CENTRO UNIVERSITÁRIO DE ANÁPOLIS – UNIEVANGÉLICA
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

AMAURI PEREIRA SOUSA

OTIMIZAÇÃO DOS PROCESSOS DE DESENVOLVIMENTO DA FÁBRICA DE TECNOLOGIA TURING COM VISTAS AOS PRINCÍPIOS DO SIX SIGMA

ANÁPOLIS – GO
2020

AMAURI PEREIRA SOUSA

OTIMIZAÇÃO DOS PROCESSOS DE DESENVOLVIMENTO DA FÁBRICA DE TECNOLOGIA TURING COM VISTAS AOS PRINCÍPIOS DO SIX SIGMA

Trabalho de conclusão de curso apresentado como requisito parcial para a conclusão da disciplina de Trabalho de Conclusão de Curso II do curso de Bacharelado em Engenharia de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA.

Orientadora: Prof. Ma. Walquíria Fernandes Marins

ANÁPOLIS – GO

2020

AMAURI PEREIRA SOUSA

OTIMIZAÇÃO DOS PROCESSOS DE DESENVOLVIMENTO DA FÁBRICA DE TECNOLOGIA TURING COM VISTAS AOS PRINCÍPIOS DO SIX SIGMA

Trabalho de conclusão de curso apresentado como requisito parcial para a conclusão da disciplina de Trabalho de Conclusão de Curso II do curso de Bacharelado em Engenharia de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA.

Aprovado pela banca examinadora em 7 de Dezembro de 2020, composta por:

_____

Prof. Ma. Walquíria Fernandes Marins

Orientadora

_____

Prof.Dr. Lucas de Almeida Ribeiro

_____

Prof. Ma. Luciana Nishi

Resumo

A otimização dos processos de desenvolvimento de *software* é uma atividade necessária para garantir um produto final correto e de acordo com o solicitado pelo cliente (PRESSMAN, 2011). Neste aspecto, ferramentas de avaliação, normas, padrões e princípios de desenvolvimento de *software* são fundamentais para avaliar os processos executados e gerar indicadores sobre os fluxos realizados, além de levantar pontos de melhorias. Este trabalho busca analisar as ferramentas e normas de avaliação do processo de desenvolvimento de *software*, definindo métodos e parâmetros de avaliação com base nos princípios de metodologias ágeis. Além disso, foi realizado um estudo de caso avaliando a Fábrica de Tecnologia Turing (FTT), levantando pontos de melhorias visando a agilidade do processo. Propondo e implantando, a partir deste estudo, ações de melhoria no processo buscando a qualidade do mesmo e visando os princípios de agilidade. Foram levantados dados sobre o processo de *software* na FTT e mensurada a qualidade do mesmo, sendo encontrado uma qualidade entre o nível 1 e 2 da escala sigma.

Palavras-chaves: Desenvolvimento de *Software*; Metodologias ágeis; Garantia de qualidade; Fábrica de *Software*.

Abstract

The optimization of software development processes is a necessary activity to guarantee a correct final product in accordance with the customer's request (PRESSMAN, 2011). In this respect, assessment tools, norms, standards and principles of software development are essential to assess the processes performed and generate indicators on the flows performed, in addition to raising points for improvement. This work seeks to analyze the tools and standards of evaluation of the software development process, defining methods and evaluation parameters based on the principles of agile methodologies. In addition, a case study was carried out evaluating the Turing Technology Factory (FTT), raising points of improvement aiming at the agility of the process. Proposing and implementing, based on this study, actions to improve the process, seeking the quality of the process and aiming at the principles of agility. Data were collected on the FTT software process and its quality was measured, with a quality found between levels 1 and 2 on the sigma scale.

Keywords*: Software Development; Agile methodologies; Quality assurance; Software factory.*

LISTA DE EQUAÇÕES

Página

Equação 1 – DPMO processo de desenvolvimento de Software da FTT 20/03/2020-14/04/2020. 52
Equação 2 – DPMO processo de desenvolvimento de Software da FTT 01/06/2020-30/06/2020 53
Equação 3 – DPMO processo de desenvolvimento de Software da FTT 13/08/2020 e 15/09/2020. 53

LISTA DE FIGURAS

Página

Figura 1 – Processos da Fábrica de Tecnologia Turing 13
Figura 2 – Como um projeto Scrum é iniciado 21
Figura 3 – Processo de desenvolvimento de software FTT. 37
Figura 4 – Sugestões para melhoria do processo de desenvolvimento de software da FTT. 46
Figura 5 – Processo de desenvolvimento de Software da FTT 2020/2. 49

LISTA DE GRÁFICOS

Página

Gráfico 1 – Período de Engenharia de Computação/Software cursado pelos integrantes da FTT. 39
Gráfico 2 – Função exercida pelos pesquisados na FTT. 40
Gráfico 3 – Tempo que os pesquisados integram a FTT. 40
Gráfico 4 – Conhecimento sobre métodos ágeis dos pesquisados. 40
Gráfico 5 – Conhecimento dos pesquisados sobre o processo de desenvolvimento de software da FTT. 41
Gráfico 6 – Conhecimento dos pesquisados sobre as fases do processo de desenvolvimento de software da FTT. 41
Gráfico 7 – Utilização do processo da FTT no desenvolvimento dos requisitos. 41
Gráfico 8 – Utilização de estratégias para melhoria da qualidade do código no processo de desenvolvimento de software da FTT. 42
Gráfico 9 – Atualização dos requisitos desenvolvidos antes de integrar ao sistema. 42
Gráfico 10 – Conflitos no momento de integrar os requisitos ao sistema. 42
Gráfico 11 – Conhecimento dos pesquisados sobre os procedimentos para não gerar conflitos durante a integração dos requisitos com o sistema. 43
Gráfico 12 – Avaliação dos pesquisados sobre a comunicação das equipes da FTT. 43
Gráfico 13 – Avaliação dos pesquisados sobre a comunicação da equipe de desenvolvimento da FTT. 43
Gráfico 14 – Avaliação dos pesquisados sobre a eficiência do processo de desenvolvimento da FTT. 44
Gráfico 15 – Avaliação dos pesquisados sobre a rotatividade de membros da equipe de desenvolvimento da FTT. 44
Gráfico 16 – Existência de padronização dos códigos desenvolvidos pela equipe de desenvolvimento. 44
Gráfico 17 – Acompanhamento dos líderes da FTT da padronização dos códigos desenvolvidos pela equipe de desenvolvimento. 45
Gráfico 18 – Qualificação dos novos membros da FTT. 45
Gráfico 19 – Avaliação dos pesquisados sobre a eficiência da qualificação dos novos membros da FTT para realizar as primeiras atividades. 45
Gráfico 20 – Existência de procedimento padrão na FTT e execução do processo de desenvolvimento de software. 46
Gráfico 21 – Existência de auditoria no processo de desenvolvimento de software da FTT. 46
Gráfico 22 – Comparação entre as métricas do processo de desenvolvimento de Software da FTT em 2020. 52

LISTA DE QUADROS

Página

Quadro 1 – Objetivos de desempenho Manufatura/Serviços – Qualidade da operação 28
Quadro 2– Metas, atributos e métricas para qualidade de software 30
Quadro 3 – DMADV Processo de desenvolvimento de Software FTT 36
Quadro 4 – Análise SWOT da FTT. 47

LISTA DE TABELAS

Página

Tabela 1 – Significado da Escala Sigma 33
Tabela 2 – Métricas do processo de desenvolvimento de Software da FTT 20/03/2020-14/04/2020. 51
Tabela 3 – Métricas do processo de desenvolvimento de Software da FTT 01/06/2020-30/06/2020. 51
Tabela 4 – Métricas do processo de desenvolvimento de Software da FTT 13/08/2020 e 15/09/2020 51

LISTA DE ABREVIATURAS E SIGLAS

AM *Agile Modeling*.

AP Atributos do processo.

ASD *Adaptive Software Development*.

AUP *Agile Unified Process*.

BPMN *Business Process Model and Notation* (Notação de Modelagem de Processos de Negócio).

DMADV Definir, Medir, Analisar, Projetar [design] e Verificar.

DMAIC Definir, Medir, Analisar, Aperfeiçoar, e Controlar.

DPMO Defeitos por milhão de oportunidades

DPU Defeitos por unidades

DSDM *Dynamic Systems Development Method*.

FDD *Feature Drive Development*.

FTT Fábrica de Tecnologia Turing.

LSD *Lean Software Development*.

MPS.BR Melhoria de Processo do Software Brasileiro.

OMG *Object Management Group*.

RAP Resultados esperados dos atributos do processo.

SQA *Software Quality Assurance* (garantia de qualidade de *software*).

SWOT *Strengths, Weaknesses, Opportunities and Threa*ts (ou FOFA – Forças, Fraquezas, Oportunidades e Ameaças).

TBA *To Be Annouced* (a ser anunciado).

TBD *To Be Determined* (a ser determinado).

UML *Unified Modeling Language* (Linguagem de Modelagem Unificada).

SUMÁRIO

Página

1. INTRODUÇÃO 12

2. FUNDAMENTAÇÃO TEÓRICA 17

2.1. Desenvolvimento de *Software* 17

2.1.1. Metodologias Ágeis 18

2.1.2. Scrum 20

2.1.3. Extreme Programming 22

2.2. Fábrica de *Software* 23

2.3. Normas de Qualidade 24

2.3.1. MPS.BR 25

2.3.2. Métricas de avaliação do processo de desenvolvimento de *software* 27

2.4. Controle e garantia de qualidade 29

2.4.1. *Six Sigma* 32

2.5. Otimização do processo de desenvolvimento de *Software* 33

3. ANÁLISE 35

3.1. Processo de desenvolvimento de *software* FTT 36

3.2. Avaliação do Processo de Desenvolvimento de *Software* 37

3.3. Alterações no processo da FTT 48

3.4. Métricas da FTT 49

4. Considerações Finais 54

REFERÊNCIAS 55

ANEXOS 59

ANEXO A – GRÁFICOS *BURNDOWN* 59

APÊNDICES 61

APÊNDICE A – QUESTIONÁRIO DE AVALIAÇÃO DA FTT APLICADO AOS INTEGRANTES DA EQUIPE DE DESENVOLVIMENTO DE *SOFTWARE* EM 2020/2 61

1. INTRODUÇÃO

A Fábrica de Tecnologia Turing (FTT) é uma fábrica acadêmica que busca desenvolver as habilidades práticas, dos alunos dos cursos de Bacharelados em Computação da UniEVANGÉLICA, através do desenvolvimento de projetos tecnológicos. Segundo a UniEVANGÉLICA (201-?, n. p.) a FTT tem como objetivo:

> Desenvolver habilidades e competências necessárias ao perfil do profissional que atua com tecnologia da informação e da comunicação. Buscar a constante atualização técnica, metodológica, em ferramentas, entre outras. Produzir *software* com qualidade. Ser um ambiente de inovação tecnológica.

Através de uma metodologia de processos híbrida a FTT busca desenvolver e entregar *softwares* com qualidade, visando a satisfação do cliente. A UniEVANGÉLICA (201-?, n. p.) expõe que a "[...] (FTT), utiliza uma metodologia híbrida para o processo de desenvolvimento de *software*, são elas: *Scrum*, *OpenUp* e *Kanban*", conforme é possível observar na Figura 1.

A metodologia *Scrum* é utilizada no gerenciamento de projetos, enquanto que o método *OpenUp* faz uma abordagem interativa durante o ciclo de vida do produto. Faria et. al. (2018, p. 7) ressalta que "A FTT optou utilizar uma metodologia ágil de desenvolvimento híbrida composta pelo *Scrum*, que tem o foco no processo gerencial e o *OpenUP*, que tem foco no processo produtivo [...]".

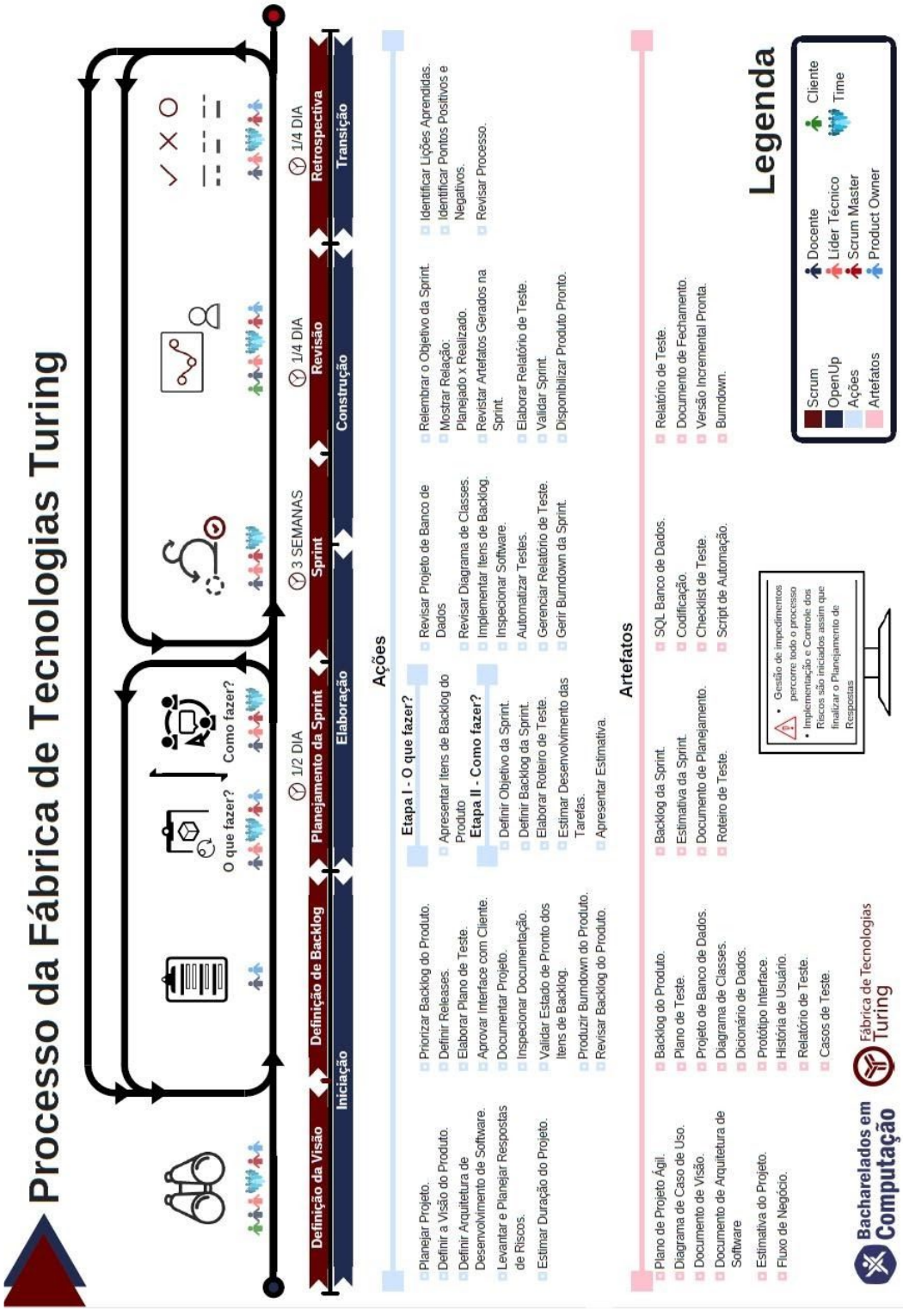
Por outro lado, a metodologia *Kanban* é aplicada no gerenciamento e foco dos projetos. Faria et. al. (2018, p. 9) explicam que "Para acompanhar o desenvolvimento de cada atividade durante a *Sprint*[1] utiliza-se o *Kanban* e para gerenciar o tempo e manter o foco e organização para a produção utiliza-se a técnica *Pomodoro*".

Entretanto, problemas relativos a falta de produtividade, qualidade, eficiência e eficácia durante o processo de desenvolvimento de *software* afetam o produto final, ou seja, o *software* (WANGENHEIM, 2009). Segundo Nuno e Oliveira (201-?,

[1] "A *sprint* é um ciclo fixo, ou iteração, durante o qual a equipe de desenvolvimento deverá transformar requisitos selecionados (histórias de usuário) em um incremento do produto potencialmente entregável". (Pham e Pham, 2011, p. 272)

p. 48) “[...] vários problemas ainda são comuns no desenvolvimento de software. Isso se deve principalmente pelo aspecto não repetitivo do desenvolvimento de produtos de *software*, o que torna a garantia da qualidade uma atividade difícil [...]”.

Figura 1 – Processos da Fábrica de Tecnologia Turing



Fonte: Faria et. al. (2018)

A garantia de qualidade no processo de desenvolvimento de *software* é algo complexo, devido as características do mesmo. Conforme expõe Oliveira (2014, p. 8)

> Isso ocorre por que um *software* não possui existência física, os clientes não sabem exatamente quais são suas necessidades, os requisitos não são imutáveis, a evolução de *hardware* e *software* são extremamente rápidas e a expectativa dos consumidores é sempre alta e pouco gerenciada. Por isso é necessário, a definição de um modelo de qualidade, com métricas tangíveis que reflitam uma forma de avaliar os produtos de *software*.

Assim, definir parâmetros e métodos de avaliação do processo de desenvolvimento de um *software* é necessário para melhorar o rendimento, a agilidade do processo, e o produto final. Portanto o seguinte problema é abordado neste estudo: Como otimizar os processos de desenvolvimento da FTT e garantir a qualidade do *software*?

O objetivo principal deste trabalho foi otimizar o processo de desenvolvimento de *software* da FTT. Este trabalho teve como objetivos específicos: analisar os processos existentes na FTT; avaliar o processo da equipe de desenvolvimento de software da FTT de acordo com modelo MPS.BR; propor melhorias no processo da equipe de desenvolvimento da FTT visando a agilidade; avaliar a implementação de melhorias nos processos da FTT; garantir a qualidade do processo de desenvolvimento da FTT.

A otimização de processos visa a melhoria do desempenho das atividades de uma empresa, procurando reduzir os custos e aumentar a lucratividade. Segundo Wangenheim (2009, p. 6) a melhoria de processos de *software* é uma “Ação executada para mudar os processos de uma organização para que eles sigam as necessidades de negócio da organização, permitindo que ela alcance suas metas de negócio mais efetivamente”.

A implementação de melhorias no processo de desenvolvimento de *software* contribui diretamente nos resultados da empresa, conforme destaca a Universidade do Vale do Itajaí (Univali) (201-?, n. p.)

> Os benefícios desta implantação podem incluir:
> - Aumento da qualidade do processo
> - Aumento da qualidade do produto
> - Maior satisfação do cliente
> - Redução do retrabalho
> - Maior produtividade
> - Redução do tempo para atender o mercado

- Maior visibilidade e controle na execução dos projetos
- Maior competitividade
- Melhor ambiente de trabalho e satisfação dos funcionários

Dessa forma, otimizar os processos de desenvolvimento traz melhorias na competitividade da empresa, além de resultados mais adequados aos objetivos da organização, visando, também, avanços na experiência do cliente. Conforme Jonnalagadda et.al. (2017, p.1, tradução nossa) “Para ser competitivo no mercado atual – em qualquer indústria – as organizações devem entregar uma experiência excepcional ao cliente”[2].

Pensando nesse quesito, ferramentas de controle do processo, verificação do cumprimento dos objetivos propostos, e adequação dos métodos implementados, são necessários para garantir o sucesso do produto desenvolvido. Jonnalagadda et.al. (2017, p.1, tradução nossa) coloca que “[...] Projetos [com métodos] ágeis são 28% mais bem sucedidos do que [métodos] tradicionais, eles ainda tem um nível de risco que pode ser direcionado tendo os controles certos para ajudar a compreender o valor do negócio [...]”[3]

Logo, ferramentas que garantam o curso adequado de uma metodologia ágil são primordiais para o bom desempenho dos processos. Jonnalagadda et.al. (2017, p.6, tradução nossa) destaca que a “Entrega de projetos ágeis inclui muitas ferramentas e técnicas para ajudar a controlar riscos típicos, tais como: analises de probabilidade [...]; gráfico de risco *burndown* [...]; analise SWOT [...]”[4]. Como o exemplo do gráfico *burndown*, no anexo A, que acompanha a evolução das atividades de um projeto através de linhas diagonais, sendo a linha azul o que foi executado (quanto mais à direita da linha vermelha mais atrasado está o projeto), e a linha vermelha o parâmetro de avaliação.

Nesse sentido, é necessário dispor de ferramentas adequadas de avaliação do processo de desenvolvimento que garantam a qualidade do *software*. De acordo com Oliveira (2014, p. 3)

---

[2] *“To be competitive in today’s marketplace – in any industry – organizations must deliver an exceptional customer experience.”*
[3] *“[…] Agile projects are 28% more successful than traditional, they still have a level of risk that can be addressed by having the right controls in place to help realize business value.”*
[4] *“Agile Project Delivery includes many tools and techniques to help control typical risks such as: Probability analysis […]; Risk burndown chart […];SWOT analysis […]”.*

> Apesar da importância dos métodos ágeis e do reconhecimento de sua contribuição positiva para a qualidade do produto e satisfação do cliente, observa-se que os modelos de garantia da qualidade existentes não aderem facilmente a esses métodos, dado a agilidade imposta pelos mesmos.

Portanto, é essencial certificar que as métricas de avaliação adequam-se aos métodos utilizados, bem como, se os resultados alcançarão os objetivos esperados, uma vez que estes interferem diretamente no resultado gerado. Oda (2018, n. p.) explica que “Os métodos são como ferramentas que você deve utilizar de acordo com as necessidades e contextos de seu negócio”. Logo, é preciso garantir que o método e a ferramenta produzirão o produto correto.

Assim, as atividades de garantia da qualidade do processo devem se adequar aos objetivos propostos, como também a metodologia utilizada, a fim de se obter um melhor resultado da avaliação e a melhoria do desenvolvimento de *software*.

Este estudo foi dividido em quatro capítulos sendo o primeiro capítulo aspectos introdutórios, os objetivos, a justificativa e a metodologia utilizada. No segundo capítulo é abordado os fundamentos teóricos que embasam o estudo, e os estudos de casos similares ao realizado neste trabalho. No terceiro capítulo é exposto o estudo realizado na FTT e analisado os dados encontrados. No quarto capítulo são feitas as considerações finais a respeito do trabalho realizado.

2. FUNDAMENTAÇÃO TEÓRICA

A avaliação do processo de desenvolvimento de *software* tem como objetivo a melhoria do processo, ou avaliar a capacidade do mesmo. Conforme coloca Morais e Vasconcelos (201-?, n.p.)

> Uma avaliação é um exame de um ou mais processos realizado por uma equipe de profissionais treinados que usam um modelo de referência com o objetivo de determinar os pontos positivos e negativos do processo. Normalmente ela é aplicada no contexto de melhoria de processos ou avaliação de capacidade.

Avaliar o processo de desenvolvimento de *software* requer ferramentas eficazes de medida. Roses e Vallerão (2013, p. 107) explicam que "[...] A grande complexidade do desenvolvimento de *software* exige que os mecanismos de controle sejam dinâmicos e flexíveis [...]".

Nesse sentido, é preciso compreender o processo de desenvolvimento utilizado para aplicar as ferramentas de controle adequadas, a fim de se obter um resultado apropriado. De modo que, a partir dos resultados adequados, a geração de melhorias sejam efetivas no processo.

2.1. Desenvolvimento de *Software*

O desenvolvimento é um processo composto por várias tarefas para se chegar ao produto final: o *software*. De acordo com Lima (2012, p. 3):

> Processo de *software* é o conjunto de tarefas necessárias para a construção de *softwares* de qualidade e inclui a definição do modelo de ciclo de vida, a estrutura organizacional, as atividades a serem executadas, as práticas a serem seguidas e os artefatos a serem produzidos.

Nesse contexto, o processo de *software* é composto por diversas atividades que resultam no produto final. Entretanto, esse processo é flexível ao trabalho à ser realizado, como Pressman (2011, p. 40) enfatiza:

> No contexto da engenharia de *software*, um processo *não* é uma prescrição rígida de como desenvolver um *software*. Ao contrário, é uma abordagem adaptável que possibilita às pessoas (a equipe de *software*) realizar o trabalho de selecionar e escolher o conjunto apropriado de ações e tarefas.

O processo de desenvolvimento conta ainda com uma metodologia que define as atividades básicas necessárias para a entrega do *software*. Pressman (2011, p. 40) explica que

> Uma *metodologia (framework) de processo* estabelece o alicerce para um processo de engenharia de *software* completo, por meio da identificação de um pequeno número de *atividades estruturais* aplicáveis a todos os projetos de *software*, independentemente de tamanho ou complexidade. Além disso, a metodologia de processo engloba um conjunto de *atividades de apoio* (*umbrela activities – abertas*) aplicáveis em todo o processo de *software*.

Logo, o processo de desenvolvimento de *software* é diretamente afetado pela metodologia de trabalho escolhida. Assim, a partir da metodologia de desenvolvimento é possível analisar quais atividades deverão ser executadas e a estrutura de trabalho a ser priorizada.

2.1.1. Metodologias Ágeis

As metodologias ágeis foram criadas visando maior adaptabilidade e o andamento mais rápido dos projetos. De acordo com Santos e Córdova (2017, p. 8) “[...] as metodologias ágeis surgiram com a intenção de priorizar mais as pessoas do que os processos, despendendo menos tempo com documentação e mais tempo com a resolução dos problemas do projeto”.

Essas metodologias seguem um conjunto de princípios, que ficou conhecido como manifesto ágil. Beck et. al. (2001, n. p.) estabeleceram os 12 princípios que orientam essas metodologias, sendo eles:

> 1. Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de *software* de valor.
> 2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
> 3. Entregar *software* funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
> 4. Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
> 5. Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
> 6. O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.
> 7. *Software* funcional é a medida primária de progresso.
> 8. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.
> 9. Contínua atenção à excelência técnica e bom *design*, aumenta a agilidade.
> 10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
> 11. As melhores arquiteturas, requisitos e *designs* emergem de times auto organizáveis.
> 12. Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

A aplicação desses princípios visam a redução de custos, de prazos de entrega, buscando a satisfação do cliente, além de produtos de qualidade. Propiciando, assim, um processo de desenvolvimento com enfoque nas reais necessidades do cliente, e executando de forma mais funcional e adequada ao problema. Mazuco (2017, p. 54) salienta que “[...] há forte evidência das práticas de metodologias ágeis sendo empregadas nas empresas e que elas estão sendo bem aceitas, tanto por parte de funcionários, alto ou baixo escalão, quanto de seus clientes [...]”.

Dentre as metodologias que aplicam os princípios de agilidade estão: *Extreme Programming*, *Adaptive Software Development* (ASD), *Scrum*, *Dynamic Systems Development Method* (DSDM), *Crystal*, *Feature Drive Development* (FDD), *Lean Software Development* (LSD), *Agile Modeling* (AM), *Agile Unified Process* (AUP),

(PRESSMAN, 2011). Este trabalho aborda somente as metodologias *Scrum*, e *Extreme Programming* uma vez que o estudo de caso está relacionado a essas.

2.1.2. Scrum

O termo *Scrum* surgiu em 1986 ao ser publicado em um artigo na *Havard Business Review*. Porém a metodologia *Scrum* foi criada somente em 1995 por Jeff Sutherland e Ken Schwaber a pedido da *Object Management Group* (OMG) onde resumiram em conjunto a experiência de anos em busca de melhorar o processo de *software* e aumentar a produtividade das equipes, (PHAM e PHAM, 2011).

O processo de desenvolvimento de *software* com *Scrum* é feito através de uma abordagem iterativa e incremental, conforme coloca Cohn (2011, p. 277)

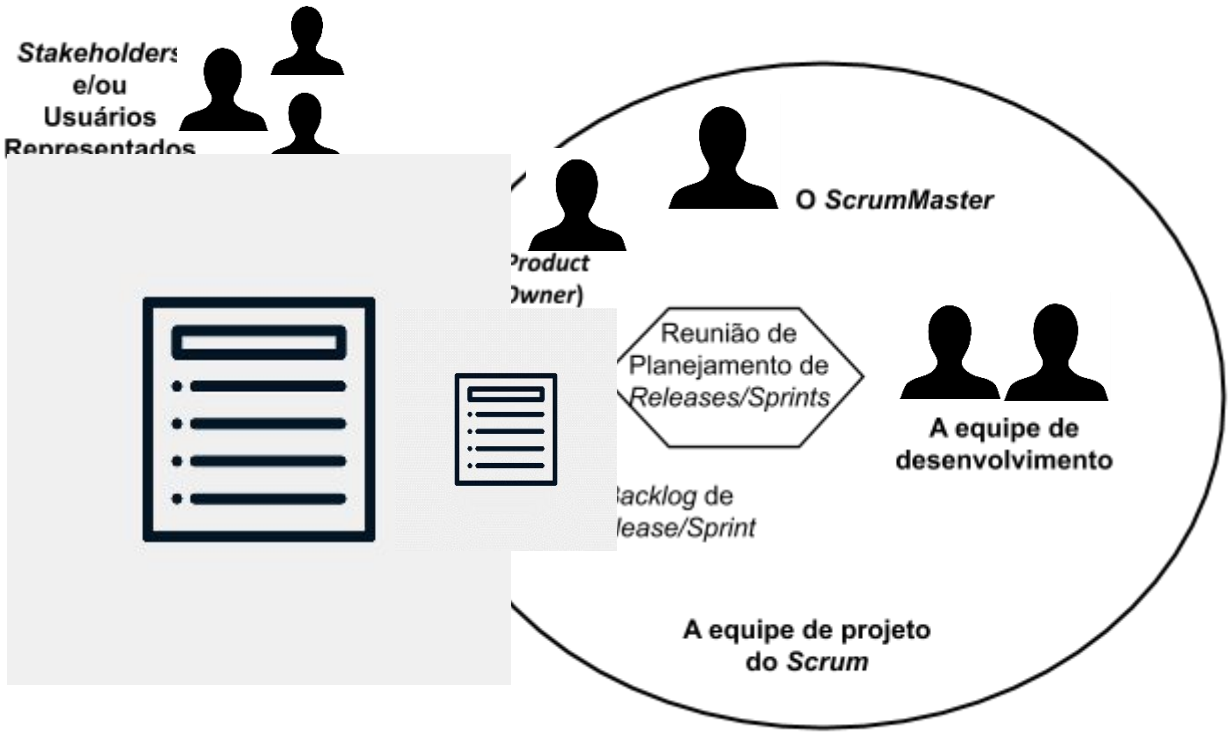
> Como todos os processos ágeis, o *Scrum* é uma abordagem de desenvolvimento de *software* iterativa e incremental. [...] Portanto em um processo incremental, desenvolvemos completamente um requisito e então passamos para o próximo. Em um processo iterativo, construímos o sistema inteiro, mas inicialmente fazemos isso de forma imperfeita, usando repetidas passagens pelo sistema todo para melhorá-lo. Os pontos fracos inerentes às abordagens só interativa ou só incremental desaparecem quando elas são combinadas, como acontece no *Scrum*.

O desenvolvimento de *software* utilizando *Scrum* inicia-se com a coleta de requisitos, obtidas através de informações fornecidas pelos *stakeholders* e/ou usuários que os representem. A figura 2 representa o início de um projeto utilizando *Scrum*.

Na figura 2 estão destacadas as pessoas envolvidas em um projeto utilizando *Scrum*, bem como a equipe que compõe o projeto. Segundo Pham e Pham (2011, p. 43) “[...] tudo começa com o *Product Owner*, que é responsável por obter informações dos *stakeholders*, ou usuários que os representem, para elaborar uma lista de requisitos e criar um *Backlog* de Produto”[5].

[5] “O *Backlog* de Produto é uma lista de requisitos priorizada, que pode incluir de tudo: de aspectos do negócio a tecnologias, questões técnicas e correções de bugs”. (Pham e Pham, 2011, p. 43)

Figura 2 – Como um projeto *Scrum* é iniciado



Fonte: Pham e Pham, 2011, p. 42

Após a fase inicial o *Product Owner* se reúne com a equipe do projeto para o planejamento de *realeses*/*sprints*, conforme ilustrado na figura 2. Pham e Pham (2011, p. 44) colocam que a

> [...] reunião de Planejamento de *Sprints*, que focaliza o "Como", a equipe de desenvolvimento tentará identificar tarefas (*tasks*) a partir das histórias previamente escolhidas e deduzir quanto tempo (em horas) a equipe levará para transformar essas tarefas em incrementos de produtos potencialmente entregáveis.

Com o planejamento pronto, o trabalho inicia-se e ocorrem as *Sprints*. Pham e Pham (2011, p. 44) explica que "Tão logo as reuniões de Planejamento de *Releases* e Planejamento de *Sprints* tenham terminado, inicia-se o trabalho propriamente dito dos *Sprints* [...], com os 15 minutos do *Daily Scrum* (Scrum Diário) ou *Daily Standup* (algo como Reunião Diária em Pé)".

A duração de uma S*print* pode variar de uma a quatro semanas, e antes da entrega de cada *Sprint* é feita uma reunião entre a equipe de desenvolvimento e o *Product Owner* para uma Revisão de *Sprint* (como um mecanismo de Inspeção e Adaptação do *Scrum*), (PHAM e PHAM, 2011).

2.1.3. Extreme Programming

O *Extreme Programming*, mais conhecido como XP, surgiu na década de 1990 com o objetivo de agilidade e satisfação do cliente. O XP é baseado em quatro valores fundamentais, sendo eles: *Feedback* (o cliente aprende com o sistema, re-avalia as suas necessidades, e gera *feedback* para a equipe de desenvolvimento), Comunicação (entre cliente e equipe, permitindo atenção aos detalhes e agilidade), Simplicidade (implementar apenas aquilo que é suficiente), e Coragem (a equipe precisa ter coragem e acreditar que o método XP será capaz de fazer o *software*) (NUNES, 2016).

O XP reúne um conjunto de técnicas e possui treze práticas importantes, Medeiros (2013) descreve essas práticas como:

Versões Pequenas: *Releases* são construídas ao longo de iterações. Uma iteração sempre alcança algum objetivo perceptível ao cliente, ou seja, não adianta usarmos uma iteração para projetarmos ou melhorarmos a arquitetura do nosso *software* se ele não vai agregar absolutamente nenhum valor ao cliente. Nada é feito que não seja imediatamente útil e necessário para não afetar os prazos de desenvolvimento. As iterações são divididas em tarefas que são a menor quantidade de trabalho que pode ser feita até que todos os testes voltem a funcionar. [...]
Jogo do Planejamento: Seu objetivo é determinar rapidamente o escopo da próxima Release, combinando prioridades do negócio e estimativas técnicas. Portanto no XP planejamos o tempo todo, diferente do que alguns pensam. Com um cliente presente ele define o escopo para a próxima Release. Dessa forma, define-se as features, prioridades e o escopo da release. [...]
Teste: A prática de teste no XP é bastante técnica, e envolve a presença do cliente no desenvolvimento e na validação de testes. O cliente compartilha com o desenvolvedor sobre o funcionamento do sistema. Os testes também se tornam as especificações da programação, visto que o teste diz o que deve estar de acordo e o que não deve estar de acordo, é como uma especificação.
Programação em pares: Trata-se de duas pessoas trabalhando com UMA máquina onde um codifica, e o outro crítica ou dá sugestões. Os pares trocam de lugar periodicamente. Essa prática é excelente e favorece comunicação e aprendizado. [...]
Projeto Simples: [...] O XP preconiza que a mudança é barata, pois ele utiliza ciclos curtos, projetos simples, refatorações, por isso mantém-se o projeto o mais simples possível, modificando-o quando for necessário suportar mais funcionalidade. [...]
Refatoração: A refatoração significa melhorar o código sem alterar sua funcionalidade. Antes de uma mudança sempre refatoramos o código para facilitar a realização de mudanças. Ou seja, se após a refatoração o código continua funcionando como anteriormente, incluímos as novas mudanças.
Propriedade Coletiva: Todos podem modificar o código a qualquer momento. Códigos não pertencem a apenas uma pessoa. A melhor forma de evitarmos problemas como trocas de pessoas da equipe e com isso a perda de conhecimento é a propriedade coletiva, onde todos mexem em todas as partes do programa e conhecem de tudo um pouco. [...]
Integração Contínua: Todo código deve ser integrado diariamente e todos testes devem passar antes e depois da integração. Se algum problema é encontrado ele deve ser corrigido imediatamente.
Cliente presente: Clientes devem estar presentes para escrevem testes de aceitação, definirem prioridades e histórias para as futuras iterações.
Semana de 40 horas: O XP preconiza que não se pode trabalhar horas extras por mais de uma semana, pois trabalho extra é sintoma de que algo está errado. Devemos manter um ritmo sustentável.
Padrões de Codificação: Todos mexem em todos os códigos, todos refatoram e todos trabalham em pares. Assim é interessante mantermos um padrão para termos algo solidificado. Por isso a melhor forma é a equipe definir um padrão de codificação sempre no início dos projetos.
Metáfora: é uma linguagem comum que todos devem possuir. Por exemplo, ao invés de descrevermos como uma certa arquitetura funciona apenas comunicamos o seu nome e todos entendem o que um programador quis dizer.
Reunião diária: é uma prática vinda do SCRUM em que todos fazem uma rápida reunião de pé para discutir o que foi feito no dia anterior, o que será feito no dia atual e se existe algum impedimento.

Essas práticas trazem dinamismo para a metodologia XP, além de simplicidade, conferindo maior interação entre toda a equipe. Silva, Santos e Shibao (2019, p.172) destacam que “Uma das principais características do XP é que o processo de desenvolvimento é a codificação do que o cliente especifica. Nenhuma ferramenta ou funcionalidade é projetada antecipadamente sem necessidade [...]”. Portanto, essa metodologia proporciona uma maior comunicação com o cliente e entregas mais rápidas e constantes.

2.2. Fábrica de *Software*

O processo de desenvolvimento de *software* não é afetado somente pela metodologia utilizada, mas também pela gestão do processo. Fernandes e Teixeira (2011, p. 75) explicam que

> A gestão estratégica do processo de *software* foca sua melhoria continua e seu alinhamento constante com o negócio, principalmente no que tange à implementação de melhorias nos processos ou novos processos e novas tecnologias, visando à entrega de funcionalidades com melhor qualidade (no prazo, custo e escopo requeridos pelo negócio) e de forma mais rápida.

Logo, a constante melhoria do processo de desenvolvimento de *software* compõe a gestão estratégica de uma fábrica de *software*, buscando a qualidade e a agilidade. A gestão para melhoria do processo realiza-se através das medições que avaliam o processo executado, Fernandes e Teixeira (2011, p. 108) colocam que

> [...] a gestão para a melhoria somente se concretiza através de medições, que permitem um entendimento do comportamento do processo. Esse entendimento é a base para que se mantenha o processo sob controle e para que sejam estabelecidas metas de melhoria. Na área de engenharia de *software*, essas medições são conhecidas como métricas.

Consequentemente, um processo de avaliação necessita de parâmetros e indicadores para definição do que é esperado de cada processo, e assim examinar o que é executado definindo uma medida para a adequação (ou falta dela) entre os

quesitos avaliados. Nesse quesito, os modelos (normas ou padrões) de qualidade oferecem um guia para a melhoria da qualidade nas operações de Fábricas de *Software*, (Fernandes e Teixeira, 2011).

2.3. Normas de Qualidade

Existem diversas normas que contribuem para a melhoria da qualidade de um *software*, estabelecendo padrões e métricas para analisar diversos aspectos conforme ressalta Maciel, Valls e Savaione (2011, n. p.)

> Padrões e normas servem para medir vários aspectos da qualidade de *software* dentre eles: a qualidade do produto, qualidade do processo de desenvolvimento e o nível de maturidade da organização desenvolvedora, com o objetivo de atingir a melhoria da qualidade contínua.

Dentre as normas de qualidade existentes destacam-se: a ABNT NBR ISO/IEC 29110; a ISO 9001; a CMM/CMMI; e a MPS.BR (Melhoria de Processo do *Software* Brasileiro), (MACIEL, VALLS e SAVAIONE, 2011). Essas normas são referências, ou regulamentações, que certificam o desenvolvimento de *software* e o produto final, garantindo a qualidade. Dentre as normas citadas este trabalho enfatiza o modelo MPS.BR, uma vez que é utilizada no estudo de caso.

2.3.1. MPS.BR

O modelo MPS.BR tem como objetivo aprimorar a avaliação do processo de *software*, como destaca Machado, Rocha e Souza (2012, p. 12-13)

> Uma das metas do Programa MPS.BR é definir e aprimorar um modelo de melhoria e avaliação de processo de *software* e serviços, visando preferencialmente às micro, pequenas e médias empresas (mPME), de forma a atender as suas necessidades de negócio e ser reconhecido nacional e internacionalmente como um modelo aplicável à indústria de *software* e serviços.

O modelo de avaliação MPS.BR foi desenvolvido em conformidade com as normas internacionais de qualidade ISO/IEC 12207:2008, ISO/IEC 20000:2011, ISO/IEC 15504-2, CMMI-DEV e CMMI-SVC, (MACHADO, ROCHA e SOUZA, 2012). Além disso, o modelo conta com níveis de maturidade do processo que vão do A até o G, sendo o nível A o mais avançado. De acordo com Machado, Rocha e Souza (2012, p. 17-18)

> Os níveis de maturidade estabelecem patamares de evolução de processos, caracterizando estágios de melhoria da implementação de processos na organização. O nível de maturidade em que se encontra uma organização permite prever o seu desempenho futuro ao executar um ou mais processos. O MR-MPS-SW [Modelo de Referência MPS para Software] define sete níveis de maturidade: A (Em Otimização), B (Gerenciado Quantitativamente), C (Definido), D (Largamente Definido), E (Parcialmente Definido), F (Gerenciado) e G (Parcialmente Gerenciado). A escala de maturidade se inicia no nível G e progride até o nível A. Para cada um destes sete níveis de maturidade é atribuído um perfil de processos que indicam onde a organização deve colocar o esforço de melhoria.

Os processos atribuídos para cada nível de maturidade são descritos através de propósitos e resultados esperados, e a capacidade de uma fábrica de *software* é avaliada de acordo com o atendimento aos atributos do processo. Conforme destaca Machado, Rocha e Souza (2012, p. 18) “A capacidade do processo é representada por um conjunto de atributos de processo descrito em termos de resultados esperados”.

Além disso, Machado, Rocha e Souza (2012, p. 18) colocam que “O atendimento aos atributos do processo (AP), pelo atendimento aos resultados esperados dos atributos do processo (RAP), é requerido para todos os processos no nível correspondente ao nível de maturidade [...]”. Os níveis de maturidade são descritos através de nove atributos do processo, (Machado, Rocha e Souza, 2012).

Este trabalho concentra-se apenas nos dois primeiros atributos do processo referente ao primeiro nível de maturidade (nível G). O primeiro atributo é o processo é executado (avalia o quanto o processo atinge o seu propósito), o segundo atributo é o processo é gerenciado (avalia o quanto o processo é gerenciado), (MACHADO, ROCHA e SOUZA, 2012).

Em relação aos resultados esperados dos atributos do processo, referente à gerência do projeto, Machado e Zabeu (2016, p. 10-29) levantam os seguintes pontos esperados

> 1- O escopo do trabalho para o projeto é definido [...]
> 2- As tarefas e os produtos de trabalho do projeto são dimensionados utilizando métodos apropriados [...]
> 3- O modelo e as fases do ciclo de vida do projeto são definidos [...]
> 4- O esforço e o custo para a execução das tarefas e dos produtos de trabalho são estimados com base em dados históricos ou referências técnicas [...]
> 5- O orçamento e o cronograma do projeto, incluindo a definição de marcos e pontos de controle, são estabelecidos e mantidos [...]
> 6- Os riscos do projeto são identificados e o seu impacto, probabilidade de ocorrência e prioridade de tratamento são determinados e documentados [...]
> 7- Os recursos humanos para o projeto são planejados considerando o perfil e o conhecimento necessários para executá-lo [...]
> 8- Os recursos e o ambiente de trabalho necessários para executar o projeto são planejados [...]
> 9- Os dados relevantes do projeto são identificados e planejados quanto à forma de coleta, armazenamento e distribuição. Um mecanismo é estabelecido para acessá-los, incluindo, se pertinente, questões de privacidade e segurança [...]
> 10- Um plano geral para a execução do projeto é estabelecido com a integração de planos específicos [...]
> 11- A viabilidade de atingir as metas do projeto é explicitamente avaliada considerando restrições e recursos disponíveis. Se necessário, ajustes são realizados [...]
> 12- O Plano do Projeto é revisado com todos os interessados e o compromisso com ele é obtido e mantido [...]
> 13- O escopo, as tarefas, as estimativas, o orçamento e o cronograma do projeto são monitorados em relação ao planejado [...]
> 14- Os recursos materiais e humanos bem como os dados relevantes do projeto são monitorados em relação ao planejado [...]
> 15- Os riscos são monitorados em relação ao planejado [...]
> 16- O envolvimento das partes interessadas no projeto é planejado, monitorado e mantido [...]
> 17- Revisões são realizadas em marcos do projeto e conforme estabelecido no planejamento [...]
> 18- Registros de problemas identificados e o resultado da análise de questões pertinentes, incluindo dependências críticas, são estabelecidos e tratados com as partes interessadas [...]
> 19- Ações para corrigir desvios em relação ao planejado e para prevenir a repetição dos problemas identificados são estabelecidas, implementadas e acompanhadas até a sua conclusão [...].

Enquanto que os resultados esperados, em relação à gerência de requisitos, Machado e Zabeu (2016, p. 32-29) colocam os seguintes pontos esperados

> 1- O entendimento dos requisitos é obtido junto aos fornecedores de requisitos [...]
> 2- Os requisitos são avaliados com base em critérios objetivos e um comprometimento da equipe técnica com estes requisitos é obtido [...]
> 3- A rastreabilidade bidirecional entre os requisitos e os produtos de trabalho é estabelecida e mantida [...]
> 4 - Revisões em planos e produtos de trabalho do projeto são realizadas visando a identificar e corrigir inconsistências em relação aos requisitos [...]
> 5 - Mudanças nos requisitos são gerenciadas ao longo do projeto [...]

Os resultados esperados constitui um referencial para a avaliação do processo de *software*. Porém outras métricas contribuem, também, para uma análise profunda do processo de desenvolvimento de *software*, garantindo também a melhoria continua.

2.3.2. Métricas de avaliação do processo de desenvolvimento de *software*

Os indicadores de desempenho constituem uma importante fonte de informação sobre o processo de desenvolvimento de *softwares*. Fernandes e Teixeira (2011, p. 75) apontam que

> [...] devemos considerar um conjunto mínimo de indicadores tais como:
> ● aprendizagem dos colaboradores;
> ● confiabilidade de entrega da operação;
> ● qualidade dos produtos;
> ● atendimento aos prazos padrões da Fábrica de *Software*;
> ● velocidade na implantação de novas linhas de serviços;
> ● flexibilidade no atendimento a flutuações de demanda;
> ● flexibilidade de atendimento a múltiplas plataformas etc.

Os indicadores definem aspectos relevantes sobre o processo de desenvolvimento em uma fábrica de *softwares*, porém é necessário analisar estes sob a perspectiva dos objetivos de desempenho. Fernandes e Teixeira (2011) levantam os objetivos de desempenho relativos a qualidade da operação, conforme exposto no quadro 1.

Observando o quadro 1 é possível identificar que as características de cada objetivo alteram conforme o produto ou serviço desenvolvido pela fábrica de

software, com exceção do último objetivo (independente da operação o objetivo é produzir com o menor custo possível). Outro objetivo, citado no quadro 1, que chama atenção é a flexibilidade na entrega, enquanto que em projetos e manutenção de *software* é mais difícil de ser atingida, em programação é mais fácil de se atingir, porém é ressaltado a necessidade de controle de tarefas e da alocação dos recursos humanos.

Quadro 1 – Objetivos de desempenho Manufatura/Serviços – Qualidade da operação

Objetivos de Desempenho Manufatura/ Serviços	Projeto de *Software*	Manutenção de *Software*	Programação
Qualidade da Operação			
Qualidade de conformidade	*Software* atende aos requisitos alocados.	Solicitação atende aos requisitos.	Programa atende às especificações.
Confiabilidade da Operação	Entregar *software* no prazo combinado.	Entregar a solicitação no prazo combinado.	Entregar o programa no prazo combinado.
Velocidade	Prazo do projeto.	Prazo de atendimento.	Prazo de programação.
Flexibilidade de variedade de produtos e serviços	Vários ambientes de *hardware* e *software*, várias metodologias; pessoal com várias habilidades; várias ferramentas de apoio ao desenvolvimento e gestão; vários domínios de aplicação.	Vários ambientes de *hardware* e *software*, várias metodologias; pessoal com várias habilidades; várias ferramentas de apoio ao desenvolvimento e gestão; vários domínios de aplicação.	Várias linguagens de programação; vários *softwares* de banco de dados; várias ferramentas para geração de massa de teste; várias ferramentas de análise de código; programadores com habilidade em várias linguagens.
Flexibilidade de entrega	Difícil de ser atingida.	Difícil de ser atingida; depende do controle de alocação de recursos humanos e do controle das tarefas.	Mais fácil de ser atingida, mas depende também do controle das tarefas e da alocação dos recursos humanos.
Flexibilidade de volume	É possível; replicação do ambiente depende de financiamento.	É possível; replicação do ambiente depende de financiamento.	É possível, depende da capacidade instalada e da estratégia de programação da produção.
Flexibilidade de introdução de novos ambientes de desenvolvimento	Velocidade é baixa em função da curva de aprendizagem, considerando um processo padrão de desenvolvimento.	Velocidade é baixa em função da curva de aprendizagem, considerando um processo padrão de desenvolvimento.	Velocidade é baixa em função da curva de aprendizagem, considerando um processo padrão de desenvolvimento.
Custo	O mais baixo custo.	O mais baixo custo.	O mais baixo custo.

Fonte: Fernandes e Teixeira (2011, p. 52)

Portanto, indicadores e métricas são relevantes ao avaliar o processo de desenvolvimento de *software*, contribuindo para atingir os objetivos de uma fábrica de *softwares*. Entretanto, esses indicadores devem ser acompanhados por mecanismos e processos de controle e garantia da qualidade, para que assim seja realizada a otimização e a melhoria continua dos processos realizados.

2.4. Controle e garantia de qualidade

O controle de qualidade é uma série de atividades que buscam auxiliar para que o *software* seja entregue alcançando os objetivos de qualidade definidos. Como explica Pressman (2011, p. 370),

> O controle de qualidade engloba um conjunto de ações de engenharia de *software* que ajudam a garantir que cada produto resultante atinja suas metas de qualidade. [...] Uma combinação de medições e realimentação (*feedback*) permite a uma equipe de *software* ajustar o processo quando qualquer um desses produtos resultantes deixe de atender às metas estabelecidas para a qualidade.

Enquanto que a garantia de qualidade de *software* ou SQA (*software quality assurance*) utiliza uma variedade de relatórios e inspeções para avaliar o controle de qualidade e implantar uma estrutura para o desenvolvimento de *software* de qualidade. Conforme ressalta Pressman (2011, p. 370),

> A garantia de qualidade estabelece a infraestrutura que suporta métodos sólidos de engenharia de *software*, gerenciamento racional de projeto e ações de controle de qualidade – todos fundamentais para a construção de *software* de qualidade. Além disso, a garantia da qualidade consiste em um conjunto de funções de auditoria e de relatórios que possibilita uma avaliação da efetividade e da completude das ações de controle de qualidade.

Dessa forma, a garantia de qualidade é composta de uma série de atividades de medição, observação e registros do processo de desenvolvimento de *software* visando medir o atingimento de metas de qualidade. Essas metas devem ser alcançáveis e buscar alcançar a qualidade dos requisitos, a qualidade do projeto, a qualidade do código, e a eficácia do controle de qualidade, (PRESSMAN, 2011).

Cada meta de qualidade possui atributos e métricas para indicar a existência de qualidade ou não, como é possível observar no quadro 2, na página seguinte, onde cada meta de qualidade apresenta um conjunto de atributos e métricas correspondente a um aspecto de qualidade do processo de *software*.

Quadro 2– Metas, atributos e métricas para qualidade de *software*

Meta	Atributo	Métrica
Qualidade das necessidades	Ambiguidade	Número de modificadores ambíguos (por exemplo, muitos, grande, amigável)
	Completude	Número de TBA (*To Be Annouced* – a ser anunciado), TBD (*To Be Determined* – a ser determinado)
	Compreensibilidade	Número de seções/subseções
	Volatilidade	Número de mudanças por requisito
		Tempo (por atividade) quando é solicitada a mudança
	Facilidade de atribuição	Número de requisitos não atribuíveis ao projeto/código
	Clareza do modelo	Número de modelos UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada)
		Número de páginas descritivas por modelo
		Número de erros UML
Qualidade do projeto	Integridade da arquitetura	Existência do modelo da arquitetura
	Completude dos componentes	Número de componentes que se atribui ao modelo da arquitetura.
		Complexidade do projeto procedural
	Complexidade da interface	Número médio de cliques para checar a uma função ou conteúdo típico
		Apropriabilidade do *layout*
Qualidade do código	Padrões	Número de padrões usados
	Complexidade	Complexidade ciclométrica
	Facilidade de manutenção	Fatores do projeto
	Compreensibilidade	Porcentagem de comentários internos
		Convenção de atribuição de variáveis
	Reusabilidade	Porcentagem de componentes reutilizados
	Documentação	Índice de legibilidade
Eficiência do controle de qualidade	Alocação de recursos	Porcentagem de horas de pessoal por atividade
	Taxa de completude	Tempo de finalização real *versus* previsto
	Eficácia da revisão	Métricas de revisão
	Eficácia dos testes	Número de erros encontrados e criticalidade
		Esforço exigido para corrigir um erro
		Origem do erro

Fonte: Pressman (2011, p. 392)

As métricas apresentadas no quadro 2 geram indicadores importantes sobre o processo de *software* mostrando pontos de erros e defeitos a serem corrigidos. Essas métricas podem ser representadas através de dados quantitativos e apresentadas através de estatísticas da garantia de qualidade. Um método de apresentar os dados do processo é através do *Six Sigma* (Seis Sigma) que consiste em utilizar análise estatística de dados para medir e melhorar o desempenho do processo, (PRESSMAN, 2011).

O método usado no *Six Sigma* é dividido em fases onde inicialmente são reconhecidos, medidos e analisados os atributos e métricas do processo. Segundo Pressman (2011, p. 394-395)

> A metodologia *Seis Sigma* define três etapas essenciais:
> - *Definir* as necessidades do cliente e os artefatos passíveis de entrega, bem como as metas de projeto através de métodos bem definidos da comunicação com o cliente.
> - *Medir* o processo existente e seu resultado para determinar o desempenho da qualidade atual (reunir métricas para defeitos).
> - *Analisar* as métricas para defeitos e determinar as poucas causas vitais.
>
> Se já existir uma gestão de qualidade, e for necessário um aperfeiçoamento, a estratégia Seis Sigma sugere duas etapas adicionais
> - *Melhorar* o processo por meio da eliminação das causas fundamentais dos defeitos.
> - *Controlar* o processo para garantir que trabalhos futuros não reintroduzam as causas dos defeitos.
>
> Essas etapas essenciais e adicionais são, algumas vezes, conhecidas como método DMAIC (**d**efinir, **m**edir, **a**nalisar, aperfe**i**çoar, e **c**ontrolar).
> Se uma organização estiver desenvolvendo uma gestão de qualidade (e não aperfeiçoando uma já existente), nas etapas essenciais são incluídas:
> - *Projetar* o processo para: (1) evitar as causas fundamentais dos defeitos e (2) atender as necessidades do cliente.
> - *Verificar* se o modelo de processos irá, de fato, evitar defeitos e atender as necessidades do cliente.
>
> Essa variação é algumas vezes denominada método DMADV (**d**efinir, **m**edir, **a**nalisar, projetar, [**d**esign] e **v**erificar).

Ainda sobre o método *Six Sigma*, são incluída duas etapas adicionais conforme a necessidade de gestão da qualidade, se a organização estiver aperfeiçoando são incluídas as fases de melhorar e controlar, caso esteja implantando são incluídas as fases de projetar e verificar. Venanzi et. al. (2017, p.

4060, tradução nossa) enfatiza que “[...] A raiz do Six Sigma é tratar os desvios nos processos, buscando alcançar o mínimo de variação possível”[6].

2.4.1. *Six Sigma*

O *Six Sigma* pode ser utilizado, ainda, como uma métrica de avaliação do processo. De acordo com Trad e Maximiano (2009, p. 650) “No aspecto estatístico, o sigma pode ser entendido como uma medida da variabilidade intrínseca de um processo – seu desvio-padrão, representado pela letra grega sigma (σ). [...]Quanto menor for o desvio padrão, melhor será o processo”.

Na metodologia *Six Sigma* duas métricas são fundamentais: o Defeitos por unidade (DPU) e o Defeitos por milhão de oportunidades (DPMO). Conforme explica Ramos (2018, p. 23)

> ● Defeitos por unidade (DPU)
> A unidade pode ser um componente, um material, uma linha de códigos, um formulário administrativo, um prazo, uma distância, dentre outros. Sendo que a DPU é o número médio de defeitos encontrados durante a produção de uma unidade, para se calcular a mesma calcula-se como sendo a relação entre o número de defeitos e a quantidade de unidades.
> A DPU geral é o número médio total de defeitos para o produto fabricado, resultado da combinação da DPU de cada etapa do processo (Equação 1).
> DPUgeral = DPU1 + DPU2 + ... + DPUn Equação 1
> ● Defeitos por milhão de oportunidades (DPMO)
> Oportunidade é qualquer ação executada ou negligenciada durante a criação de uma unidade de trabalho, são as causas do problema, cometendo um erro capaz de gerar insatisfação do cliente (Equação 2). A DPMO é a relação da DPUgeral com o número de oportunidades.
> DPMO = (DPUgeral)/Número de oportunidades por unidade) × 1.000.000 Equação 2

Logo, o DPU indica o número médio de defeitos encontrados na fabricação de cada item do processo, enquanto o DPMO mostra o percentual que o DPU afeta o processo como um todo. O DPMO está relacionado com a escala sigma, como é mostrado na tabela 1

---

[6] *“The root of Six Sigma is to treat the deviations in the processes, aiming to achieve the minor possible variance”*.

Tabela 1 – Significado da Escala Sigma

Taxa de Acerto	Taxa de Erro	Defeitos por Milhão de Oportunidades (DPMO)	Escala Sigma
30,9%	69,1%	691.462	1,0
69,1%	30,9%	308.538	2,0
93,3%	6,7%	66.807	3,0
99,38%	0,62%	6.210	4,0
99,977%	0,023%	233	5,0
99,99966%	0,00034%	3,4	6,0

Fonte: Trad e Maximiano (2009, p. 650)

Como é possível observar na tabela 1 quanto mais alto o nível na escala sigma menor a taxa de erro encontrada e menos defeitos por milhão de oportunidades aparecem. Portanto, quanto mais alto o nível sigma melhor é o processo.

2.5. Otimização do processo de desenvolvimento de *Software*

A utilização de metodologias de garantia de qualidade para a otimização do processo de *software* é encontrada em alguns estudos. Como no trabalho de Oliveira (2014), onde foi avaliada a qualidade no desenvolvimento de *software* com metodologias ágeis, bem como no trabalho de Mezzomo (2015), onde foi desenvolvido um *framework* para avaliação e melhoria do processo de *software*, e também no trabalho de Miranda (2016), onde foram analisadas as atividades de desenvolvimento de *software* em um centro de dados de uma universidade pública.

No trabalho de Oliveira (2014) foram realizadas entrevistas e aplicados *surveys* em quatro empresas de desenvolvimento de *softwares*, foi utilizada a analise SWOT (*Strengths, Weaknesses, Opportunities and Threats*), ou análise FOFA (Forças, Fraquezas, Oportunidades e Ameaças), para identificar a relação entre as práticas de garantia de qualidade e a qualidade do produto final. A partir dos dados levantados Oliveira (2014) desenvolveu uma ferramenta web para auxiliar na avaliação da qualidade dos projetos que utilizam metodologias ágeis.

Enquanto que no trabalho de Mezzomo (2015) foi realizado um levantamento bibliográfico sobre avaliação e melhoria do processo de desenvolvimento de *software*, dando enfoque nos modelos MR-MPS-SW, CMMI-DEV e a norma ISO/IEC 12207. A partir do levantamento bibliográfico Mezzomo (2015) elaborou e

desenvolveu um framework para avaliação e melhoria do processo de *software*, que foi avaliado e analisado de forma qualitativa por um grupo de avaliadores.

Por outro lado o trabalho de Miranda (2016) inicia-se por um estudo bibliográfico acerca do processo de desenvolvimento de *software*, seguido posteriormente por um estudo de caso de um centro de dados de uma universidade pública. O estudo de caso realizado por Miranda (2016) aborda o detalhamento da organização, a modelagem, analise e desenho do processo de desenvolvimento de *software* realizado pela mesma, e também faz a comparação com os modelos de processos contidos na literatura.

3. ANÁLISE

O trabalho iniciou-se de um estudo bibliográfico, através de uma pesquisa exploratória a respeito de normas, métodos e parâmetros de avaliação de processos de desenvolvimento de *software*. Além de estudar as metodologias ágeis e seus princípios, bem como, indicadores e métricas de avaliação adequados para esse método de desenvolvimento.

Foi realizado um estudo de caso na FTT com base no método *Six Sigma*, onde foram definidas ações para avaliação do processo. Além da pesquisa bibliográfica, o estudo explorou, inicialmente, a estrutura da FTT. Onde foram desenhados e especificados o processo de desenvolvimento de *software* que já era executado no período do estudo, e buscou-se entender os procedimentos e atividades da equipe de desenvolvimento.

O estudo de caso contou com uma avaliação da FTT pelos membros que integram a equipe de desenvolvimento de *software*. O questionário foi elaborado com base no estudo bibliográfico e implementado na ferramenta *google* f*orms*. A avaliação possuía estrutura fechada com questões de resposta obrigatória. As respostas auxiliaram a identificar as características e a execução dos processos da FTT.

Além disso, a avaliação e a observação das atividades da FTT ajudaram a identificar os pontos fortes e também os pontos de falha ou que necessitavam de alterações para aumentar a produtividade. Logo, esses pontos foram discutidos juntamente com a liderança da FTT, onde foram sugeridas e implementadas alterações no processo.

Durante o estudo foram monitoradas as atividades da FTT gerando métricas, dados quantitativos, do processo. Esses dados foram sistematizados e agrupados, gerando indicadores de qualidade, através do cálculo utilizado na metodologia *Six Sigma*. Os indicadores auxiliaram no entendimento dos pontos de falha e erros na execução do processo de desenvolvimento de *software* da FTT.

A análise do processo de desenvolvimento de *software* da FTT foi realizada levando em consideração o método *Six Sigma*, representado no quadro 3, onde foram definidas ações a serem efetivadas durante todo o estudo de caso.

Quadro 3 – DMADV Processo de desenvolvimento de Software FTT

Fase	Ação
Definir	Desenhar e especificar processo de desenvolvimento de *Software*.
Medir	Utilizar métricas para mensurar quantitativamente o processo da FTT.
Analisar	Agrupar dados e avaliar necessidades de correções.
Projetar	Desenhar e definir alterações no processo.
Verificar	Avaliar qualidade do processo.

Fonte: Autores (2019).

Essas ações tiveram como intuito garantir a qualidade do processo implementado na FTT, e caso fossem encontrados erros e falhas elaborar soluções e aplica-las ao processo. Além disso, a análise buscou acompanhar e gerar dados sobre as atividades de desenvolvimento de *software* da FTT.

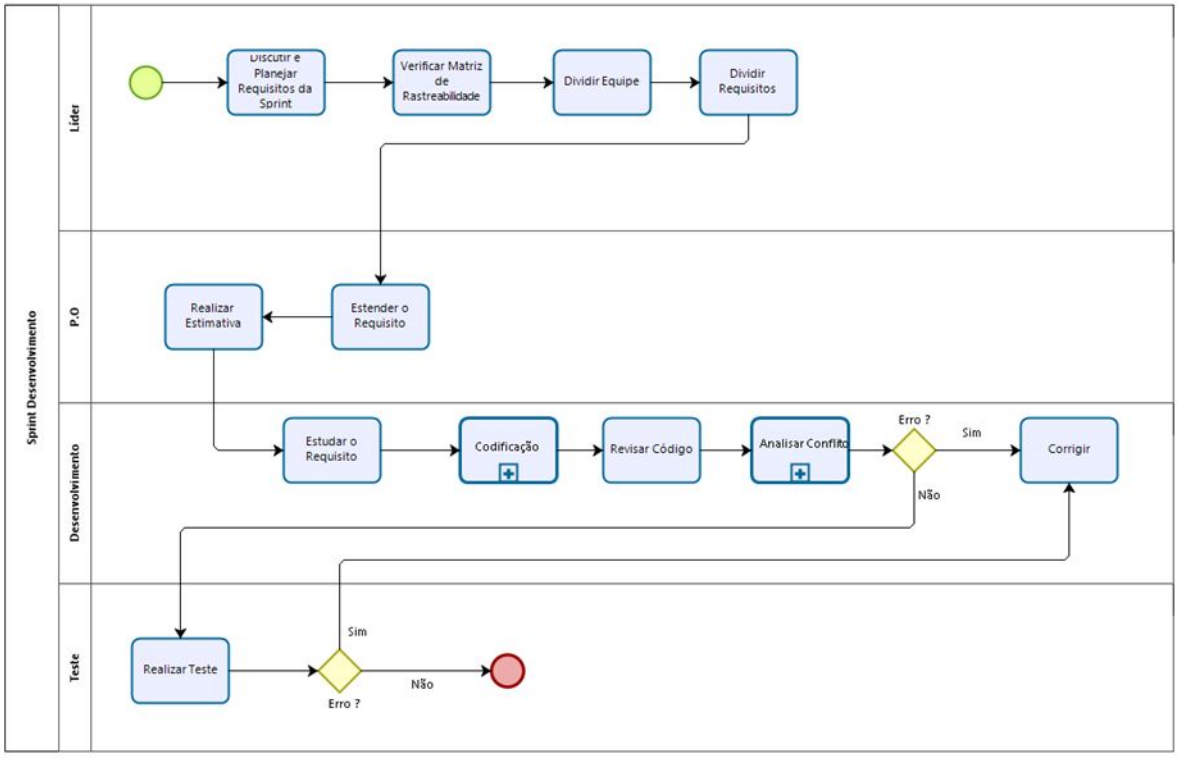
3.1. Processo de desenvolvimento de *software* FTT

O processo de desenvolvimento de *software* da FTT, que era realizado no início do estudo, pode ser observado através da Figura 3, desenhada pelos autores por meio da ferramenta Bizagi, que utiliza o BPMN (*Business Process Model and Notation* - Notação de Modelagem de Processos de Negócio) para descrever processos através de uma série de ícones.

Como é notado na Figura 3, a *sprint* de desenvolvimento iniciava-se com o líder do processo realizando o planejamento das atividades, a divisão dos requisitos, dentre outras ações. Após essas atividades, o P.O. (*Product Owner*) entendia os requisitos e realizava as estimativas, passando os requisitos para equipe de desenvolvimento que interpretava os requisitos e codificava. Dentro da equipe de desenvolvimento era realizada, ainda, a revisão do código, a análise dos conflitos e correções no código.

Após a análise dos conflitos (caso não fossem encontrado erros) o *software* era entregue a equipe de teste que realizavam testes no mesmo. Caso fosse encontrado algum erro o *software* voltava para equipe de desenvolvimento para correção, caso contrário as atividades eram finalizadas.

Figura 3 – Processo de desenvolvimento de software FTT.



Fonte: Autores (2019).

3.2. Avaliação do Processo de Desenvolvimento de *Software*

Com base nas observações dos processos realizados pela equipe de desenvolvimento de *software* da FTT, e nos resultados esperados dos atributos do processo no nível G do modelo MPS-BR, foi elaborado um questionário (disponível no apêndice A), que foi implementado na ferramenta *google forms* e enviado no início do segundo semestre de 2020 para os integrantes da equipe de desenvolvimento de *software* da FTT responderem.

O questionário foi estruturado com perguntas fechadas, com exceção da última questão que era aberta. Todas as questões eram de resposta obrigatória, ou seja não era possível terminar a avaliação enquanto todas as perguntas não fossem respondidas. Todas as questões fechadas ofereciam alternativas para qualificar o processo de desenvolvimento de *software* da FTT.

A pesquisa foi respondida por seis integrantes da equipe de desenvolvimento da FTT. As seis primeiras perguntas do questionário visaram identificar os pesquisados, os resultados estão ilustrados nos gráficos de 1 a 6. A maioria dos pesquisados cursam o quarto período do curso de Engenharia de *Software*, todos os pesquisados são membros da equipe de desenvolvimento de *Software* da FTT, a maioria integra a FTT a mais de 1 ano.

Metade dos pesquisados possuem conhecimento básico em metodologias ágeis e a outra metade conhecimento intermediário. Todos os pesquisados conhecem o processo e as fases de desenvolvimento de *Software* da FTT. As respostas as perguntas iniciais do questionário indicaram que os pesquisados possuem um conhecimento relevante sobre a FTT e o processo de desenvolvimento de *Software*.

Entre a sétima e a decima terceira pergunta do questionário foram abordadas questões referentes a rotina de desenvolvimento de *Software* da FTT, os resultados estão ilustrados nos gráficos de 7 a 13. Os pesquisados, em sua maioria, utilizam o processo já estabelecido pela FTT para o desenvolvimento dos requisitos. A maioria dos pesquisados afirmaram que são utilizadas estratégias para melhoria da qualidade do código.

Metade dos pesquisados atualizam o código antes de fazer a integração dos requisitos. A maioria dos pesquisados afirmaram que raramente há conflitos no momento de integrar os requisitos desenvolvidos. Por outro lado, a maioria desconhece, ou conhece parcialmente o procedimento para não gerar conflitos durante a integração dos requisitos ao sistema. Os pesquisados consideraram a comunicação tanto das equipes da FTT quanto entre a equipe de desenvolvimento como boa ou ótima.

Vale destacar que mesmo a maior parte dos pesquisados afirmarem seguir os procedimentos da FTT, há a necessidade de enfatizar os processos com os membros da equipe de desenvolvimento. Visto que, em busca da melhoria continua é preciso corrigir os desvios visando a qualidade. Além disso, reforçar as estratégias para melhoria da qualidade do código, para que todos os membros tenham conhecimento. Como também, treinar os membros para integração dos requisitos ao sistema, ainda que alguns membros tenham conhecimento do procedimento, a maioria ainda tem conhecimento parcial, ou nenhum conhecimento. Corroborando

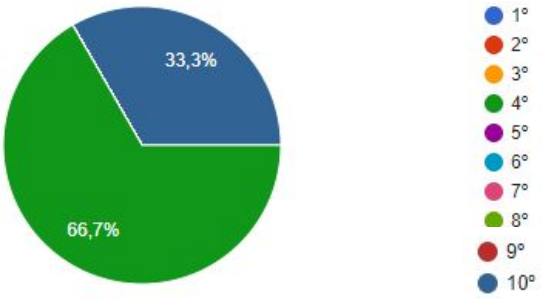
essa necessidade, apenas a metade dos pesquisados sempre atualizam os requisitos antes de integrar.

As últimas perguntas do questionário, da decima quarta a vigésima pergunta, abordaram de forma geral sobre o processo de desenvolvimento, a estrutura da FTT e a opinião dos pesquisados, os resultados estão ilustrados nos gráficos de 14 a 21 e na figura 4. A maioria dos pesquisados consideram o processo como parcialmente suficiente. A rotatividade dos membros da FTT foi considerada como média ou alta. Todos os pesquisados afirmaram existir padronização nos códigos desenvolvidos pela equipe, e que os lideres verificam essa padronização.

A maioria dos pesquisados afirmou que há qualificação dos novos membros da equipe, porém a maioria afirmou que a qualificação foi parcialmente suficiente, ou não foi suficiente, para realizar as atividades iniciais na FTT. Todos os pesquisados afirmaram que os processos são especificados conforme um procedimento padrão, e a maioria afirmou que o processo é executado conforme o estabelecido. Todos os pesquisados afirmaram, ainda, que existe auditoria dos processos da equipe de desenvolvimento da FTT, para verificar se o processo estabelecido está sendo seguido. Nenhum pesquisado sugeriu melhorias para o processo de desenvolvimento de *software*, apenas um dos pesquisados sugeriu mudanças no teste de *software*.

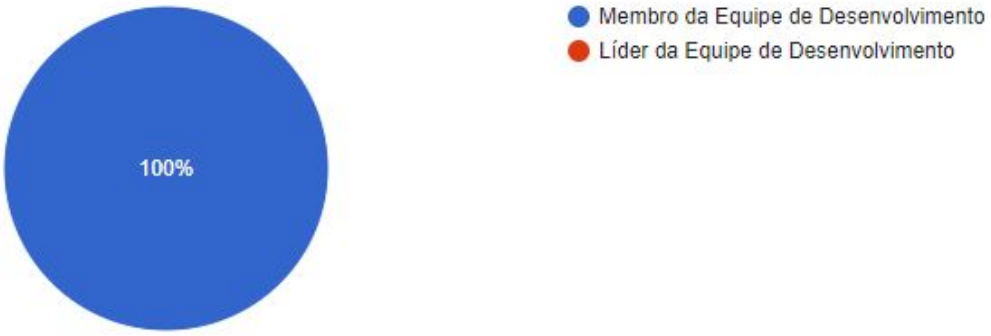
Ressalta-se que os procedimentos de SQA são necessários para assegurar um processo de desenvolvimento mais assertivo, logo estabelecer e realizar continuamente processos de garantia da qualidade, e compartilhar os resultados com a equipe, é fundamental. Portanto, buscar o aprimoramento contínuo do processo, com a participação da equipe, pode aumentar a percepção de eficácia do trabalho da FTT pelos membros. Além disso, reforçar o treinamento e o acompanhamento dos novos integrantes, principalmente nas primeiras atividades a serem desenvolvidas. Essas atividades de aprimoramento do processo auxiliam na identificação de possíveis desvios entre o que foi estabelecido e o que está sendo executado, e assim garantir um resultado adequado.

Gráfico 1 – Período de Engenharia de Computação/*Software* cursado pelos integrantes da FTT.



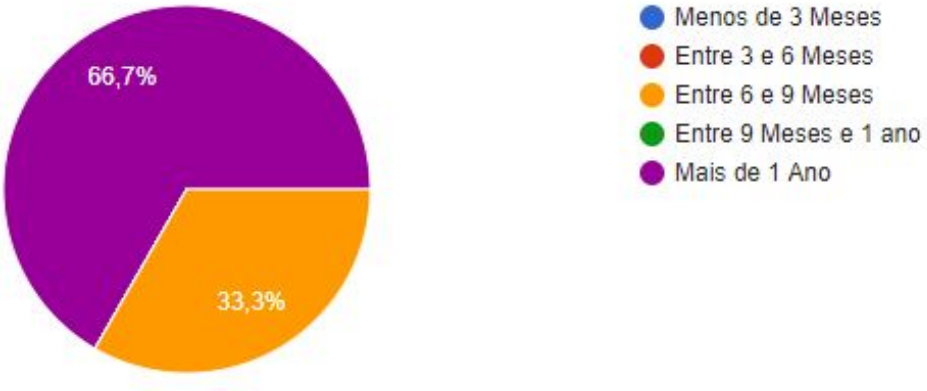
Fonte: Autores (2020).

Gráfico 2 – Função exercida pelos pesquisados na FTT.



Fonte: Autores (2020).

Gráfico 3 – Tempo que os pesquisados integram a FTT.



Fonte: Autores (2020).

Gráfico 4 – Conhecimento sobre métodos ágeis dos pesquisados.

Fonte: Autores (2020).

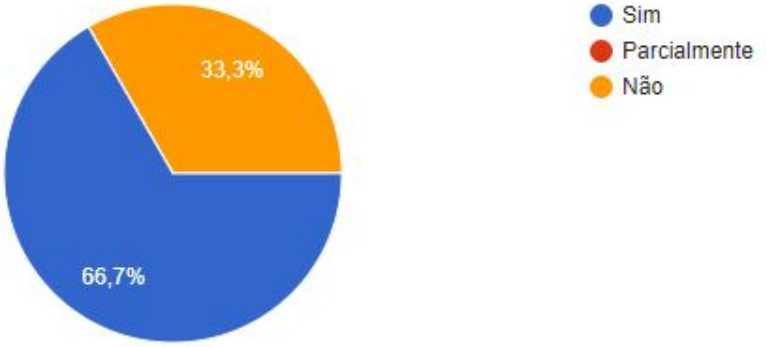
Gráfico 5 – Conhecimento dos pesquisados sobre o processo de desenvolvimento de *software* da FTT.

Fonte: Autores (2020).

Gráfico 6 – Conhecimento dos pesquisados sobre as fases do processo de desenvolvimento de *software* da FTT.

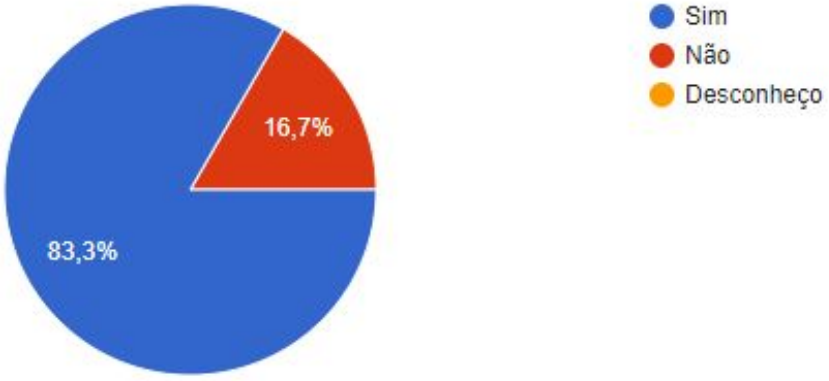
Fonte: Autores (2020).

Gráfico 7 – Utilização do processo da FTT no desenvolvimento dos requisitos.



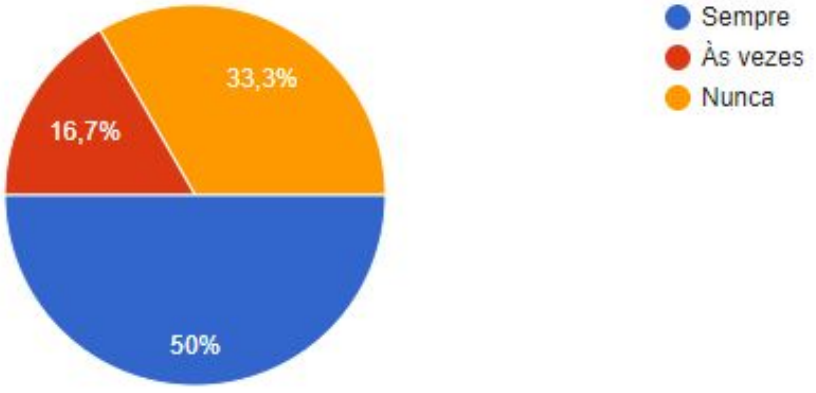
Fonte: Autores (2020).

Gráfico 8 – Utilização de estratégias para melhoria da qualidade do código no processo de desenvolvimento de software da FTT.



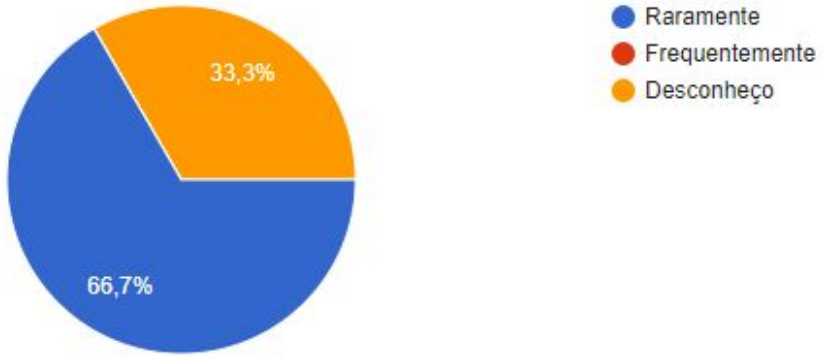
Fonte: Autores (2020).

Gráfico 9 – Atualização dos requisitos desenvolvidos antes de integrar ao sistema.



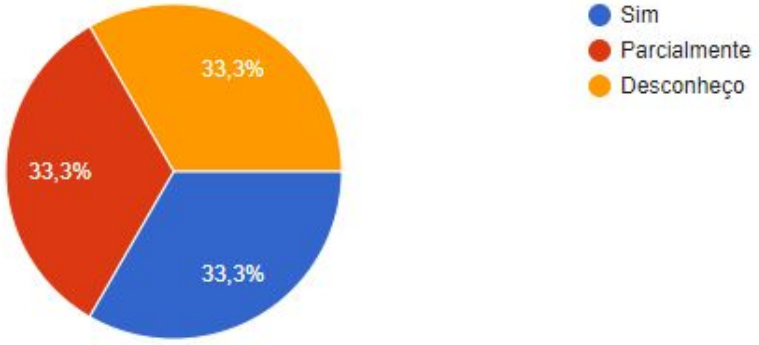
Fonte: Autores (2020).

Gráfico 10 – Conflitos no momento de integrar os requisitos ao sistema.



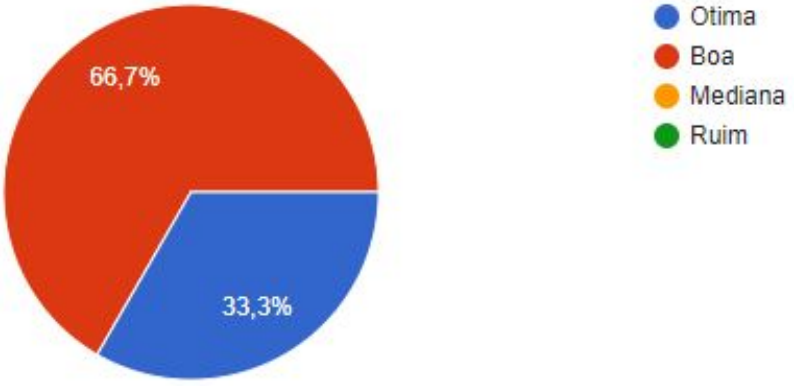
Fonte: Autores (2020).

Gráfico 11 – Conhecimento dos pesquisados sobre os procedimentos para não gerar conflitos durante a integração dos requisitos com o sistema.



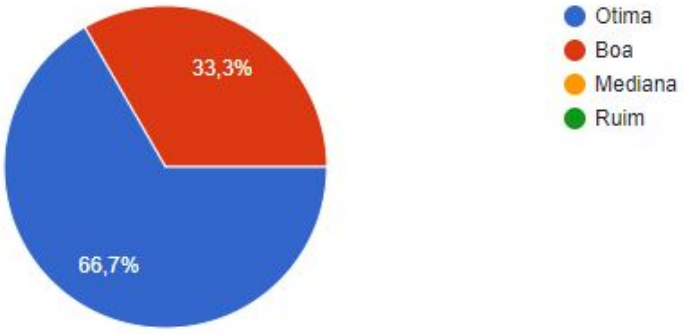
Fonte: Autores (2020).

Gráfico 12 – Avaliação dos pesquisados sobre a comunicação das equipes da FTT.



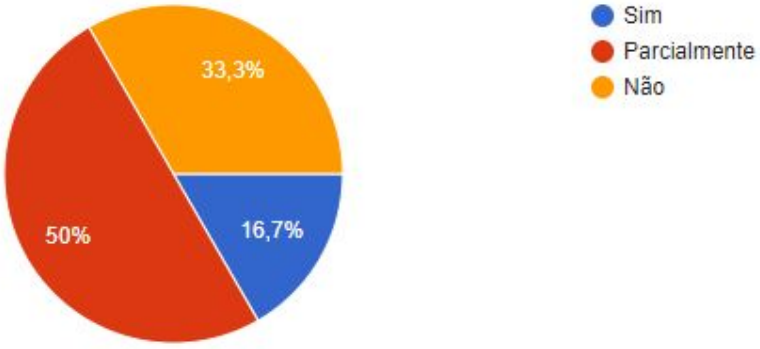
Fonte: Autores (2020).

Gráfico 13 – Avaliação dos pesquisados sobre a comunicação da equipe de desenvolvimento da FTT.



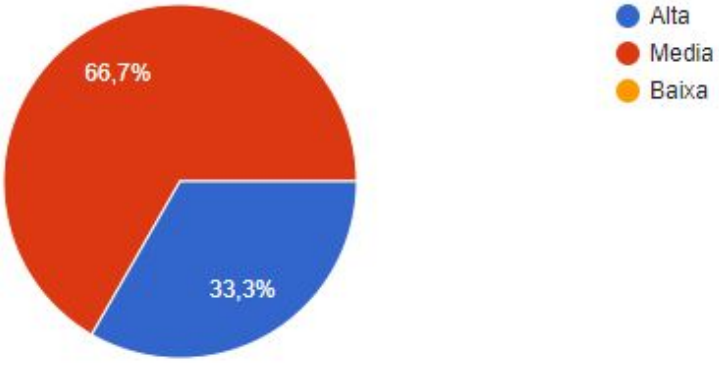
Fonte: Autores (2020).

Gráfico 14 – Avaliação dos pesquisados sobre a eficiência do processo de desenvolvimento da FTT.



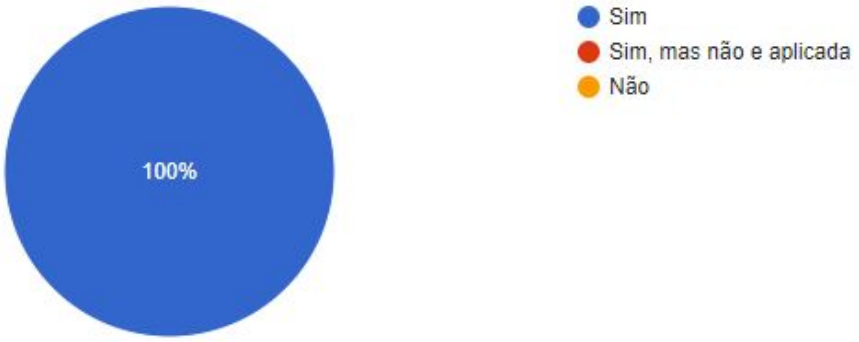
Fonte: Autores (2020).

Gráfico 15 – Avaliação dos pesquisados sobre a rotatividade de membros da equipe de desenvolvimento da FTT.



Fonte: Autores (2020).

Gráfico 16 – Existência de padronização dos códigos desenvolvidos pela equipe de desenvolvimento.

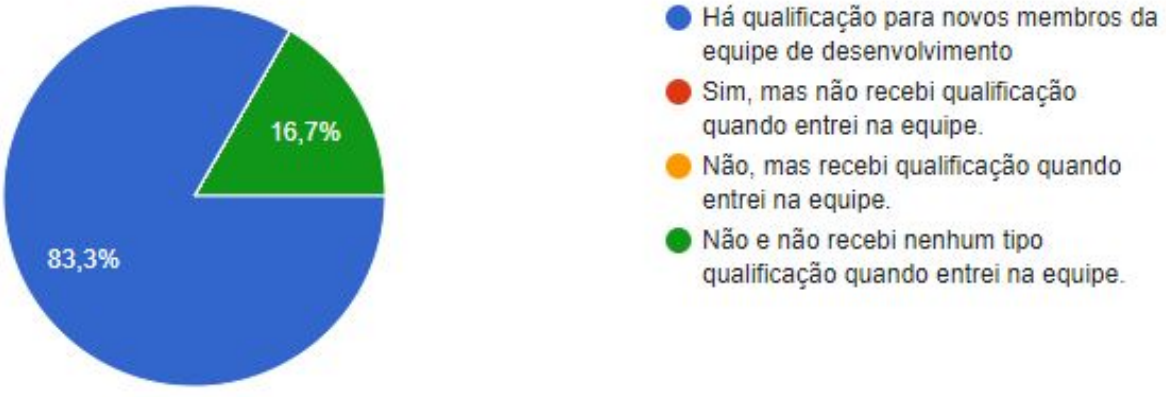


Fonte: Autores (2020).

Gráfico 17 – Acompanhamento dos líderes da FTT da padronização dos códigos desenvolvidos pela equipe de desenvolvimento.

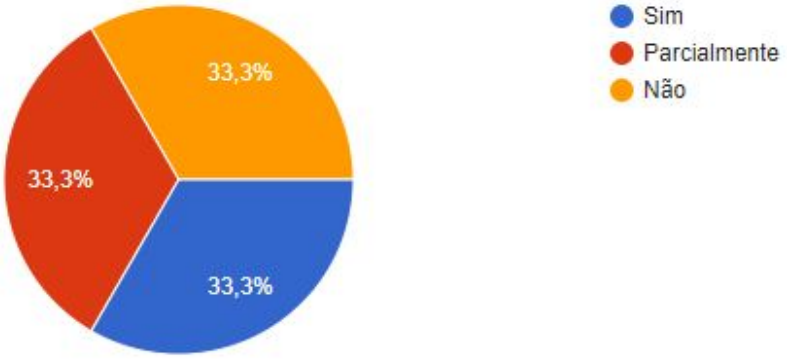
Fonte: Autores (2020).

Gráfico 18 – Qualificação dos novos membros da FTT.



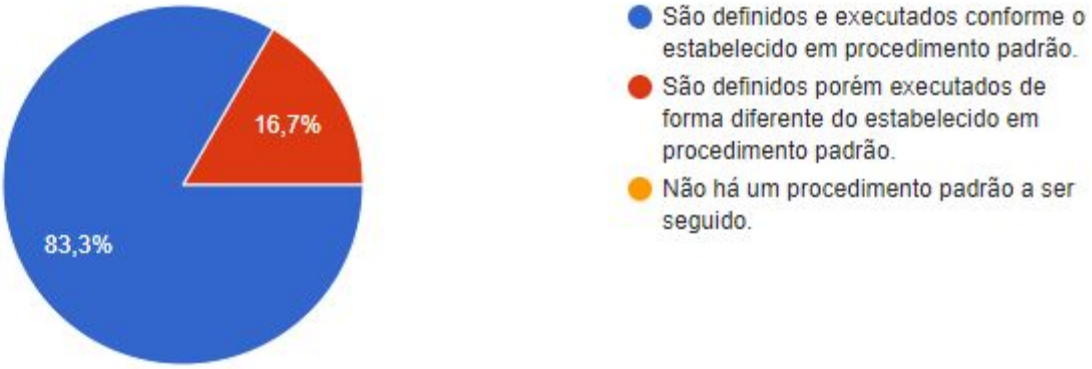
Fonte: Autores (2020).

Gráfico 19 – Avaliação dos pesquisados sobre a eficiência da qualificação dos novos membros da FTT para realizar as primeiras atividades.



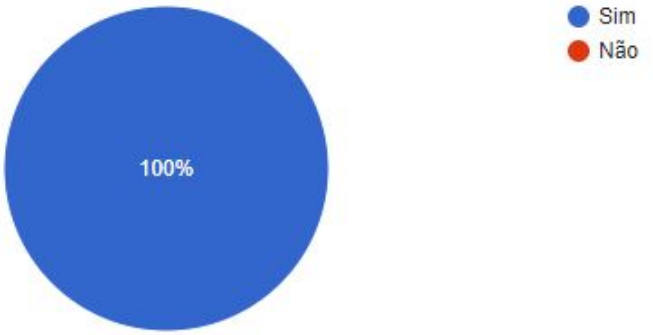
Fonte: Autores (2020).

Gráfico 20 – Existência de procedimento padrão na FTT e execução do processo de desenvolvimento de *software*.



Fonte: Autores (2020).

Gráfico 21 – Existência de auditoria no processo de desenvolvimento de *software* da FTT.



Fonte: Autores (2020).

Figura 4 – Sugestões para melhoria do processo de desenvolvimento de *software* da FTT.

Fonte: Autores (2020).

A partir dos dados da pesquisa realizada foi elaborado o quadro 4, que contém a análise *SWOT* (ou análise FOFA). A análise FOFA consiste em uma ferramenta de diagnóstico que auxilia na identificação de relações entre as práticas de SQA e a qualidade do processo, (OLIVEIRA, 2014). Como é exposto no quadro 4, a FTT possui algumas forças internas, a existência um processo estabelecido, bem como uma boa comunicação entre as equipes, e abertura dos membros a aprendizagem.

Além disso, apresenta a possibilidade de inserção de novos métodos e linguagens, por se tratar de uma fábrica acadêmica há uma abertura maior para inovação, bem como a alta demanda por tecnologia e um mercado de *softwares* altamente dinâmico colabora para a expansão e melhoria da FTT. Logo, existe uma amplitude de oportunidades que a FTT pode explorar e fomentar gerando um aprendizado ainda maior para os membros.

Por outro lado, a FTT apresentou os seguintes pontos de fraqueza (ou melhora), o treinamento mostrou-se insuficiente para os integrantes executarem as atividades, alguns membros também necessitam de acompanhamento e outros atualizarem-se das atividades realizadas na FTT, como também necessita uma maior participação do membros nos processos de SQA. Além disso, a FTT enfrenta ameaças externas como melhores oportunidades no mercado de trabalho, que causa a evasão dos integrantes, a elevada necessidade de gestão e controle que o processo de desenvolvimento de *software* exige, e limitação dos recursos por depender da administração da instituição a qual a FTT está vinculada.

Quadro 4 – Análise *SWOT* da FTT.

	Fatores Positivos	**Fatores Negativos**
Fatores Internos	Forças: ● Existência de procedimento padrão para desenvolvimento. ● Boa comunicação entre as equipes. ● Abertura dos membros a aprendizagem.	Fraquezas: ● Treinamento insuficiente. ● Membros com necessidade de acompanhamento do trabalho. ● Necessidade de participação dos membros nos processos de SQA.
Fatores Externos	Oportunidades: ● Possibilidade de inserção de novos métodos e linguagens fomentando a inovação tecnológica. ● Alta demanda por tecnologia. ● Mercado de *softwares* altamente dinâmico.	Ameaças: ● Integrantes encontram oportunidades melhores no mercado de trabalho. ● Necessidade elevada de controle e gestão de processos. ● Limitação de recursos.

Fonte: Autores (2020).

Durante o estudo e a avaliação da FTT foram observadas algumas atividades e procedimentos que poderiam ser alterados visando a agilidade e produtividade da equipe. Esses pontos foram discutidos e sugeridos à liderança da FTT, e apresentados no tópico seguinte.

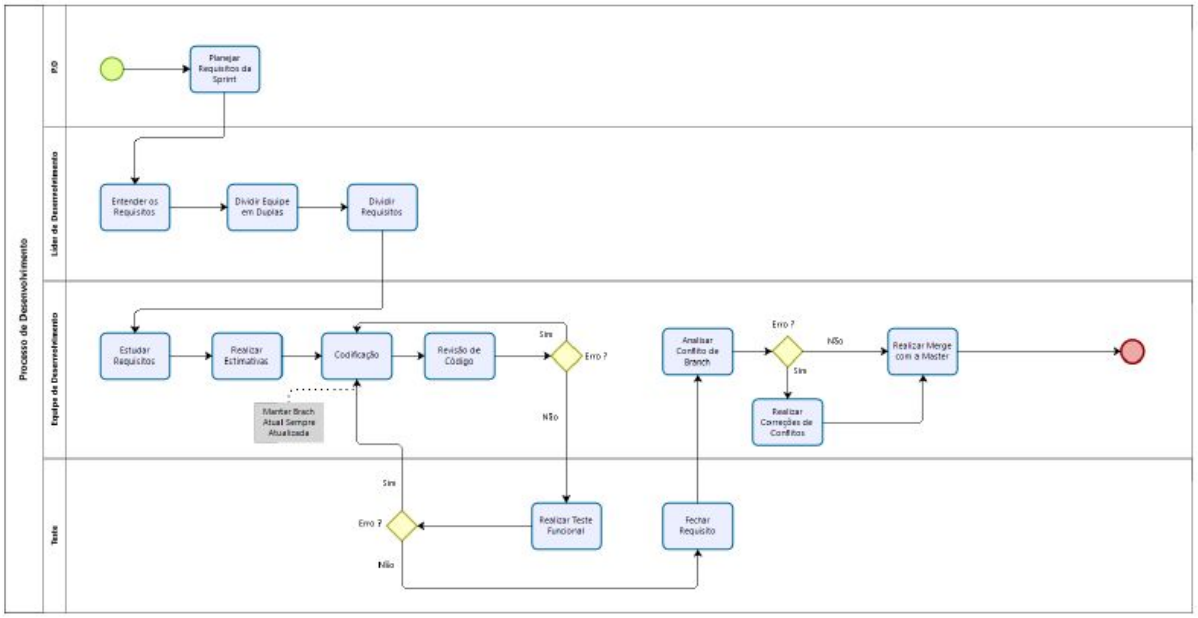
2.3. Alterações no processo da FTT

Durante o estudo da FTT ocorreram alterações no processo visando a qualidade e a agilidade do desenvolvimento de *software*. Os novos procedimentos foram elaborados e desenhados pelos autores juntamente com os líderes da FTT. O novo processo foi validado e implementado no início do segundo semestre de 2020, o processo pode ser observado na figura 5.

Ao comparar a Figura 3 com a Figura 5 verifica-se que houveram alterações nas atividades de desenvolvimento e de teste. Houve a inclusão de atividades para equipe de desenvolvimento e o processo não finaliza-se na equipe de teste (como ocorria no início do estudo). Após o P.O. planejar os requisitos da *sprint*, o líder da equipe de desenvolvimento entende os requisitos, divide a equipe em duplas e divide os requisitos.

A equipe de desenvolvimento estuda os requisitos, codifica e revisa os códigos, que caso não possua erros são encaminhados para a equipe de teste que realiza o teste funcional. Caso não sejam encontrados erros nos códigos, durante o teste, são fechados os requisitos e os códigos encaminhados de volta para equipe de desenvolvimento que analisa o conflito de *branch*, caso encontrado erros são realizadas correções e após esse procedimento realizada a *merge* com a *master* e encerrado o processo.

Figura 5 – Processo de desenvolvimento de Software da FTT 2020/2.



Fonte: Autores (2020).

O novo processo retrata as atividades de integração dos requisitos ao sistema, e evidência o fluxo de atividades até a finalização do código, mostrando as atividades que são realizadas em caso de erro. Além disso, o processo enfatiza a atividade de atualização dos requisitos visando a diminuição de erros referentes a códigos desenvolvidos diferente do especificado, diminuindo a necessidade de alterações e exclusões. As atividades da equipe de desenvolvimento da FTT foram monitoradas e mensuradas de forma quantitativa durante o ano de 2020, e os resultados geraram métricas que serviram de base para calcular o nível de qualidade do processo, essas informações são discutidas no tópico seguinte.

2.4. Métricas da FTT

As atividades da FTT foram monitoradas durante o primeiro semestre de 2020, no período de 20/03/2020 a 30/06/2020, e durante o segundo semestre de 2020. A finalidade do acompanhamento das atividades foi obter dados quantitativos do processo buscando gerar números que qualificassem o processo de desenvolvimento da FTT. Durante o período observado foram realizados *commits*[7] em 32 dias no primeiro semestre de 2020. Enquanto que no segundo semestre foram realizados *commits* em apenas 2 dias, 13/08/2020 e 15/09/2020.

Os dados encontrados foram agrupados, os referentes ao primeiro semestre foram igualmente divididos em dois períodos de análise 20/03/2020-14/04/2020 e 01/06/2020-30/06/2020, nos quais houveram 16 dias de *commits* em cada período. Os resultados foram distribuídos nas tabelas 2 e 3. Enquanto que os dados referentes ao segundo semestre foram agrupados em uma única analise, e o resultado descrito na tabela 4.

Durante os *commits* foram observadas três situações: linhas de código alteradas, linhas de código adicionadas e linhas de código excluídas. No primeiro período acompanhado, tabela 2, obteve um total de 15033 linhas de código.

[7] Alterações no projeto de *software* em que a equipe de desenvolvimento da FTT desenvolvia durante o estudo. Cada *commit* contém modificações no *software* que pode ser: linha de código alterada, linha de código adicionadas e linha de código excluída.

Enquanto que no segundo período, tabela 3, obteve um total de 5415 linhas de código. Já no período referente ao segundo semestre de 2020, tabela 4, obteve um total de 70 linhas de código.

Ainda que para fins analíticos a diferença no total de linhas de código dos períodos gere distorções de análise, é possível observar as diferenças proporcionais de alterações, adições e exclusões de linhas de código. Fazendo a comparação do aumento ou da diminuição que cada item representa do total. Analisando as linhas alteradas, do primeiro para o segundo período houve uma redução quase inexpressiva de 0,03%, enquanto que comparando o segundo com o terceiro período houve um aumento de 10,32%.

Examinando as linhas de código adicionadas do primeiro para o segundo período houve uma redução de 13,81%, e do segundo para o terceiro período uma redução de 11,75%. Referente as linhas de código excluídas do primeiro período para o segundo período houve um aumento de 13,84% e do segundo para o terceiro período houve um aumento de 1,43%.

Vale ressaltar que a diferença no total do número de linhas de código dos períodos pode gerar uma análise ambígua, uma vez que não há uma proximidade nos números produzidos. Ainda assim, os números encontrados podem ser analisados e comparados, conforme como pode ser observado no gráfico 22.

Tabela 2 – Métricas do processo de desenvolvimento de *Software* da FTT 20/03/2020-14/04/2020.

Alterações encontrados no processo	Quantidade	Porcentagem do total
Linhas de código alteradas	648	4,00%
Linhas de código adicionadas	12440	82,70%
Linhas de código excluídas	2005	13,30%
Total de linhas de código	**15.093**	**100%**

Fonte: Autores (2020).

Tabela 3 – Métricas do processo de desenvolvimento de *Software* da FTT 01/06/2020-30/06/2020.

Alterações encontrados no processo	Quantidade	Porcentagem do total

Linhas de código alteradas	215	3,97%
Linhas de código adicionadas	3730	68,89%
Linhas de código excluídas	1470	27,14%
Total de linhas de código	**5415**	**100%**

Fonte: Autores (2020).

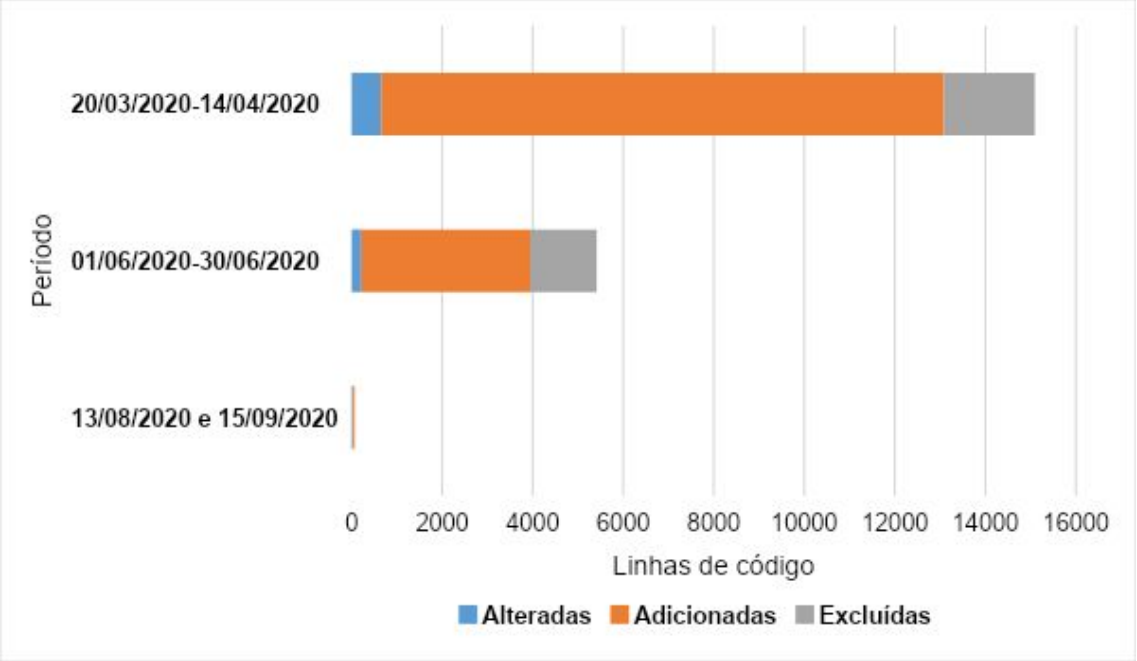
Tabela 4 – Métricas do processo de desenvolvimento de *Software* da FTT 13/08/2020 e 15/09/2020

Alterações encontrados no processo	Quantidade	Porcentagem do total
Linhas de código alteradas	10	14,29%
Linhas de código adicionadas	40	57,14%
Linhas de código excluídas	20	28,57%
Total de linhas de código	**70**	**100%**

Fonte: Autores (2020).

No gráfico 22 é possível visualizar a diminuição drástica do número de linhas de códigos, que pode ser justificada pelos períodos analisados. O período referente a 01/06/2020-30/06/2020 coincide com o final do primeiro semestre, e o período referente ao segundo semestre não houve um número considerável de *commits* comparado com os demais períodos analisados.

Gráfico 22 – Comparação entre as métricas do processo de desenvolvimento de *Software* da FTT em 2020.



Fonte: Autores (2020).

A partir dos dados quantitativos sistematizados nas tabelas 2, 3 e 4 foram elaboradas as equações 1, 2 e 3 apresentando o DPMO dos períodos analisados. O cálculo foi realizado somando o número de linhas alteradas e excluídas (considerando estas como o DPU) e dividindo pelo número total de linhas de códigos do período e multiplicando por 1.000.000.

Equação 1 – DPMO processo de desenvolvimento de Software da FTT 20/03/2020-14/04/2020.

$$DPMO = (DPUgeral)/Número\ de\ oportunidades\ por\ unidade) \times 1.000.000$$

$$DPMO = (2653/15033)\ x\ 1.000.000$$

DPMO=176.478

Equação 2 – DPMO processo de desenvolvimento de Software da FTT 01/06/2020-30/06/2020

$$DPMO = (DPUgeral)/Número\ de\ oportunidades\ por\ unidade) \times 1.000.000$$

$$DPMO = (1685/5415)\ x\ 1.000.000$$

DPMO=311.173

Equação 3 – DPMO processo de desenvolvimento de Software da FTT 13/08/2020 e 15/09/2020.

$$DPMO = (DPUgeral)/Número\ de\ oportunidades\ por\ unidade) \times 1.000.000$$

$$DPMO = (30/70)\ x\ 1.000.000$$

DPMO=428.571

Referente ao primeiro período foi constatado um DPMO de 176.478 o que corresponde a uma taxa de erro 17,64%, em comparação a escala sigma (tabela 1) o processo de desenvolvimento da FTT estaria no nível 2. Enquanto que no segundo período observado o DPMO foi de 311.173, totalizando uma taxa de erro de 31,12%, comparando com a escala sigma (tabela 1) o processo da FTT estaria no nível 1. A respeito do terceiro período analisado, O DPMO foi de 428.571, totalizando uma taxa de erro de 42,85%, comparando com a escala sigma (tabela 1) o processo da FTT estaria no nível 1.

O cálculo do DPMO não difere por volume, ou seja, independente da diferença no total de linhas de código produzidas em cada período o cálculo leva em consideração a proporção de erros por milhões de oportunidades. Logo, houve uma redução da qualidade da produção da FTT. Por outro lado, para uma avaliação mais profunda da FTT, seria necessário uma série de dados que pudessem estabelecer (quantitativamente) um padrão de qualidade do processo de desenvolvimento de *software* da FTT.

Portanto, há a necessidade de reforçar os procedimentos de garantia da qualidade, além da revisão periódica dos processos da FTT, buscando, assim, aumentar o nível de qualidade no desenvolvimento de *software*. Além disso, gerar dados que auxiliem nas atividades e no planejamento

3. Considerações Finais

Através do estudo e avaliação do processo de desenvolvimento de *software* buscou-se estimular o uso de ferramentas, métodos e normas de qualidade para garantia da qualidade desse processo. Como também, estimular a melhoria continua e a otimização dos processos com o objetivo de assegurar que o *software* desenvolvido tenha qualidade.

Considerando que as atividades de garantia da qualidade devem adaptar-se ao modelo de processo de desenvolvimento executado, e ainda assim devem atender os objetivos a serem alcançados, essas são atividades essenciais para gerar dados sobre os processos realizados pela empresa. Logo os procedimentos de garantia da qualidade contribuem para correções e otimizações no processo.

Analisando o estudo de caso realizado, apesar das fraquezas e ameaças presentes na FTT, a mesma possui pontos de força e oportunidades a serem exploradas. Nesse sentido, reforçar os procedimentos de avaliação e de SQA auxiliariam a FTT a melhorar o processo de desenvolvimento e a utilizar esse potencial.

Nesse sentido, a FTT, de acordo com os dados levantados, realiza um processo com qualidade entre o nível 1 e 2 da escala sigma. Ainda que erros sejam encontrados e existam pontos a serem otimizados no processo, o processo consegue ser parcialmente eficaz, conforme os próprios membros da FTT opinaram. Portanto, as atividades de garantia da qualidade podem ajudar na assertividade do processo de desenvolvimento de *software* da FTT.

Deste modo, as ferramentas de avaliação, indicadores e métricas do processo, levantados neste trabalham, contribuiriam com os integrantes da FTT a entenderem as atividades realizadas, e como o processo está sendo executado e os resultados que estão sendo gerados, e assim, certificando se o processo foi executado de forma correta.

Portanto, a partir deste trabalho espera-se que os procedimentos de SQA sejam utilizados de forma frequente na FTT. Além de estimular o uso das atividades de garantia da qualidade nos processos de desenvolvimento de *software*. Assim como, este trabalho procurou contribuir para o estudo de normas, procedimentos e atividades de garantia da qualidade.

REFERÊNCIAS

BECK, Kent; BEEDLE, Mike; BENNEKUM, Arie van; COCKBURN, Alistair; CUNNINGHAM, Ward; FOWLER, Martin; GRENNING, James; HIGHSMITH, Jim; HUNT, Andrew; JEFFRIES, Ron; KERN, Jon; MARICK, Brian; MARTIN, Robert C.; MELLOR, Steve; SCHWABER, Ken; SUTHERLAND, Jeff; THOMAS, Dave. **Manifesto para o desenvolvimento ágil de software**. 2001. Disponível em: < https://www.manifestoagil.com.br/index.html>.

CAMPOS, Ester Lima de. ***Burndownchart* – Mede o progresso da *sprint* e dá indicativos do processo de trabalho da equipe**. 2012. Disponível em: <http://blog.myscrumhalf.com/2012/01/burndown-chart-medindo-o-progresso-de-sua-sprint-e-trazendo-indicativos-do-processo-de-trabalho-da-equipe/>.

COHN, Mike. **Desenvolvimento de *Software* com *Scrum*: aplicando métodos ágeis com sucesso**. Tradução: Aldir José Coelho Corrêa da Silva; revisão técnica: Rafael Prikladnicki. Porto Alegre, Bookman, 2011.

FARIA, Arthur De Aquino; ANDRADE, Gustavo Campos De; CORRÊA, JONAS SANTOS; AMORIM, Leonardo Duarte; ROLIM, Victor Hugo Ferreira; SILVA, Welington Tierry Da. **Guia Prático De Funcionamento Da Fábrica De Tecnologias Turing (FTT)**. Anápolis: UniEVANGÉLICA, 2018.

FERNANDES, Aguinaldo Aragon; TEIXEIRA, Descartes de Souza. **Fábrica de *Software*: Implementação e gestão de operações**. 1. ed. São Paulo, Atlas, 2011.

JONNALAGADDA, Ganesh; SHAFEY, Sarah; LYNAH, Wes; MASSETTI, Marco; MAERTEN, Patrick; WIECZOREK, Simon; LANDZAAT, Sander; PROBST, Manuel; SCHUSTER, Andrew; OXBOROUGH, Chris; BAINS, Ravinder; BONSER, Matthew. ***Agile Project Delivery Confidence Mitigate project risks and deliver value to your business***. Jul. 2017. Disponível em: <https://www.pwc.com/gx/en/actuarial-insurance-services/assets/agile-project-delivery-confidence.pdf>.

LIMA, Alessandra. **Processo de Desenvolvimento de Software – PDS**. Instituto Nacional de Tecnologias da Informação; Coordenação-Geral de Planejamento, Orçamento e Administração Coordenação de Desenvolvimento, Infraestrutura e Suporte – CODIS; Processo de Desenvolvimento de Software (PDS - ITI), Nov. 2012. Disponível em: < http://www.iti.gov.br/images/institucional/politicas/PDS_ITI.pdf>.

MACHADO, Cristina Ângela Filipak; ROCHA, Ana Regina C.; SOUZA, Gleison dos Santos. **MPS.BR - Melhoria de Processo do Software Brasileiro: Guia Geral MPS de Software**. 2012. Disponível em: <https://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_Geral_Software_2012-c-ISBN-1.pdf >.

MACHADO, Cristina Ângela Filipak; ZABEU, Ana Cecilia. **MPS.BR - Melhoria de Processo do Software Brasileiro: Guia de Implementação – Parte 1: Fundamentação para Implementação do Nível G do MR-MPS-SW:2016**. 2016. Disponível em: <https://www.softex.br/wp-content/uploads/2018/11/Guia-de-Implementa%C3%A7%C3%A3o-de-Software-%E2%80%93-Parte-01.pdf>.

MACIEL, Ana Carla Fernandes; VALLS, Carmem; SAVOINE; Márcia Maria. **Análise da Qualidade de Software Utilizando as Normas 12207, 15504, ISO 9000-3 e os Modelos CMM/CMMI e MPS.BR**. Revista Científica do ITPAC, Araguaína, v.4, n.4, out. 2011. Disponível em: <https://assets.itpac.br/arquivos/Revista/44/5.pdf>.

MAZUCO, Alan Saulo da Costa. **Percepções de Práticas Ágeis em Desenvolvimento de Software: Benefícios e Desafios**. Dissertação (Mestrado - Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, Brasília, 2017. Disponível em: <https://repositorio.unb.br/bitstream/10482/25332/1/2017_AlanSaulodaCostaMazuco.pdf >.

MEDEIROS, Higor. **Práticas em XP: *Extreme Programming***. (2013). Disponível em:<https://www.devmedia.com.br/praticas-em-xp-extreme-programming/29330>.

MEZZOMO, Leonardo Possamai. **Um framework para avaliação e melhoria do processo de software aderente ao cmmi, mr-mps-sw e iso/iec 12207**. Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Ciências Exatas e Naturais, Programa de Pós-Graduação em Ciência da Computação, Belém, 2015. Disponível em: <http://ppgcc.propesp.ufpa.br/Disserta%C3%A7%C3%B5es_2015/Leonardo%20Possamai%20Mezzomo_Disserta%C3%A7%C3%A3o.pdf>.

MIRANDA, Mayllon Melo de. **Análise das Atividades de Desenvolvimento de Software no Centro de Processamento de Dados de uma Universidade Pública**. Universidade de Brasilia, Faculdade de Tecnologia, Departamento de Engenharia de Produção, Brasília, novembro de 2016. Disponível em: <https://bdm.unb.br/handle/10483/17754>.

MORAIS, Lenildo; VASCONCELOS, Audrey. **Modelos de Maturidade para Processos de Software: CMMI e MPS.BR**. 201-?. Disponível em: <https://cin.ufpe.br/~processos/TAES3/Livro/00-LIVRO/08-CMMI_MPSBR_v6_CORRIGIDO.pdf>.

NUNES, R. D. **A Implantação das metodologias ágeis de desenvolvimento de software scrum e extreme programming(XP): uma alternativa para pequenas empresas do setor de tecnologia da informação**. ForScience, v. 4, n. 2, (2016). Disponível em: < http://www.forscience.ifmg.edu.br/forscience/index.php/forscience/article/view/117>.

NUNO, Claudinei Di; OLIVEIRA, Elisamara de. **Engenharia de Software**. 201-?. Disponível em: <https://www.academia.edu/22497095/Engenharia_de_Software_Engenharia_de_Software?auto=download>.

ODA, Luciana Sayuri. **O que é metodologia ágil e como ela beneficia nos processos de sua empresa de tecnologia?**. Nov. 2018. Disponível em: <https://blog.sebrae-sc.com.br/metodologia-agil/>.

OLIVEIRA, Bruno Henrique**. Qualidade de software no desenvolvimento com métodos ágeis**. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2014. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-12082014-165244/pt-br.php>.

PHAM, Andrew; PHAM, Phuong-Van. ***Scrum* em ação: gerenciamento e desenvolvimento ágil de projetos de *software***. Tradução: Edgard B. Damiani. São Paulo, Novatec Editora, Cengage Learning, 2011.

PRESSMAN, Roger S. **Engenharia de *Software*: uma abordagem profissional**. Tradução: Ariovaldo Griesi; Mario Moro Fecchio; revisão técnica Reginaldo Arakaki; Julio Arakaki; Renato Manzan de Andrade. 7 ed. Porto Alegre, AMGH, 2011.

RAMOS, Renan de Oliveira. **Aplicação do método Seis Sigma como indicativo da qualidade de obras de construção civil**. Universidade Federal de Uberlândia, Faculdade de Engenharia Civil, Uberlândia, 2018. Disponível em: <https://repositorio.ufu.br/bitstream/123456789/22170/3/Aplica%C3%A7%C3%A3oM%C3%A9todoSeis.pdf>.

ROSES, Luís Kalb; VALLERÃO, Alexandre Guido. **Monitoramento e Controle de Projetos de Desenvolvimento de Software com o *Scrum*: Avaliação da**

Produção Científica. **Revista de Gestão e Projetos - GeP**, [S.l.], v. 4, n. 2, p. 100-127, aug. 2013. ISSN 2236-0972. Disponível em: <http://www.revistagep.org/ojs/index.php/ gep/article/view/154>.

SANTOS, Deisy Braz dos; CÓRDOVA, Paulo Roberto. **Combinação De Métodos Ágeis No Processo De Desenvolvimento De Software: Um Estudo De Caso**. *Ignis* vol.06, n.01, p. 06-23, jan./ abr 2017. Disponível em: <https://periodicos.uniarp.edu.br/ignis/article/view/1133/0>.

SILVA, Paulo Roberto da; SANTOS, Mario Roberto dos; SHIBAO, Fabio Ytoshi. **Desenvolvimento de Softwares: CMMI e Metodologias Ágeis**. Revista Livre de Sustentabilidade e Empreendedorismo, v. 4, n. 3, p. 157-184, mai-jun, 2019. Disponível em:<http://relise.eco.br/index.php/relise/article/viewFile/239/229>.

TRAD, Samir; MAXIMIANO, Antonio Cesar Amaru. **Seis sigma: fatores críticos de sucesso para sua implantação**. Rev. adm. contemp., Curitiba, v. 13, n. 4, p. 647-662, Dec. 2009. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1415-65552009000400008&lng=en&nrm=iso>.

UNIEVANGÉLICA. **Fábrica de Tecnologia Turing-FTT**. 201-?. Disponível em: <http://inf.unievangelica.edu.br/views/about.html>.

UNIVERSIDADE DO VALE DO ITAJAÍ (UNIVALI). **Avaliação e Melhoria de Processos de Software**. 201-?. Disponível em: <https://www.univali.br/pos/mestrado/mestrado-em-computacao-aplicada/linhas-de-pesquisa/Paginas/default.aspx >.

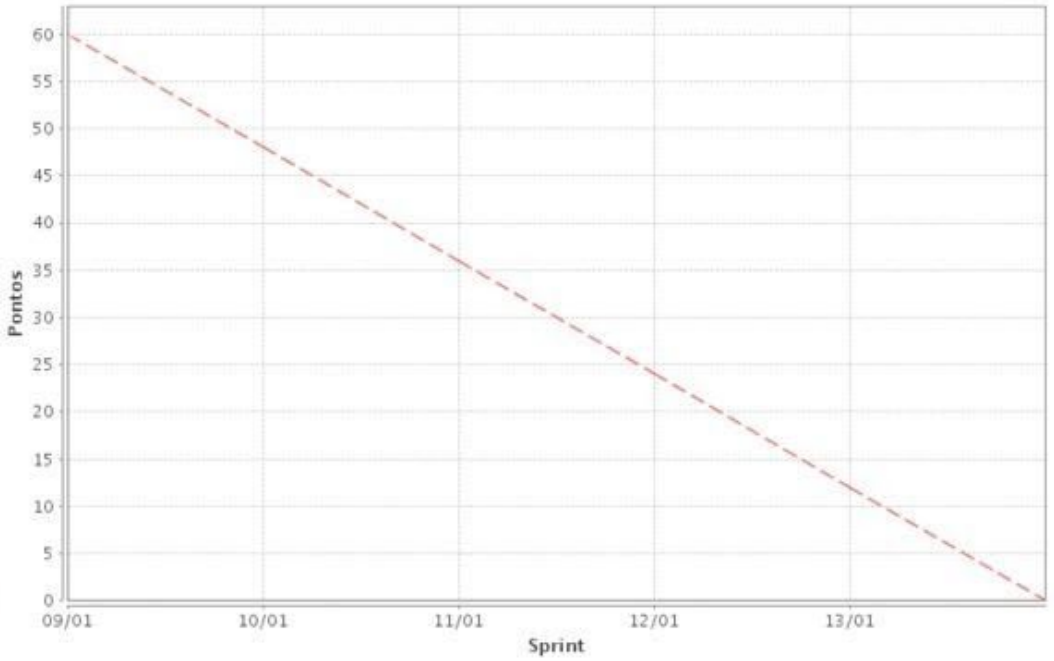
VENANZI, Delvio; FAUSTINO, Diogo Luis; SILVA, Orlando Roque Da; HASEGAWA, Haroldo Lhou. ***Lean Six Sigma – Multiple Case Study***. Revista GEINTEC, Aracaju/SE. Vol.7, n.4, p.4059-4073, out/nov/dez – 2017. Disponível em: <http://www.revistageintec.net/index.php/revista/article/view/1105>.

WANGENHEIM, Christiane Gresse von. **Melhoria de Processo de *Software***. SECCOM-UFSC, 2009. Disponível em: <http://www.inf.ufsc.br/~c.wangenheim/download/SECCOM2009-talk-vpdf.pdf>.

ANEXOS

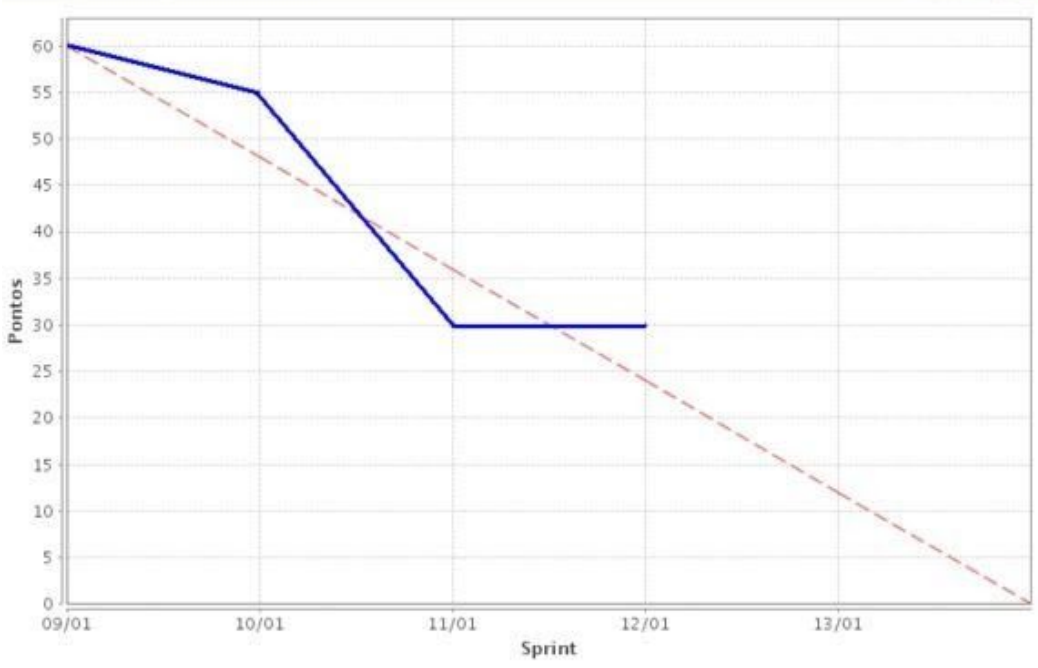
ANEXO A – GRÁFICOS *BURNDOWN*

Gráfico 1 – Início do projeto



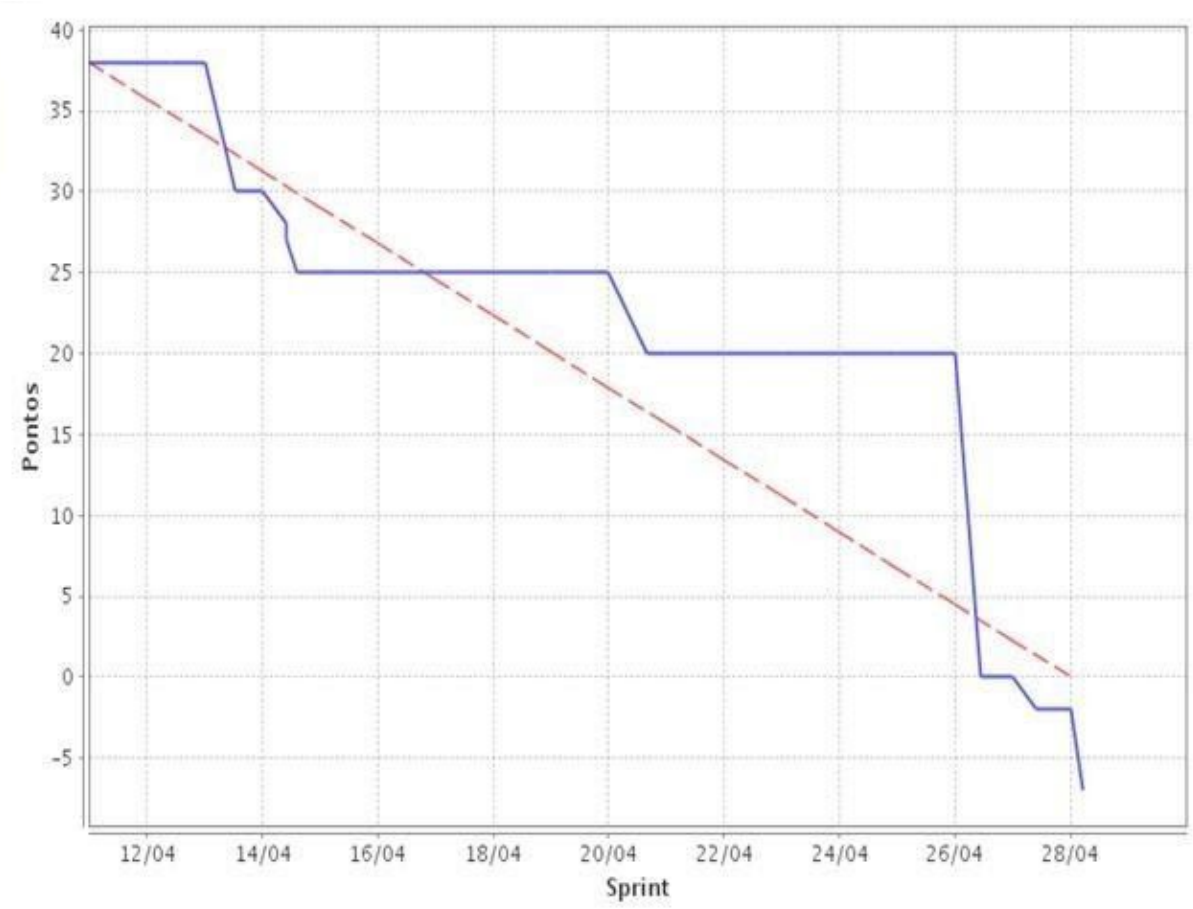
Fonte: Campos (2012, n.p.)

Gráfico 2 – Evolução do projeto



Fonte: Campos (2012, n.p.)

Gráfico 3 –Finalização do projeto



Fonte: Campos (2012, n.p.)

APÊNDICES

APÊNDICE A – QUESTIONÁRIO DE AVALIAÇÃO DA FTT APLICADO AOS INTEGRANTES DA EQUIPE DE DESENVOLVIMENTO DE *SOFTWARE* EM 2020/2

Qual Período está cursando? *

- ◯ 1°
- ◯ 2°
- ◯ 3°
- ◯ 4°
- ◯ 5°
- ◯ 6°
- ◯ 7°
- ◯ 8°
- ◯ 9°
- ◯ 10°

Qual a função que exerce na Fábrica de Tecnologias Turing (FTT). *

- ◯ Membro da Equipe de Desenvolvimento
- ◯ Líder da Equipe de Desenvolvimento

Há Quanto tempo integra a FTT? *

- ◯ Menos de 3 Meses
- ◯ Entre 3 e 6 Meses
- ◯ Entre 6 e 9 Meses
- ◯ Entre 9 Meses e 1 ano
- ◯ Mais de 1 Ano

Qual seu Nivel de conhecimento sobre métodos ágeis de desenvolvimento de software? *

- ◯ Nenhum
- ◯ Basico (Apenas conhece como funciona o método)
- ◯ Intermediario (Conhece a metodologia e sabe como aplicá-la parcialmente)
- ◯ Avançado (Conhece a metodologia e sabe aplicá-la completamente)

Conhece o processo atual da equipe de desenvolvimento existente na FTT? *

- ◯ Sim
- ◯ Parcialmente
- ◯ Desconheço

Conhece as fases do processo desenvolvimento da FTT? *

- ◯ Sim
- ◯ Parcialmente
- ◯ Desconheço

Utiliza o processo já estabelecido na FTT para realizar o desenvolvimento dos requisitos? *

- ◯ Sim
- ◯ Parcialmente
- ◯ Não

Ao longo do processo de desenvolvimento de software é utilizada alguma estratégia para melhoria da qualidade de código? *

- ◯ Sim
- ◯ Não
- ◯ Desconheço

Antes de integrar os requisitos desenvolvidos você realiza a atualização deles? *

- ◯ Sempre
- ◯ Às vezes
- ◯ Nunca

No momento de integrar os requisitos desenvolvidos para o sistema, há conflitos? *

- ◯ Raramente
- ◯ Frequentemente
- ◯ Desconheço

Em relação aos conflitos no momento de integrar os requisitos desenvolvidos com o sistema, conhece os procedimentos para não gerar conflitos durante a integração? *

- ◯ Sim
- ◯ Parcialmente
- ◯ Desconheço

Como você avalia a comunicação entre as equipes da FTT? *

- ◯ Otima
- ◯ Boa
- ◯ Mediana
- ◯ Ruim

Como você avalia a comunicação entre os integrantes da equipe de desenvolvimento da FTT? *

- ◯ Otima
- ◯ Boa
- ◯ Mediana
- ◯ Ruim

Como membro da equipe de desenvolvimento da FTT você considera o processo realizado atualmente como suficiente? *

- ◯ Sim
- ◯ Parcialmente
- ◯ Não

Como avalia a rotatividade de membros relacionados a equipe de desenvolvimento? *

- ◯ Alta
- ◯ Media
- ◯ Baixa

Existe padronização nos códigos elaborados pela equipe de desenvolvimento. *

- ◯ Sim
- ◯ Sim, mas não e aplicada
- ◯ Não

Os lideres da equipe de desenvolvimento da FTT verificam a padronização dos códigos. *

- ◯ Sim
- ◯ Não
- ◯ Parcialmente

Há qualificação para novos membros da equipe de desenvolvimento? *

- ◯ Há qualificação para novos membros da equipe de desenvolvimento
- ◯ Sim, mas não recebi qualificação quando entrei na equipe.
- ◯ Não, mas recebi qualificação quando entrei na equipe.
- ◯ Não e não recebi nenhum tipo qualificação quando entrei na equipe.

Caso tenha recebido qualificação para exercer sua função na FTT, esta foi suficiente para compreender e realizar atividades atribuídas na primeira Sprint? *

- ◯ Sim
- ◯ Parcialmente
- ◯ Não

Em relação aos processos da equipe de desenvolvimento da FTT, são especificados conforme algum procedimento padrão? Quando executados seguem conforme o estabelecido ou são executados de maneira contrária? *

- ◯ São definidos e executados conforme o estabelecido em procedimento padrão.
- ◯ São definidos porém executados de forma diferente do estabelecido em procedimento padrão.
- ◯ Não há um procedimento padrão a ser seguido.

Existe algum tipo de auditoria (Scrum Master ou Qualidade) para verificar se o processo da equipe de desenvolvimento da FTT segue conforme o estabelecido em procedimento padrão? *

- ◯ Sim
- ◯ Não

Você possui alguma sugestão para a melhoria do processo de desenvolvimento na FTT? Se sim, descreva. *

Sua resposta