na produtividade. Logo evitam-se jornadas superiores a 40 horas.
- **Metáforas:** A equipe se comunica sobre o desenvolvimento de software por meio de metáforas, caso consiga encontrar uma que realmente faça sentido dentro do contexto e que a facilite a comunicação.
- **Cliente presente:** O cliente ou seu representante devem estar disponíveis em tempo integral para sustentar o time de desenvolvimento. Na metodologia o cliente é membro do time de desenvolvimento e é responsável pelos requisitos do sistema.
- **Reuniões em Pé:** As reuniões serão realizadas em pé para não se perder o foco e produzir reuniões mais rápidas, somente abordando as tarefas realizadas e as que serão executadas pela equipe no futuro.
- **Time Coeso:** A equipe de desenvolvimento é formada de pessoas multidisciplinares e engajada. Os membros, preferencialmente, devem possuir habilidades relacionadas com diversas áreas do projeto.

1.9.5.2 Processos

A figura 10 mostra as seis fases com as suas respectivas atividades e práticas, conforme Beck (1999), as quais são:

- **Exploratória:** Os clientes escrevem as histórias que eles gostariam que fossem incluídas na primeira versão a ser lançada. Uma história irá corresponder a uma funcionalidade a ser adicionada no programa. Ao mesmo tempo a equipe de desenvolvimento deve se adaptar com ferramentas, tecnologias e práticas que serão utilizadas no projeto. Deve se testar a tecnologia a ser utilizada e explorar as possibilidades de arquitetura para o sistema. Esta fase deve ter duração de poucas semanas a poucos meses.

Figura 10: Ciclo de Vida do processo XP



Fonte – Abrahamsson et al. (2002)

- **Planejamento:** Nessa fase serão priorizadas as histórias e se delimitará o escopo da primeira versão do programa a ser liberada. Os programadores devem estimar o tempo necessário para a implementação de cada história e sendo possível agendar a primeira entrega.

- **Iterações:** São as várias iterações do sistema até que a primeira entrega seja realizada. Elaborado na fase de planejamento, o cronograma é dividido em tarefas menores, iterações, que serão concluídas de uma a quatro semanas. Na primeira iteração serão escolhidas histórias que implementem a arquitetura básica do sistema. O cliente pode definir quais histórias serão selecionadas em cada iteração. Executados ao final de cada iteração, os testes funcionais serão elaborados pelo cliente. O sistema ficará pronto para a produção ao final da última iteração.
- **Produção:** Essa fase necessita de testes extras e verificações de performances antes que o sistema seja lançado para uso do cliente. Podem surgir algumas alterações que podem ser incluídas no lançamento da próxima versões. As correções podem ser feitas em interações mais curtas. Alterações não aprovadas devem ser documentadas para implantação futura, se for o caso.
- **Manutenção:** Na primeira versão do sistema, o projeto deve manter produção e desenvolvimento das próximas iterações.
- **Morte do projeto:** Ocorre quando não existe mais histórias a serem implementadas. Isso acontece quando o sistema satisfaz as necessidades do cliente e atende critérios de desempenho e confiabilidade. A escrita da documentação do sistema ocorre nessa etapa. Pode ocorrer também, se o sistema não estiver entregado, que o resultado esperado ou o orçamento não sejam suficiente para continuar o projeto.

1.9.5.3 Papéis e responsabilidades

Existem sete papeis no XP relacionados com diversas tarefas e objetivos:

- **Programador:** Escrevem testes e mantém o código o mais simples possível. O sucesso do XP está conectado a comunicação e coordenação com outros programadores e membros da equipe.

- **Cliente:** Responsável pelas histórias e escritas de testes funcionais. Prioriza a implementação dos requisitos. Decide quando cada requisito está definido.
- **Analista de Teste:** Oferece suporte ao cliente para elaboração de testes funcionais. Realiza testes funcionais com regularidade, fornece publicidade aos resultados e mantém as ferramentas de teste.
- **Redator Técnico:** Analisa as estimativas feitas pela equipe e fornece o *feedback* sobre a precisão das estimativas, para o melhor preparo da equipe em projetos futuros. Observa a evolução de cada iteração e avalia se a meta será alcançada dentro dos limites de recurso e tempo, ou se é necessária uma alteração no processo.
- **Treinador:** Normalmente é a pessoa com o maior conhecimento do processo de desenvolvimento, dos valores e práticas do XP, pois é a responsável pelas questões técnicas do projeto. O treinador verifica o comportamento da equipe frente ao processo XP, aponta os erros cometidos pela equipe.
- **Consultor:** O consultor é um membro externo que detém algum conhecimento técnico específico necessário para o projeto. O consultor guia a equipe para solucionar seus problemas.
- **Gerente –** Responsável por assuntos administrativos do projeto, motiva a participação do cliente nas atividades do desenvolvimento. O gerente irá atuar como filtro excluindo os desenvolvedores de assuntos não relevantes e de aspectos que somente serão implementados em futuras fases do projeto. Para atuação como gerente e necessário conhecimento e que valorize as práticas do XP.

2 METODOLOGIA ÁGIL E QUALIDADE

As seções anteriores definiram conceitos de qualidade de software, garantia da qualidade de software e algumas métricas foram apresentadas. É fácil perceber que a grande parte dos conceitos foram elaborados com base em metodologias tradicionais de desenvolvimento de software. Logo é necessário se revisar as pesquisas em torno dos métodos ágeis e de diferentes aspectos de qualidade.

2.1 Qualidade em metodologia ágil.

Em 2007 Stamelos e Sfetos (2007) publicou um livro sobre o título " *Agile Software Quality Assurance".* O livro é composto pela argumentação de diferentes autores separado em 12 capítulos, sobre a garantia da qualidade em metodologias ágeis. Em seu primeiro capítulo ele discorre sobre quais parâmetros de qualidade podem ser aplicados nas metodologias ágeis. Em capítulos subsequentes ele apresenta uma ferramenta para garantia da qualidade dentro do contexto ágil.

Huo et al (2004) analisou as técnicas de garantia de qualidade e a frequência de utilização em práticas ágeis. O autor constatou que existem técnicas de garantia da qualidade, algumas estão acopladas as fases do desenvolvimento e outras separadas na forma de práticas de suporte. Huo et al (2004) afirma também que a frequência com que essas práticas são utilizadas é maior do que no desenvolvimento em cascata e que essas práticas estão disponíveis, devido a característica da metodologia ágil, em estágios iniciais do projeto.

Mnkandla et al. (2006) levantou em seu artigo a possibilidade de existir atividades que podem melhorar a alta qualidade já alcançada pelos métodos ágeis e que existem técnicas diferentes para métodos ágeis específicos.

Em 2006 um estudo conduzido por Dyba et al. (2008) analisou a relação entre a qualidade empírica de software e métodos ágeis. A pesquisa teve como base estudos que demostraram alta qualidade de análise. A pesquisa foi baseada em uma

metodologia empírica bem descrita. Essa pesquisa encontrou cinco estudos que examinam a qualidade do produto. Como exemplo um estudo mostrou que os efeitos da adoção da metodologia XP aumentaram em 50% a produtividade, melhorou em 65% a qualidade pré-lançamento e aumentou em 35% a qualidade pós lançamento (Lyman et al. 2004).

Siniaalto et al. (2007) liderou experimentos para analisar a influência que diferentes métodos ágeis causavam na qualidade do software. Como exemplos se teve o estudo que comparou a utilização de *test-driven development* - TDD em três projetos de desenvolvimento. Como resultado se constatou que o efeito do TDD no projeto do programa não foi tão evidente como o esperado, mas a abrangência para os testes foi significantemente superior do que técnicas de desenvolvimento que executam os testes somente após a conclusão do código.

Um estudo prático comparou a garantia de qualidade no XP com o modelo espiral mostrou que atividades de melhora da qualidade estão embutidas no XP e que em grande parte ocorrem em paralelo. E a frequência com que essas atividades ocorrem no XP é superior à frequência da ocorrência no modelo espiral pois a iteratividade é uma característica natural na XP. A pesquisa mostrou que um processo não é superior ao outro. O estudo conclui que quando comparamos as falhas ocorridas em projetos do modelo espiral com projeto do modelo XP, percebeu-se que a densidade de falha no espiral era maior. (Hashmi et al. 2007).

2.2 Padrões de qualidade e a metodologia ágil.

Quando analisamos as pesquisas relativas à utilização de padrões de qualidade consagrados no mercado, como CMMI, ISO, Six Sigma (Anderson 2005; Alegria et al. 2007; Nawrocki et al. 2002ª), percebemos que grande parte das iniciativas tende a buscar modificar ou adaptar a metodologia ágil para que ele se torne aderente as restrições presentes no padrão. E a grande razão para isso ocorrer é o fato de que alguns princípios vão de encontro a processos chaves presentes em alguns padrões (Royce, 2002).

Como exemplo podemos citar a iniciativa proposta por Anderson (2005) e Alegria et al (2007) que tentaram combinar a metodologias como SCRUM e XP para

satisfazer as restrições presentes nos níveis 1 e 2 do CMMI. Como resultado foi possível o desenvolvimento de um ciclo de vida ágil que atendia a todos os níveis de CMMI (Anderson, 2005). Alegria et al. (2007) percebeu que para alcançar o nível 2 do CMMI era necessário corrigir uma série de problemas presentes na metodologia.

Outro exemplo é o mapeamento feito por Silva e Hoentsch (2009) entre as metodologias ágeis FDD e Scrum e o modelo de qualidade MPS.BR que afirma que tanto a FDD como o Scrum não conseguem atender alguns níveis de processos presentes no padrão. Sugerindo que a adoção tanto do FDD como do SCRUM é inapropriada para empresas que pretendem galgar sistematicamente todos os níveis de maturidade presentes no MPS.BR.

Como percebemos, todas as pesquisas sugerem que se modifiquem, ou se adapte, a metodologia para que atenda aos requisitos presentes no padrão, mas a grande questão que permanece é: é necessário se alterar a metodologia ou um novo padrão deve ser criado.

2.3 Métricas e a metodologia ágil.

Quando buscamos trabalho relativos a métricas e a metodologia ágil percebe que pouco ainda se tem publicado e grande parte do material encontrado está presente em blogs e fóruns.

Jeffries (2004) sugeriu uma métrica que poderia ser utilizada em metodologias ágeis. Chamada de “ *Running Testing Features Metric (RTF)*” que irá mostrar a cada momento no projeto a quantidade de funcionalidades que passaram pelos teste de aceite. Ele afirma que o RTF deve ser aplicado de forma a aumentar linearmente do início do projeto ao seu final.

Outra métrica comumente utilizada por times ágeis é a velocidade. Velocidade dentro do contexto ágil pode ser definida como a quantidade de trabalho que o time pode executar durante uma iteração. Pode ser calculada pela divisão das tarefas completadas por cada time. Pode ser usada para comparar a produtividade de dois times por meio de aceleração que a velocidade de cada time através do tempo (Amber, 2008a).

Hartmann et al. (2006) discute sobre qual métrica é apropriada para em um ambiente ágil e recomenda o foco em medir as entregas ao invés das entradas, usando um conjunto de métricas pequeno e de fácil coleta que se apoiam nos princípios da agilidade.

3. SURVEY – QUALIDADE E METODOLOGIAS ÁGEIS

3.1 Considerações Iniciais

Observando o panorama do desenvolvimento ágil percebemos que todos os praticantes afirmam que a utilização das técnicas presentes no modelo ágil melhora a qualidade dos produtos desenvolvidos. Mas percebe-se também a falta de uma técnica para avaliar como os processos ágeis atendem os requisitos necessários para a qualidade de software. Ainda é difícil afirmar que fatores, práticas e ferramentas presentes em um contexto de desenvolvimento ágil apresentam impacto direto na qualidade do produto desenvolvido.

O *Survey* tem como objetivo analisar o impacto das metodologias ágeis na qualidade do software desenvolvido assim como o comprometimento dos desenvolvedores com o processo, pois se acredita que esse envolvimento seja um fator fundamental para a qualidade do produto no desenvolvimento através de metodologias ágeis.

O *Survey* é uma estratégia de estudo conhecida por realizar estudos empíricos que fornece uma descrição quantitativa ou numérica sobre a fração de uma amostra, por meio de questionários. (CRESWELL, 2008).

3.2 Metodologia

O *Survey* de qualidade em metodologias ágeis foi desenvolvido e operacionalizado por meio da ferramenta *Survey Monkey* on-line. As questões foram desenvolvidas com base na literatura sobre qualidade de software e nos processos de desenvolvimento ágil.

O questionário para coleta de dados foi dividido em:

1. Informações básicas: Tem como objetivo classificar os participantes.
2. Processo de desenvolvimento: Tem como objetivo conhecer as atividades do processo de desenvolvimento e a documentação gerada.
3. Garantia da Qualidade: objetiva identificar as atividades e artefatos de garantia de qualidade.

4. Teste de Software: Identificar as atividades e artefatos de teste de software.
5. Satisfação do Cliente: Identificar as atividades de aceite são executadas e se a empresa se preocupa com elas.
6. Qualidade do software: Identificar quais são os principais indicadores da qualidade do software.
7. Métricas e Medidas: Identificar quais as principais métricas e medidas utilizadas no processo de desenvolvimento.

A Quadro 4 mostra as questões utilizadas no s*urvey*.

Quadro 4: Questões do *survey*

1.Informações Básicas	
A quanto tempo você trabalha com TI?	Múltipla Escolha
Com qual metodologia ágil você já trabalhou?	Múltipla Escolha
A quanto tempo você trabalha com metodologias de desenvolvimento ágil?	Múltipla Escolha
2.Processo de Desenvolvimento	
Quais das seguintes práticas ágeis você utiliza?	Múltipla escolha
Qual a documentação gerada na metodologia em relação a gestão do escopo do projeto?	Questão Aberta
Qual a documentação gerada na metodologia em relação ao versionamento do código do projeto?	Questão Aberta
3. Garantia da Qualidade	
Quais as atividades são executadas pela sua empresa para o gerenciamento de qualidade de software?	Questão Aberta
Existe na sua empresa equipe própria de controle de qualidade?	Múltipla Escolha
4.Teste de Software	
Como são gerados os casos de teste?	Questão Aberta

A empresa possui equipe de teste dedicada?	Múltipla escolha
5.Satisfação do Cliente	
Como é o processo de aceite pelo usuário final?	Questão Aberta
Com que frequência a empresa avalia a satisfação do cliente?	Múltipla Escolha
São feitas revisões dos planos de projetos, processos ou especificações de novos produtos e serviços? Com base em que dados essas revisões são feitas?	Questão aberta
6. Qualidade	
Qual(is) indicador(es) de desempenho dos processos utilizados no desenvolvimento de software são usados pela empresa?	Questão Aberta
Qual a porcentagem média de sucesso dos projetos de desenvolvimento ágil?	Múltipla Escolha
A utilização de metodologias ágeis afeta sua produtividade?	Múltipla Escolha
A utilização de metodologias ágeis afeta a qualidade do sistema desenvolvido?	Múltipla Escolha
A utilização de metodologias ágeis afeta o custo dos sistemas desenvolvidos?	Múltipla Escolha
Em média, quantas revisões pós entrega, um sistema construído por metodologia ágil tem que passar antes de ser completamente aceito pelos usuários finais?	Múltipla Escolha
Como você avalia a qualidade do código gerado?	Múltipla Escolha
7. Métricas e Medidas	
Como são coletadas as métricas/medidas de qualidade?	Múltipla Escolha
Qual métrica ou medida é utilizada?	Múltipla Escolha

Fonte – O autor (2017)

Para resolução do questionário foram selecionados desenvolvedores que trabalham utilizando metodologias ágeis de três empresas, uma do setor público e duas do setor privado. O questionário também foi publicado em redes sociais com o foco no estudo de metodologias ágeis. O questionário ficou disponível entre os dias 12 de setembro de 2017 ao dia 01 de outubro de 2017.

O questionário foi traduzido para o inglês e também distribuídos em grupos presentes em redes sociais (Linkedn e Facebook) onde o foco é o estudo e a disseminação de conhecimentos relativos ao desenvolvimento de software utilizando metodologias ágeis.

O Survey foi respondido por um total de 47 pessoas sendo desses 19 usuários de língua portuguesa e 28 usuários de língua inglesa.

3.3 Analise dos Dados Quantitativos.

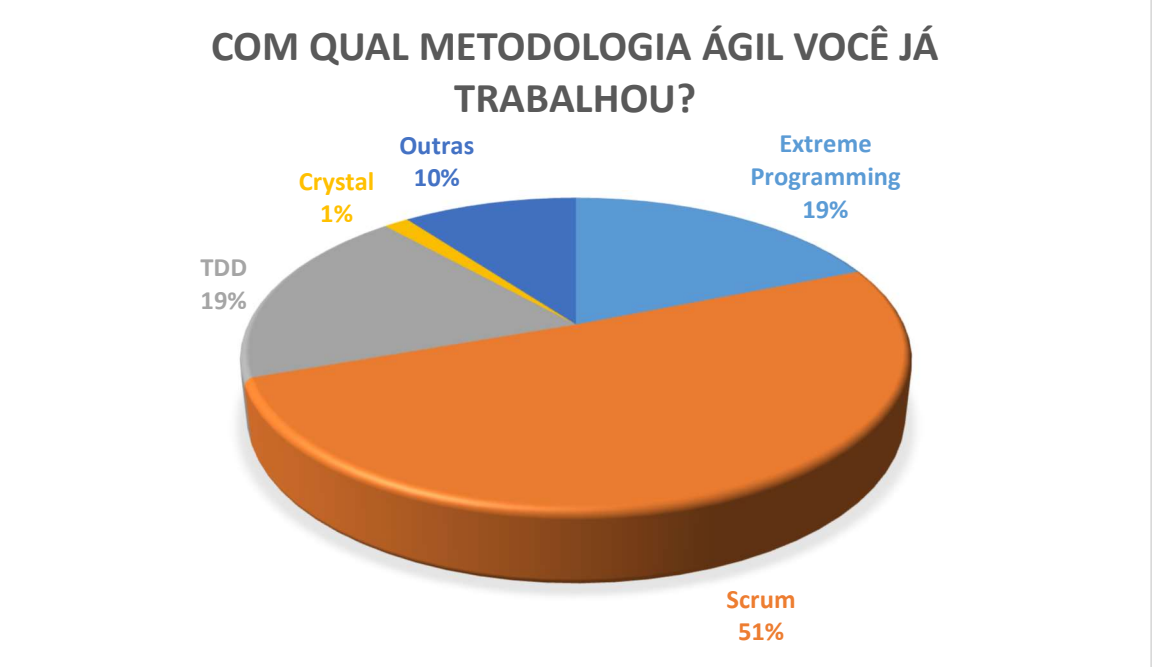
Os gráficos 1 e 2 descrevem respectivamente o tempo de experiência do profissional na área de Tecnologia da informação e o tempo de trabalho com as metodologias ágeis.

Gráfico 1: Distribuição do tempo de experiência em Tecnologia da Informação entre os participantes do *survey*.

Gráfico 2: Distribuição do tempo de experiência com metodologias ágeis entre os participantes do *survey*.

No gráfico 3 podemos observar que a metodologia ágil mais utilizada entre os participantes é o Scrum, com mais da metade da amostra já tendo trabalhado com o método.

Gráfico 3: Distribuição da utilização dos métodos ágeis entre os participantes do *survey*.



Quando analisamos os dados de controle de qualidade e testes, percebemos que a maioria das corporações que os respondentes estão alocados não possui equipe própria para controle de qualidade e não possuem equipe dedicada de testes. O que nos permite inferir que a equipe de desenvolvimento é a responsável pelas atividades de controle de qualidade e testes. Nos gráficos 4 e 5 mostram respectivamente esse cenário.

Gráfico 4: Distribuição dos participantes do *survey* que possuem na empresa equipe própria de garantia de qualidade.

Gráfico 5: Distribuição dos participantes do *survey* que possuem na empresa equipe própria de teste de software.

Depois de observar se a empresa possuía ou não equipe dedicada a teste, a pesquisa questionou a frequência em que a satisfação do cliente é medida. O resultado (gráfico 6) mostra que a maioria dos respondentes avalia sempre a satisfação do cliente em relação ao sistema que foi entregue.

Gráfico 6: Frequência de avaliação da satisfação do cliente de acordo com os participantes do *survey.*

No gráfico 7 podemos perceber que a taxa de sucesso de desenvolvimento de sistemas quanto se utiliza de uma abordagem ágil é alta, pois quase 80% dos respondentes afirma que a taxa de sucesso gira em torno de 51% a 100%.

No gráfico 8 podemos perceber que a adoção de metodologias ágeis aumenta a produtividade significativamente, pois a maioria da amostra afirma que a adoção de uma metodologia ágil aumentou ou aumentou muito sua produtividade. Assim como no gráfico 9 podemos perceber que a qualidade também é afetada positivamente pela adoção do modelo ágil o que suporta a hipótese de que a metodologia ágil pode produzir um software de boa qualidade.

Gráfico 7: Porcentagem média de sucesso dos projetos de desenvolvimento ágil de acordo com os participantes dos *survey*.

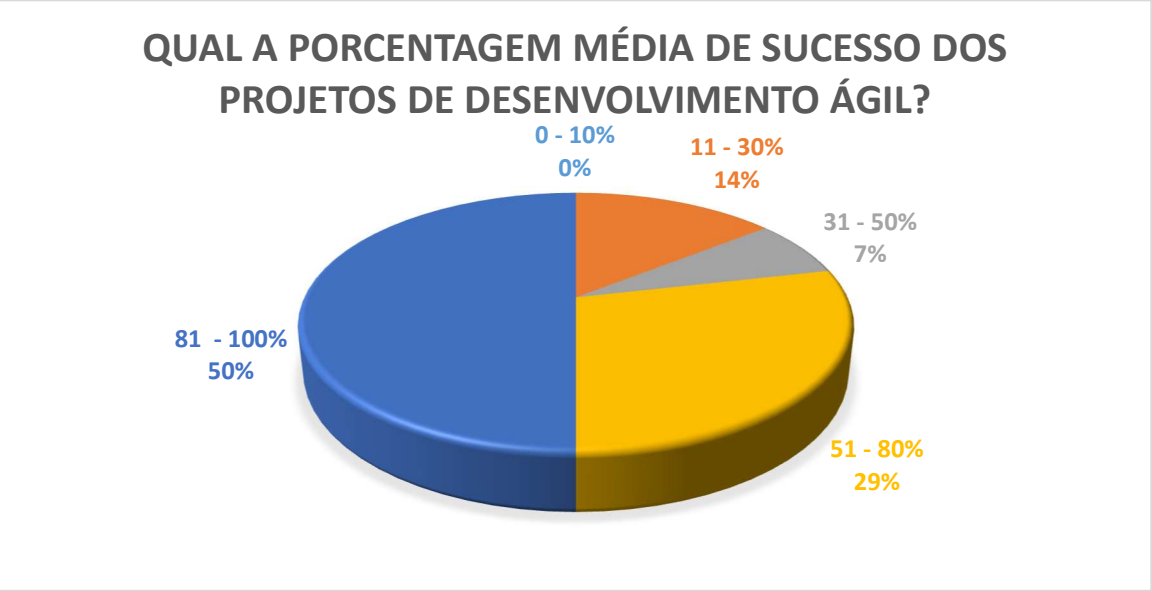


Gráfico 8: Distribuição do aumento da produtividade utilizando metodologias ágeis de acordo com os participantes do *survey*.

Gráfico 9: Aumento da qualidade do sistema desenvolvido por meio de metodologias ágeis de acordo com os participantes do *survey*

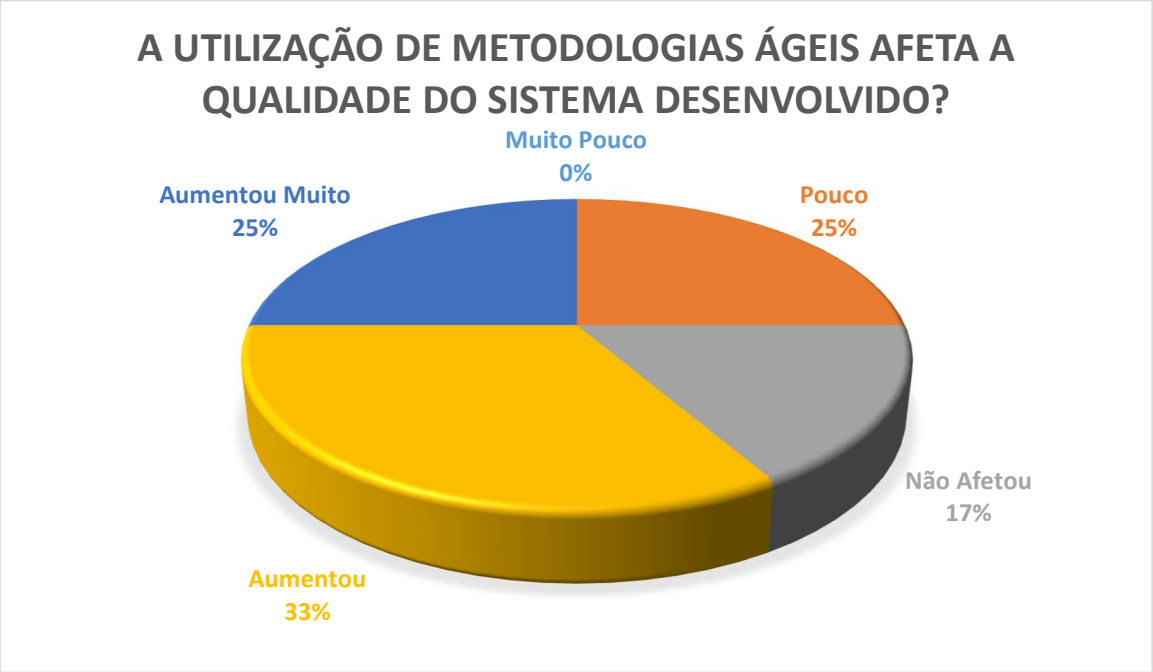
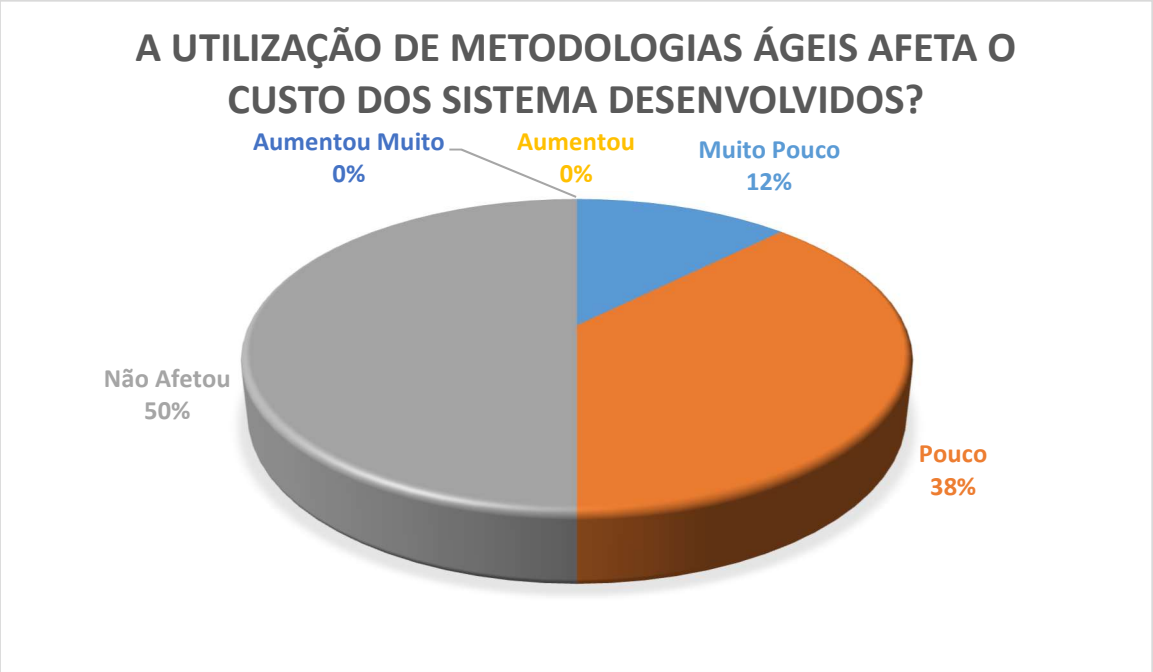


Gráfico 10: Impacto no custo de desenvolvimento utilizando metodologias ágeis de acordo com os participantes do *survey*.



Quando tratamos das revisões que o sistema tem que passar antes de ser totalmente entregue ao usuário final percebemos que o cenário apresentado pela Gráfico 11 mostra que boa parte dos sistemas é revista antes de ser totalmente entregue ao usuário.

Gráfico 11: Média (%) de revisões pós-entregas em sistemas desenvolvidos com auxílio de metodologias ágeis de acordo com os participantes do survey.

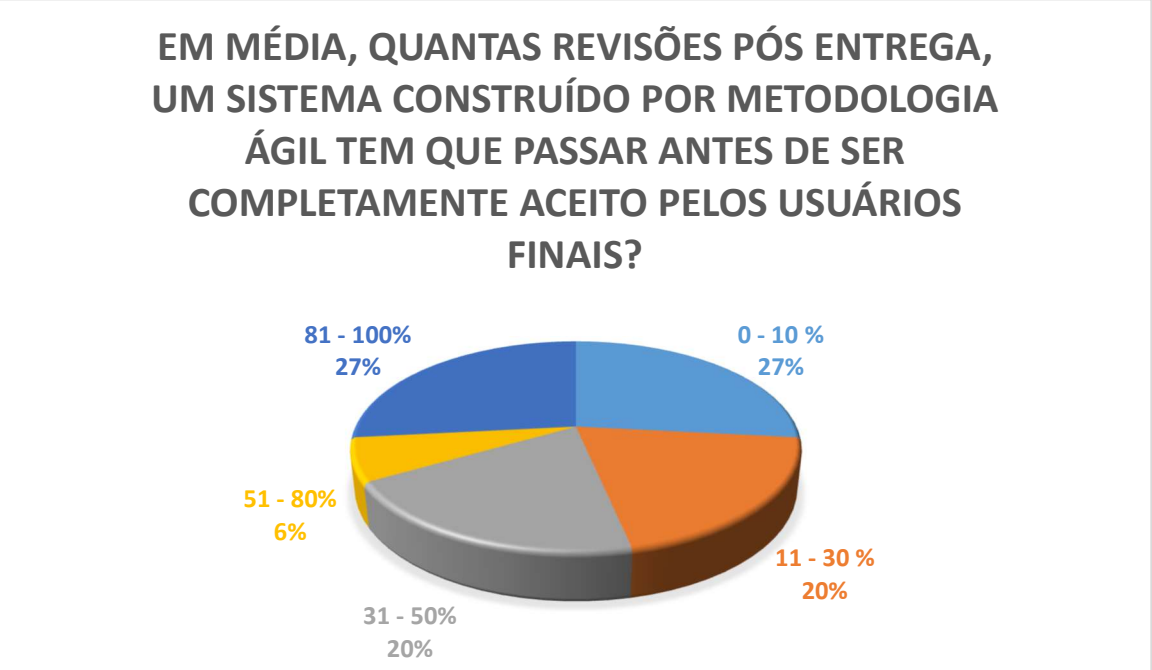
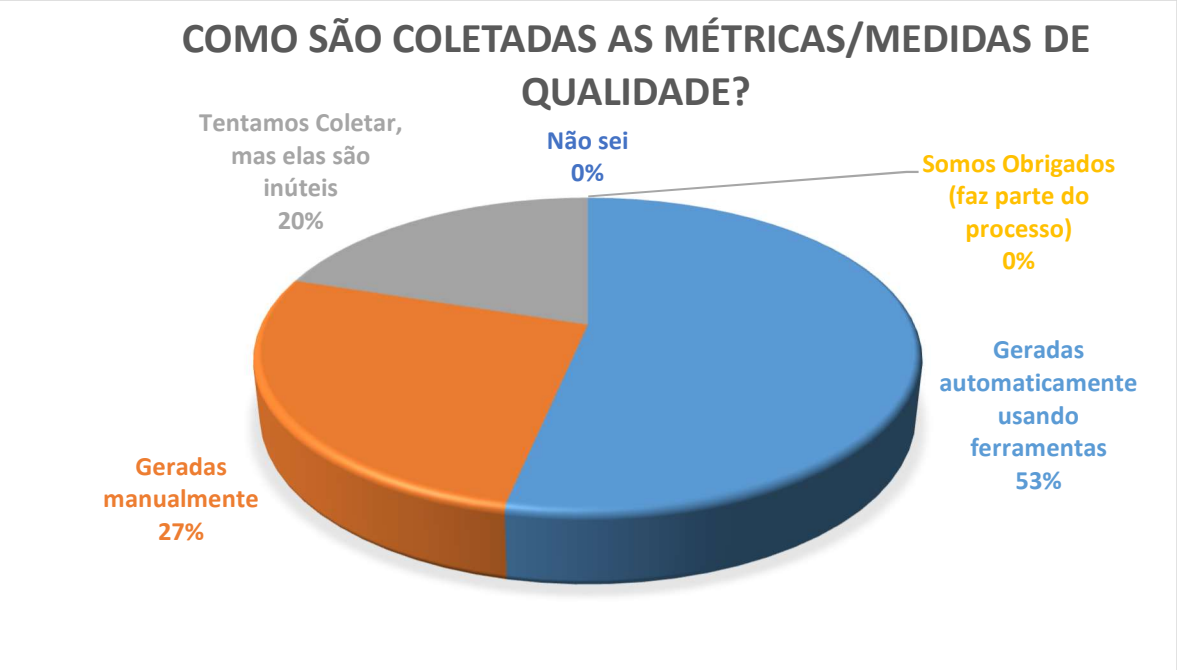


Gráfico 12: Qualidade do código desenvolvido através de metodologias ágeis de acordo com os participantes do *survey.*

As duas últimas questões do *survey* tem o foco em medidas e métricas. A primeira questiona sobre a coleta de medidas ou métricas de qualidade em sua empresa. O resultado mostra que 53% são coletadas automaticamente com o auxílio

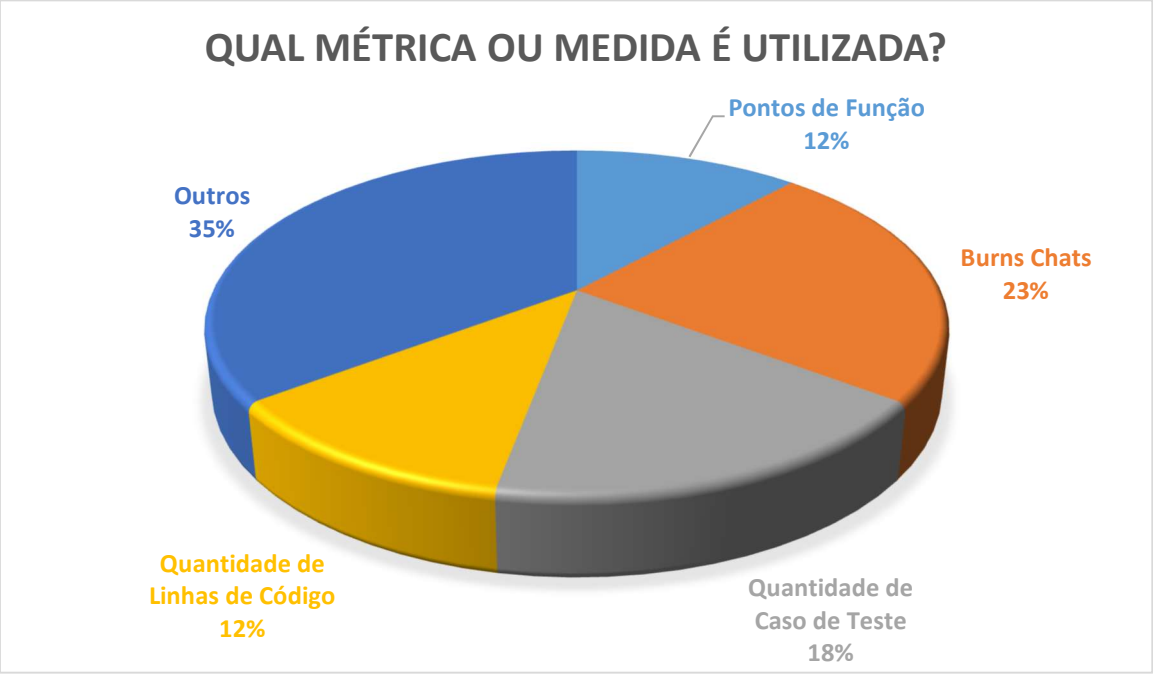
de ferramentas contra 27% das que são coletadas manualmente. Os detalhes estão presentes no gráfico 13.

Gráfico 13: Forma de coleta de métricas de qualidade no desenvolvimento ágil de acordo com os participantes do *survey*.



A última questão da pesquisa mostrou uma lista com métricas comumente utilizadas no processo de desenvolvimento de sistemas. E o resultado mostrou que os participantes usam de forma variada as métricas e que as métricas também são utilizadas quando se desenvolve de forma ágil. O gráfico 14 mostra a distribuição das métricas utilizadas no desenvolvimento ágil de software.

Gráfico 14: Métricas/Medidas utilizadas no desenvolvimento ágil de acordo com os participantes do *survey*.



3.4 Analise dos Dados Qualitativos

Nas questões abertas da *survey* foi possível extrair informações que reforçam a tese de que o comprometimento do desenvolvedor com a metodologia ágil auxilia na qualidade do produto fazendo com que o desenvolvedor se preocupe com o processo de desenvolvimento.

Dentre as respostas, identificamos que os desenvolvedores estão preocupados em gerar documentação para o processo de desenvolvimento, na maioria das respostas, percebemos que eles utilizam o auxílio das práticas presentes na metodologia ágil para elaborar a documentação, muitos citaram que a documentação é criada com o auxílio do *backlog* do produto, que é a lista de requisitos técnicos e de negócio, ordenados por prioridade, junto com outras técnicas como *burn charts* ou *roadmap*.

Outro ponto levantado pelos participantes foi a utilização de ferramentas para o controle de versões. Eles relataram que normalmente utilizam o Git, que é um sistema de controle de versionamento de arquivos.

Quando focamos nas atividades de gerenciamento e controle de qualidade de software percebemos que atividades como os testes (unitário, de carga e de integração), refatoração, programação em pares e retrospectivas, são utilizadas como ferramentas de qualidade. Mostrando que os modelos ágeis foram criados para o alcance de altos níveis de qualidade. Analisando especificamente a atividade de testes percebeu-se que os casos de teste são gerados através das histórias do usuário.

Percebe-se que as práticas presentes no modelo ágil são utilizadas como atividades de controle de qualidade, como exemplo podemos citar as reuniões Scrum, participação ativa do cliente, programação em pares. De fato, o resultado é animador, pois vai ao encontro do descrito na literatura que afirma que as principais atividades de controle de qualidade são os testes, revisões e refatorações.

É interessante observar pelas respostas que nos projetos desenvolvidos por meio de metodologias ágeis existe governança no processo, pois a maioria dos respondentes cita a utilização de retrospectivas como ferramenta de revisão dos planos, processos ou especificações de novos produtos o que indica, novamente, uma cultura de documentar no ambiente de desenvolvimento ágil.

E finalmente, podemos afirmar que o survey fortalece a tese que um dos grandes fatores de qualidade no processo de desenvolvimento ágil é o comprometimento do desenvolvedor com sua atividade, com a utilização das práticas e artefatos presentes na metodologia, assim como o conhecimento da metodologia em si. Nota-se que os participantes acreditam no processo que eles utilizam como a melhor forma de se criar um produto de alta qualidade.

CONCLUSÃO

A principal motivação desse projeto foi o estudo e analise, dentro do modelo de desenvolvimento ágil, da qualidade do software. Apesar de estarem bem difundidos e utilizados pela indústria, ainda falta meios eficientes e diretos de avaliar a qualidade de seus processos e produtos. Podemos observar, também, que o modelo de avalição da qualidade é diferente do utilizado em metodologias tradicionais.

Um *survey* tentar entender o impacto das metodologias ágeis na qualidade e como o comprometimento dos desenvolvedores que utilizam o processo ágil afeta a qualidade do produto desenvolvido. Com a análise dos resultados foi possível perceber que existem atividades de garantia de qualidade bem definidas no processo e que atividades como teste e refatoração são executadas várias vezes durante o processo de desenvolvimento. Isso é devido à natureza iterativa e incremental do processo de construção do sistema. Percebeu-se que o desenvolvedor ágil possui alto conhecimento do processo de desenvolvimento que utiliza, e que a qualidade é de extrema importância e de responsabilidade, não de uma pessoa ou equipe única, mas de todo o time de desenvolvimento.

Pode-se afirmar que quando metodologias ágeis possuem um bom processo de qualidade ela afetará de forma diretamente proporcional a satisfação do cliente, alcançando assim também a satisfação dos stakeholders. Somado a isso, podemos afirmar que quando se melhora a produtividade, teremos melhorias também na satisfação do cliente e na qualidade.

Na metodologia ágil pessoas são a chave para o sucesso do projeto. Assim sendo, o papel do processo em um método ágil é servir de suporte para que o time de desenvolvimento determine a melhor maneira de conduzir o trabalho, além disso, a agilidade preconiza a comunicação face a face com o time e com o cliente que deve estar sempre presente no processo de desenvolvimento. Em um ambiente de desenvolvimento ágil é possível se entender que quem toma a decisão não é tão importante quanto a colaboração para a criação de informação que direciona decisões certas. Um projeto ágil é um ecossistema (Cockburn e Highsmith, 2001a.), no qual processos, pessoas e ambiente tem que interagir de forma dinâmica, com foco na entrega e na satisfação do cliente.

Entre as dificuldades encontradas para realização dessa pesquisa, para a coleta de dados, devido à natureza do tema, algumas questões tinham que ser abertas o que “afugentava” os respondentes fazendo com que o questionário fosse abandonado antes da sua conclusão. Outro fator limitante do survey foi a população pesquisada, pois devido ao curto prazo para coleta de dados, não foi possível generalizar os resultados, mas os estudos e a comparação com outras pesquisas conduzidas por Ambler (2006, 2007, 2008) mostram uma tendência dos dados que servem como base para novas pesquisas.

Por fim, uma sugestão para trabalhos futuros é realizar uma pesquisa para aprofundar a existência e o funcionamento da governança no desenvolvimento de software utilizando metodologias ágeis, analisando como as retrospectivas são executadas após cada iteração, se as retrospectivas são documentadas e for fim se existem uma cultura e uma forma sistemática de documentação no ambiente ágil.

REFERÊNCIAS

ABBAS, N**. Agile software assurance: A empiracal study**. IN: Empirical Software Engineering and Measurement, 2007. ESEM 207. First International Symposium on, 2007.

ABBAS, N. **Software Quality and Governance in Agile Software Development**. Tese de Doutoramento, University of Southampton – Faculty of Engineering, Science, and Mathematics School off Electronics and Computer Sicience, 2009.

ABRAHAMSSON, P. et al. **Agile Software Development Methods**. VTT technical Research Centre of Finland, 2002.

ABRAN, A.; MOORE, J. W. **Guide to the Software Engineering Body of Knowledge**. Angela Burgess, 2004.

AGILE MANIFESTO: **http://agilemanifesto.org/**. Acesso em: 01 ago. 2017.

AMBLER, S. (2006). " Results from Scott Ambler´s 2006 Agile Adoption Rate Survey": www.ambysoft.com. Acesso em 01 nov. 2017.

AMBLER, S. (2007). " Results from Scott Ambler´s 2007 Agile Adoption Rate Survey": www.ambysoft.com. Acesso em 02 nov. 2017.

AMBLER, S. (2008). " Results from Scott Ambler´s 2008 Agile Adoption Rate Survey": www.ambysoft.com. Acesso em 02 nov. 2017.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. **The Goal Question Metric Approach**. 1994.

BECK, K. **Embracing Change With Extreme Programming. Computer**, v. 10, n. 32, p. 70–77, 1999a.

BECK, K. **Extreme Programming Explained: Embracing Change**. Addison Wesley, 1999b.

BECK, K.; ANDRES, C. **Extreme Programming Explained: Engrace Change**. 2nd ed. Addison-Wesley Professional, 2004.

BECK, K.; FOWLER, M. **Planning Extreme Programming**. 1 ed. Boston, MA, USA: Addison Wesley, 139 p., 2001.

BOEHM, B. **Get Ready for Agile Methods, with Care. Computer**, v. 1, 2002a.

BOEHM, B. **Get ready for Agile Methods, with Care**. Computer, v. 1, 2002b.

BOEHM, B.; TURNER, R. **Balancing Agility and Discipline: A Guide for the Perplexed.** Addison-Wesley Longman Publishing Co. Inc., 2003.

BOEHM, B. W. et al **Characteristics of Software Quality**. North-Holland, 1978.

BUDD, T. A. **Mutations analysis: Ideas, examples and prospects. Computer Program Testing** North-Holand Publishing Company, p. 129–148, 1981.

CAROSIA, J.S. **Levantamento da qualidade do processo de software com foco em pequenas organizações.** Dissertação de Mestrado, INPE – Instituto Nacional de Pesquisas Espaciais, 2004.

COHEN, D.; LINDVALL, M.; COSTA, P. An **Introduction to Agile Methods**. Advances in Computers, 2004.

COCKBURN, A. **Agile Software Development.** 1 ed. Boston, MA, USA: Addison Wesley, 2002.

COCKBURN, A. **Crystal Clear a Human Powered Methodology for Small Teams**. Addison Wesley, 2005.

COCKBURN, A.; Highsmith, J. **Agile Development: The people factor**. Computer, 2001a.

COCKBURN, A.; HIGHSMITH, J. **Agile Software Development: The business of innovation**. Computer, 2001b.

COCKBURN, A. et al **Agile Manifesto**, http://agilemanifesto.org/. 2001.

CRESWELL, J. W. **Research design: Qualitative, quantitative, and mixed methods approaches**. SAGE Publications, Inc, 2008.

CROSBY, P.B. **Quality is free**. New York: McGraw-Hill, 1992.

DELAMARO, M.; MALDONADO, J.; JINO, M. **Introdução ao teste de software**. Editora
Campos, 2007.

DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F. G. **Hints on test data selection: Help for the practicing programmer**. IEEE Computer, v. 11, n. 4, p. 34–43, 1978.

FOWLER, M. **The New Methodology**,
http://www.martinfowler.com/articles/newMethodology.html, 2005. Acesso em:
02 jul. 2017.

GALIN, D. **Software Quality Assurance: From Theory to Implementation**. Addison Wesley, 2003

GILLIES, A. C. **Software Quality: Theory and Management**. Lodon: Chapman & Hall, 2002a.

GILLIES, A. C. **Software Quality: Theory and Management**. Lodon: Chapman & Hall, 2002b.

GUASPARI, John. **Tempo: a dimensão da qualidade**; São Paulo. IMAN, 1994.

HARROLD, M. J. **Testing: A roadmap.** Em: Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), New York, NY, USA: ACM Press, 2000.

HIGHSMITH, J. **Adaptive Software Development: a Collaborative Approach to Managing,** Complex Systems. Dorset House Publishing Co., 2000.

HUO, M. et al, **Software Quality and Agile Methods**, COMPSA,04, 2004

IEEE (1990). IEE Standard 610.12-1990 **Glossary of Software Engineerng Terminology**. New York, the Institute of Electrical and Electronics Engineers.

IEEE (1990). IEE Standard 610.12-1990 **Glossary of Software Engineerng Terminology. New Yor**k, the Institute of Electrical and Electronics Engineers.

JURAN, J. M; FEO, J. A. D. **Juran´s quality handbook, sixth edition: The complete guide to performance excellence**. Mcgraw_hill, 2010.

JONES, C. **Apllied Software Measurement: Assuring Productivity and Quality**. McGraw-Hill, INC (1991).

KAN, S. H. **Metrics and Models in Software Quality Engineering**. Boston, MA,: Addison-Wesley Longman Publishing Co, 2002.

LARMAN, C. **Agile and Iterative Development: A Manager´s Guide.** Pearson Education, Inc, 2004.

MALDONADO, J. C. **Critérios potenciais usos: Uma contribuição ao teste estrutural de software**. Tese de Doutoramento, Universidade de Campinas, 1991.

MARTIN, J. **Rapid Application Development,** Macmillan Publishing Co., Inc. 1991.

MCCALL, J. A., P **An Introduction to Software Quality Metrics**. Petrocelli, 1979.

MYERS, G. J. et al, **The art of software testing**. New Jersey: John Wiley and Sons, 2004.

MNKANDLA, E.; DWOLATZKY, B. **Defining Agile Software Quality Assurance**. In: International Conference on Software Engineering Advances (ICSEA'06), 2006a.

MNKANDLA, E.; DWOLATZKY, B. **Defining agile software quality assurance**. In: International Conference on Software Engineering Advances (ICSEA'06), 2006b.

NBR ISO/IEC 9126-1. **Software Engineering – Product quality**. Associação Brasileira de Normas Técnicas, Rio de Janeiro, 2003.

OLIVEIRA, Bruno Henrique. **Qualidade de software no desenvolvimento com métodos ágeis**. Tese de Mestrado. Universidade de São Paulo, 2014.

GALIN, D. **Software Quality Assurance: From Theory to Implementation**. Addison Wesley, 2003

GILLIES, A. C. **Software Quality: Theory and Management**. Lodon: Chapman & Hall, 2002a.

GILLIES, A. C. **Software Quality: Theory and Management. Lodon**: Chapman & Hall, 2002b.

PRESSMAN R.S. **Engenharia de Software – Uma Abordagem Profissional** – 7º Edição. McGraw-Hill, 2011.

ROCHA, A. R. C. **Um modelo para avaliação da qualidade de especificações**. Tese de Doutoramento, PUC Rio, 1983.

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. **Qualidade de Software: Teoria e Prática**. São Paulo: Prentice Hall, 2001.

ROYCE, W. **Software Project Management**, Addison-Wesley,1998.

ROYCE, W. **CMM vs. CMMI: From Conventional to Modern Software Management Rational Software**, 2002.

SINIAALTO, M. ; ABRAHAMSSON, P. **A Comparative Case Study on the Impact of Test-Driven Development on Program Design and Test Covarage**.(2007).

SILVA, F.G; HOENTSCH, S.C.P; SILVA, L. **Uma análise das Metodologias Ágeis FDD e Scrum sob a Perspectiva do Modelo de Qualidade MPS.BR**. Universidade Federal de Sergipe, 2009.

SOMMERVILLE, I. **Engenharia de Software**. 9º Edição. Peason Addison Wesley, 2013.

SCHWABER, K. **"Scrum Development Process"**, OOPSLA'95 Workshop on Business Object Design and Implementation. Springer-Verlag. (1995)

STAMELOS, J. G. e SFETOS P. **Agile Software development Quality Assurance**, IGI Global, 2007

TELES, V. M. a. **Um estudo de caso da adoção de práticas e valores do Extreme Programming**. Tese de Doutoramento, Universidade Federal do Rio de Janeiro, 2005.

WATTS, R. **Measuring Software Quality,** NCC/BlackWell, 1987

ZHU, H.; HALL, P. A. V.; MAY, J. H. R. **Software unit test coverage and adequacy**. ACM Comput. Surv.1997