UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Leonardo Ferreira Gonzalez

Análise comparativa sobre duas metodologias de gestão ágil: Extreme Programming e Scrum

UBERLÂNDIA

2021

Leonardo Ferreira Gonzalez

Análise comparativa sobre duas metodologias de gestão ágil: Extreme Programming e Scrum

Trabalho de Conclusão de Curso da Engenharia de Controle e Automação da Universidade Federal de Uberlândia - UFU - Campus Santa Mônica, como requisito para a obtenção do título de Graduação em Engenharia de Controle e Automação

Universidade Federal de Uberlândia – UFU

Faculdade de Engenharia Elétrica

Orientador: Márcio José da Cunha

UBERLÂNDIA

2021

Gonzalez, Leonardo

Análise comparativa sobre duas metodologias de gestão ágil: Extreme Programming e Scrum / Leonardo Ferreira Gonzalez – UBERLÂNDIA

Orientador: Márcio José da Cunha

Trabalho de Conclusão de Curso – Universidade Federal de Uberlândia – UFU Faculdade de Engenharia Elétrica . 2021.

Inclui bibliografia.

1. Gestão Ágil 2. Gestão de projetos 3.Extreme Programming 4. Scrum I. Márcio José da Cunha. II. Universidade Federal de Uberlândia. III. Faculdade de Engenharia Elétrica. IV. Engenharia de Controle e Automação.

Dedico esse trabalho a
minha mãe que
sempre foi um pilar
forte para quaisquer
que fossem os
objetivos que
colocasse à minha
frente, por sempre me
apoiar e fazer questão
de ser uma pessoa
presente no meu
crescimento

Agradecimentos

Agradeço muito a minha mãe Ester Machado Ferreira, meu pai Ramiro Gonzalez dos Santos por todos os ensinamentos e apoio para que eu tivesse oportunidade de fazer uma graduação. Agradeço também aos meus amigos que fizeram parte crucial de minha jornada na graduação, agradeço também a empresa contida no estudo de caso em questão pela oportunidade e incentivo. Por fim agradeço ao orientador Márcio José da Cunha.

“O melhor modo de se encontrar é perdendo-se a serviço de outros.”,
(Mahatma Gandhi)

RESUMO

O desempenho dos negócios empresariais, em tempos de competição e globalização, está cada vez mais relacionado com o sucesso dos projetos. O sucesso dos projetos está diretamente atrelado às interações que existem entre tecnologia, processos e pessoas.

A gerência de projetos se dá justamente nesse intermédio e nessas interações, é também a aplicação de ferramentas sob uma perspectiva estratégica que visa aumentar a produtividade e efetividade de uma solução. Quando analisamos o setor de software, pesquisadores e profissionais discutem e questionam as metodologias atuais há anos, esse questionamento foi o que trouxe uma migração da gestão tradicional em cascata para uma gestão ágil, feita por iterações.

Esse trabalho traz como intuito explorar de maneira comparativa duas metodologias de desenvolvimento ágil de projetos analisando de forma crítica sua literatura e, de forma mais específica, analisar o impacto do desenvolvimento orientado a testes para qualidade do software desenvolvido por cada uma delas. Para complementar a análise serão colocados também informações e resultados práticos de sua implementação no caso de uma empresa de software do ramo de CPM e BI. As metodologias escolhidas para análise são Scrum e Extreme Programming e a demonstração prática dos processos desenvolvidos serão feitas via modelagem de notação BPMN.

Palavras-chave: Gestão de Projetos; Extreme Programming; Metodologia Ágil; Scrum; Otimização; Processos;

ABSTRACT

The performance of corporate businesses, in times of competition and globalization, is increasingly related to the success of projects. The success of the projects is directly linked to the interactions that exist between technology, processes and people.

A project design takes place just in between and in these interactions, it is also an application of tools from a strategic perspective that aims to increase the productivity and effectiveness of a solution. When we analyze the software sector, researchers and professionals have been discussing and questioning current methodologies for years, this questioning was what brought about a migration from traditional cascading management to agile management, made by iterations.

This work aims to comparatively explore two agile project development methodologies, critically analyzing their literature and, more specifically, analyzing the impact of test-driven development for the quality of the software developed by each of them. To complement the analysis, information and practical results of its implementation will also be placed in the case of a software company in the field of CPM and BI. The methodologies chosen for analysis are Scrum and Extreme Programming and the practical demonstration of the developed processes will be done via BPMN notation modeling.

Keywords: Project Management. Extreme Programming. Agile. Scrum. Optimization. Processes.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

BI	Business Intelligence
BPMN	Business Process Model and Notation
CCD	Conventional Code Development
CPM	Corporate Performance Management
ITLD	Iterative Test-Last Development
MVP	Produto viável mínimo
PMBoK	Project Management Body of Knowledge
PMI	Project Management Institute
PO	Product Owner
QA	Quality Assurance
TDD	Test Driven Development
UI	User Interface
UX	User Experience
XP	Extreme Programming

Sumário

1 Introdução

Na atualidade, em praticamente todos os negócios e áreas, existe um papel essencial realizado por algum software. A Engenharia de Software é a área que estuda e entrega teorias técnicas e ferramentas visando a organização, produtividade e qualidade no desenvolvimento de um software. (FALBO, 2014) O atingimento desse objetivo no escopo principalmente da organização depende da ordenação de pessoas e processos, sem ordenação é impossível que uma equipe tenha consciência das tarefas e resultados que são esperados por cada um, como também de qual é o papel individual de cada membro para atingimento de um objetivo único. Processos Waterfall foram os primeiros a serem propostos dentro da Engenharia de Software. (VALENTE, 2020)

1.1 Justificativa

O método Waterfall foi introduzido pela primeira vez em um artigo de Wiston Royce de 1970 e em pouco tempo se tornou a principal ferramenta da época para o desenvolvimento de software. O método pauta-se em uma crença de que o processo de desenvolvimento de um software pode ser previsto por etapas sequenciais, de forma simplificada contendo um momento único para coleta de requisitos, um único para o design e outro, também único, para validação. (SZALVAY, 2004) Um estudo feito em 1994 chamado CHAOS Report feito pela Standish Group fornece dados que representam a efetividade geral da metodologia na época:

- 9% dos projetos de grandes empresas entregam os projetos com custo e tempo planejados inicialmente;
- 31.1% dos projetos de TI são cancelados antes de serem finalizados;
- 16.2% dos projetos sem distinção são entregues com custo e tempo planejados inicialmente.

A forma como o Waterfall se inseriu como metodologia de desenvolvimento de software e na Engenharia de Software tem suas origens de uma abordagem Taylorista. Quando aplicada ao desenvolvimento de software compõe na metodologia Waterfall uma priorização do planejamento inicial sobre mudanças que se adicionam

ao cenário de um projeto, pautado em um método científico que coloca seus esforços em uma repetição pragmática, tais influências são raízes primordiais para o fracasso de sua implementação no meio. (MAURER, 2007)

O desenvolvimento de sistemas de software é atividade dentro da comunidade de Engenharia de Software que considera puramente o conhecimento tecnológico, ou técnico. Esse viés acompanha a formação indicada e valorização dos profissionais no mercado. Entretanto, a atividade de desenvolvimento de software é composta por níveis de complexidade que extrapolam as barreiras técnicas, que envolvem habilidades como por exemplo negociação ou comunicação, habilidades então que são conectadas com outras áreas de conhecimento. A transformação dessa visão para que a discussão sobre a Engenharia de Software traga também conceitos de ciências humanas e sociais é algo necessário, transformação essa que nos últimos anos tem sido alvo de mais pesquisas e aproximando os desafios do desenvolvimento de software como desafios de complexidade tecnológica mas também social. O texto de Herbsleb ilustra bem essa discussão. (PRIKLADNICKI, 2007)

> *Uma compreensão mais profunda do nosso próprio campo nos conduz à interseção de várias disciplinas científicas – um espinhento emaranhado intelectual que muitos dentre nós preferem evitar a destrinchar. Compreender como se pode melhorar a engenharia de software requer um aprofundamento da nossa compreensão quanto a duas dimensões: 1) a dos princípios e práticas efetivos em engenharia de software; 2) a de como tais princípios e práticas se alinham frente ao modo como seres humanos funcionam cognitiva, social e culturalmente. Em minha opinião, a pesquisa atualmente realizada em engenharia de software tem progredido de forma estável quando se trata da primeira dimensão, porém correndo o tempo todo o risco de resultar irrelevante ao negligenciar as realidades interpostas pela segunda dimensão. Tendemos a assumir que seres humanos poderão mudar, e simplesmente o farão de todas as maneiras que se mostrarem necessárias. Todavia, o funcionamento humano não é assim tão maleável, e, para imenso prejuízo de nossas pesquisas, ignoramos tal fato.*
>
> HERBSLEB, Beyond Computer Science, 2005

Quando imaginamos os materiais utilizados para criação de um produto do meio, da concepção final do desenvolvimento de um software temos materiais extremamente voláteis: expectativas traduzidas em requisitos de um sistema que expressa uma vontade de um usuário que é incerta, pois o mesmo ainda desconhece o produto final, ou a comunicação sinérgica entre diversos outros softwares que

precisam comunicar com o programa que será desenvolvido. Por esses motivos podemos pensar que o desenvolvimento de software é um processo puramente intelectual, a transformação dos materiais envolvidos se traduzem em representações de diversas maneiras dos pensamentos e soluções que compõem o processo. (SCHWABER, 2004)

Dentro desse contexto e discussão surgem as metodologias de desenvolvimento ágil, trazendo uma nova perspectiva sobre como evoluir a forma de se desenvolver software. Tal perspectiva atrela-se a uma mudança cultural de uma sociedade que aplicava uma lógica Taylorista para o desenvolvimento de software para começar a enfatizar mais pessoas do que processos, a apresentar um modelo de gestão preparado para a complexidade que os novos desafios apresentam. Para isso foge-se então da previsibilidade absoluta e controle para um conjunto de valores, princípios e práticas que pudessem auxiliar na tomada de decisões de um ambiente que está em constante transformação. (WILLI, 2014)

1.2 Objetivo

Esse trabalho vem com intuito de fazer uma análise comparativa sobre duas metodologias de gestão ágil, o Extreme Programming e o Scrum. Essa análise tem como objetivo clarificar o entendimento acerca dos métodos, assim como a aplicação dos mesmos, auxiliando no entendimento de suas ferramentas e no entendimento das vantagens e desvantagens de cada método. A análise se pauta em uma revisão literária de ambas as metodologias e também em um estudo de caso real feito dentro de uma empresa de tecnologia do ramo de CPM e BI da cidade de Uberlândia que aplica hoje metodologias ágeis para gerência de seu projeto e produto único.

1.3 Organização do trabalho

O trabalho inicia sua análise no capítulo 2 a partir de uma introdução teórica que prioriza uma literatura clássica sobre o assunto, com autores que fizeram parte da construção do manifesto ágil, em um cenário onde ainda não haviam impactos do mercado sobre o conteúdo de forma comercial. Essa introdução fornece uma visão geral sobre os papéis (cargos) existentes em ambas as metodologias, os valores, os processos e sobre as principais ferramentas que auxiliam no processo de desenvolvimento.

Em seguida no capítulo 3 é especificado como o trabalho pretende abordar as duas partes de sua análise, uma inicial que soma e sintetiza estudos que trazem a discussão mais recente sobre o assunto priorizando uma análise sobre a qualidade do software resultante de cada metodologia pela divergência de utilização do TDD (Test-Driven Development ou Desenvolvimento Orientado a Testes). Em uma segunda parte contextualiza-se o caso da empresa analisada, trazendo então a estrutura da equipe, a cultura da empresa e seus processos.

No capítulo 4 a estrutura é divida e ordenada da mesma forma que a apresentação da metodologia, inicialmente na análise comparativa teórica e em seguida discute-se o caso analisado da empresa escolhida.

Para análise comparativa é feita uma crítica sobre o perfil de cada uma das literaturas de forma geral e depois aprofunda-se nos impactos em qualidade externa, qualidade interna e produtividade no desenvolvimento de software por cada metodologia a partir de um dos principais pontos de divergência de ambas as metodologias, a utilização do TDD.

E por fim, na análise da implementação das metodologias na empresa escolhida fez-se um paralelo do que é ou não utilizado dentro da empresa, assim como uma análise crítica das escolhas tomadas pela empresa de acordo com o que as metodologias fornecem como orientação e estudos já contextualizados no trabalho.

2 Referencial teórico

2.1 Gerência de Projetos

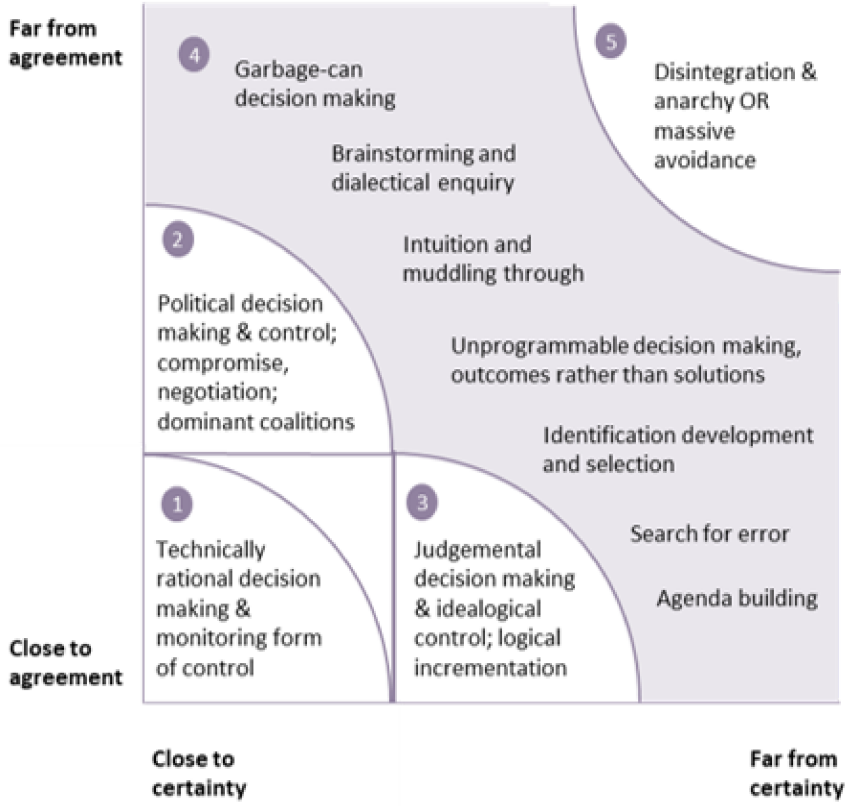
O gerenciamento de projetos é um conjunto de ferramentas gerenciais que permite que a empresa desenvolva um conjunto de habilidades, destinados ao controle de eventos não repetidos, únicos e complexos, dentro de um determinado tempo, custo e qualidade (VARGAS, 2009). Dentre as várias definições existentes para o conceito de gerenciamento de projetos esse conceito consegue se colocar como um meio termo entre duas perspectivas. De uma primeira ótica temos o projeto como uma entrega final planejada, composta pelas etapas da abordagem tradicional: 1. Iniciação; 2. Planejamento; 3. Execução; 4. Monitoramento e Controle; 5. Encerramento. Podendo citar o exemplo mais clássico dessa abordagem que seria a Metodologia Cascata (Waterfall), sendo das abordagens tradicionais a que atualmente é ainda a mais utilizada. (PMBOK, 2018)

Um grande divisor de águas na história do gerenciamento de projetos foi quando através de instituições como o PMI (Project Management Institute) foi-se desenvolvido o PMBoK, que é basicamente um guia que contém um conjunto de conhecimentos e boas práticas avaliadas por um conjunto de especialistas que pretende dar um passo a passo estruturado para melhor execução de uma gerência de projetos de sucesso. Atualmente, principalmente quando pensamos em projetos de tecnologia, que envolvem o desenvolvimento de software, a preferência por especialistas tem sido uma abordagem mais dinâmica sobre a gerência de projetos, e foi após a década de 90 com a criação das metodologias ágeis que surgiu um novo e maior nicho de conceitos e metodologias, especializadas para resolver os desafios que as áreas de tecnologia apresentam para os gestores.

Quando pensamos na gestão de projetos aplicada ao desenvolvimento de software precisamos ter uma abordagem que adeque e especialize as visões e análises gerais para esse escopo. Existem perguntas e particularidades que são únicas do meio, o processo de desenvolvimento de software é complexo, tentar prever essa complexidade através de um planejamento robusto é extremamente difícil por esse motivo, mas é justamente nesse ponto que metodologias como o método cascata mantinham o centro de sua estratégia. Documentos com centenas

de páginas que explicariam exatamente como cada passo deveria ser tomado, tentavam assim prever cada dificuldade que o projeto enfrentaria até o final de sua entrega.

Figura 1 - Matriz de Stacey



Fonte: Stacey RD (2002)

A imagem acima representa a Matriz de Stacey, desenvolvida e publicada por Ralph Douglas Stacey, a mesma foi feita para auxiliar no processo de tomada de decisões de gestão através de uma categorização dos ambientes provenientes entre a intensidade de duas variáveis representadas nas axis, sendo elas:

- **Eixo Y:** representa o nível de acordo existente entre o objetivo final e o que está sendo proposto é o que responde a pergunta do “O que” o projeto se propõe. Trazendo para o universo do desenvolvimento de software essa axis nos daria os requisitos de um projeto.
- **Eixo X:** representa o nível de certeza. Geralmente nos dá a pergunta do “Como”, representa também a conexão entre causa e efeito do projeto, o quanto o que está sendo proposto está sob controle das ferramentas e conhecimentos dispostos pela equipe. Trazendo para o universo do

desenvolvimento de software essa axis representaria para nós a tecnologia.

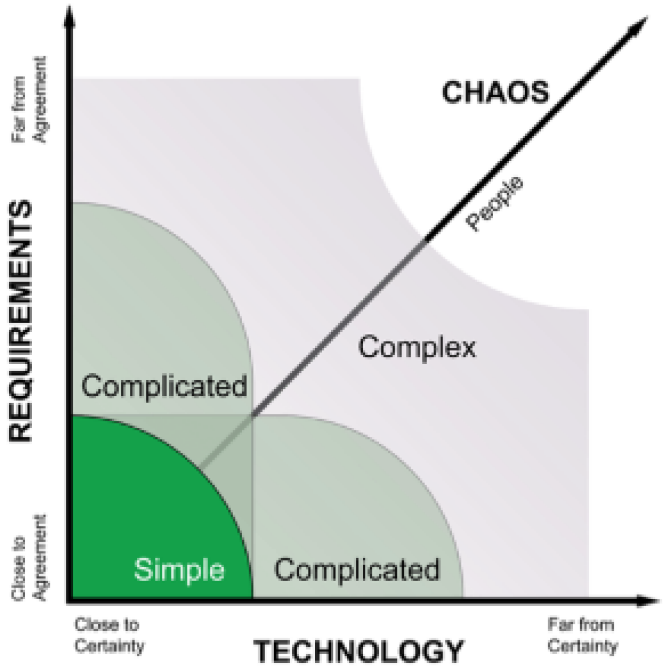
Conforme se há mais ou menos acordo entre os requisitos e mais ou menos tecnologia disponível para concretizar esses requisitos entramos nas seguintes áreas (numerais respectivos às zonas da imagem 2):

1. **Alto nível de acordo, alto nível de certeza:** é uma zona de conforto para gestores, é onde temos a maior facilidade de se usar dados de projetos e situações passadas para o planejamento e execução. É onde planos detalhados conseguem ter maior poder de previsibilidade, já que os requisitos são claros e as tarefas simples de acordo com a tecnologia disposta. (STACEY, 2002)
2. **Baixo nível de acordo, alto nível de certeza:** geralmente associada a momentos onde o processo de tomada de decisões precisa ser mais político do que técnico, os requisitos podem estar tendo conflitos com o cliente final, ou não foram pensados de forma a atender a todas as partes envolvidas e por isso até mesmo sua entrega final não seria satisfatória. Aqui habilidades como negociação se tornam importantes e é necessário que o gestor faça coalizões para garantir acordo entre o escopo do projeto e a necessidade do cliente. (STACEY, 2002)
3. **Alto nível de acordo, baixo nível de certeza:** esses casos onde a causa-efeito dos projetos é incerta as decisões a serem tomadas pelos gestores encontram-se mais no sentido de encontrar uma visão mais realista para abordagem do problema. As decisões devem partir de questionamentos sobre a correspondência entre budget e tempo disponíveis em relação aos objetivos traçados, principalmente quando o produto é um produto inovador também é necessário questionar a maturidade da própria tecnologia que está disponível no mercado. (STACEY, 2002)
4. **Zona da complexidade:** combinação entre baixos níveis de certeza e de acordo, normalmente pode sugerir um caminho para decisões de base precária, principalmente quando há pouca flexibilidade em relação aos prazos, métodos, contratos ou objetivos traçados. É a zona e área onde se tem conceitos como o “Garbage-can”, “Systems Thinking” e “Concurrent Engineering” são considerados adequados, assim como habilidades como

adaptabilidade, criatividade e agilidade por parte dos gestores. (STACEY, 2002)

5. **Baixo nível de acordo, baixo nível de certeza:** baixíssimos níveis de certeza e de acordo, logicamente a zona onde independente da abordagem, metodologia ou linha adotada é uma zona para se evitar dentro da gestão de um projeto. É a zona de caos e onde existe maior fuga por parte da equipe e inclusive da gestão para resolução dos problemas, sem ordenação, baixa produtividade. (STACEY, 2002)

Figura 2: Matriz de Stacey adaptada para o desenvolvimento de software



Fonte: Alex Ballarin (2019)

A matriz de Stacey é uma boa abordagem para ilustração do impacto da complexidade no processo de tomada de decisão por parte dos gestores e inclusive sobre o viés de pensamento entre o paradoxo de controle. Metodologias como Gestão em Cascata (Waterfall) pautam-se em uma previsibilidade imensurável, os prazos colocados subestimam as dificuldades dessa análise inicial que provê esse orçamento e planejamento. Por esse motivo é possível pensar que as zonas que a metodologia de Gestão em Cascata trilha seu melhor estado quando a complexidade não envolve uma baixa certeza, ou seja, as tarefas têm uma causa-efeito mais previsível, mas não acho que essa seja uma afirmação justa com as metodologias de Gestão em Cascata, mas o que deve ser de fato destacado é o

potencial de evolução da adoção de valores como revisão e adaptação das metodologias ágeis em relação a metodologia em cascata para ter mais reação a essas imperfeições que existem de origem tecnológica ou de escopo.

O paradoxo de controle na verdade, segundo Philip J. Streatfield em seu livro The Paradox of Control in Organizations, deve ser evitado, ao invés então de colapsar entre visões diferentes de “o gestor tem o controle” e “o gestor não tem o controle” é mais valioso, segundo Philip fazer sentido desse paradoxo.

> Os gerentes estão no controle das organizações em que trabalham? Minha experiência agora me sugere que esta é a pergunta errada. A principal capacidade de gestão não é estar "no controle", mas sim a capacidade de participar criativamente na formação de um significado transitório, o que permite que todos os membros de uma organização continuem vivendo com a ansiedade gerada pela mudança. É esse significado que cria uma sensação de ordem, coerência, padrão ou controle. A capacidade de participar criativamente na construção de significado se desenvolve à medida que os gerentes lutam para lidar com o paradoxo do controle, usando mecanismos de controle legítimos como ferramentas em uma dinâmica mais ampla de interação comunicativa auto-organizada. Acredito que os profissionais de administração aprimoram e desenvolvem continuamente a capacidade de conviver com o paradoxo à medida que fazem sua prática, mesmo que não tenham consciência disso.
>
> (Streatfield, 2001, p. 136)

A introdução por base nesses conceitos contextualizados ao ambiente e desafios do desenvolvimento de software fornecem uma visão sobre os questionamentos que começavam a ser feitos aos conceitos de gestão da época. As metodologias ágeis tiveram papel crucial neste debate e na transformação desempenhada anos depois para Engenharia de Software como um todo.

2.2 Metodologias Ágeis

Por mais que as metodologias ágeis sejam um conceito que até hoje estão em voga no mundo da tecnologia, as mesmas são datadas de meados de 1990. O conceito de agilidade na gestão não demorou a ser difundido entre especialistas e rapidamente se tornou parte crucial na gestão dos projetos de desenvolvimento de software. Diferente de vertentes como PMBoK (Project Management Body of Knowledge) comenta que essas metodologias fazem uma transição do conceito de projetos, que antes, como exemplo do próprio PMBoK eram definidos como um

“esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo” para uma perspectiva onde possuem um processo muito mais incremental e contínuo, com mais adaptabilidade.

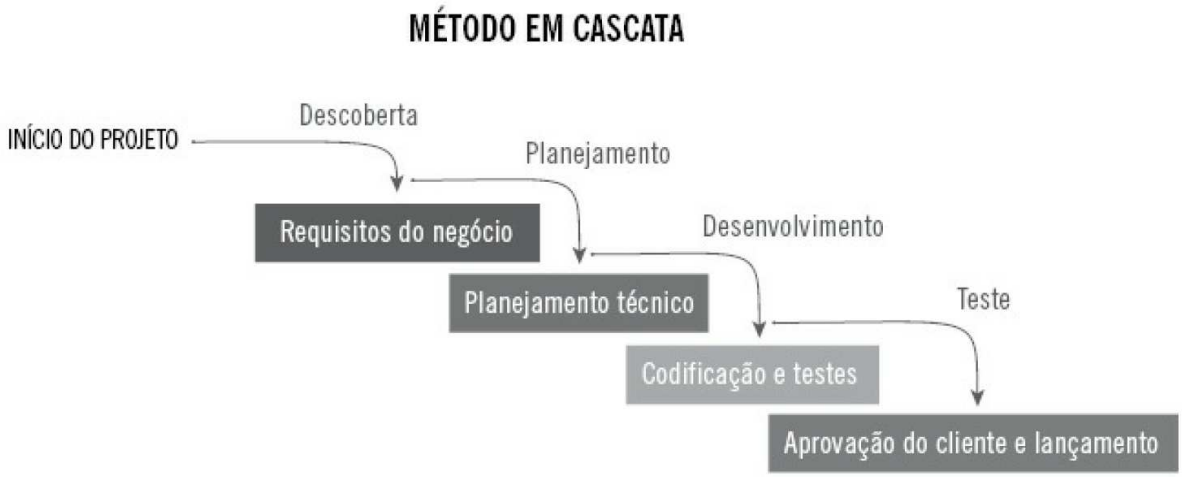
Toda e qualquer metodologia de desenvolvimento ágil segue os princípios do manifesto ágil, que diz: "*Estamos descobrindo maneiras melhores de desenvolver software, fazendo-os nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:*

Indivíduos e interações mais que processos e ferramentas
Software em funcionamento mais que documentação abrangente
Colaboração com cliente mais que negociação de contratos
Responder a mudanças mais que seguir um plano

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.” (Agile Manifesto, 2001)

As metodologias ágeis partem de uma época onde o desenvolvimento de software era baseado principalmente em um tipo de abordagem, a Waterfall (ou desenvolvimento em cascata).

Figura 3: Métodos de gestão em cascata



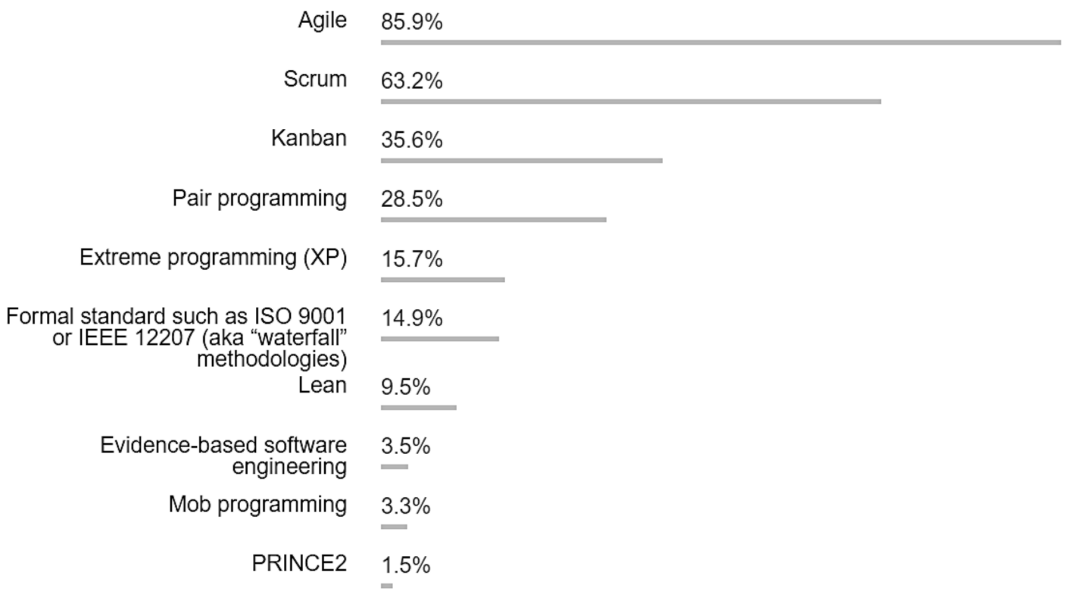
Fonte: Jeff Sutherland (2014)

Esse tipo de método utilizava a organização através dos chamados “diagramas de Gantt”, em homenagem a Henry Gantt. Com a facilidade de acesso a

computadores ficou também mais fácil a criação de diagramas complicados que buscavam ao máximo prever cada desafio e problema que seria encontrado até o destino final: a entrega de um produto que condiz com os requisitos e planejamento construídos ao início do projeto. (SUTHERLAND, 2014)

Essa é a primeira de muitas distinções existentes entre a abordagem em cascata com as metodologias ágeis, pegando como exemplo um dos pontos do manifesto que ilustra essa divergência: “Responder a mudanças mais que seguir um plano”. Essa valorização da velocidade de adaptação e compreensão que o planejamento inicial precisa e vai se transformar é um primeiro passo para transição árdua que as empresas passaram. Hoje vivemos em um cenário onde mais de 85% das empresas dentro do ramo da tecnologia da informação utilizam metodologias ágeis para gestão de seus projetos, segundo a Stack Overflow Survey realizada com mais de 50 mil respondentes em 2018.

Figura 4: Utilização das metodologias de gestão de projeto para software



Fonte: Stack Overflow Survey (2018)

2.3 Scrum

O termo Scrum foi criado em referência ao jogo de rugby e a maneira como um time trabalha junto para avançar com a bola no campo. Alinhamento cuidadoso, unidade de propósito, clareza de objetivo, tudo se unindo. Trata-se de uma metáfora perfeita para o que uma equipe deseja fazer. (SUTHERLAND, 2014)

O Scrum acredita e aceita que somos criaturas extremamente influenciadas pelo hábito e pela repetição, a metodologia foi pensada de forma a utilizar da maneira mais eficaz os padrões buscados por seres humanos em processos de repetição aperfeiçoados com o passar do tempo. (SUTHERLAND, 2014) O Scrum acredita que o desenvolvimento de software é uma atividade complexa de tal forma que a previsibilidade direta premeditada de seus projetos é estatisticamente improvável pelas abordagens tradicionais de modelagem definitiva, característica do planejamento de modelos como o do já mencionado de Gestão por Cascata. (SCHWABER, 2004)

A engenharia em si aceita uma abordagem em processos onde um determinado nível de imprecisão é aceitável, como sabemos em um carro a todo momento as rodas oscilam, os cilindros vibram e os freios tremem. Na construção de um veículo existe um nível de precisão aceitável que determina quando aquele projeto estará cumprindo seu papel essencial, e dentro também de um processo de construção de um veículo podemos notar que encontra-se hoje um conjunto de etapas previsíveis imutáveis que permitem que o resultado final de qualidade, ou seja, que essa taxa de imprecisão e erro mantenha-se estável. O que acontece quando não podemos encontrar esse conjunto fixo e estático de etapas previsíveis, quando a repetição de processos nos leva a resultados diferentes (principalmente negativos) é a aplicação de uma abordagem chamada controle de processos empíricos, e é essa abordagem que o Scrum utiliza. (SCHWABER, 2004)

> É típico adotar a abordagem de modelagem definida (teórica) quando os mecanismos subjacentes pelos quais um processo opera são razoavelmente bem compreendidos. Quando o processo é muito complicado para a abordagem definida, a abordagem empírica é a escolha apropriada.
>
> B. A. Ogunnaike e W. H. Ray,
> Process Dynamics, Modeling, and Control (1994)

Essa abordagem é baseada em três blocos essenciais: transparência, inspeção e adaptação. Transparência significa que quaisquer que sejam as variáveis que determinam o processo devem estar visíveis para aquelas que possuem influência sobre as mesmas, e o que está visível também deve ser compatível com a realidade, dentro da metodologia do Scrum esse bloco está muito presente na definição do que é uma tarefa finalizada acordado entre as partes envolvidas no

projeto. Inspeção significa que as várias partes de um processo definido precisam ser constantemente revisadas, de forma a detectar variâncias inaceitáveis, dentro da metodologia do Scrum existem rotinas específicas de análise atribuídas a parte de juízo competente adequado, os objetos de análise são os artefatos do Scrum. E por fim a adaptação, a mesma é consequente aos processos de inspeção, caso a inspeção feita em um dos eventos explicitar algum problema no andamento em um dos processos em andamento uma ação deve ser tomada de forma imediata para minimizar riscos futuros. (SCHWABER, 2004)

2.3.1 Valores do Scrum

Dentro das metodologias ágeis os valores representam uma bússola para toda e qualquer decisão tomada. Diferente de outras partes do conteúdo pode-se dizer que são a parte mais adaptável e flexível para qualquer realidade, por mais que os papéis, ritos e outras partes da metodologia já sejam pensados para trabalhar em sinergia com esses valores.

Coragem: Os membros da equipe devem ter a coragem de fazer a coisa certa e trabalhar para resolver seus problemas. (SCHWABER, 2004)

Foco: Todos devem ter um foco único no trabalho da Sprint e nas metas do time. (SCHWABER, 2004)

Comprometimento: Todos devem se comprometer pessoalmente para atingimento das metas do time. (SCHWABER, 2004)

Respeito: Os membros da equipe devem se respeitar entre si de forma que todos sejam pessoas capazes e independentes. (SCHWABER, 2004)

Abertura: O time e todas as partes envolvidas devem concordar em ter transparência acerca do trabalho que deve ser feito e também nos desafios enfrentados. (SCHWABER, 2004)

2.3.2 Papéis

Product Owner: em suma é a pessoa responsável por gerir o Backlog do produto e maximizar o valor de um projeto, é o representante de todas as partes envolvidas dentro de um projeto. (SCHWABER, 2004)

Scrum Master: Ken Schwaber comenta em seu livro que poderia ter continuado com a nomenclatura padrão para o cargo e definir esse papel como “gestor do projeto”, porém foi escolhido esse nome justamente pois era do seu interesse explicitar as diferenças existentes nas responsabilidades de um gestor de projetos tradicional e a proposta dessa função. A autoridade do Scrum Master é de certa forma indireta, seu papel se impõe basicamente para garantir o cumprimento correto das regras e práticas do Scrum no projeto, por isso dentre todos os envolvidos em um time de Scrum ele é a pessoa que indubitavelmente deve possuir um conhecimento concreto sobre a metodologia. O Scrum Master é o responsável pelo sucesso do projeto e pela maximização dos benefícios da utilização do Scrum. (SCHWABER, 2004)

Time: conjunto de pessoas com habilidades multidisciplinares autogeridas e com objetivo de desenvolver software a cada Sprint. (SCHWABER, 2004)

2.3.3 Eventos e regras

O Scrum é caracterizado por uma série de eventos cíclicos que compõem um processo de gestão estruturado e completo, chamados de ritos, cada um contém regras específicas que incluem tempo de duração, forma de facilitação, objetivos e resultado esperado.

Sprint: o Scrum assim como todo método ágil é um método iterativo, Sprint é o nome dado para a iteração do Scrum. Ela ocorre em um intervalo fechado de 30 dias consecutivos, esse intervalo é pensado para permitir um intervalo mínimo para que o time consiga desenvolver incrementos ao produto potencialmente implementáveis e ao mesmo tempo é o máximo considerado para que o cliente final não perca interesse no desenvolvimento do projeto. Acredita-se no Scrum que esse intervalo também é o máximo para que não seja necessário o desenvolvimento de

documentações complexas para acompanhamento do projeto. (VALENTE, 2020)

Sprint planning meeting: reunião que marca o início de uma Sprint, onde o planejamento da mesma acontece em intervalo fechado de 8 horas, dividido em duas etapas. A primeira etapa tem a figura chave do Product Owner, que prepara uma apresentação onde se apresentam histórias aos desenvolvedores e os mesmos definem se possuem tempo necessário para implementá-las no intervalo de uma iteração. (SCHWABER, 2004) Com as histórias para Sprint definidas acontece a segunda etapa, protagonizada pelos desenvolvedores, nesse momento as histórias são quebradas em tarefas, nesse momento acontece também uma estimativa para as mesmas, caso aconteça de uma história mostrar-se mais complexa nessa análise mais completa a história pode ser removida da Sprint. (VALENTE, 2020)

Daily Scrum meeting: reunião diária fechada a um tempo máximo de 15 minutos independente da quantidade de membros existentes naquele time. A mesma possui o seguinte conjunto de características e regras:

- Deve ocorrer sempre no mesmo dia e no mesmo horário. Orienta-se para colocá-la como a primeira atividade do dia sempre. (SCHWABER, 2004)
- Cada membro do time deve responder às seguintes perguntas somente:
    - O que você fez desde a última reunião diária no que se relaciona a este projeto?
    - O que você fará entre essa reunião e a próxima?
    - O que te impede de performar da forma mais otimizada possível? (SCHWABER, 2004)

Sprint review: reunião feita com propósito de facilitar um momento para que o time apresente os resultados de uma Sprint, tanto para o Product Owner quanto para o cliente. Essa reunião tem tempo fechado de 4 horas, as apresentações são focadas em novas funcionalidades do produto, então o processo de desenvolvimento não é interessante aqui. Se alguma entrega em parte ou por completo se mostrou incompleta ou fugiu do objetivo inicial ela pode ser recolocada no Backlog de produto para ser retrabalhada em uma próxima Sprint. (SCHWABER, 2004)

Sprint retrospective meeting: é considerada a última atividade de uma Sprint, tem

tempo fixo fechado de 3 horas e tem como objetivo realizar uma revisão de como o trabalho se deu na última Sprint, ela é feita entre a equipe completa do Scrum, nela participam somente PO, Scrum Master e o time. (SCHWABER, 2004) A reunião é iniciada através da resposta de cada membro da equipe às seguintes perguntas:

- O que ocorreu bem na última Sprint?
- O que pode ser melhorado na próxima Sprint?

É importante ressaltar que nesse momento anotações devem ser feitas pelo Scrum Master e tarefas reais são construídas para serem colocadas no Backlog de Produto como atividades de alta prioridade. É um momento de feedback também de se propor mudanças e melhorias para os processos, para qualquer ponto de comunicação ou problema que possa ter ocorrido na última Sprint. (VALENTE, 2020)

2.3.4 Artefatos

Backlog de produto: lista considerada contínua, ou seja, eterna, onde os requisitos para um produto que está sendo desenvolvido evoluem assim como todas as outras partes de um projeto, enquanto o produto e o projeto existirem, existirá um Backlog para o mesmo de forma a torná-lo mais útil, competitivo e adequado ao mercado. O Product Owner é o responsável pelo Backlog de produto, mantendo o mesmo disponível, com um conteúdo bem escrito e com priorização feita. (SCHWABER, 2004)

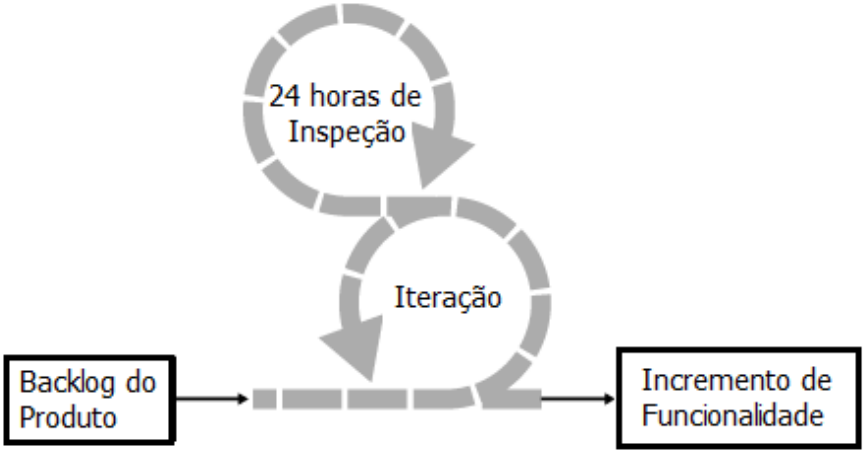
Backlog da Sprint: a cada Sprint, parte do Backlog de produto, com base na priorização escolhida e das demandas do cliente e do mercado, é escolhida para compor o trabalho proposto daquela Sprint. Considera-se que o trabalho sendo executado em uma Sprint deve ser sempre representado em tempo real para que o time consiga acompanhar seu desempenho continuamente. Para construção do Backlog os requisitos vindos do Backlog de produto são quebrados de forma que estejam representados por tarefas que devem ter um tempo de execução de 4 até 16 horas, tarefas que ultrapassam uma estimativa de 16 horas precisam de maiores análises pela equipe de desenvolvimento de forma que se tornem mais granulares. O Backlog da Sprint só pode ser alterado pelo time após sua definição. (SCHWABER, 2004)

Incrementos de funcionalidades do produto potencialmente implementáveis: todo trabalho precisa ser feito em incrementos, então todo requisito, quebrado em tarefas no backlog da Sprint se torna um incremento quando seu valor é entregue ao cliente com base na **Definição de Pronto** acordada entre as partes responsáveis do projeto. (MACIEIRA, 2018) Essa definição basicamente diz quais são os critérios mínimos para entrega de uma tarefa, por exemplo se ela precisa ser completamente testada, bem estruturada, em código bem escrito, construída em arquivo executável, com documentação especificada, entre outros pontos que são variáveis com base na necessidade do projeto em questão. (SCHWABER, 2004)

2.3.5 Funcionamento

O Scrum concentra seu funcionamento em uma prática incremental e iterativa, como representado na imagem abaixo. O círculo inferior representa para nós as iterações de desenvolvimento do produto com suas diversas atividades, o resultado de cada iteração deve ser sempre um novo incremento de funcionalidade(s). O círculo superior representa basicamente as Daily meetings, reuniões diárias que promovem inspeção entre das atividades entre membros e adaptações rápidas, tais reuniões são parte de todas as iterações. O Backlog do produto é a fonte primária de requisitos e direcionamento para o projeto. Esse ciclo é o esqueleto do Scrum que se repete até que o projeto finalize ou que o produto não mais exista.

Figura 5: Esqueleto dos processos do Scrum



Fonte: Ken Schwaber (2004)

Além desse esqueleto acredita-se que o Scrum começa de fato através de uma visão de produto criada pelo Product Owner, essa visão então deve ser lapidada em requisitos funcionais e não funcionais, possibilitando assim o primeiro passo do fluxo completo da metodologia, que é a construção do Backlog do produto. A partir do Backlog do produto o PO prioriza essa lista com base em uma análise de quais requisitos tem maior potencial de geração de valor para o produto e então essa lista é particionada e organizada em releases. (SCHWABER, 2004)

Todo trabalho no Scrum é realizado em Sprints, cada uma representa uma iteração e, como dito, é feita com duração fixa de 30 dias, a Sprint é iniciada através da Sprint planning meeting onde o PO apresenta uma proposta do que será trabalhado na próxima Sprint, a partir dessa apresentação o time seleciona de toda apresentação quais requisitos poderão ser feitos no ciclo que está sendo planejado. A segunda parte da Sprint planning meeting é destinada ao time e é focada unicamente no planejamento da Sprint, o time então fica responsável por transformar cada um dos requisitos apresentados em incrementos de funcionalidade potencialmente implementáveis, momento onde as expectativas e os requisitos selecionados do Backlog de produto são transformados no Backlog da Sprint. (SCHWABER, 2004)

Como atividade constante de desenvolvimento da Sprint acontecem as Daily meetings, mantendo a equipe focada nos objetivos traçados e também garantindo que quaisquer bloqueios sejam desfeitos imediatamente. Ao final da Sprint acontece o Sprint Review, momento onde a equipe apresenta os resultados da Sprint para o PO e qualquer parte interessada que queira participar, tal reunião é um momento informal porém importante para que de forma colaborativa sejam traçados novos caminhos para o time. Após a Sprint Review o Scrum Master realiza a Sprint Retrospective meeting, facilitando dessa forma uma revisão interna sobre a qualidade da aplicação do Scrum na última Sprint, levantando e sugerindo propostas para tornar o processo mais agradável e efetivo. Após esse passo remarca-se a próxima Sprint planning meeting para que o ciclo se torne completo. (SCHWABER, 2004)

2.4 Extreme Programming

O Extreme Programming foi usado pela primeira vez na década de 1990 em um projeto da Chrysler, o Chrysler Compreensive Compensation (C3), o projeto mudou para a abordagem XP quando Kent Beck foi trazido para aumentar a performance e possivelmente aproximar a equipe da entrega final. O projeto na época estava há 2 meses da entrega final há 5 meses, a equipe se atentou continuamente para quem teria acesso a informação antes de dizer qualquer coisa para resguardo individual, as pessoas com as quais ele teve contato estavam claramente cansadas e irritadas.

Kent comenta em seu livro “Extreme Programming Explained: Embrace Change” que assim como geralmente acontece nos casos de consultoria o cliente já sabe a resposta antes de contratar. Após dois dias em uma reunião com o CIO da Chrysler a escolha tomada foi de largar tudo que havia sido feito e recomeçar todo projeto do zero com um time menor. Segundo própria definição, consolidada ao longo do primeiro dia de alinhamento do projeto e metodologia representavam o básico do Extreme Programming: um sistema sempre implantável no qual as funcionalidades, escolhidas pelo consumidor, são adicionadas e automaticamente testadas em um ritmo fixo constante.

2.4.1 Valores do Extreme Programming

Comunicação: o XP acredita que comunicação é essencial para que seja criado um senso de time e cooperação efetiva, não somente, comunicação dentro do XP é considerada como peça chave que tem potencial tanto destrutivo para o sucesso da equipe quanto também para solucionar problemas, já que se pode, por exempo, escutar de membros da equipe experiências de aprendizado que se conectam aos desafios atuais. O XP enfatiza a preferência pela comunicação face a face e também a utilização de ferramentas visuais (como quadro branco). (K. BECK, 1999)

Simplicidade: Nas palavras de Kent Beck “To make a system simple enough to gracefully solve only today's problem is hard work”, o que será feito é exatamente o que foi pedido, nada mais. É talvez um dos valores mais intensos do Extreme Programming, tem o propósito de evitar ao máximo o desperdício de recursos,

construindo um sistema que seja mais fácil de ser revisado, alterado e consertado em caso de bugs. Simplicidade também significa que devemos colocar unicamente os requisitos no sistema no qual se tem certeza da necessidade, no XP não se tenta explorar possibilidades futuras. Entende-se também que simplicidade só faz sentido em contexto, então se uma equipe é especialista em Java tomar um caminho de solução via Python talvez não seja o caminho mais simples. (K. BECK, 1999)

Feedback: o XP acredita no feedback como parte essencial da agilidade em sua metodologia, o feedback é incentivado e pode se colocar em diversas situações, ele pode vir do código, através dos testes que são requisito primordial do XP. Esse feedback pode vir também como feedback do consumidor, por exemplo, ao priorizar que os testes funcionais sejam escritos juntamente com o usuário, e por fim o feedback vale para a equipe por práticas como Planning Poker ou Spikes para entendimento estratégico sobre um desafio e alinhamento entre as partes envolvidas. (K. BECK, 1999) Não existe caminho certo, nem para o produto, para os processos ou para o código, é preciso estar preparado para deixar partes do sistema para trás e ajustar a rota de acordo com o andamento, e isso deve ser feito de forma rápida. (VALENTE, 2020)

Coragem: Kent Beck definiu como coragem “Effective action in the face of fear”, essa definição é colocada com reforço para o seu significado em sinergia com os outros valores. (KENT BECK, 1999) É necessário coragem para ter as conversas que explicitam problemas organizacionais e de processos que reduzem a efetividade do time, é necessário coragem também para descartar soluções que não estão funcionando para procurar outras mais simples, é necessário coragem para aceitar os feedbacks e construir ações concretas em cima deles. (FOWLER, 2000)

Respeito: todos os outros valores se não possuem respeito como base não funcionam, respeito nesse significado remete-se a preocupação verdadeira com o projeto e com todos os membros da equipe de forma que todos entendem a importância de cada papel e ser humano como igualmente significativas, do contrário o papel não existiria e aquela pessoa não estaria lá. (K. BECK, 1999)

2.4.2 Princípios

O XP entende que valores somente são muito abstratos para guiar diretamente comportamentos, por isso complementa esses valores com princípios que buscam tornar esses valores mais práticos. Todos eles são pensados de forma que atendam tanto a demandas pessoais do time que desenvolve o projeto quanto ao sucesso do projeto, logo pensados com base nas necessidades do negócio.

Humanidade: o primeiro princípio é a humanidade, o XP entende que o desenvolvimento de software baseia-se em um pilar único de capital humano, então os processos necessários para construção de produto baseiam-se fundamentalmente em adaptá-lo a necessidades humanas. Um bom desenvolvedor para se ter um bom ambiente de trabalho precisa ter necessidades básicas cumpridas como por exemplo: segurança no emprego (medo de perder o emprego afeta significativamente a performance), possibilidade de evoluir suas habilidades crescendo como profissional, trabalhar por objetivos compartilhados, receber reconhecimento por seu trabalho, e por fim a habilidade da empatia, entender e ser entendido nunca em um sentido de estender a barreira profissional que um trabalho deve ter, inclusive por isso também se valoriza um limite de 40 horas de trabalho semanais sempre dentro do XP. (K. BECK, 1999)

Economicidade: esse princípio está conectado com a praticidade que o XP tenta trazer ao focar seus requisitos de negócio na essência da necessidade do cliente, para ser simples através de um design que priorize um processo incremental. Ao mesmo tempo o XP é uma metodologia que se caracteriza pela evolução incremental principalmente para projetos longos, de escopo aberto, dessa forma a prática também representa o forte valor econômico de um software tem quando ele possibilita opções de novas melhorias e funcionalidades para aquele projeto, ele não é limitado a um único propósito, existe ali continuidade. (K. BECK, 1999)

Benefício Mútuo: as atividades e decisões tomadas em um projeto devem sempre contemplar todas as partes interessadas, portanto ao mesmo tempo que um bom ambiente de trabalho é entregue (Humanidade) um produto com alto valor agregado é também entregue (Economicidade). (VALENTE, 2020) Esse princípio se aplica a práticas importantes, como por exemplo, assim como escrever testes automatizados

permite que um código mais limpo seja entregue a futuros desenvolvedores que o utilizarão ajudam também o próprio desenvolvedor a arquitetar suas próprias soluções. Ou também quando conceitos complexos são representados por metáforas coerentes faz com que o indivíduo consiga tanto ser melhor entendido pela equipe e conseguir apoio para as dificuldades técnicas quanto também a equipe a entender o código em questão. (K. BECK, 1999)

Auto-similaridade: esse princípio nos traz uma visão de cautela com os entendimentos de causa e efeito que fazemos, não é porque algo funcionou em uma situação que funcionará em outra. Pode parecer contraditório com a ideia de que crescimento e aprendizado contínuo ao longo de um projeto, pois de fato isso acontece, porém é sobre entender que cada situação é única, e as ações que partem de cada uma delas deve ser adaptada ao cenário. (K. BECK, 1999)

Melhoria contínua: o XP acredita que a perfeição não existe no desenvolvimento de software. Significa fazer o melhor que é possível hoje, buscando sempre como fazer melhor amanhã, e isso vale para as histórias, para o design e para o processo também. (K. BECK, 1999)

Diversidade: o XP acredita que um ambiente de conflito é essencial pois assim se criam oportunidades, duas ideias e pensamentos diferentes apresentam a possibilidade de análise, debate e escolha, e isso gera valor para equipe. É importante lembrar que o conflito aqui está associado ao embate de opiniões e ideias, sem respeito é impossível que essas interações aconteçam de forma produtiva e positiva. (K. BECK, 1999)

Reflexão: é essencial que equipes consigam avaliar se o trabalho que vem sendo feito está sendo produtivo ou não, ela se reflete nas reuniões após interações, Releases e outros momentos cíclicos destinados a esse processo. Porém a reflexão na metodologia se encaixa também em outro sentido, o princípio pode ser também levado adiante, não somente aplicado ao trabalho mas também em conversas despretensiosas não relacionadas ao desenvolvimento de software permitem acesso a diferentes formas de ver e lidar com o mundo, naturalmente as associações acontecerão através da criatividade. (K. BECK ,1999)

Flow: o XP acredita que o software deve estar em constante evolução e avaliação, essa construção contínua representa esse princípio do Flow, esse princípio se torna prática ao passo que a equipe valoriza uma frequência de deploy e builds diariamente, se possível inclusive várias vezes em um único dia. O desenvolvimento orientado a testes e a cobertura de testes automatizados são partes essenciais para garantir velocidade e confiança nesse processo. (K. BECK, 1999)

Oportunidade: problemas devem ser considerados como oportunidades para mudança dentro XP, deve-se fugir da ideia de sobrevivência e sim considerar soluções que realmente trarão aprendizados e evolução das pessoas que compõem a equipe e do processo, a cada desafio a equipe deve sair melhor. Esse crescimento contínuo se torna realidade através das práticas do XP, afinal todas elas tem um perfil de resolução dos problemas de forma definitiva e não paliativa. (K. BECK, 1999)

Redundância: redundância significa garantia. Sabemos que problemas difíceis dentro do desenvolvimento de software podem ter diversas soluções válidas possíveis, redundância significa ter mais de uma barreira de proteção para problemas que poderiam gerar desastres. Desastres corroem a confiança, Kent Beck nos dá a entender que o custo da implementação de soluções redundantes é pago com folga através da manutenção da confiança entre os membros da equipe. A redundância também pode ser vista na própria metodologia a partir do momento que tem diversas práticas que preveem um software de qualidade, podemos citar: desenvolvimento orientado a testes, proximidade valorizada com o consumidor, deploy diário, integração contínua ou a programação em par. (K. BECK, 1999)

Falha: o XP acredita que a falha é essencial para o processo de aprendizado, principalmente quando equipes estão tendo dificuldades de ter sucesso em uma implementação ou caminho a ser seguido. Esse princípio foge da ideia constantemente criticada do desperdício, pelo simples fato de que se a solução correta e o design adequado estivessem disponíveis eles já estariam sendo implementados e a conversa, a dúvida e o receio não existiriam. (K. BECK, 1999)

Qualidade: acredita-se que um projeto que prioriza qualidade também possui um tempo de entrega mais rápido, aumento na produtividade incremental e consequentemente consegue também cumprir mais prazos, um desafio até hoje recente no desenvolvimento de software. Para o XP negligenciar os processos de testes e as práticas que garantem um bom desenvolvimento para acelerar uma entrega resulta apenas na construção de débito técnico e um legado no código, esse legado aumenta de forma desproporcional ao tempo “adiantado”. (K. BECK, 1999)

Pequenos passos: realizar grandes mudanças de forma repentina é algo que pode ser tentador para muitos times, principalmente em momentos que demandam melhorias súbitas, momentos desafiadores. Existe uma pergunta que Kent Beck comenta ter feito constantemente, “Qual é o mínimo que você pode fazer que está sem dúvidas na direção correta?”, essa pergunta pode despertar preocupação, afinal dessa maneira o time poderia estar fadado à estagnação talvez, porém pequenos passos são formas de trazer também o foco para coisas possíveis, próximas e que tem impacto direto. Pequenos passos podem ser representados de diversas formas, como por exemplo pela priorização do teste antes do desenvolvimento, onde cada teste é feito de cada vez, a integração contínua que traz de horas em horas os testes e mudanças feitas no desenvolvimento do projeto. (ANDRES, 1999)

Responsabilidade aceita: dentro do XP responsabilidades não podem ser nomeadas, precisam ser aceitas. Esse princípio reflete a responsabilidade individual do desenvolvedor que se colocou de prontidão para tarefa e inclusive estimou o esforço necessário para a mesma. Responsabilidade também se compartilha autoridade e autonomia, a pessoa responsável pela história é também responsável pelo design, implementação e teste referentes a tarefa. (K. BECK, 1999)

2.4.3 Papéis

Testadores: papel que ajuda o consumidor a escolher e escrever testes funcionais automatizados que vão representar feedback instantâneo ao processo de desenvolvimento. (KENT BECK, 1999)

Gestores de projeto: funcionam como meio de campo para todo processo de

desenvolvimento, sendo o papel que mais tem potencial para mudar positivamente os processos de desenvolvimento, organizando e adaptando cada ciclo às necessidades da equipe. São responsáveis também por manter as partes interessadas de um projeto (clientes, programadores, executivos, gestor de produto) atualizadas e atua como facilitador da comunicação entre essas partes de forma a evitar também interações desnecessárias. (K. BECK, 1999)

Gestores do produto: gestores de produto têm o papel de direcionar todo processo de desenvolvimento, traduzindo as necessidades do negócio em histórias para organização e priorização das mesmas em Releases, interações e ciclos na esteira de desenvolvimento. Um plano dentro do XP representa o que pode acontecer e não necessariamente o que vai acontecer, então gestores de produto são responsáveis também por ajudar o time ajustando as histórias de forma que estejam em um escopo no qual seja possível se atingir valor e implementar funcionalidades no tempo disponível. (K. BECK, 1999)

Usuários: usuários no processo de desenvolvimento do XP quando existe um envolvimento real são responsáveis por tomar decisões ativas dentro do processo de desenvolvimento, através da escrita de testes funcionais com a equipe de testes e consegue fornecer feedbacks valiosos para o valor do projeto conforme as entregas acontecem. É importante que os usuários escolhidos para que se mantenha essa comunicação sejam pessoas que entendam sua responsabilidade e conheçam de plataformas similares a que está sendo desenvolvida pelo projeto. (K. BECK, 1999)

Desenvolvedores: desenvolvedores estimam histórias, transformam histórias em tarefas, automatizam processos de desenvolvimento, escrevem testes, desenvolvem funcionalidades e trabalhando em pares prezam por uma evolução contínua do design do sistema. (K. BECK, 1999)

Existem outros papéis definidos que podem ser encontrados no Livro Extreme Programming: Embrace Change. Os mesmos são: RH, Executivos, Escritores técnicos, Arquitetos, Designers de Iteração.

2.4.4 Definições gerais e práticas do Extreme Programming

Termo	Definição
Iteração	Período de tempo fixo, tipicamente entre 1 e 3 semanas. Ao fim de cada iteração novas funcionalidades em pleno funcionamento devem ser entregues ao consumidor.
Release	Um determinado número de interações forma uma Release. Tem duração de 1 até 3 meses, a razão dessa variação no período é devido a necessidade de entrega do software em produção nesse período, publicamente, ao cliente.
Histórias de usuário	Uma história de usuário é um pequeno parágrafo que define um requisito do sistema ou funcionalidade que deve ser entendido e acordado entre desenvolvedores, testers, consumidores. Uma história de usuário deve ser pequena o suficiente para que a cada iteração possa fazer múltiplas.
Testes de aceitação	Testes focados na regra de negócio proposta nas histórias.
Planning Game	Processo dentro do XP utilizado no planejamento onde são clarificados os requisitos necessários para o sistema e o time de desenvolvimento especifica quanto cada história custará de tempo da equipe e qual a capacidade produtiva disponível por dia/semana/mês.
Releases pequenas	Releases pequenas facilitam o feedback de usuários finais e partes interessadas do negócio.
Semana de 40 horas	Prática dentro do XP que representa a crença de que o trabalho além das 8 horas diárias a longo prazo afeta a qualidade do software, já que indivíduos cansados estão mais propensos a cometer erros.

(ANGELA MICHELLE MARTIN, 2009)

O Extreme Programming tem seu sucesso atribuído principalmente a forma como enfatiza a satisfação do cliente. Ao invés de entregar tudo que o cliente poderia em um futuro longínquo querer, o software foca em entregar somente o que o cliente precisa e como ele precisa. O Extreme Programming empodera seus desenvolvedores a mudanças de requisitos no projeto mesmo em etapas finais do desenvolvimento. (DON WELLS, 2013)

Outro pilar importante do sucesso do XP está na qualidade do projeto, na metodologia a mesma está diretamente ligada à forma rigorosa com que o XP lida com os testes, através de um conjunto de regras tais como:

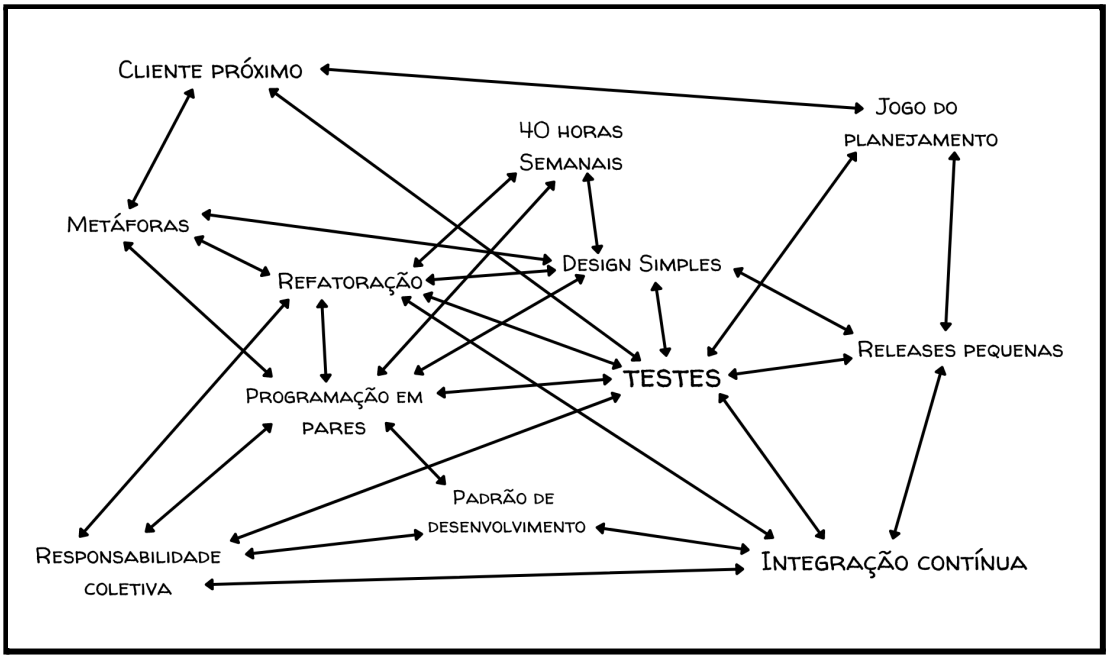
- Todos os códigos devem ter testes unitários;
- Todos os códigos devem ter seus testes unitários colocados a prova antes de ir para uma Release;
- Quando um Bug é encontrado, um teste automatizado deve ser criado antes do desenvolvimento do próprio Bug (acredita-se que um bug não é um erro na lógica mas sim um teste ainda não escrito);
- Testes de aceitação são rodados constantemente e seus resultados devem estar visíveis a todos. (KEN AUER, 2001)

No XP existem recomendações também específicas em relação ao ambiente de trabalho esperado, começando pelo tamanho da equipe, recomenda-se diretamente que a metodologia seja aplicada para equipes pequenas, de até 10 desenvolvedores. Além disso conta-se com um ambiente que preza por um espaço de trabalho informativo que traga as principais mensuráveis de um projeto de XP, no caso para os desenvolvedores poderia-se citar: histórias pendentes, em andamento e concluídas; quando para os executivos seria o número de Bugs em produção (não previstos) e o intervalo de tempo entre o desenvolvimento de um projeto e o retorno financeiro. (Marco Tulio Valente, 2020)

Na gestão de um projeto quatro variáveis devem ser levadas em consideração: custo, qualidade, tempo e escopo. O custo é uma das variáveis mais independentes dessa equação, qualidade como já mencionado anteriormente pode ser sacrificada para se ganhar velocidade, porém o efeito é temporário e o gestor pode esperar uma perda em velocidade logo depois. Tempo e escopo são as variáveis que mais se relacionam entre si, aumentando a abrangência de uma solução com poucos requisitos o tempo necessário pode aumentar de forma exponencial. O desenvolvimento de um software necessita um acordo entre a empresa e o cliente, o XP acredita que dessas quatro variáveis três devem ser estáticas: o custo, a qualidade e o tempo. O escopo manter-se variável resulta para o projeto uma proteção para os desenvolvedores, garante um envolvimento do cliente na evolução de um projeto, mantendo o mesmo sempre atrelado a resolução da dor central do cliente. (Martin Fowler, 2000)

Em suma o XP se fixa em um conjunto de valores, princípios e práticas. Os valores e princípios provêem o contexto para o método, porém é através das práticas que identificamos verdadeiramente e discutimos o XP. Kent Beck comenta que nenhuma das práticas do XP é completamente nova. Ao invés disso, a inovação desse método está na utilização das práticas em um nível extremo e garantindo que estejam sendo utilizadas da forma mais intensa o possível. (M. MARTIN, 2009)

Figura 6: Práticas do Extreme Programming



Fonte: O autor, adaptação e tradução feita em referência a Kent Beck (2000)

Dentre as principais práticas de programação podemos citar:

Desenvolvimento Orientado a Testes (TDD): prática que prevê que para cada código desenvolvido deve haver antes um teste automatizado desenvolvido para a mesma. O XP acredita que através dessa prática é possível prever se a funcionalidade tem um escopo incompleto, uma arquitetura clara e incentiva confiança entre membros da equipe. (K. BECK, 1999)

Integração contínua: a adição de modificações em código implementadas por desenvolvedores na versão que será colocada em testes é feita através dos pulls, ou builds, que integram novas alterações ao que já foi desenvolvido e testado. Tais integrações por interagirem muitas vezes com os mesmos trechos de código podem

gerar os chamados conflitos, conflitos em integração são impossíveis de serem completamente evitados, porém podem ser reduzidos. No XP prega-se uma ideia de integração contínua, que consiste em garantir que um código que está sendo produzido seja “buildado”, ou integrado em uma frequência diária, evitando por exemplo que desenvolvedores fiquem muito tempo isolados trabalhando em um mesmo trecho do código e que dessa forma façam as adaptações necessárias de forma rápida e contínua. (VALENTE, 2020)

Programação em pares: dentro do XP acredita-se que todas as tarefas que exijam codificação sejam feitas em pares. Dentro da metodologia existem dois papéis principais: o líder, que fica com o teclado e mouse, e ao outro desenvolvedor cabe a função de revisor e questionador do trabalho do líder. (Marco Tulio Valente, 2020) Os papéis variam e dessa forma a equipe consegue manter uma responsabilidade compartilhada e uma normalização das expectativas sobre o que significa qualidade de software. (K. BECK, 1999)

Design incremental: o XP acredita em histórias de usuário simples, da mesma forma a simplicidade se aplica ao design do sistema, a arquitetura proposta deve pensar sempre na forma mais simples de se chegar ao objetivo esperado. Conforme as histórias vão sendo validadas nas interações e Releases espera-se uma evolução tanto do entendimento das dores do cliente por trás da história quanto da maturidade da equipe técnica em relação a solução implementada, e essa se traduz em tarefas de refatoração, prática essencial para o design incremental. (VALENTE, 2020)

Outras práticas e artefatos podem ser encontradas no Livro Extreme Programming: Embrace Change. Tais práticas são: Whole Team, Informative Workspace, Energized Work, Sit together, Pairing and Personal Space, Stories, Weekly Cycle, Quarterly Cycle, Slack, Ten-minute Build, Release planning, Iteration planning, Testes de programador e Gráficos de Burndown.

2.4.5 Funcionamento e processos

Se o Scrum tem o início de seu processo de desenvolvimento partindo do preenchimento do Backlog de produto, no XP, esse processo inicia-se com o

levantamento das histórias de usuário, por preferência esse processo deve ser feito com envolvimento do cliente. De forma completa, para que o processo seja iniciado, as seguintes informações precisam ser organizadas:

- Uma lista de histórias prontas, escritas em conjunto com o cliente;
- Orçamento de esforço esperado por histórias em pontos ou horas;
- Tempo total disponível da equipe por iteração, ou seja, quantos pontos ou horas poderão ser alocados por iteração;
- Tempo de duração de uma iteração completa;
- Quantidade de interações de uma Release. (VALENTE, 2020)

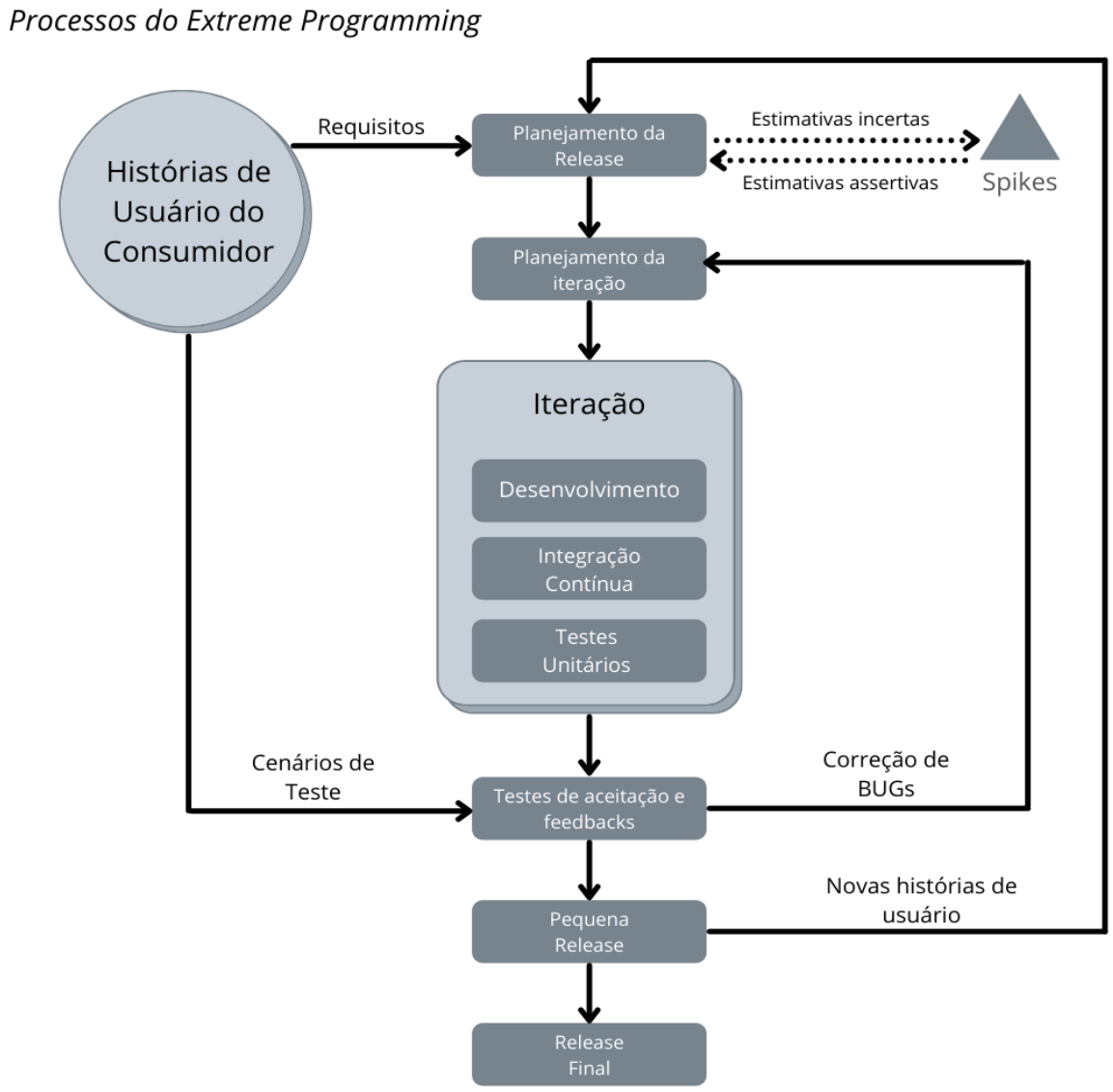
Tendo reunidas essas informações inicia-se o **planejamento da Release**, onde é feita uma priorização das histórias com influência direta do cliente, e onde serão escolhidas quais histórias serão implementadas nas iterações da próxima Release. Após a priorização feita, o representante do cliente e a equipe conseguem ter uma visão completa de como alocar cada história nas iterações referentes a Release em questão, dessa forma, respeitando os limites de horas ou pontos por iteração, alocam-se histórias por iteração, completando assim o planejamento de uma Release. (VALENTE, 2020)

Através do XP uma metodologia foi criado um conceito importante chamado Spike, trabalhado no planejamento de uma Release. Histórias ou tarefas definidas como Spike são atividades que não tem como objetivo entregar o código como no desenvolvimento comum do XP, o objetivo dessa abordagem é facilitar aos desenvolvedores a ter maior entendimento sobre uma história que é ou muito complexa ou muito longa. Dessa forma, esse estudo auxilia no processo de se estimar o custo de uma história, facilitando então aos gerentes e clientes no futuro encaixar o momento correto de desenvolvimento ou até mesmo descartar a possibilidade de se implementar aquela história, dado o resultado. (CHROMATIC, 2003)

Após a organização de histórias por iterações inicia-se o processo de **planejamento da iteração**, o objetivo dessa etapa é transformar as histórias em tarefas que estejam descritas em atividades de programação, com uma organização que priorize o aspecto técnico da implementação que será feita. Com as tarefas preparadas o time decide em conjunto qual desenvolvedor será alocado para cada

uma, após essa alocação a implementação tem seu início. (VALENTE, 2020)

Figura 7: Processos do Extreme Programming



Fonte: O autor

Uma iteração tem seu fim quando todas as histórias alocadas naquela iteração são implementadas e aprovadas pelo cliente, então além da validação feita pelos testes planejados necessita-se dessa aprovação para considerar uma iteração completa. (VALENTE, 2020) Após a iteração ser completa ela é colocada em uma pequena Release, dessa forma realimentando novas histórias de usuários para futuras Releases, com base em uma nova perspectiva adquirida da visão de parte do produto pronto, em funcionamento e aprovada. (FOWLER, 2000)

3 Metodologia

A metodologia será descrita com base nas duas etapas escolhidas, a primeira de ordem teórica trará como se deu a escolha dos materiais de estudo e quais serão os objetos de análise. A segunda etapa trará o contexto da empresa analisada com base na estrutura da equipe, da cultura da empresa e dos processos da mesma.

3.1 Análise comparativa

Para analisar as diferenças entre a qualidade do software desenvolvido por ambas as metodologias, deseja-se no trabalho realizar uma soma e síntese de estudos existentes sobre a utilização e efetividade de ambos os métodos, assim como a análise crítica do autor do trabalho. Ao contrário da contextualização teórica, que teve suas fontes centradas nos livros originários das metodologias, serão priorizadas referências com data de publicação recentes, de artigos e estudos que contenham uma visão mais atual e madura sobre a aplicação das metodologias.

Não é possível analisar todos os aspectos e interações possíveis das ferramentas oferecidas pelas metodologias em um único trabalho, então serão feitos recortes em partes chave de ambas, os recortes feitos trarão observações sobre os seguintes pontos:

- Análise crítica sobre a literatura de ambas as metodologias;
- O impacto das práticas de engenharia e na qualidade do software, focando principalmente no TDD (Test Driven Development) ou Desenvolvimento Orientado a Testes;

3.2 Estudo de caso

O estudo do caso foi feito levando em consideração conversas tidas com membros da equipe de desenvolvimento, antigos participantes da indústria e leva em consideração também a participação do autor do trabalho em cargo descrito abaixo.

3.2.1. Estrutura e equipe

A empresa em questão contava com um time da indústria de desenvolvimento de software de 4 desenvolvedores back-end, 4 desenvolvedores front-end, 3 testers, sendo 2 especializados em teste manual (caixa-preta) e 1 especializado em teste automatizado (caixa-branca) e também 1 analista de processos. O responsável pela gerência dividia seu trabalho como gerente de projetos e em parte como Product Owner, fazendo um papel fluido entre o que deveria ser o Scrum Master e o PO no Scrum e do Gestor de Projetos e de Produtos no XP. As responsabilidades atribuídas a cada pessoa eram as seguintes:

Líder back-end: responsável pela arquitetura do sistema a nível de back-end, guardava a responsabilidade técnica a respeito do back-end da aplicação, atuava como líder direto dos desenvolvedores back-end e cuidava das implantações on-premises (instalações feitas em servidores do cliente consumidor) nos clientes também e toda parte de comunicação entre a aplicação e os diferentes bancos de dados.

Líder front-end: responsável pela arquitetura do sistema a nível de front-end, responsável técnico pelo front-end da aplicação, atuava como líder direto dos programadores especialistas em front-end e prezava pela implementação de um software com boas práticas a respeito da experiência do usuário.

Programadores: desenvolvedores da equipe de front-end e desenvolvedores da equipe de back-end.

Testers (caixa-preta): testers que atuavam em testes de UI (User Interface), sem nenhuma ferramenta auxiliar de testes, em bases controladas eles simulavam a utilização real com referência direta das histórias de usuário construídas nas tarefas.

Tester (caixa-branca): um tester trabalhando em testes automatizados para as questões mais básicas do sistema, focado no desenvolvimento de testes de serviço, realizando rotinas que seriam rodadas sempre que uma nova funcionalidade entrasse em uma das branches de controle.

Analista de processos: responsável por auxiliar Product Owner assumindo parte dos

processos de interação com outras áreas, alimentando o Backlog com tarefas com uma categoria: Bugs. Esse levantamento era baseado em uma análise com resultado previsto e alinhado entre as partes, facilitando a priorização da tarefa pelo PO.

Product Owner/Gestor de projetos: responsável tanto pela questão de priorização das tarefas do Backlog para construção das Releases, trabalhava também escrevendo as histórias de usuário e escopos para implementação das novas funcionalidades e melhorias no sistema. Era responsável também por guiar o único processo cíclico que acontecia, que eram as Daily meetings.

3.2.2. Cultura

A cultura organizacional da empresa traz valores muito presentes na realidade de startups, provenientes de sua origem e perfil de seus fundadores. O ambiente contava com informalidade e pessoalidade, com clima leve de se trabalhar que se estendia a todas as áreas, a proximidade entre pessoas da equipe era grande, com bons relacionamentos interpessoais.

Ao mesmo tempo que a empresa sempre trazia uma urgência muito grande para o crescimento e performance, as oportunidades de ascensão profissional e recompensas sempre acompanharam muito bem os que mantinham o nível de dedicação esperado. Em termos práticos contava com plano de desenvolvimento individual e facilidade de contato entre diferentes níveis hierárquicos, principalmente para propostas de melhoria e sugestões de evolução de processos, regras, ou qualquer aspecto do funcionamento da empresa.

Acredito que a mentalidade das pessoas da empresa desde sua fundação foi muito global, os benchmarkings feitos eram sempre de concorrentes internacionais de peso, o referencial de comparação sempre foi bem desafiador. Ao mesmo tempo que o nível de excelência mínimo do software quando falamos em qualidade, performance, design e infraestrutura era sempre elevado. Por fim observei que haviam oportunidades de aprendizado no ecossistema local de empresas do ramo que estão próximas que poderiam ser aproveitadas mais do que de fato eram.

Mesmo que não possa trazer características mais específicas acerca da missão e visão da empresa, posso trazer aqui os valores, que são os seguintes: Paixão, inovação, união, liberdade, UX/UI, protagonismo, atenção aos detalhes e

diversidade. Esses valores foram levantados através de pesquisas feitas com funcionários de forma aberta, sendo priorizados de acordo com um método similar ao Treemapping, que utilizava de uma análise de repetição dos valores mencionados pela própria equipe.

3.2.3. Processos

Os processos da empresa foram divididos em blocos de acordo com as interações que a indústria de software tinha internamente entre sua equipe e o restante das áreas. Os processos descritos serão os seguintes:

A) Processos para identificação, lançamento e categorização dos BUGs

B) Processos para Gestão dos BUGs do tipo HOTFIX (rápida solução)

C) Processos para demandas de tarefas para evolução do produto

D) Processos de Testes e Validação em QA (Quality Assurance)

E) Processos para atualização das Releases em produção

Antes de entrar nos pontos mais específicos os seguintes pontos da metodologia podem ser especificados:

- A plataforma utilizada para lançamento e gestão das tarefas foi o ClickUp;
- O termo DEV representa o ambiente (branch) imediata após a finalização do desenvolvimento, a menos estável e controlada, assim que novos códigos são adicionados vão primeiramente para lá para então serem testados;
- As cores representadas a seguir nos fluxogramas representam os status que utilizamos na plataforma para organização e mostra para cada situação em que status cada uma das tarefas se encontraria no momento, foi construído dessa forma para ser mais didático para o desenvolvedor, sendo os status então:
    a. Azul: tarefa a fazer;
    b. Vermelho: tarefa em desenvolvimento;
    c. Amarelo: tarefa em teste;
    d. Laranja: tarefa com BUGs encontrados na primeira etapa de testes (DEV);

e. Verde claro: tarefa validada na primeira etapa de testes (DEV);

f. Rosa: tarefa em QA;

g. Marrom: tarefa com BUGs encontrados na segunda etapa de testes (QA);

h. Azul escuro: tarefa validada na segunda etapa de testes (QA);

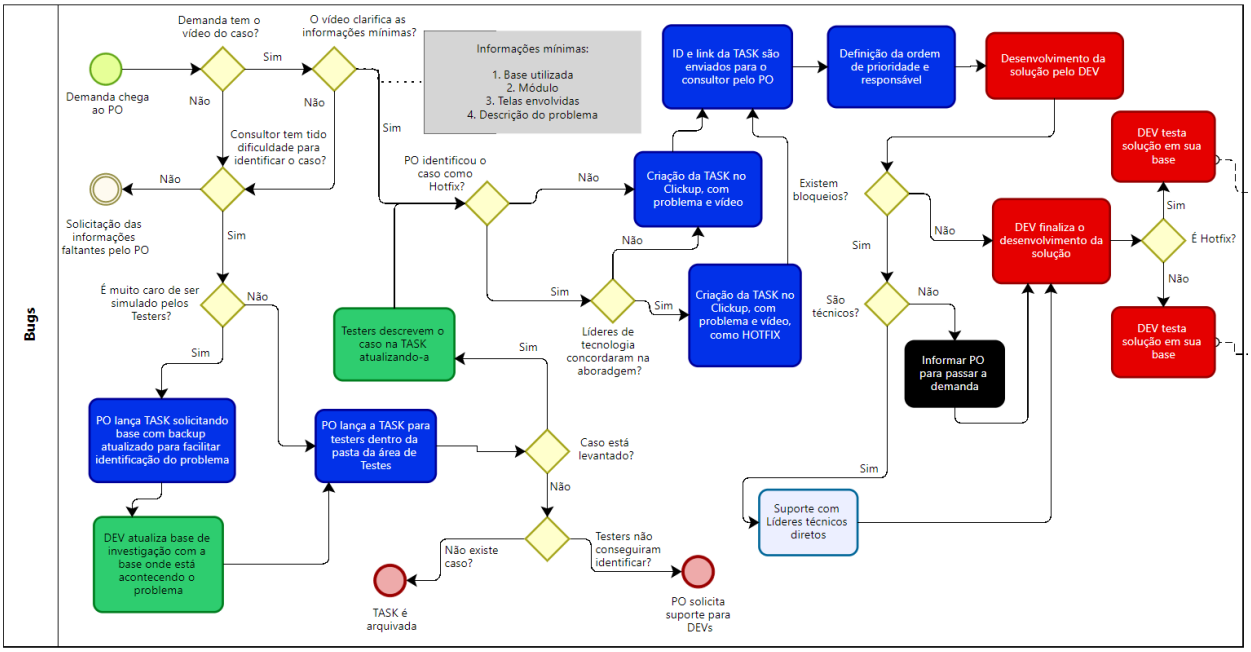
i. Roxo: tarefa colocada em Release de estabilização;

j. Cinza: tarefa com BUGs encontrados na Release de estabilização;

k. Verde: tarefa completamente validada e colocada em produção.

A) Processos para identificação, lançamento e categorização dos BUGs

Figura 8: Processos para identificação, lançamento e categorização dos BUGs



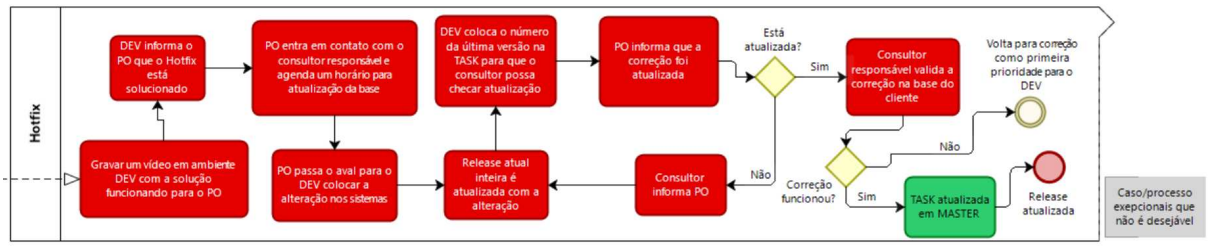
Fonte: O autor

Um dos maiores desafios da indústria da empresa analisada era na diminuição do tempo gasto na análise das solicitações de correção por BUG. A plataforma em questão continha configurações possíveis infinitas, se tratava de um software muito flexível e por isso o crivo entre uma demanda se tratar de uma configuração equivocada do usuário final ou a quebra de uma regra de negócio muitas vezes não eram imediatamente claras. O tempo de análise para tal ficava envolto no tempo dos desenvolvedores, que deveria ser completamente aplicado às Releases.

Os crivos e análises dos problemas são feitos então pela Analista de Processos, garantindo tempo útil dos desenvolvedores. Vale ressaltar também a análise dupla para definição de Bugs que requerem solução rápida, ou como chamados: “Hotfix”; Os mesmos passavam por duas análises: uma visão de prioridades pensando no impacto nos clientes finais (geralmente os Bugs quebram uma regra de negócio prevista) e futuramente uma outra análise pensando no impacto técnico de uma alteração em código que seria feita de forma imediata, ou seja, sem os processos completos de teste previstos, principalmente pensando já nos impactos da solução necessária.

B) Processos para Gestão dos BUGs do tipo HOTFIX (rápida solução)

Figura 9: Processos para Gestão dos BUGs do tipo HOTFIX



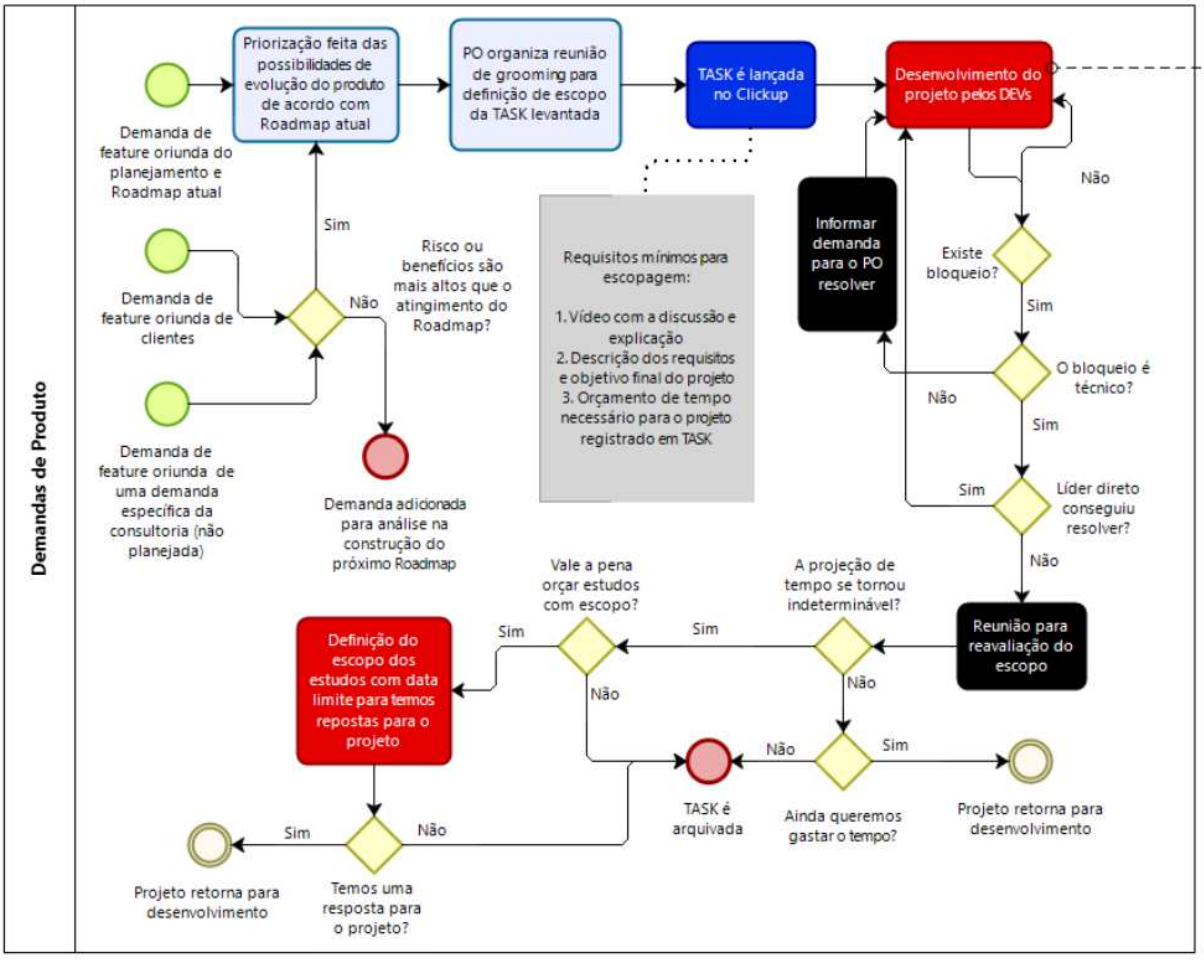
Fonte: O autor

Os BUGs de Hotfix como já dito são BUGs que não poderiam esperar todo fluxo de validação e testes da esteira de desenvolvimento, dessa forma os processos desenvolvidos nesse caso se destacam pelo aumento da proximidade do consultor ou cliente com a indústria para rápida identificação e acompanhamento. Dessa forma para mitigar os riscos desse tipo de abordagem adota-se um processo de validação da solução entregue de prontidão, e com a homologação validada diretamente pelo consultor ou cliente que trouxeram a demanda, a Release de produção é atualizada com as alterações feitas.

Existe também uma abordagem de resolução do problema em termos de arquitetura do software que procura trazer a alteração mínima em código para aquele resultado esperado, de forma que se possa planejar para o futuro uma resolução definitiva para o problema, uma que possa inclusive englobar outros possíveis casos do BUG. Essas situações tem um resultado negativo esperado e só existem pois nesse caso existe um valor muito alto agregado na velocidade da entrega da correção.

C) Processos para demandas de tarefas para evolução do produto

Figura 10: Processos para demandas de tarefas para evolução do produto



Fonte: O autor

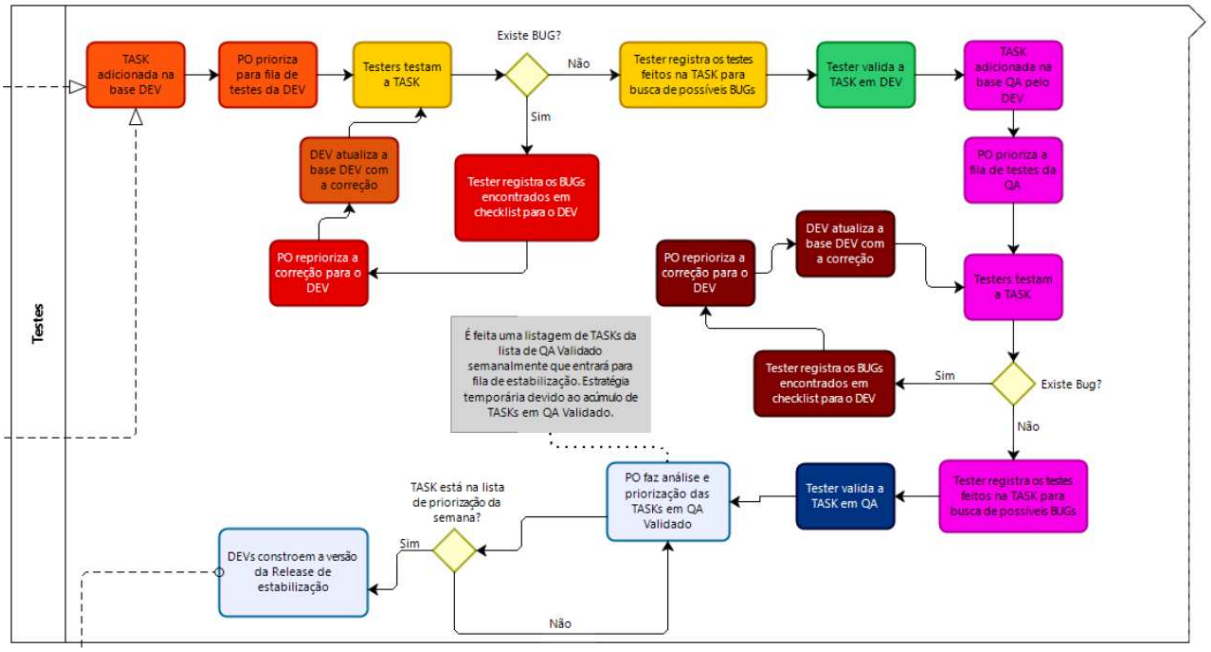
No início desse processo conseguimos notar as três principais origens de tarefas que compõe o Backlog de Produto da empresa: tarefas incluídas dentro do caminho de desenvolvimento do produto compondo o Roadmap de produto, demandas de novas funcionalidades acordadas com clientes, demandas que vinham de orientação do time interno de projetos (consultores). O primeiro ponto a ser ressaltado aqui é o crivo inicial entre essas tarefas, essa averiguação é feita pelo PO e nesse momento precisam ser avaliados riscos e benefícios de um desvio do planejamento (Roadmap de produto).

Não estão colocados Bugs nessa lógica pois os mesmos possuem um tratamento particular para priorização.

O segundo ponto a ser ressaltado nesse processo é a tratativa existente para bloqueios técnicos, a empresa entende que as estimativas feitas com base no estudo inicial da tarefa tem uma margem de erro esperada. Essa tratativa faz com que quando o nível de complexidade de uma tarefa se distancie consideravelmente do avanço atingido em seu desenvolvimento de forma contínua é preciso reavaliar seu custo, o custo de uma tarefa considerado nesta análise é o tempo de trabalho gasto pelos desenvolvedores. Após reavaliação desse custo há dois destinos possíveis, o retorno do projeto para desenvolvimento (em alguns casos com simplificação do escopo da tarefa, buscando atingir um Produto viável mínimo) ou então arquivamento dessa tarefa que significa uma pausa, então aquela entrega é descartada para o planejamento da próxima Release.

D) Processos de Testes e Validação em QA

Figura 11: Processos de Testes e Validação em QA



Fonte: O autor

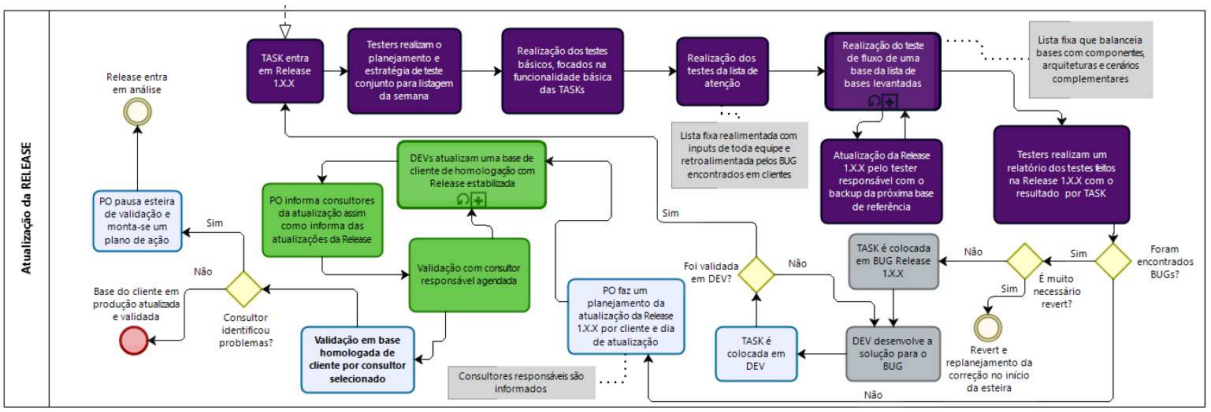
Para garantir maior segurança ao cliente final existe uma etapa de testes constituída de duas etapas, logo existem duas referências de “galhos” ou “branches”, da definição para o universo de software, branch seria um galho da árvore de desenvolvimento. É uma cópia do código derivado de um certo ponto do trunk (tronco)

que é utilizado para a aplicação de mudanças no código, preservando a integridade do código no trunk (tronco) principal (versão Release de produção).

Esses blocos da versão que vão passando por etapas de validação até chegarem na produção justamente para evitar que uma versão que chegue para o cliente final contenha erros. O fluxo de testes tem sua priorização proporcionalmente de acordo com a prioridade das tarefas entregues na Release, os testes feitos nessa etapa são puramente testes humanos, então testes de UI (User Interface).

E) Processos para atualização das Releases em produção

Figura 12: Processos para atualização das Releases em produção



Fonte: O autor

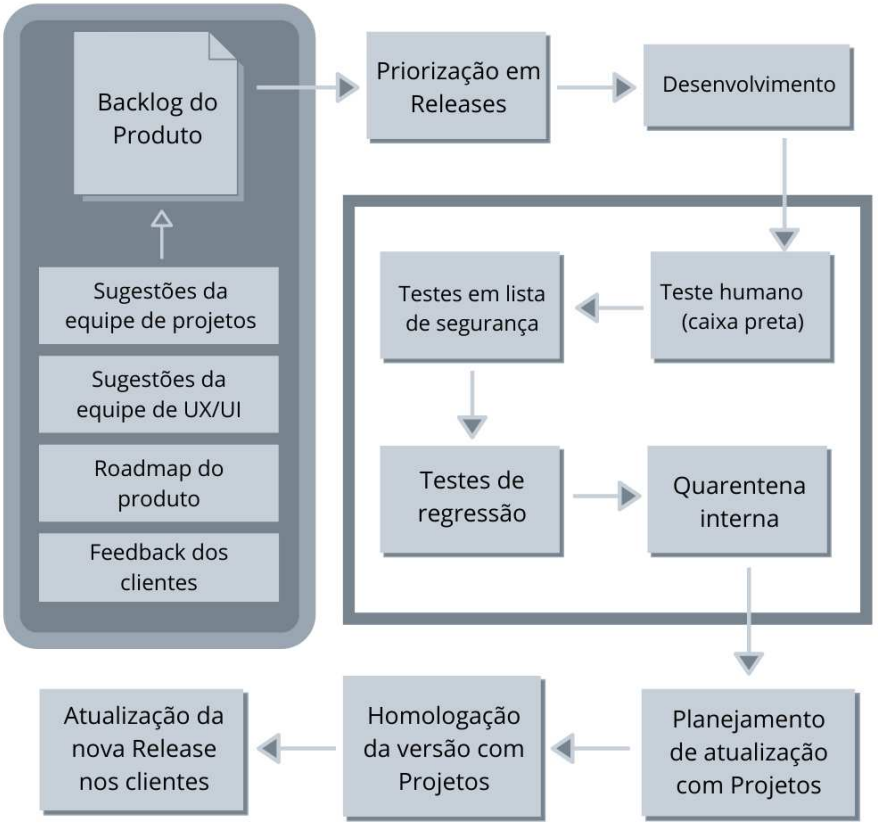
O processo de atualização das versões da aplicação no servidor de um cliente ou na base em nuvem de um cliente é um processo que necessita da maior cautela possível, logo aqui existem etapas de testes que foram colocadas de forma adicional dentro do processo. Os testes feitos nessa etapa são também humanos, porém para evitar redundância nos testes funcionais já feitos em DEV e em QA a empresa adotou os seguintes procedimentos para reforçar seus processos de teste:

1. Realização dos testes da lista de segurança: casos de teste manuais construídos de forma a explorar os erros mais recorrentes da plataforma, especialmente e de forma mais rígida para os que acontecem e são notificados por clientes.

2. A segunda são os testes de regressão, nessa etapa organiza-se aplicações reais da equipe interna, projetos já prontos que rodam em produção são colocados através de um backup em base controlada e com a versão do código com a branch que está em estabilização, em seguida essas aplicações são utilizadas simulando uma utilização intensa e o mais próximo o possível de um usuário real. A escolha de quais projetos vão ser escolhidos para realização dos testes de regressão é feita através de uma análise que leva em consideração uma combinação que traga a maior variedade de combinação da utilização dos componentes e funcionalidades da ferramenta.

Os processos como descritos em tópicos dão uma perspectiva específica dos blocos, porém sob uma análise geral da esteira de desenvolvimento podemos representar o processo completo da seguinte forma simplificada:

Figura 13: Fluxo da esteira de desenvolvimento da empresa analisada



Fluxo da esteira de desenvolvimento da empresa analisada (simplificado)

Fonte: O autor

Os processos de testes para Bugs de correção rápida foram os primeiros que utilizaram de uma aproximação com os consultores e clientes como processo de homologação para feedback rápido sobre o resultado da implementação feita no código. O segundo processo que utilizou dessa aproximação com cliente foi na homologação de Releases feitas pela equipe de projetos e pelo cliente final e os últimos os testes feitos em lista de segurança e os testes de regressão.

Os testes feitos em lista de segurança representam uma estratégia que consistia em construir uma sequência de passos fixos antes da entrada de uma Release em produção que garantisse que as principais regras de negócio, ou que regras que apresentassem para aquela versão um possível risco previsto por alterações feitas fossem testadas através de um teste de UI (caixa-preta) garantindo que os pontos chave do produto estejam seguros na próxima versão. Os testes de regressão por definição representam uma técnica que busca evitar que partes do programa desenvolvido já testadas apresentem erros, a empresa construiu processos de testes de regressão incomuns, ainda eram feitos via ação humana, testes de UI, feitos através de aplicações construídas para que uma utilização simples de 45-60 minutos pela equipe de testes pudessem testar uma taxa maior que 90% das funcionalidades que o programa oferecia, dessa forma tendo os comportamentos e resultados esperados era possível identificar falhas indiretas principalmente (pensando que os testes diretos das funcionalidades da Release já haveriam sido testados a essa altura).

A empresa utilizava do programa para gestão interna de seus processos e utilizava desse fator como ferramenta de testes para seu software. A equipe quase que integralmente entende o funcionamento de forma avançada, e dessa forma antes de uma versão chegar para o cliente final a Release era atualizada para a própria equipe usar, então setores não ligados a indústria, como comercial, projetos, design, entre outros… Essa equipe então é tratada nessa situação como clientes com apenas os bônus dessa relação para equipe da indústria. Nesse momento de quarentena onde a versão nova está somente com a equipe interna, a indústria consegue então ter um feedback rápido sobre as funcionalidades trazidas e também sobre possíveis erros que estejam ocorrendo na versão, tendo tempo hábil para corrigir antes da versão chegar para os clientes.

4 Resultados e Discussões

Assim como na metodologia foi feita uma divisão da análise em dois blocos: análise comparativa teórica e estudo do caso; Também avaliamos aqui os resultados em dois blocos: um primeiro seriam os resultados obtidos da análise teórica ressaltando diferentes características de ambas as metodologias no sucesso de uma operação de desenvolvimento e em um segundo para avaliação da implementação feita pela empresa das metodologias de gestão ágil em seus processos e os resultados obtidos.

4.1 Análise das diferentes características das metodologias de gestão ágil

Uma das primeiras características de ambas as literaturas do Scrum e do XP quando analisadas são acerca da estrutura que cada uma delas traz. O Scrum através de livros como Agile Project Management with Scrum (Ken Schwaber) ou Essential Scrum: A Practical Guide to the Most Popular Agile Process (Kenneth S. Rubin) exemplificam uma abordagem de literatura e orientação mais rígida no que deve ser feito processualmente para que a metodologia funcione bem. Essa visão e análise da literatura é também compartilhada por uma palestra de Dave Thomas chamada Agile is Dead da Goto; Conference em 2015, autor do livro O Programador Pragmático e um dos programadores que assinou o manifesto ágil em 2001.

Em contraste a essa posição, a literatura e linguagem do XP conseguem dar também enfoque em outros aspectos da gestão além dos processos gerenciais e na estrutura das reuniões como o Scrum faz. (M. MARTIN, 2009) Um deles é através dos princípios colocados, forma escolhida de tangibilizar os valores em comportamentos. Ao mesmo tempo, onde o XP mais se preocupa em trazer detalhes e direcionamentos são nas práticas de engenharia trazidas, sendo elas:

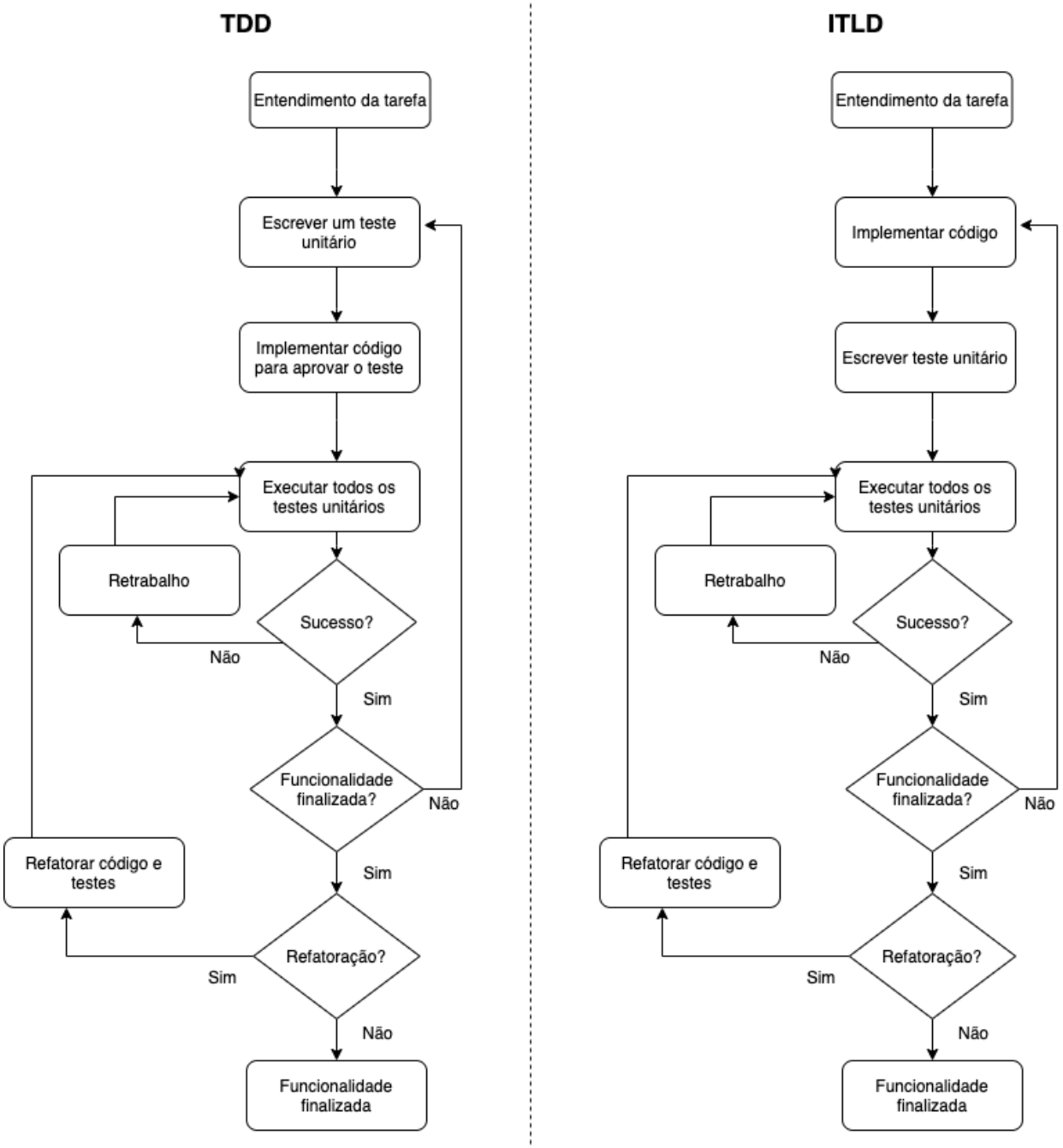
- Programação em par;
- Desenvolvimento orientado a testes;
- Integração contínua;
- Refatoração;

- Testes de aceitação;
- Releases pequenas.

Segundo Dave Thomas, o Scrum e sua literatura ao longo dos anos pode ter seguido esse caminho devido ao viés comercial que o movimento “Agile” tomou. Entende-se que as metodologias provenientes dessa visão (Scrum, XP, Kanban, Lean, entre outras) foram utilizadas cada vez mais como um produto que alimentou um mercado de livros, conferências e treinamentos e que o Scrum foi uma das metodologias que mais recebeu essa influência e consequentemente se tornou a cara desse processo de transformação que migrou a gestão de projetos de desenvolvimento de software do modelo tradicional cascata para uma utilização quase que unânime das metodologias ágeis atualmente, como já mostrado. Então entende-se, segundo Thomas, que um modelo que simplifica o entendimento dos conceitos por trás da gestão por metodologias ágeis a passos simples é mais fácil de ser vendido e replicado. (Dave Thomas; Agile is Dead da Goto; Conference em 2015)

Uma das implicações da rigidez e completude das práticas de engenharias trazidas pelo XP em relação ao Scrum pode ser a qualidade do software desenvolvido por ambas as metodologias. O Scrum utiliza o chamado ITLD, ou Desenvolvimento de Teste por Iterações (aqui testa-se ao final do desenvolvimento), e o XP utiliza o TDD, ou Desenvolvimento Orientado a Testes. A prática do TDD, originária do XP, preza por uma abordagem cíclica ao desenvolvimento dos testes unitários de forma que esses são escritos antes da solução. (Davide Fucci, 2016). Essa prática tem como objetivo a entrega de uma maior qualidade no software e produtividade para os programadores. (KARAC, 2018)

Figura 14: Representação dos processos de TDD e ITLD



Fonte: Baculi G. e Vilela P. (2021)

Dadas as promessas da aplicação dessa prática, que hoje é amplamente utilizada não somente em abordagens diretamente ligadas ao XP, há cerca de 20 anos diferentes estudos tentam questionar e validar empiricamente a efetividade da mesma. Foram feitas análises nesse estudo para tentar sintetizar esse conhecimento construído ao longo dos anos, as análises levam em consideração três variáveis de impacto pela aplicação do TDD:

- Qualidade externa: na literatura é geralmente considerada como a quantidade de defeitos encontrados pelos testes executados no estudo. (SANTOS, 2018)
- Qualidade interna: refere-se a facilidade de se integrar novas partes e funcionalidades em um projeto. Geralmente associadas a fatores como: esforço gasto em manutenção, flexibilidade de alteração de partes antigas do projeto, clareza do código, entre outros. (FREEMAN, 2010)
- Produtividade: produtividade pode ser definida como a razão de uma unidade de saída (produção) para uma de unidade de entrada usada para produzir a saída. (IEEE Standard for Software Productivity Metrics, 1993). Simplificando, no desenvolvimento de software especificamente, a unidade produzida geralmente tem dois tipos de características, ou é orientada a quantidade de linhas produzidas ou a quantidade de requisitos entregues.

Tabela 1: Estudos em formato de síntese analisados

Estudos em formato de síntese			
Estudo	Autor	Ano	Resultado
Test-Driven Development: uma revisão sistemática	Gustavo Baculi Benato e Plínio Roberto Souza Vilela	2021	Qualidade externa: melhorou Qualidade interna: inconclusivo Produtividade: inconclusivo
The Effects of Test Driven Development on Internal Quality, External Quality and Productivity: A systematic review	Wilson Bissi, Adolfo Gustavo Serra Seca Neto, Maria Claudia Figueiredo e Pereira Emer	2016	Qualidade externa: melhorou Qualidade interna: melhorou Produtividade: melhorou
Does the performance of TDD hold across software companies and premises? A group of industrial experiments on TDD	Adrian Santos, Janne Järvinen, Jari Partanen, Markku Oivo e Natalia Juristo	2018	Produtividade: ligeiramente pior ou inconclusivo
What Do We (Really) Know about Test-Driven Development?	Itir Karac e Burak Turhan	2018	Inconclusivo

Fonte: O autor

Tabela 2: Experimentos individuais analisados

Experimentos individuais					
Estudo	Autor	Tipo do documento	Duração do Experimento	Ano	Resultado
Analysis of the Impact of Test Based Development Techniques (TDD, BDD, and ATDD) to the Software Life Cycle	Luis Alberto Cisneros Gómez	Tese de mestrado	1 mês	2018	Qualidade externa: melhorou Qualidade interna: melhorou Produtividade: melhorou
Impact of Process Conformance on the Effects of Test-driven Development	Davide Fucci, Burak Turhan e Markku Oivo	Artigo	3 horas	2014	Inconclusivo para todas as variáveis; notou-se apenas aumento no número de testes desenvolvidos
A Longitudinal Cohort Study on the Retainment of Test-Driven Development	Simone Romano, Danilo Caivano, Giuseppe Scanniello, Davide Fucci, Burak Thuran, Maria Teresa Baldassarre e Natalia Juristo	Artigo	5 meses	2018	Qualidade externa: inconclusivo Qualidade interna: inconclusivo Produtividade: inconclusivo
A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last?	Hakan Erdogmus, Burak Turhan, Markku Oivo, Natalia Juristo e Davide Fucci	Artigo	5 horas (por tarefa analisada, processo não contínuo)	2016	Qualidade externa: melhorou Qualidade interna: melhorou Produtividade: melhorou

Fonte: O autor

Apesar dos estudos terem em sua maior parte se colocado de forma positiva em relação aos impactos da implementação do TDD, ainda existem críticas sobre a proposta colocada por Kent Beck em 2003. Porém da mesma forma existem pouquíssimos estudos e artigos que trazem a proposta de ITLD como vantajosa em relação a metodologia do TDD, para quaisquer das variáveis analisadas.

De fato é um campo de estudo empírico complexo e de difícil padronização, que conta cada vez mais com técnicas estatísticas para avaliar as lacunas deixadas pela análise dos pesquisadores de engenharia de software. (BACULI, 2021) Dos estudos mencionados nesse trabalho considerados inconclusivos nota-se em diversos

estudos um padrão de sugestão ao final dos estudo por uma análise de longo prazo que contenha objetivos semelhantes aos das pesquisas de avaliação do TDD nos critérios avaliados aqui.

Softwares como já comentado no trabalho são atividades de complexidade únicas, como produto de uma ação puramente intelectual com alto impacto pelos indivíduos que compõem sua produção. O desenvolvimento de um software em todos os seus aspectos recebe influência do tempo, podendo-se dizer que envelhece conforme mais funcionalidades e linhas de código são adicionadas em um projeto. (VALENTE, 2020) As duas primeiras Leis de Lehman ilustram bem esse conceito para nós:

- Mudança contínua: um sistema deve ser continuamente adaptado ou do contrário seu uso será progressivamente menos satisfatório. (LEHMAN, 2001)

- Complexidade incremental: à medida que um sistema sofre alterações sua complexidade cresce e sua evolução se torna dificultada, a não ser que um trabalho seja feito para contrapor essa tendência. (LEHMAN, 2001)

Projetos de longo prazo acabam sofrendo maior influência desse aumento de complexidade, e principalmente por esse motivo é onde se conseguiria ver com mais clareza os impactos de uma ação de contraposição ao aumento de complexidade. A contraposição a complexidade mencionada pela segunda lei pode ser feita através de atividades como a refatoração.

Refatoração pode ser definida como “uma mudança feita na estrutura interna de um software de forma a tornar sua compreensão mais fácil e sua modificação menos custosa sem alterar seu comportamento externo observável”. (Martin Fowler, 2018) Essa definição está no livro “Improving Design of Existing Code” escrito por um dos autores do Extreme Programming e pode ser considerado o principal livro sobre refatoração da área desde sua primeira versão em 1999. É caracterizado principalmente por um processo de refatoração que fornece processos e orientações para que coberturas de teste acompanhem a atividade de forma segura, além de estar conectado com princípios do XP como Pequenos Passos e práticas como Integração Contínua e o próprio TDD.

Das críticas analisadas ao processo de Desenvolvimento Orientado a Testes

avaliadas nesse trabalho existem duas visões que podem ser ressaltadas, no artigo “A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last?” Comenta-se que houve uma evolução nos aspectos analisados em termos de qualidade e produtividade, porém tal evolução foi atribuída a outro fator, o principal mencionado em sua análise foi devido aos ciclos curtos de desenvolvimento e integração que seu processo valorizava. Um outro estudo mencionado, “A Longitudinal Cohort Study on the Retainment of Test-Driven Development”, apesar de ter tido resultados inconclusivos apresentou que os grupos que utilizavam TDD possuíam um tempo de refatoração significativamente inferior, facilitando um processo de alterações incrementais, fato que motivou a equipe a propor ao final do estudo sugestões de análises de longo prazo sobre a evolução da qualidade interna de casos de utilização do TDD como metodologia.

Os artigos em sua maioria executam seus experimentos em um curto período de tempo, apesar de ter sido encontrado um exemplo com tempo de análise de 5 meses acredito que possam ocorrer variações significativas com a evolução contínua na escala de anos. O estudo de caso mencionado neste trabalho mostra a realidade de uma empresa que contém um único produto, dessa forma o software em questão está quase a 3 anos sendo alterado, e continuará sendo alterado da mesma maneira por muitos outros anos.

Esses estudos e conclusões reforçam as recomendações existentes para equipes que pretendem utilizar XP. Se uma cobertura de testes mais completa e ciclos menores fornecem maior flexibilidade é seguro dizer que o Extreme Programming tem suas vantagens mais notadas em ambientes onde existem uma grande quantidade de alterações propostas para o escopo, assim como novas funcionalidades. Ao mesmo tempo, a metodologia tem seu diferencial dentro dos processos de refatoração, pode-se dizer que possui mais vantagem também na contraposição mencionada para segunda Lei de Lehman, então para projetos de longo prazo.

Essas características de um projeto se colocam de forma análoga quando pensamos no processo de desenvolvimento através do Scrum. Principalmente pelo fato de que a metodologia possui ciclos de desenvolvimento maiores (30 dias) em relação ao XP (1-2 semanas) e não preza pelo investimento em um processo de desenvolvimento de testes contínuo, que principalmente quando em escopos de

análise menores pode significar ocasionar em uma percepção de aumento de produtividade. (SANTANA, 2019)

4.2 Análise da implementação das metodologias no estudo de caso

Segundo a própria empresa não existe uma metodologia única que se coloca como representante do que é aplicado, existem ferramentas e entendimentos tanto da metodologia do Scrum quanto do XP dentro de seus processos, apesar do XP ser a mais almejada como referência final.

Um primeiro ponto avaliado sobre os processos da empresa em questão é que não existem processos formais de revisão além da reunião diária que é feita entre toda equipe da indústria. Esse único momento de revisão é pautado nas três perguntas base do Scrum e tem seu impacto mais voltado para o foco da equipe nas tarefas da Release e na tratativa de bloqueios diários. Não ter processos de revisão de entrega de Releases estruturado, nem um momento de avaliação rotineiro da aplicação com os clientes faz com que o processo não tenha uma entrada cíclica de novas funcionalidades no Backlog de produto, e também não permite momentos formais de reflexão, feedback e evolução dos processos pela equipe como um todo.

A empresa por mais que já tenha um tester automatizado (desenvolvedor de testes de serviço) o mesmo entrou de forma tardia na equipe, foi o último profissional contratado, por esse fator ainda está passando por uma fase de adaptação para entender a arquitetura do software, as regras de negócio do programa para então dar início ao desenvolvimento dos testes de serviço. Por mais que o tester especializado não esteja atuando em testes que iriam compor os testes de serviço, a equipe de desenvolvimento poderia já estar trabalhando com os testes unitários.

Testes de serviço compõem o que Martin Fowler (um dos autores do XP) já citou como Subcutaneous Test (Testes subcutâneos) os testes que operam logo abaixo da interface de usuário. Os mesmos são focados em uma necessidade de um teste de ponta a ponta que necessite simular um comportamento do usuário final que

expõe uma regra de negócio da aplicação, tais testes são feitos de forma automatizada. Segundo Sean Stolberg em seu artigo Enabling Agile Testing Through Continuous Integration os testes subcutâneos podem ser aplicados com sucesso em sua análise como testes de aceitação focados em um processo de regressão através da ferramenta NUnit. Ferramenta compatível com uma arquitetura Java e Javascript do sistema e condizente com demanda que é característica da empresa analisada no estudo de caso.

Os testes unitários por definição podem ser descritos como:

> *Um teste de unidade é um pedaço de código escrito por um desenvolvedor que contempla uma área muito pequena e específica de funcionalidade do código que está sendo testado.*
>
> Pragmatic Unit Testing in Java with JUnit, Andrew Hunt and David Thomas (2015)

Testes unitários são considerados dentro do Extreme Programming como um dos principais pilares da metodologia. São testes feitos em classes específicas e tem escopo focados apenas no resultado esperado para as classes em questão do sistema, a ideia é que todas as classes do sistema estejam com testes unitários que assegurem o pleno funcionamento de todas as partes do sistema e forneça alerta rápido para quaisquer alterações trazidas em novas Releases. Não só isso, mas o XP acredita que os testes unitários devem ser desenvolvidos antes de qualquer código, melhorando assim a arquitetura do sistema, a qualidade final e a flexibilidade para refatoramento. (K. BECK, 1999)

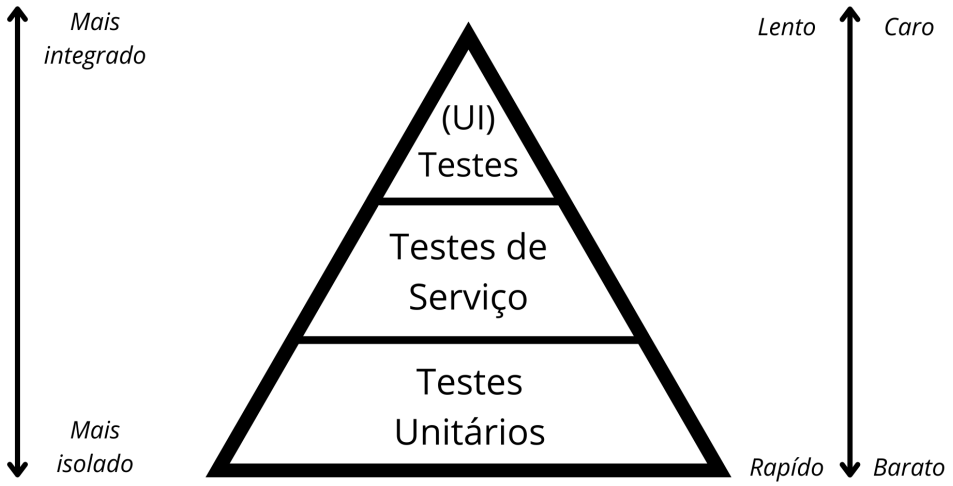
Pode-se dizer que esses pontos de falha em testes unitários e de serviço representam o maior distanciamento entre os processos da empresa e a metodologia principal pela a qual ela se orienta, o Extreme Programming. Avaliando as causas juntamente com a equipe pode-se dizer também que o principal fator que leva a esse distanciamento sejam os seguintes:

- Prazos apertados de entrega que continuamente estimulam a criação de débito técnico e legado em código para futuras refatorações e correções;
- Entrada tardia do profissional de testes automatizados, causando maior tempo necessário para adaptação aos códigos já desenvolvidos;

- Mudança de cultura dos desenvolvedores para desenvolvimento dos testes unitários antes do desenvolvimento das funcionalidades.

Sob uma perspectiva da Pirâmide de Cohn, existe uma proporção direta entre necessidade de integração e velocidade dos testes, sendo que integração significa também maior esforço envolvido e consequentemente maior investimento. Essa proporção se dá de forma que os testes de interface são os que representam maior custo e menor velocidade e os testes unitários o menor custo e maior velocidade. (SPINELLIS, 2018)

Figura 15: Adaptação da pirâmide de testes de Cohn



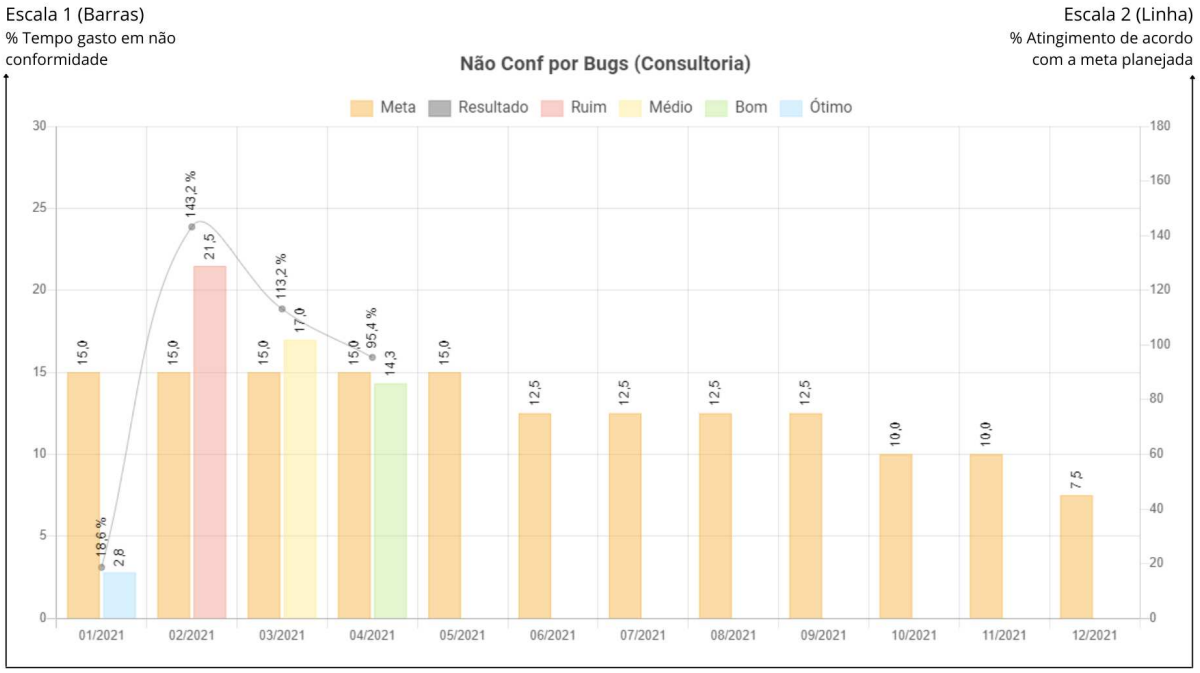
Fonte: o Autor

A empresa tem atualmente todos os seus esforços reunidos em testes de interface, não tendo então um dos pilares do XP contemplados e ao mesmo tempo utilizando o caminho de maior esforço e menor retorno de acordo com o conceito. Os testes de interface representavam um caminho de rápida implementação pela falta de um profissional especializado nos primeiros momentos de evolução do produto e se tornaram ao longo do tempo o processo único para preencher toda a lacuna do processo de testes da esteira de desenvolvimento.

Uma das prioridades da empresa no final do ano passado era a melhoria da qualidade de seu software, reduzindo as falhas não previstas, dessa forma as evoluções nos processos de testes foram tomadas através do aumento da

proximidade com o cliente e com a equipe interna que utiliza o software através dos processos mencionados anteriormente referentes aos Bugs de correção rápida, dos testes de regressão e também dos testes por lista de segurança.

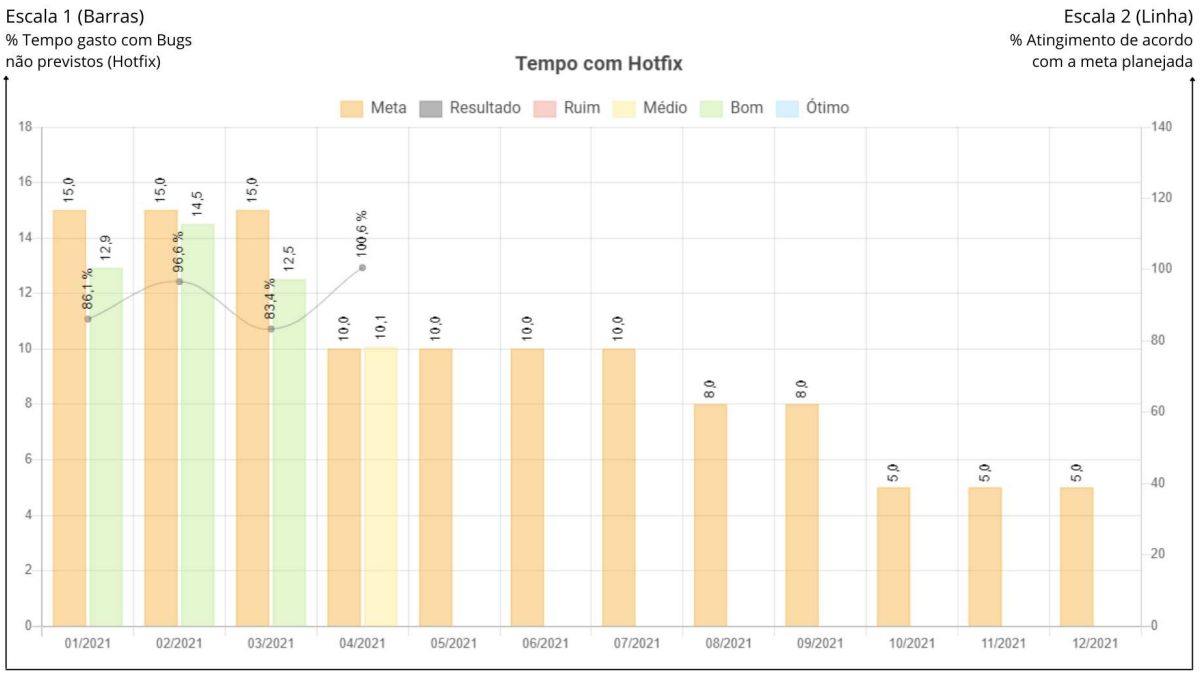
Figura 16: Gráfico de evolução do tempo improdutivo da empresa devido a Bugs



Fonte: Dados da empresa analisados

O gráfico acima contém um indicador que representa o tempo gasto na empresa pela equipe de projetos (consultores) que foi improdutivo devido a Bugs encontrados na plataforma em relação ao tempo total gasto no desenvolvimento dos projetos de CPM e BI. As barras laranjas representam a meta mensal e a linha o atingimento feito, com proporção inversa, logo pretende-se manter o resultado realizado abaixo da meta estabelecida. O mês de Janeiro por ter sido um mês atípico na metodologia de coleta dos dados apresenta um resultado que pode ser desconsiderado para análise, porém é possível notar uma melhora contínua na proporção de tempo improdutivo devido a Bugs ao longo dos meses de Fevereiro, Março e Abril.

Figura 17: Gráfico da proporção do tempo da indústria gasto em Bugs não previstos



Fonte: Dados da empresa analisados

O indicador da empresa representado acima representa uma proporção entre o tempo gasto pela indústria com a correção de Bugs em Hotifx, ou seja, trabalhando em Bugs não previstos pelos processos de teste. Assim como o indicador de tempo não produtivo este representa uma mensurável direta para qualidade do software implantado em produção, e é possível também notar uma melhoria considerável nos meses de Fevereiro, Março e Abril, então pode-se dizer que os processos de teste que atuavam diretamente para redução do tempo gasto em Bugs não previstos que seriam os testes da lista de segurança (para evitar problemas graves retroativos), os testes de regressão construídos pela abordagem de UI e a proximidade com a equipe de projetos para homologação das Releases tiveram resultados positivos para as métricas de qualidade adotadas pela empresa.

As estratégias de testes ativas que foram adotadas pode-se dizer que são pensadas para atacar impactos indiretos no sistema advindos da adição de uma nova funcionalidade. Tais testes em UI representam um nível de esforço desproporcional do que a abordagem através dos testes automatizados, já que pegando o exemplo dos testes graves retroativos podemos notar que são testes para falhas já encontradas, que são repetidos por entrega de Release manualmente, o desenvolvimento

automatizado do teste com aquele objetivo reduziria significativamente o esforço demandado.

A cultura da empresa nasceu como uma startup, e tendo boa parte das lideranças sendo composta por pessoas mais jovens, de 20 a 30 anos é também muito diferente de empresas tradicionais, que traziam consigo um viés de pensamento mais similar ao das metodologias de gestão tradicionais, como Gestão em Cascata. Essa cultura sem dúvidas facilita a assimilação de uma visão de gestão já mais alinhada com a proposta que o manifesto ágil traz. Um outro ponto de análise relevante é quando pensamos na relação entre os valores identificados na empresa inicialmente e os valores do Scrum e do XP podemos ver que existem pontos de similaridade importantes.

Paixão e protagonismo (valores mencionados da empresa analisada) se colocam na prática dentro das atividades da indústria através de comportamentos acordados como responsabilidade aceita do XP, e também se conectam com o valor de comprometimento do Scrum. Liberdade e atenção aos detalhes se colocam em prática através da comunicação constante sobre o escopo para adaptações e evoluções no mesmo, semelhança também notada com o valor de melhoria contínua do XP. O valor de UX/UI se conecta diretamente com a proporção do Backlog de produto destinado a tarefas que são de melhorias de usabilidade e da interface com o usuário, são parte também de uma ponte direta com o cliente final da empresa analisada.

Considerando que uma das prioridades da empresa era em relação a qualidade existem diversas práticas não colocadas dentro de seus processos que poderiam influenciar diretamente no resultado final de seu produto. O tester automatizado poderá futuramente implementar os testes de serviços, porém é essencial que sejam tomadas medidas para que se altere as prioridades do desenvolvimento, de forma que os programadores utilizem uma abordagem orientada a testes (TDD), já que menos de 1% dos métodos e classes está atualmente coberta por testes unitários e segundo a pirâmide de Cohn é um investimento de baixo custo e alto valor agregado.

Nota-se também funções divididas entre o gestor de produto e o gestor de projetos em uma única pessoa. Tal fato, segundo o responsável, afeta diretamente as

tarefas de gestão do produto, de forma que não há tanto tempo para dedicação às histórias de usuário, ao contato com o cliente e a possíveis benchmarkings para trazer propostas de evolução do produto. Como sugestão pode-se dizer que a Analista de Processos ou algum membro da equipe possa ser capacitado a ocupar a posição de gestor(a) de projetos.

Se preza-se por qualidade também podemos dizer que os processos de correção em Hotfix aplicados pela empresa fazem com que o código desenvolvido não passe pelas etapas de teste que geraram a evolução das mensuráveis de qualidade da empresa analisadas nas Figuras 16 e 17. Dessa forma é correto dizer que são processos que devem ser ao máximo evitados, por saber que a empresa não aplicava processos de refatoração constante nem possui cobertura completa de testes automatizados é recomendável que no planejamento de próximas Releases atividades como desenvolvimento de testes e refatoração sejam priorizadas como atividades necessárias para cumprimento de um débito técnico e evitar avanços descontrolados de complexidade no código.

Nota-se também que existem processos que poderiam ser divididos entre diferentes tipos de equipes, um exemplo não mostrado na metodologia é acerca dos processos de implantação do produto em servidores de clientes. Tal responsabilidade poderia ser assumida por uma equipe especializada em infraestrutura e redes de computadores, principalmente por sabermos que tais atividades possuem planejamento variável com a aquisição de novos clientes e torna-se impraticável incluí-la nos processos de desenvolvimento por processos iterativos feitos para contemplar as necessidades de um produto.

5 Conclusão e Trabalhos Futuros

Este trabalho analisou de forma comparativa duas metodologias de desenvolvimento ágil, explicitando pontos de discrepância entre os resultados obtidos por cada uma sobre uma análise teórica, os pontos fortes de cada uma das literaturas e também uma contextualização prática de uma implementação mista das metodologias em uma empresa de software do ramo de CPM e BI.

O Scrum ao longo da análise apresentou-se como uma metodologia que compõe uma série de orientações precisas para o aspecto de gestão de processos e inclusive com orientações mais detalhadas de como inclusive guiar uma reunião. Por outro lado o XP apresenta uma literatura mais completa na descrição de seus valores com um diferencial singular, uma tangibilização da aplicação desses valores através dos princípios descritos, não só isso mas também apresenta diferente do Scrum práticas de Engenharia de Software e com direcionamentos específicos para como se programar, testar, refatorar, entre outros pontos.

Pode-se dizer que ambas as metodologias são complementares, porém é claro também ao final do trabalho que o estudo de caso apresentado não trouxe exemplos e contexto suficiente para uma conclusão ou análise de eficiência dessa utilização mista. Essa afirmação parte do fato de que as análises, resultados e recomendações foram feitas comparando-se recortes da implementação aplicados à estrutura da equipe e os papéis recomendados, ou então a estratégia de testes, ou aos processos e ciclos da esteira de desenvolvimento. Por mais que estejam todos conectados ao conteúdo do trabalho e às discussões exploradas, ainda existe a falta de um parecer que leve todas as dimensões da aplicação em consideração.

A utilização do TDD (Desenvolvimento Orientado a Testes) segundo síntese e soma dos estudos analisados mostrou em geral um resultado positivo em qualidade externa e interna para sua implementação em relação ao ITLD. Os experimentos aqui analisados tinham um tempo curto de análise, de até 5 meses, em paralelo a projetos que envolvem a construção de um produto contínuo os impactos das primeiras Leis de Lehman aqui aplicadas tornam-se mais críticas já que variam com o tempo, por isso

recomenda-se que em futuros trabalhos e experimentos maiores períodos de tempo sejam analisados, para que levem esse fator em consideração, não só pela influência das Leis de Lehman mas também pela ausência de estudos de longo prazo na literatura da área.

Notou-se uma melhoria nas mensuráveis de qualidade da empresa analisada nos meses de Fevereiro, Março e Abril, porém os resultados analisados foram correlacionados somente com a implementação dos testes de UI que continham como objetivo uma análise dos impactos indiretos das funcionalidades implementadas em outras partes do sistema. Pode-se dizer que em trabalhos futuros com objetivos semelhantes se escolha um único ponto de análise, nesse caso para avaliar-se criticamente a utilização das metodologias e sua efetividade poderiam levar mais fatores em consideração. Esse trabalho focou bastante nas práticas e orientações para testes pois acredita-se que hoje seja a de maior impacto na empresa, porém outras facetas poderiam ter seus impactos analisados, como por exemplo as práticas de engenharia pregadas pelo XP adotadas na empresa com diferentes nuances como: Programação em Pares, Integração Contínua e Design Incremental.

Este trabalho possibilitou um estudo acadêmico com conexão e aplicação diretas no mercado de trabalho através do autor que as escreveu e do ambiente e desafios aos quais sua atividade profissional o impõe. Temáticas que trazem também conhecimentos de gestão podem ser cada vez mais integradas ao ambiente de engenharia, afinal é sabido que engenheiros muitas vezes têm papéis e funções que fogem de um embasamento puramente tecnológico.

Referências

BARBOZA FILHO, F. U. N., de CARVALHO, M. M., & RAMOS, A. W. **Gerenciamento de projetos: o impacto do uso dos indicadores de desempenho no resultado do projeto**. 2009.

Kent Beck, com Cynthia Andres, **Extreme Programming Explained**: Embrace Change, Addison-Wesley. 2000.

VARGAS, Ricardo. **Gerenciamento de projetos: estabelecendo diferenciais competitivos**. 7. ed. Rio de Janeiro: Brasport, 2009.

ANGELA MICHELLE MARTIN, **The Role of Customers In Extremme Programming Projects**. Victoria University of Wellington, 2009.

Nagy Ramadan Darwish; **ENHANCEMENTS IN SCRUM FRAMEWORK USING EXTREME PROGRAMMING PRACTICES**, International Journal of Intelligent Computing and Information Science. IJICIS, Vol.14, No. 2 April 2014. https://doi.org/10.21608/ijicis.2014.15773

A Guide to the Project Management Body of Knowledge. Project Management Institute ANSI/PMI 99-001-2017.

Stacey RD; **Strategic management and organisational dynamics: the challenge of complexity** (Prentice Hall, Harlow), 3rd ed.; 2002.

Alex Ballarin Latre. **Contextless Scrum: A Principles or Rules Driven Framework**. April 8, 2019. Disponível em: <https://www.scrum.org/resources/blog/contextless-scrum-principles-or-rules-driven-framework>. Acesso em 2 de Maio de 2021.

Stacey RD. **Strategic management and organisational dynamics: the challenge of complexity**. 3rd ed. Harlow: Prentice Hall, 2002.

Kent Beck, and Martin Fowler. **Planning Extreme Programming**, Addison-Wesley. 2001.

Philip J. Streatfield. **The Paradox of Control in Organizations;** Routledge; October 12, 2001.

Babatunde Ogunnaike e Willis Harmon Ray; **Process Dynamics, Modeling, and Control**; 1994.

Ham Vocke; **The Practical Test Pyramid**; 2018.
Disponível em: <https://martinfowler.com/articles/practical-test-pyramid.html>. Acesso em 25 de Maio de 2021.

Sean Stolberg; **Enabling Agile Testing Through Continuous Integration**; Pacific Northwest National Laboratory, 2009 Agile Conference. DOI 10.1109/AGILE.2009.16 https://doi.org/10.1109/AGILE.2009.16

Andrew Hunt and David Thomas; **Pragmatic Unit Testing in Java with JUnit**; 2003.

Diomidis Spinellis; **State-of-the-Art Software Testing**; 2018. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8048644> Acesso em 17 de Abril de 2021.

Don Wells; **Extreme Programming: A gentle introduction**. 2013 Disponível em: <http://www.extremeprogramming.org/> Acesso em 13 de Maio de 2021.

HERBSLEB, J. D.; **Beyond Computer Science**. In: Proceedings of the 27th International Conference on Software Engineering (ICSE), St. Louis, Missouri, EUA; 2005

Rafael Prikladnicki, Renato Willi, Fabiano Milani; **Métodos Ágeis para Desenvolvimento de Software**. 2014. ISBN 978-85-8260-208-9

Frank Maurer e Grigori Melnik; **Agile Methods: Crossing the Chasm**; 29th International Conference on Software Engineering; 2007. https://doi.org/10.1109/ICSECOMPANION.2007.18

Victor Szalvay; **An Introduction to Agile Software Development**; 2004.

The Standish Group: Chaos Chronicles Version; West Yarmouth, MA. 1994.

Ricardo de Almeida Falbo; **Engenharia de Software: Notas de Aula**. UFES - Universidade Federal do Espírito Santo; 2014.

Marco Tulio Valente; **Engenharia de Software Moderna**; Editora Moderna, primeira edição. 2020.

K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marrick, R.C. Martin, S. Mellor,K. Schwaber, J. Sutherland, and D. Thomas. (2001), **Manifesto for Agile Software Development**; Disponível em: <http://agilemanifesto.org/> Acesso em: 10 de Abril de 2021.

Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK®) 6ª edição, em Português (Brasil). ISBN-10: 1628251921; ISBN-13: 978162825192 Project Management Institute. 2018.

Ken Auer and Roy Miller. **Extreme Programming Applied: Playing To Win**; Addison–Wesley. 2001

Chromatic (Shane Warden); **Extreme Programming Pocket Guide**; Ed. O'Reilly

Media; 2003.

Steve Freeman, Nat Pryce; **Growing Object-Oriented Software, Guided by Tests**; 2010.

IEEE Standard for Software Productivity Metrics; 1993; IEEE Std 1045-1992

M. M. Lehman. **Rules and Tools for Software Evolution Planning and Management. Annals of Software Engineering**; 2001.

Gustavo Baculi Benato e Plínio Roberto Souza Vilela; **Test-Driven Development: uma revisão sistemática**; 2021

Wilson Bissi, Adolfo Gustavo Serra Seca Neto, Maria Claudia Figueiredo e Pereira Emer; **The Effects of Test Driven Development on Internal Quality, External Quality and Productivity: A systematic review**; 2016. https://doi.org/10.1016/j.infsof.2016.02.004

Adrian Santos, Janne Järvinen, Jari Partanen, Markku Oivo e Natalia Juristo; **Does the performance of TDD hold across software companies and premises? A group of industrial experiments on TDD**; 2018. https://doi.org/10.1007/978-3-030-03673-7_17

Itir Karac e Burak Turhan; **What Do We (Really) Know about Test-Driven Development?;** 2018. https://doi.org/10.1109/MS.2018.2801554

Luis Alberto Cisneros Gómez; **Analysis of the Impact of Test Based Development Techniques (TDD, BDD, and ATDD) to the Software Life Cycle**; 2018. https://doi.org/10.1145/2652524.2652526

Davide Fucci, Burak Turhan e Markku Oivo; **Impact of Process Conformance on the Effects of Test-driven Development**; 2014. https://doi.org/10.1145/3239235.3240502

Davide Fucci, Burak Turhan e Markku Oivo; **A Longitudinal Cohort Study on the Retainment of Test-Driven Development**; 2018

Hakan Erdogmus, Burak Turhan, Markku Oivo, Natalia Juristo e Davide Fucci; **A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last?**; 2016.

Stack Overflow Survey, 2018. Disponível em: <https://insights.stackoverflow.com/survey/2018> Acesso em: 05 de Maio de 2021.

MACIEIRA, Leonardo Gabriel; **Metodologia ágil para gestão e planejamento de projetos: Implantação do Scrum e do Framework Jira na qualidade em uma empresa do setor aeronáutico**; 2018

Martin Fowler, with Kent Beck; **Refactoring, Improving the Design of Existing Code**; 2018.