INSTITUTO FEDERAL
SANTA CATARINA

CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE SAÚDE E SERVIÇO
CURSO SUPERIOR DE TECNOLOGIA EM
GESTÃO DA TECNOLOGIA DA INFORMAÇÃO

BRUNO JAIME NASCIMENTO

DESENVOLVIMENTO DE SOFTWARE E METODOLOGIAS: Fatores e Dificuldades que Influenciam na Adoção de Metodologias de Desenvolvimento

Florianópolis - SC
2019

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA — CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE SAÚDE E SERVIÇOS
CURSO SUPERIOR DE TECNOLOGIA EM GESTÃO DA TECNOLOGIA DA INFORMAÇÃO

BRUNO JAIME NASCIMENTO

DESENVOLVIMENTO DE SOFTWARE E METODOLOGIAS: fatores e dificuldades que influenciam na adoção de metodologias de desenvolvimento

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia do Estado de Santa Catarina como parte dos requisitos para obtenção do título de Tecnólogo em Gestão da Tecnologia da Informação.

Professor Orientador: Underléa Cabreira, Dra.
Coorientador: Cleverson Tabajara Vianna, Dr.

FLORIANÓPOLIS, NOVEMBRO DE 2019

Ficha de identificação da obra elaborada pelo autor.

Nascimento, Bruno Jaime
Desenvolvimento de Software e Metodologias : fatores e dificuldades que influenciam na adoção de metodologias de desenvolvimento / Bruno Jaime Nascimento ; orientação de Underléa Cabreira; coorientação de Cleverson Tabajara Vianna. – Florianópolis, SC, 2019.
69 p.
Trabalho de Conclusão de Curso (TCC) – Instituto Federal de Santa Catarina, Câmpus Florianópolis. CST em Gestão da Tecnologia da Informação. Departamento Acadêmico de Saúde e Serviços.
Inclui Referências.

1. Desenvolvimento de Software. 2. Metodologias de Desenvolvimento. 3. Desenvolvimento Ágil. 4. Metodologias Ágeis. I. Cabreira, Underléa . II. Vianna, Cleverson Tabajara. III. Instituto Federal de Santa Catarina. Departamento Acadêmico de Saúde e Serviços. IV. Título.

DESENVOLVIMENTO DE SOFTWARE E METODOLOGIAS: FATORES E DIFICULDADES QUE INFLUENCIAM NA ADOÇÃO DE METODOLOGIAS DE DESENVOLVIMENTO

BRUNO JAIME NASCIMENTO

Este trabalho foi julgado adequado para obtenção do Título de Tecnólogo em Gestão da Tecnologia da Informação e aprovado na sua forma final pela banca examinadora do Curso Superior de Tecnologia em Gestão da Tecnologia da Informação do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis, 5 de novembro de 2019.

Felipe C. Soares, Doutor
Coordenador do CST em Gestão da Tecnologia da Informação
Instituto Federal de Santa Catarina

Banca Examinadora:

Ungerléa Cabreira, Doutora
Orientadora
Instituto Federal de Santa Catarina

Fernando Ferreira Aguiar, Mestre

Marcos André Pisching, Doutor
Instituto Federal de Santa Catarina

AGRADECIMENTOS

A meus pais e a minha família, por todo o incentivo e exemplo ao longo dos anos.

A minha namorada Larissa, por todo o amor e apoio incondicional.

A meu grande amigo Fernando, por todo o auxílio e camaradagem.

A minha orientadora Underléa, por todo o apoio e direcionamento que tornaram este trabalho possível.

A todos os professores e mestres que me acompanharam nesta caminhada.

Ao IFSC, por todas as oportunidades concedidas.

RESUMO

Este trabalho foi realizado com o objetivo de investigar os motivos que levam a empresas e gestores ainda desenvolverem projetos de software sem o uso de metodologias de desenvolvimento, apesar das evidências, tanto anedóticas quanto acadêmicas, dos benefícios de seu uso. Para isto, foi realizada uma revisão de literatura a respeito de metodologias de desenvolvimento, suas características e princípios, assim como uma enquete sobre o uso e aplicação de metodologias em empresas e suas dificuldades. Com base na revisão de literatura, foi desenvolvido e aplicado um questionário a profissionais da área de TI que atuam em empresas da Grande Florianópolis, procurando verificar o uso de metodologias em suas empresas, os desafios envolvidos no processo de desenvolvimento de software e as dificuldades identificadas pelos participantes na adoção de metodologias de desenvolvimento. Através deste trabalho, foi concluído que os principais fatores que impedem a adoção de metodologias de desenvolvimento estão relacionadas a resistência à mudanças por parte das organizações e à falta de conhecimento especializado sobre metodologias de desenvolvimento ou como aplicá-las.

Palavras Chave: Desenvolvimento de Software, Metodologias de Desenvolvimento, Desenvolvimento Ágil, Metodologias Ágeis.

ABSTRACT

This work was conducted with the goal of investigating the reasons why companies and managers still develop software without the use of software development methodologies, despite ample evidence, both anecdotal and academic, of the benefits of their use. For this, a literature review was conducted on software development methodologies, their characteristics and principles, and a literature search was conducted on studies involving the use of and adoption of methodologies in companies and the difficulties involved. Based on the literature review, a questionnaire was developed and applied to IT professionals working in companies in Florianópolis, seeking to verify the use of methodologies in their companies, the challenges involved in the software development process and the difficulties identified by the participants in the adoption of methodologies. Through this work, it was concluded that the main factors that prevent the adoption of software development methodologies are related to organization's resistance to change and the lack of specialized knowledge about software development methodologies or how to apply them.

Keywords: Software Development, Software Development Methodologies, Agile Development, Agile Methodologies.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

XP	*eXtreme Programming*
PTA	*Processo de Transformação/Transição Ágil*
GT	*Grounded Theory*
VCS	*Version Control System*
SEBRAE	*Serviço Brasileiro de Apoio às Micro e Pequenas Empresas*

SUMÁRIO

1. INTRODUÇÃO

Atualmente, a demanda por serviços de Tecnologia da Informação e ferramentas de software está em constante crescimento. Empresas de desenvolvimento de softwares são do interesse de empreendedores e investidores; a maioria dos mercados de software tem alcance global e é dominada por um punhado de produtos, por isso, existem oportunidades significativas para lucros e crescimento a longo prazo para uma empresa que leva o produto certo ao mercado (CROWNE, 2002). Procurando suprir esta demanda, milhares de empresas ofertando serviços de TI são criadas todos os anos, no Brasil e no mundo. No entanto, grande parte delas não consegue sobreviver neste ambiente extremamente competitivo. Segundo dados do Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE, 2016), a taxa de sobrevivência de micro e pequenas empresas que trabalham no ramo de desenvolvimento de software de computador sob encomenda é de 66%, no período dos dois primeiros anos. Ou seja, de cada dez empresas criadas, quase quatro fecham as portas em menos de dois anos. Segundo uma pesquisa realizada pelo *Standish Group (1995)*, 31,1% dos projetos de desenvolvimento de software são cancelados antes de serem concluídos. Outros resultados indicam que 52,7% dos projetos custarão 189% de suas estimativas originais (PINTO, 2002). Um estudo mais recente realizado pelo mesmo grupo em 2015 revela que 60% dos projetos são finalizados fora do prazo definido e 56% extrapolam o seu orçamento. Percebemos então que projetos de desenvolvimento de software frequentemente enfrentam atrasos e custam mais do que o planejado. O desenvolvimento de software é altamente imprevisível. Somente 10% dos projetos de software são entregues com sucesso, dentro do orçamento e prazo estimados (PINTO, 2002).

No início dos anos 90, devido ao largo crescimento das indústrias de software e hardware, percebeu-se que metodologias de gerenciamento de projetos são úteis para alcançar excelentes resultados na produção de produtos de software e hardware. Devido à adoção destas metodologias, organizações se tornaram mais

eficientes ao produzir produtos de alta qualidade dentro de um prazo e orçamento especificados (AKBAR, et al., 2018).

Ainda segundo Akbar, et al. (2018, p.8066):

> É prioridade de toda organização escolher metodologias de desenvolvimento de software de baixo custo que podem fornecer melhores práticas para desenvolver produtos de alta qualidade e cumprir os requisitos da organização. Portanto, metodologias de desenvolvimento de software são consideradas fatores-chave para o sucesso e progressão das organizações.

A literatura tradicionalmente enxerga o uso de metodologias de desenvolvimento de software como benéficas para melhorar tanto o processo quanto o produto resultante do desenvolvimento de sistemas (FITZGERALD, 1998).

1.1.Definição do Problema

Embora muitas sejam as recomendações, bem como o reconhecimento dos benefícios na utilização de uma metodologia de desenvolvimento de software, ainda há gestores que preferem não adotar metodologias. Mais do que isto: em um estudo realizado por Fitzgerald (1998), dentre 776 organizações pesquisadas, 60% destas não utilizavam metodologia alguma e dentre estas, 79% não pretendiam adotar uma metodologia. Deste modo, em busca de compreender quais são os fatores que influenciam nesta decisão, temos como objetivo fundamental neste trabalho responder a seguinte pergunta de pesquisa: *quais os principais fatores que dificultam na adoção de metodologias de desenvolvimento de software pelos gestores e organizações de TI?*

1.2.Justificativa

Nas décadas iniciais do desenvolvimento de sistemas, projetos frequentemente utilizavam métodos não sistemáticos, um processo hoje chamado de

"*code-and-fix"* resumido por Christophe Thibuat como, "um ano escrevendo código e um ano depurando". Esta abordagem não estruturada funciona bem para sistemas pouco complexos, mas à medida que a complexidade aumenta, torna-se mais difícil adicionar novos recursos e *bugs* são mais difíceis de corrigir (AWAD, 2005). Com o aumento na complexidade dos sistemas sendo desenvolvidos, abordagens metodologicamente mais disciplinadas foram defendidas (FITZGERALD, 1998). A solução para estes problemas veio na forma da criação de metodologias. As metodologias impõem um processo disciplinado ao desenvolvimento de software com o objetivo de tornar o desenvolvimento mais previsível e eficiente (AWAD, 2005).

Segundo Fitzgerald (1998), alguns dos conceitos fundamentais que sustentam o uso de metodologias são:

- O desenvolvimento de sistemas é um processo muito complexo. O uso de metodologias pode fornecer uma subdivisão reducionista desse processo em etapas plausíveis e coerentes;
- Ao tornar a tarefa de desenvolvimento mais visível e transparente, metodologias podem facilitar o gerenciamento de projetos e o controle do processo de desenvolvimento, assim reduzindo riscos e incertezas;
- Elas podem fornecer uma estrutura para a aplicação de técnicas e recursos em momentos apropriados durante o processo de desenvolvimento;
- Possibilita a padronização no processo de desenvolvimento. Além disso, pode levar a maior produtividade e qualidade, pois os requisitos de recursos podem ser previsto e disponibilizados como e quando necessários;

As primeiras metodologias estabelecidas, que hoje são consideradas as metodologias tradicionais, são centradas no planejamento, onde o trabalho começa com a elicitação e documentação de um conjunto completo de requisitos do sistema, seguido de desenvolvimento e inspeção de design arquitetural. Devido a esses aspectos excessivamente trabalhosos, estas metodologias tornaram-se conhecidas

como “metodologias pesadas” (AWAD, 2005).

Métodos tradicionais, como Modelo em Cascata, Modelo V e Modelo Racional Unificado (RUP, *Rational Unified Process*), presumem que é possível identificar no início do projeto todos os requisitos do usuário. Isso, por sua vez, deveria reduzir a probabilidade de incerteza dos requisitos. No entanto, o fato é que requisitos mudam (O’DONNELL; RICHARDSON, 2008). De acordo com Mikael Lindvall et al. (2004) “o uso de, o interesse em e as controvérsias sobre os métodos ágeis aumentaram dramaticamente” e este aumento tem sido atribuído a pequenas organizações reconhecendo os metodologias tradicionais como muito “incômodas, burocráticas e inflexíveis”.

Tem sido argumentado que métodos ágeis são otimizados para lidar com mudanças e atender a solicitações de clientes de modo rápido e eficiente (O’DONNELL; RICHARDSON, 2008). Com base nos desafios atuais do desenvolvimento de software, as metodologias Ágeis são uma opção interessante e viável para se obter qualidade, controle de orçamento do projeto e entrega de valor frequente e contínua (CAMPANELLI, PARREIRAS, 2015).

Uma pesquisa realizada pela Shine Technologies (O’DONNELL; RICHARDSON, 2008) com um grupo de empresas quanto a aplicação de métodos ágeis concluiu que:

- 93% relataram aumento de produtividade;
- 88% produziram softwares com qualidade melhor ou significantemente melhor;
- 83% relataram um aumento na satisfação de negócios.

1.3.Objetivos

1.3.1. Geral

O objetivo geral deste trabalho é identificar os principais fatores que influenciam a não adoção de metodologias de desenvolvimento de software por parte das organizações e gestores de projetos.

1.3.2. Específicos

Elaborar e aplicar um questionário enumerando possíveis fatores que influenciam nas decisões de desenvolvimento de software dos gestores de empresas ou gestores de projetos em não utilizar metodologias de desenvolvimento de software;

Analisar e apresentar os resultados obtidos;
Apresentar os fatores encontrados através do questionário aplicado e da revisão de literatura que influenciam positiva e negativamente na aplicação de uma metodologia ágil de desenvolvimento de software em uma organização;

Elencar os desafios e problemas que ainda impedem a aplicação de um processo de desenvolvimento de software.

2. METODOLOGIA

Esta seção está separada pelos tópicos “Caracterização da Pesquisa”, na qual se apresenta a metodologia adotada para a condução da pesquisa, e “Procedimentos Metodológicos” que descreve as etapas realizadas, e a revisão bibliográfica para obter embasamento teórico.

Para Creswell e Clark (2007) metodologia é o estudo da organização dos caminhos a serem percorridos e dos instrumentos utilizados na elaboração de uma pesquisa ou estudo científico. Para Fonseca (2002) os Procedimentos Metodológicos se concentram em demonstrar as etapas do estudo, como revisão sistemática, criação de modelos, métodos ou técnicas utilizadas para a conclusão do trabalho.

2.1. Caracterização da Pesquisa

Este Trabalho de Conclusão de Curso é caracterizado como uma pesquisa básica ou científica, com o intuito de entender o problema e seu contexto, e gerar novos conhecimentos para o avanço da ciência, sem aplicação prática prevista (GIL, 2003).

Do ponto de vista de sua abordagem e objetivos, este trabalho de conclusão de curso pode ser avaliado como abordagem qualitativa, por considerar que exista uma relação entre mundo e o objeto em estudo que não foi expressa em números, mas analisada de modo intuitivo,, e enquadra-se como pesquisa descritivo-exploratória por proporcionar maior familiaridade com um problema, envolvendo levantamento bibliográfico e técnicas de coletas de dados padronizados por meio de questionário ou observação (GIL, 2003; LAKATOS; MARCONI, 2003).

2.2. Procedimentos Metodológicos

Dos procedimentos metodológicos, este trabalho foi desenvolvido por meio de

revisão bibliográfica do estado da arte abrangendo metodologias de desenvolvimento de softwares, que estruturou as hipóteses levantadas para construção de um questionário (ver anexo A) aplicado a equipes atuantes em projetos de software. As empresas foram selecionadas de modo aleatório na grande Florianópolis tendo como único requisito atuar na área de desenvolvimento de software. Dentre os respondentes tiveram 9 empresas completando um total de 16 respostas as quais são apresentadas no capítulo de análise e discussão dos resultados. O questionário foi aplicado entre 18 de setembro de 2019 até 10 de outubro de 2019, totalizando um período de aproximadamente três semanas.

2.3. Organização do Trabalho

Este trabalho está estruturado em 5 capítulos. No capítulo 3, Revisão de Literatura, os conceitos fundamentais referentes a desenvolvimento de software e metodologias de desenvolvimento são apresentados, e no capítulo 4, Análise e Discussão dos Resultados, são apresentados os resultados da pesquisa e sua análise.

3. REVISÃO DE LITERATURA

3.1.Desenvolvimento de Software

Desenvolvimento de Software começou como uma atividade desorganizada, comumente definida como "*code-and-fix"*, ou seja, desenvolver e corrigir. Os softwares eram escritos com pouco planejamento, e o design do sistema era determinado em base de muitas decisões de curto prazo. Esta prática funcionava bem para pequenos sistemas, mas quando os sistemas aumentavam em complexidade, aumentava também a dificuldade de se adicionar novas funcionalidades e falhas nos códigos eram mais difíceis de resolver. Este estilo de desenvolvimento foi utilizado por muitos anos, até que uma alternativa foi introduzida: Metodologia. Metodologias procuram estabelecer um processo disciplinado no desenvolvimento de softwares, com o objetivo de o tornar mais previsível e eficiente (AWAD, 2005).

Determinar a metodologia de desenvolvimento de software a ser utilizada em um projeto é uma decisão crítica. Condições de mercado e pressões competitivas requerem que as organizações diminuam tempos de ciclo, reduzam custos e melhorem a qualidade dos softwares (KAKKAR, 2006).

Por vezes, a decisão de uma metodologia pode ser baseada em um viés de marketing ou literatura, que apoiam práticas novas ou apoiadas pela indústria. Outras vezes, organizações podem contar com padrões já utilizados por questões de consistência e repetibilidade. É improvável que a escolha de uma metodologia de desenvolvimento de software seja uma questão simples ou determinística (0, 2016). Hawrysh e Ruprecht (2000) afirmam que uma única metodologia não é capaz de funcionar para todo o espectro de diferentes projetos, e a gestão do projeto deve definir a sua natureza e selecionar a metodologia de desenvolvimento mais adequada ao projeto em questão.

As metodologias chamadas de Pesadas ou Orientadas a Documentação são consideradas como as maneiras tradicionais de desenvolver software. Estas metodologias são baseadas em uma série sequencial de passos, como definição de

requisitos, criação de soluções, realização de testes e implantação (AWAD, 2005). Alguns dos modelos de metodologias pesadas mais conhecidos são o modelo em Cascata, o modelo Espiral e o modelo chamado *Rational Unified Process* (RUP), ou em português, Processo Racional Unificado.

Metodologias pesadas requerem a definição e documentação de um conjunto fixo de requisitos, definidos no começo do projeto. Elas assumem que o cliente tem conhecimento dos requisitos desde o início e que eles permaneceram os mesmos (TORTAMIS, 2004).

Em contraste com as metodologias tradicionais, as metodologias leves, também conhecidas como ágeis, são menos estruturadas. Elas fornecem orientação e limites, enquanto uma metodologia pesada define cada atividade e documentação em detalhes (AWAD, 2005).

Nandhakumar e Avison (1999) argumentam que metodologias de desenvolvimento de software tradicionais são tratadas como uma "ficção necessária para apresentar uma ilusão de controle ou prover um status simbólico, e são mecanicistas demais para serem de utilidade nas atividades detalhadas do dia-a-dia de um desenvolvedor de sistemas de uma organização. Segundo Awad (2015), profissionais julgavam esta visão centrada em processos frustrante e vivenciavam problemas quando a taxa de mudanças era relativamente baixa. Como resultado, diversos consultores desenvolveram independentemente metodologias e práticas para aceitar e reagir às mudanças inevitáveis que estavam vivenciando. Essas metodologias e práticas são baseadas em aprimoramentos iterativos; técnicas que foram introduzidas em 1975 e que mais tarde se tornaram conhecidas como metodologias ágeis.

3.1.1 Metodologias Pesadas

Nos primeiros anos do desenvolvimento de software, geralmente os desenvolvedores não seguiam nenhuma prática ou rotina em particular, apenas questionavam o usuário quais eram as suas necessidades e partiam disto. Este processo, chamado hoje de "*code-and-fix*", cobre os passos básicos de

desenvolvimento de sistemas: análise e desenvolvimento, seguido de testes e correção de erros (BALLE, et al, 2018). Segundo Boehm (1988), este tipo de prática tinha três grandes problemas:

- Depois de um certo número de correções, o código tornava-se tão mal estruturado que as correções subseqüentes eram muito caras/trabalhosas. Isto ressaltou a necessidade de uma fase de design antes do desenvolvimento
- Freqüentemente, até mesmo softwares bem projetados correspondiam tão mal às necessidades dos usuários que eram simplesmente rejeitados ou dispendiosamente refeitos. Isso tornou a necessidade de um levantamento de requisitos antes da fase de desenvolvimento evidente
- O código era caro para corrigir devido à má preparação para testes e modificações. Isto tornou claro a necessidade de planejamento para estas fases

Segundo Fowler (2005, p. 1):

> O software é escrito sem muito planejamento e o design do sistema é definido a partir de muitas decisões de curto prazo. Isto funciona surpreendente bem enquanto o sistema é pequeno, mas à medida que o sistema cresce, torna-se cada vez mais difícil adicionar novos recursos ao sistema. Além disso, os erros tornam-se cada vez mais predominantes e cada vez mais difíceis de corrigir.

Ainda segundo Fowler, o movimento original para tentar solucionar os problemas do modo *code-and-fix* introduziu a noção de metodologia. Essas metodologias, inspiradas em metodologias já conhecidas de engenharia, impõem um processo disciplinado ao desenvolvimento de software, com o objetivo de tornar o desenvolvimento de software mais previsível e mais eficiente. Elas procuram fazer isso desenvolvendo um processo detalhado com forte ênfase no planejamento.

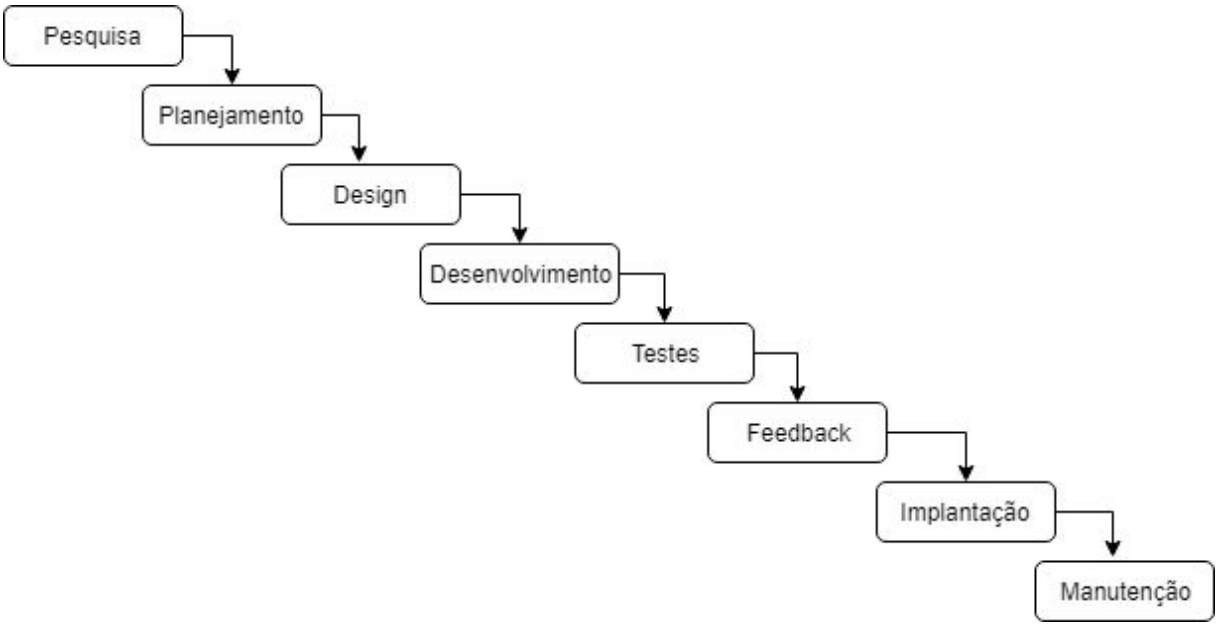
3.1.2 Método Cascata

O Método em Cascata é a primeira metodologia geralmente reconhecida como sendo dedicado ao desenvolvimento de software. Seus princípios são pela primeira vez descritos por Royce, embora o termo Cascata *(Waterfall)* não tenha sido usado em seu artigo (DESPA, 2014). O método é assim chamado porque prevê uma sequência em cascata de uma atividade a para outra, abordando os problemas anteriormente encontrados em desenvolvimento de software, adicionando etapas que cobrem verificação, análise de risco e desenvolvimento (BALLE, et al, 2018).

A metodologia (Figura 1) enfatiza uma progressão linear estruturada entre fases definidas. Cada fase consiste em um conjunto definido de atividades e entregas que devem ser realizadas antes que a fase seguinte possa começar (AWAD, 2005). Segundo Awad, as fases são sempre chamadas de forma diferente, mas a ideia básica é que a primeira fase tenta capturar *o que* o sistema fará, seus requisitos de sistema e software; a segunda fase determina *como* isso será projetado. A terceira fase é onde os desenvolvedores começam a escrever o código, a quarta fase é o teste do sistema e a fase final é focada em tarefas de implementação, como treinamento e documentação.

O *feedback* do proprietário do projeto é recebido apenas após o projeto for completamente desenvolvido e testado. O método em cascata é adequado para projetos de desenvolvimento de softwares de baixa complexidade, onde os requisitos são claros e um planejamento detalhado pode ser facilmente elaborado para o todo projeto (DESPA, 2014).

Figura 1: Metodologia em Cascata



Fonte: Adaptado de Despa(2014)

3.1.3 Manifesto Ágil

O “Movimento Ágil” surgiu com a publicação do “Manifesto para Desenvolvimento Ágil de Software”, elaborado em 2001 por um grupo de profissionais e consultores da área. De acordo com o *website* do Movimento Ágil: “Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar”:

- *Indivíduos e interações* acima de processos e ferramentas
- *Software em funcionamento* acima de documentação abrangente
- *Colaboração com o cliente* acima de negociação de contratos
- *Responder a mudanças* acima de seguir um plano

Os 12 princípios do desenvolvimento de software Ágil, definidos pelo Manifesto, são:

- Nossa maior prioridade é satisfazer o cliente através da entrega

contínua e adiantada de software com valor agregado.

- Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
- Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
- Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
- O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
- Software funcionando é a medida primária de progresso.
- Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
- Contínua atenção à excelência técnica e bom design aumenta a agilidade.
- Simplicidade, a arte de maximizar a quantidade de trabalho não realizado, é essencial.
- As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Para Verret (2018), de um ponto de vista prático, os princípios acima se traduzem em práticas de projeto de software que empregam alto envolvimento do usuário para gerar consenso e criar decisões (Ramesh, Cao e Baskerville, 2010). Além disso, metodologias ágeis adotam uma abordagem iterativa para o desenvolvimento, em oposição a modelos em cascata que utilizam uma abordagem relativamente linear e seqüencial (Sureshchandra e Shrinivasavadhani, 2008).

3.1.4 Metodologias Ágeis

Segundo Highsmith e Cockburn (2001, p.122):

> O que há de novo nos métodos ágeis não são as práticas que eles usam, mas o reconhecimento das pessoas como os principais impulsionadores do sucesso do projeto, associado a um foco intenso na eficácia e manobrabilidade. Isso produz uma nova combinação de valores e princípios que definem uma visão de mundo ágil.

Os métodos ou metodologias Ágeis são caracterizados por ciclos de desenvolvimento mais curtos, alta interação com o cliente, entregas incrementais, redesign frequente com acomodação de mudanças necessária pela coleta dinâmica de requisitos dos usuários. Apesar de diversas metodologias de desenvolvimento de software partilharem dos princípios ágeis elaborados pelo Manifesto Ágil, elas diferem umas das outras em diversos parâmetros (MATHARU et al., 2015). Algumas das mais conhecidas metodologias ágeis são: *Scrum*, *Extreme Programming* (em português: Programação Extrema, também conhecido pela sigla XP), Desenvolvimento *Lean* (ou “enxuto), entre outras. Abaixo serão detalhados algumas destas metodologias.

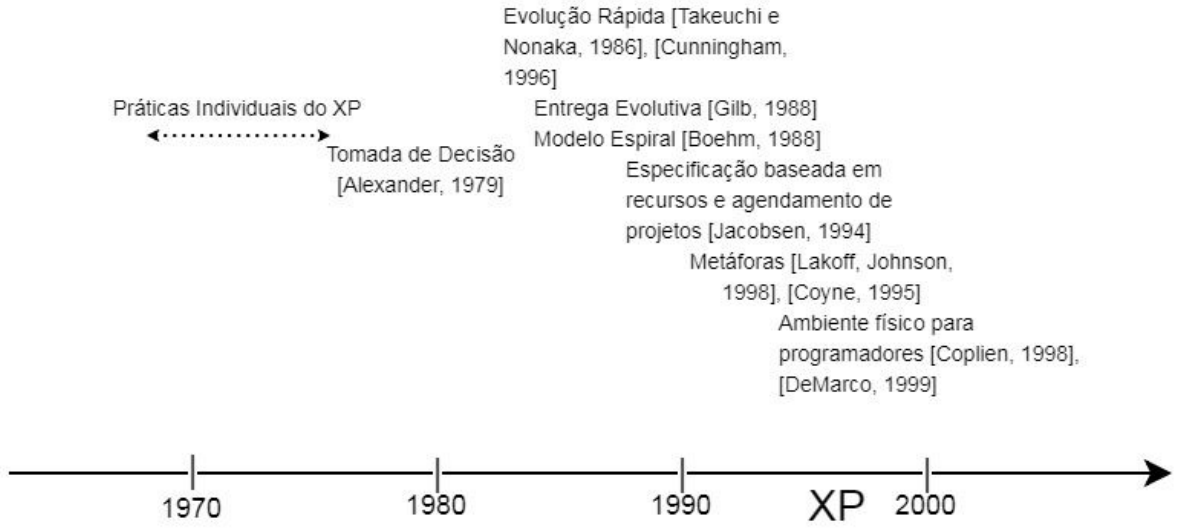
3.1.5 Extreme Programming (XP)

A XP é uma metodologia leve de desenvolvimento criada por Kent Beck, Ward Cunningham e outros profissionais e consultores da área. O método evoluiu dos problemas causados pelos longos ciclos de desenvolvimento das metodologias tradicionais (KHAN; QURESHI; KHAN, 2011). Em vez de planejar, analisar e projetar para o futuro longínquo, o XP explora a redução no custo de alteração de software para realizar todas essas atividades um pouco por vez, durante o desenvolvimento do software (BECK, 1999).

A metodologia teve seu início como uma tentativa de aplicar práticas e metodologias das décadas anteriores consideradas eficientes nos processos de desenvolvimento de software (Figura 2). Após diversas experiências bem sucedidas,

a metodologia XP foi “teorizada” a partir dos princípios e práticas chaves utilizadas. Apesar de as práticas individuais do XP não serem novidade, elas foram coletadas e organizadas para funcionarem umas com as outras de maneira inovadora, assim formando uma nova metodologia (Abrahamsson, *et al,* 2002).

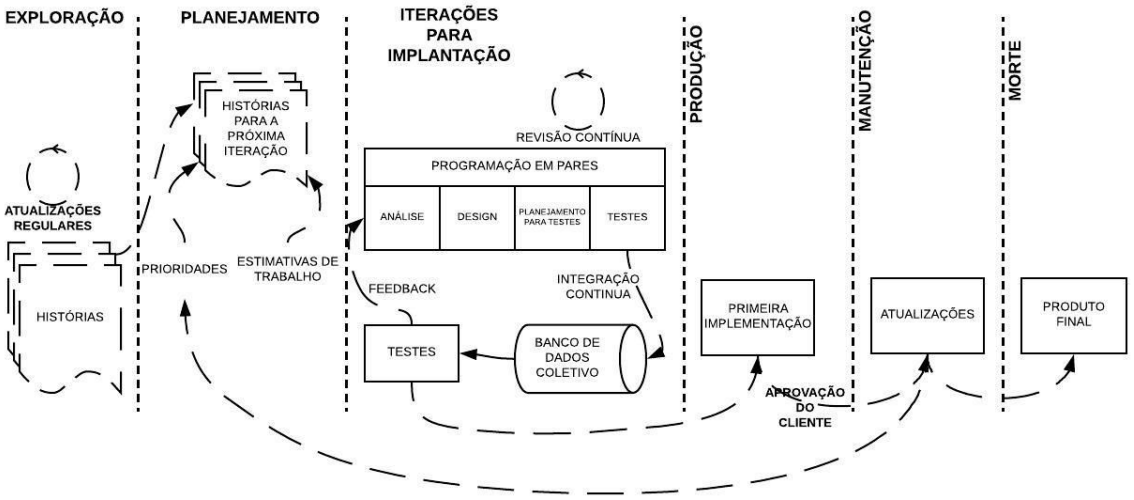
Figura 2: Raízes do processo XP



Fonte: Adaptado de Abrahamsson, *et al* (2002)

O processo XP (Figura 3) pode ser caracterizado por curtos ciclos de desenvolvimento, planejamento incremental, *feedback* contínuo, dependência em comunicação e design evolucionário (BECK, 1999). Com estas características, programadores XP reagem ao ambiente em constante mudança com muito mais coragem (AWAD, 2005). Ainda, segundo Williams (2003), membros de uma equipe XP dedicam poucos minutos a programação, poucos minutos a gestão do projeto, poucos minutos em design, poucos minutos em *feedback* e poucos minutos no desenvolvimento da equipe, todos os dias. O termo ‘extrema’ provém de levar estes princípios e práticas a níveis extremos (AWAD, 2005).

Figura 3: Processo XP



Fonte: Adaptado de Awad (2005)

Os princípios que regem a XP foram definidos por Beck em 1999 e resumidas por Awad em 2005, e estão detalhados abaixo:

- **Planejamento** – O cliente decide o escopo e timing de implantação baseado em estimativas fornecidas pelos programadores. Os programadores desenvolvem apenas as funcionalidades solicitadas pelos requisitos desta iteração;
- **Pequenas Implementações** – O sistema é posto em produção em poucos meses, antes do problema inteiro ser resolvido. Atualizações são feitas com frequência - desde diariamente até mensalmente;
- **Metáforas** – A forma do sistema é definida através de uma metáfora ou conjunto de metáforas compartilhadas entre o cliente e os programadores;
- **Design Simplificado** – A ênfase é desenvolver as soluções mais simples possíveis e complexidade desnecessária e códigos extras devem ser removidos imediatamente;
- ***Refactoring*** – Envolve a reestruturação do sistema através da remoção de duplicidade de código, melhora da comunicação,

simplificação e adição de flexibilidade no sistema, sem nunca alterar a funcionalidade do programa;

- **Programação em Pares** – Todo código que entrará em produção deve ser escrito por dois programadores, em um computador;
- **Propriedade Coletiva** – Nenhum programador é dono ou responsável por segmentos individuais do código, qualquer pessoa poderá alterar qualquer parte do código a qualquer momento;
- **Integração Contínua** – Um novo trecho de código deve ser integrado ao sistema assim que estiver pronto. Quando integrado, o sistema é reconstruído e todos os testes devem ser realizados para que as mudanças sejam aceitas.
- **Semana de 40 horas** – Nenhum membro da equipe deve realizar horas extras mais que duas semanas seguidas. Qualquer necessidade de horas extras deve ser encarada como sinal de problemas.
- **Acesso ao Cliente** – O cliente deve estar disponível a equipe de desenvolvimento a qualquer momento.
- **Padrões de Programação** – Regras para programação existem e devem ser seguidas pelos programadores para garantir consistência e melhorar a comunicação entre a equipe de desenvolvimento;

O ciclo de vida de um projeto XP, definido por Beck (2001) e resumido por Awad (2005) é dividido em seis fases: Exploração, Planejamento, Iterações para Implantação, Produção, Manutenção e Morte.

Na fase de Exploração, o cliente cria cartões de histórias, definições curtas de funcionalidades a serem implementadas no sistema. Na fase de Planejamento, uma ordem de prioridade é criada para cada história e um cronograma para o lançamento da primeira versão do software é definido.

Em Iterações para Implantação, a primeira iteração da equipe de desenvolvimento é criar uma arquitetura do sistema como um todo, e continuamente integrar e testar o seu código.

Testes adicionais e verificação de performance do sistema prévios ao lançamento são realizados na fase de Produção.

Ideias previamente adiadas e sugestões encontradas neste momento são documentadas para serem implementadas nas atualizações do sistema feitas na fase de Manutenção.

Por fim, quando todas as histórias elaboradas pelo cliente forem implementadas, toda a documentação necessária do sistema é escrita e nenhuma outra mudança de arquitetura, design ou código é necessária, o projeto chega na fase de Morte.

Dentro do processo XP são definidos diferentes atores para tarefas e propósitos diferentes. Os papéis de cada ator foram definidos por Beck (2001) e resumidos por Abrahamsson, *et al,* em 2002, conforme abaixo:

- **Programador** – O programador desenvolve o programa e os testes do modo mais simples e definitivo o possível. A primeira questão que torna o XP bem-sucedido é se comunicar e coordenar com outros programadores e membros da equipe de forma eficaz.
- **Cliente** – O cliente escreve as histórias e os testes funcionais, e decide quando cada requisito está satisfeito. O cliente define também a prioridade de implementação para os requisitos.
- ***Tester*** – O *tester* ajuda o cliente a escrever os testes funcionais. Eles executam testes funcionais regularmente, transmitem os resultados e mantêm as ferramentas de teste.
- ***Tracker*** – O *tracker* é responsável pelo *feedback* no XP. Ele traça as estimativas feitas pela equipe e dá *feedback* sobre a precisão delas, a fim de melhorar as estimativas futuras. Ele também rastreia o progresso de cada iteração e avalia se a meta é alcançável dentro das restrições de tempo e recursos fornecidas ou se são necessárias alterações no processo.
- ***Coach*** – O *coach* é a pessoa responsável pelo processo como um todo. Uma boa compreensão do XP é fundamental nesta função, permitindo que o coach guie o outros membros da equipe no decorrer do processo.
- **Consultor** – O consultor é um membro externo conhecimentos

técnicos específicos necessários ao projeto. O consultor auxilia a equipe a solucionar problemas específicos.

- **Gestor** – O gestor toma as decisões. Para poder realizar esta função, ele se comunica com a equipe do projeto para determinar a situação atual e para distinguir quaisquer dificuldades ou deficiências no processo.

Para Khan, Qureshi e Khan (2011) o modelo XP, com base em suas características, é indicado para projetos de pequeno-médio porte, com alto envolvimento do cliente. O modelo coloca alta importância na coesão da equipe e seu foco é o produto final. Os autores também destacam que a sua aplicação pode não se adequar a projetos maiores onde a documentação é essencial. Além disso, ressaltam que a falta de documentação do modelo é um problema frequente em projetos, e que o modelo de programação em pares é dispendioso. Os autores vão ao ponto de afirmar que, quando o método XP falha, geralmente é devido a falta de documentação de suas técnicas.

3.1.6 *Scrum*

Scrum é um processo iterativo e incremental para o desenvolvimento de qualquer produto ou gestão de qualquer projeto de software. *Scrum* foca em como membros de uma equipe devem trabalhar para produzir a flexibilidade de sistema necessária para funcionarem em um ambiente em constante mudança (AWAD, 2005). Segundo Schwaber e Beedle (2002), a primeira referência ao termo *'Scrum'* na literatura é de um artigo de Takeuchi e Nonaka (1986) chamado '*The New New Product Development Game'* ("O Novo Jogo Para Desenvolvimento de Novos Produtos", tradução livre), onde é apresentado um processo de desenvolvimento de produto adaptativo, rápido e auto-organizado, originado no Japão (Abrahamsson, *et al,* 2002). O termo *scrum* foi retirado do esporte de equipes de *rugby*, onde o *scrum* é um método de reinício de jogada, onde os dois times se enfrentam de cabeça baixa e um time deve empurrar o outro para retomar a bola, com a equipe inteira

tendo que trabalhar em conjunto para alcançar o objetivo.

A primeira versão do *Scrum* foi apresentada em 1995 por Ken Schwaber e Jeff Sutherland. Desde 2009 um documento público chamado *The Scrum Guide* (O Guia *Scrum*, tradução livre) é mantido por Schwaber e Sutherland. A última atualização ao documento foi feita em 2017.

No *Scrum*, projetos são divididos em breves intervalos de trabalho, conhecidos como *sprints.* Cada *sprint* possui tipicamente de uma a quatro semanas de duração. O objetivo do *Scrum* é fornecer o máximo de software de qualidade possível dentro de uma série (3-8) de *sprints* (Beedle, *et al, 1998).* A duração de uma *sprint* é definida no início do projeto e nunca estendida, cada uma resultando em um produto potencialmente utilizável (KHAN; QURESHI; KHAN, 2011). A duração dos *sprints* varia de acordo com diferentes autores, mas nunca passa de 30 dias.

O processo *Scrum* envolve um *Scrum Master*, o Dono do Produto ou cliente e a Equipe *Scrum*. O papel principal do *scrum* master é eliminar impedimentos. A Equipe *Scrum* é uma multidisciplinar, composta de desenvolvedores, testers e outros especialistas de diversos campos necessários para o projeto (SRIVASTAVA; BHARDWAJ; SARASWAT, 2017). O método não requer ou fornece qualquer método/prática específica de desenvolvimento de software a ser utilizada. Em vez disso, requer certas práticas de gerenciamento e ferramentas em diferentes fases do *Scrum* para evitar o caos gerado por imprevisibilidade e complexidade (AWAD, 2005).

As práticas chave do método foram descritas por Ken Schwaber e Mike Beedle em 2002 no livro *Agile Software Development with Scrum* e resumidas por Awad (2005), conforme abaixo:

- *Backlog* do Produto*:* Uma lista priorizada de todas as funcionalidades e mudanças que ainda devem ser feitas no no sistema. O Dono do Produto é o responsável por manter esta lista.
- *Sprints:* Os *sprints* são ciclos de desenvolvimento de no máximo 30 dias, que devem produzir uma funcionalidade ou incremento no

sistema. As ferramentas da equipe são as reuniões de planejamento de *Sprints*, *Backlog* de *Sprint* e reuniões *Scrum* diárias.

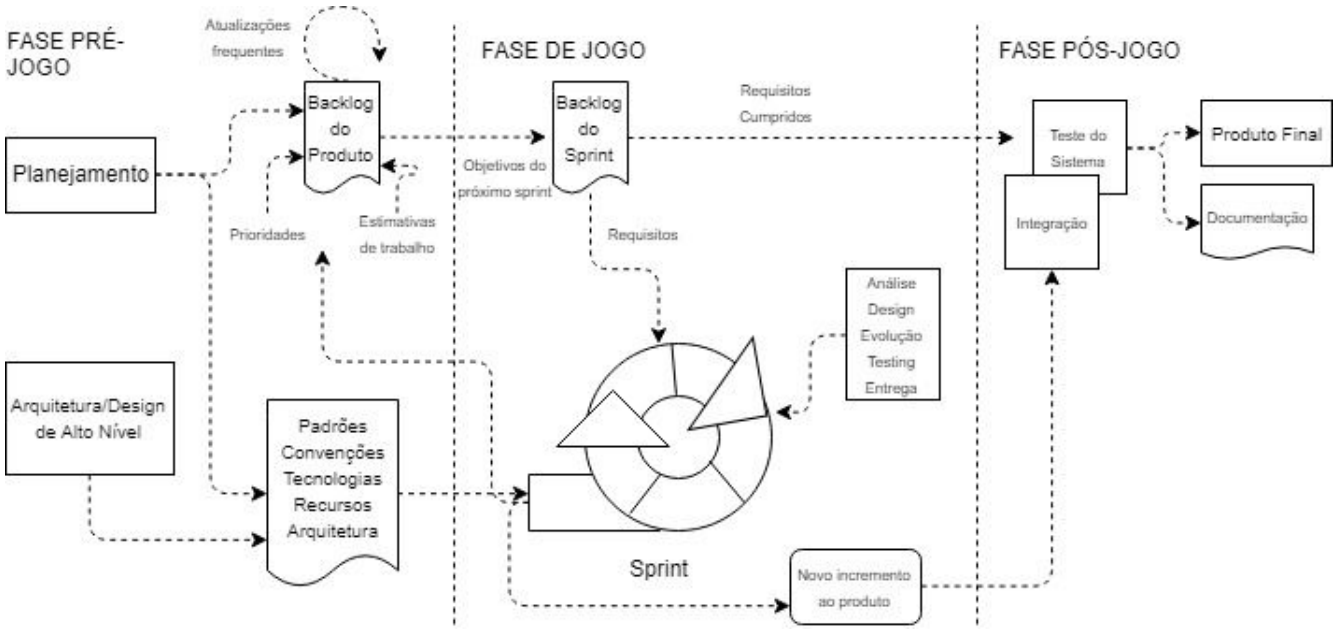
- Reunião de Planejamento de *Sprint*: As reuniões de planejamento de *Sprint* são realizadas juntamente com os clientes, usuários, a gerência, o Dono do Produto e a Equipe *Scrum*, onde um conjunto de objetivos e funcionalidades são definidos para o *sprint*. Em seguida, o *Scrum* Master e a Equipe *Scrum* definem como o produto será implementado durante o *sprint*.
- *Backlog* do *Sprint*: É a lista de funcionalidades determinadas para um *sprint*. Quando todas as funcionalidades estiverem completas, uma nova iteração do sistema é entregue.
- *Scrum* Diário: É uma reunião diária de aproximadamente 15 minutos, que tem o objetivo de acompanhar o progresso da Equipe *Scrum* e abordar quaisquer problemas ou obstáculos encontrados pela equipe.

O processo *Scrum* (Figura 4) foi definido por Schwaber em 1995 e dividido em três fases:

- ***Pregame* (Pré-jogo):**
  - Planejamento - Nesta fase é criado o *Backlog* do Produto, uma lista contendo todos os requisitos conhecidos até o momento. Esta lista é constantemente atualizada com itens mais novos e detalhados e novas ordens de prioridades. Nesta fase também são definidas as equipes, ferramentas e outros recursos, além de avaliações de riscos e questões de controle e treinamentos necessários (Abrahamsson, *et al,* 2002).
  - Arquitetura - Aqui será projetado como os itens do *backlog* serão implementados. Esta fase inclui modificação na arquitetura do sistema e design de alto nível (SCHWABER, 1995). Uma reunião de revisão do projeto é realizada para discutir as propostas para a implementação e decisões são tomadas com base nesta revisão. Além disso, os planos preliminares para o conteúdo dos lançamentos são preparados (Abrahamsson, *et al,*

2002).

Figura 4: Processo *Scrum*



Fonte: Adaptado de Abrahamsson, *et al* (2002)

- ***Game* (Jogo):**
    - *Sprints* de Desenvolvimento - A fase de desenvolvimento (também chamada de fase do jogo) é a parte ágil do Abordagem *Scrum* (Abrahamsson, *et al,* 2002). Aqui serão realizados os *sprints* de desenvolvimento que, com constante respeito às variáveis de tempo, requisitos, qualidade, custo e competição, resultaram no lançamento de uma nova funcionalidade do sistema (SCHWABER, 1995). Cada *sprint* inclui as fases tradicionais do desenvolvimento de software: levantamento de requisitos, análise, design, evolução e entrega. A arquitetura e o design do sistema evoluem durante o os *sprints* de desenvolvimento (Abrahamsson, *et al,* 2002). Quando todos os requisitos definidos no *backlog* de produto forem cumpridos, esta fase chega ao fim.
- ***Postgame* (Pós-jogo):**

- Término - Quando todos os requisitos definidos forem cumpridos e não existem mais melhorias a serem realizadas ou problemas a serem resolvidos, o projeto entra nesta fase. O sistema está pronto para ser entregue e a preparação para isto é feita nesta fase, com as etapas de integração, testes de sistema e documentação (Abrahamsson, *et al,* 2002).

Schwaber (1995) também define uma lista de tarefas a serem realizadas em cada fase:

- **Planejamento:**
  - Levantamento do *backlog* de produto
  - Definição da data de entrega e funcionalidades dos lançamentos
  - Seleção do lançamento mais apropriado para desenvolvimento imediato
  - Mapeamento de pacotes de produtos (objetos) para itens de *backlog* na versão selecionada.
  - Definição de equipe(s) de projeto para a construção da nova versão
  - Avaliação e controles de riscos apropriados
  - Revisão e possível ajuste de itens e pacotes de *backlog*
  - Validação ou seleção de ferramentas de desenvolvimento e infraestrutura
  - Estimativa do custo de lançamento, incluindo desenvolvimento, material de garantia, marketing, treinamento e distribuição
  - Verificação da aprovação da gestão e financiamento
- **Arquitetura:**
  - Revisão dos itens do *backlog* de produto
  - Identificar as alterações necessárias para implementar os itens do *backlog*
  - Realizar análise de domínio na medida necessária para criar, aprimorar e atualizar modelos de domínio para refletir o novo contexto e requisitos do sistema

    - Refinar a arquitetura do sistema para suportar o novo contexto e requisitos
    - Identificar quaisquer problemas no desenvolvimento ou implementação das mudanças
    - Reunião de revisão de projeto, cada equipe apresentando abordagens e mudanças para implementar cada item do *backlog*. Atribuir alterações conforme necessário
- **Desenvolvimento:** Esta fase é um ciclo iterativo de trabalho de desenvolvimento. A gerência determina quando os requisitos de tempo, concorrência, qualidade e funcionalidades foram atendidos, as iterações são concluídas e o projeto entra na fase de encerramento. Essa abordagem também é conhecida como Engenharia Concorrente (SCHWABER, 1995). O desenvolvimento consiste nos seguintes macro processos:
    - Reunião com a(s) equipe(s) para revisar os planos de lançamento
    - Distribuição, revisão e ajuste dos padrões com os quais o produto irá conformar
    - *Sprints* iterativos, até que o produto seja considerado pronto para distribuição
- **Término:** Quando a gerência do projeto determina que todos os requisitos definidos no *backlog* do produto foram cumpridos e o produto não necessita de mais ajustes ou correções, o ciclo de *sprints* se encerra e o projeto entra nesta fase. Segundo Schwaber (1995), esta fase prepara o produto desenvolvido para lançamento. Integração, teste do sistema, documentação do usuário, preparação de material de treinamento e marketing são realizados nesta fase.

Conforme mencionado anteriormente, um *sprint* é um período onde um conjunto de atividades serão realizadas, com duração geralmente de uma a quatro semanas (máximo de 30 dias). Segundo Schwaber (1995), o intervalo é baseado na complexidade do produto, avaliação de risco e grau de supervisão desejado. A

velocidade e a intensidade da *sprint* são impulsionadas pela duração selecionada do *sprint*. O risco é avaliado continuamente e controles de risco e respostas adequados são colocados em prática. Cada *Sprint* consiste em uma ou mais equipes executando o seguinte:

- **Desenvolvimento** - Definir as alterações necessárias para a implementação de requisitos de *backlog* em pacotes, abrindo os pacotes, realizando análise de domínio, projetar, desenvolver, implementar, testar e documentar as mudanças. Desenvolvimento consiste nos micro processos de descoberta, invenção e implementação
- **Encerramento** - Fechando os pacotes, criando uma versão executável das mudanças e como eles implementam os requisitos do *backlog*
- **Revisão** - Todas as equipes se reúnem para apresentar o trabalho e analisar o progresso, levantando e resolvendo problemas, adicionando novos itens ao *backlog*. O risco é revisado e respostas apropriadas definidas
- **Ajustes** - Consolidar as informações coletadas da reunião de revisão nas áreas afetadas pacotes, incluindo aparência e propriedades diferentes

3.1.7 Desafios de Implementar uma Metodologia Ágil

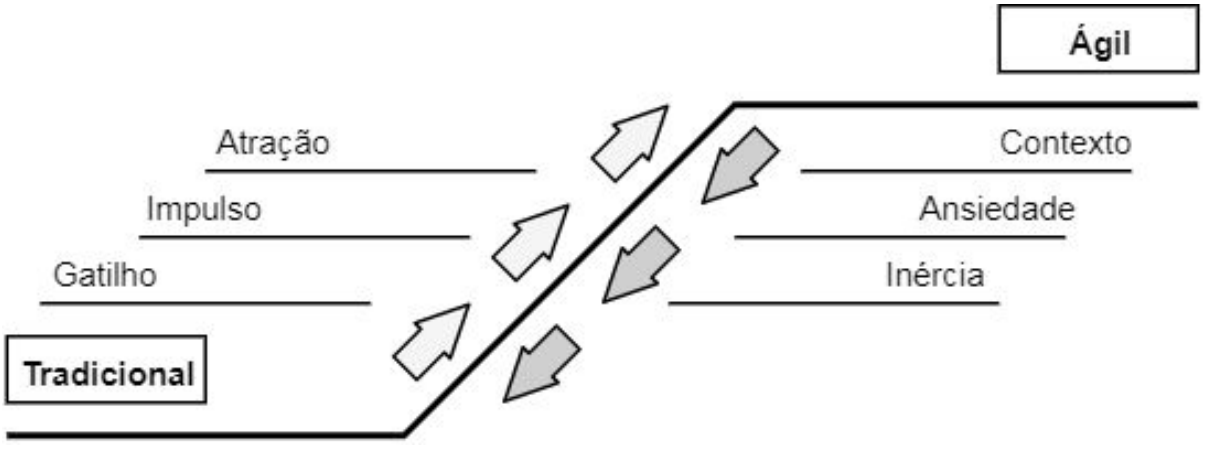
Segundo Verret (2018), embora os benefícios do Método Ágil tenham sido bem documentados, a implementação de métodos ágeis pode ser desafiadora (SVENSON, HOST, 2012). Schwaber, Laganza e D'Silva (2007) relataram que os autores de pesquisas sobre a implementação de métodos ágeis observaram que os adotantes ágeis geralmente desconhecem o que realmente significa adotar um método ágil em sua organização e muitas vezes desconhecem a amplitude da mudança necessária. Pesquisas recentes sobre desafios comuns que as organizações podem encontrar ao tentar introduzir sistemas ágeis em um ambiente não-ágil revelaram diversas áreas preocupantes (GREGORY, et al., 2015). Problemas incluem relutância em fazer as mudanças necessárias nos processos do projeto que a adoção de metodologias ágeis exige, mal-entendidos pela gerência e outras pessoas na organização quanto ao significado do termo Ágil e seus

processos associados, desafios com a comunicação entre a(s) equipe(s), falta de confiança entre a gerência do projeto e a equipe de desenvolvimento e discordâncias ao priorizar requisitos (GREGORY et al., 2015).

Outros desafios incluem disponibilidade limitada do cliente, documentação mínima de aplicativos e projetos, orçamento incorreto e estimativas de tempo resultantes de solicitações de mudanças, limitações contratuais que não permitem ajustes não especificados no projeto e discordâncias quanto aos requisitos funcionais (INAYAT et al., 2015). Desafios específicos ao gerenciamento incluem a incapacidade dos gerentes em deixar de microgerenciar os projetos e até mesmo o uso do termo Ágil como uma desculpa para pressionar as equipes a trabalharem mais (GREGORY et al., 2015).

Um estudo realizado por Hohl et al. (2016) identificou seis categorias de forças que agem durante a transição de métodos tradicionais para métodos ágeis. Os autores caracterizam como "gatilho" ("*trigger")*, "impulso" (*"push")* e "atração" (*"pull")* as forças que levam à adoção de métodos ágeis e, em contraste, "inércia", "ansiedade" e "contexto" como forças opostas à adoção (figura 5).

Figura 5: Forças agindo na adoção de uma Metodologia Ágil



Fonte: Adaptado de Hohl, *et al* (2016)

Segundo Hohl et al. (2016, p.471):

> Uma força gatilho inicia uma mudança e empurra um indivíduo ou uma organização em direção à adoção de um método ágil. Uma força de impulso empurra um indivíduo ou uma organização no sentido de adotar práticas ágeis baseadas em questões ou demandas. Uma força de tração vem entrar

> em vigor quando indivíduos ou organizações são atraídos para a adoção ágil com base na atratividade de uma situação futura. Forças de inércia, como hábitos, impedem as pessoas de experimentar algo novo e portanto, impede a adoção ágil. As forças da ansiedade representam medos que poderiam impedir a adoção do método ágil. Muitas vezes, as incertezas em torno de novas situações causam ansiedade. As forças do contexto resultam de restrições e obstáculos no ambiente (como estruturas organizacionais ou barreiras do processo) e impede a transição.

Para realizar este estudo, os autores (Hohl et al., 2016) selecionaram um grupo de 14 profissionais da área para realizar uma série de entrevistas. A escolha dos entrevistados foi realizada com base em dois critérios: primeiro, o entrevistado deveria ter uma experiência de trabalho de vários anos. A o tempo de experiência variou entre 3 a 20 anos, com uma experiência de trabalho média de 16 anos. Segundo, o entrevistado já deveria utilizar práticas ágeis para desenvolvimento de software. Com base nestas entrevistas, foram identificados não apenas as forças em ação durante a transição para o método Ágil, mas também os principais desafios (e possíveis abordagens para solucioná-los) ligados a estas forças (Quadro 2).

O processo de transição para um método ágil é complexo e requer muito esforço da gestão e equipes, adaptação cultural, lida com egos e resistência à mudança e exige patrocínio da alta gerência (CAMPANELLI, PARREIRAS, 2015). De tal maneira, é compreensivo que organizações sintam-se intimidadas com o processo, especialmente se já passaram por tentativas mal-sucedidas. Campanalli e Parreiras (2015) concluem que a escolha da estratégia para a adoção de métodos ágeis é um componente chave para que a organização aproveite os benefícios trazidos pelo método Ágil e supere os problemas comuns encontrados no processo de transição.

Quadro 1: Desafios e Abordagens para Soluções

Forças	Desafios	Abordagem da Solução
Inércia	Falta de compreensão do aplicabilidade de métodos ágeis em um contexto específico	Organizar treinamento ágil específico ao contexto
	É necessária uma mudança de mentalidade para adotar práticas ágeis	Colaborar com especialistas (externos)
	É necessário mais esforço de comunicação	Recomendações de boca a boca e cultura de feedback
	Processo de desenvolvimento atual é visto como satisfatório	Explicar os benefícios do método ágil
	Aceitação limitada para reestruturação organizacional	Integração lenta e gradual do método ágil
Ansiedade	A gerência não quer ceder responsabilidades	Redefinir a função da gerência; desenvolvedores devem receber mais responsabilidade
	Falta de clareza sobre como fornecer estimativas corretas sobre os esforços de desenvolvimento ao aplicar práticas ágeis	Encurte o intervalo de estimativas
	Tema que defeitos relevantes ao cliente permaneçam no software entregue	Fornecer um ambiente seguro para falhas e desenvolver protótipos
Contexto	Processos rígidos e inflexíveis	Definir interface de transição entre método ágil e processos tradicionais
	Hierarquia na organização	Transforme hierarquias em networks
	Sincronização frequente com fornecedores e controle de qualidade	Mais desenvolvimento de software interno
	Desenvolvimento de software distribuído globalmente	Uso de simulações que substituem partes indisponíveis
	Manter os benefícios da reutilização de software	Reorganizar o desenvolvimento da linha de produtos

Fonte: Adaptado de Hohl, *et al* (2016)

3.1.8 Adotando uma Metodologia Ágil

A transição do desenvolvimento de software tradicional para um método Ágil

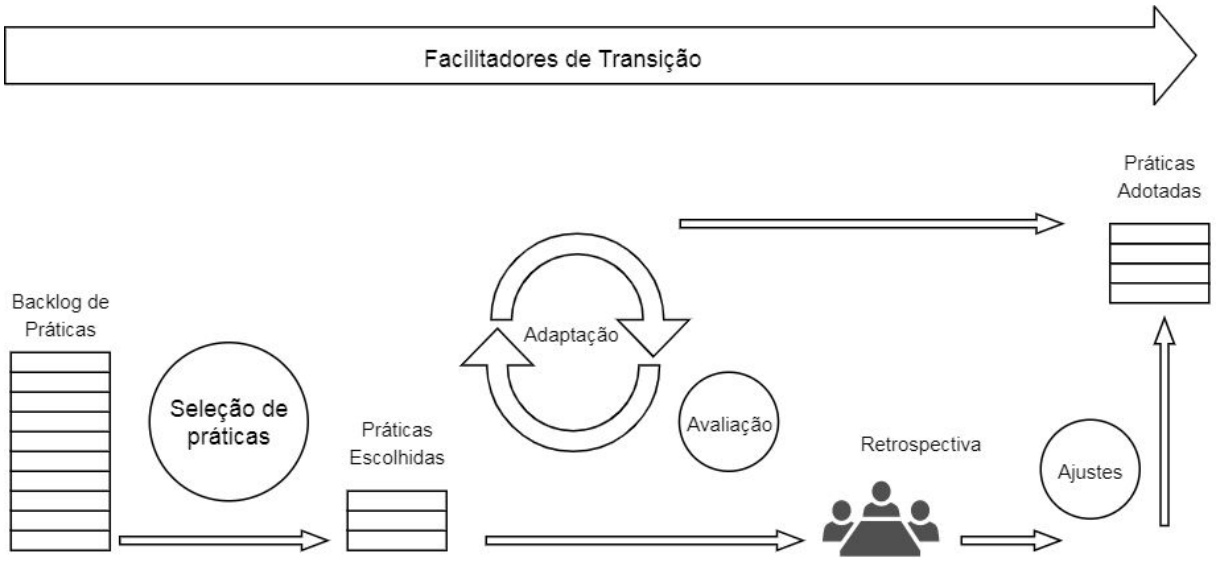
exige é uma tarefa complexa e laboriosa. Deixar os métodos tradicionais e adotar um método Ágil, um processo conhecido como Processo de Transformação/Transição Ágil (PTA, do original *Agile Transformation/Transition Process*, ATP), afeta quase todos os aspectos das empresas de software e requer colaboração e envolvimento praticamente todos os stakeholders envolvidos no processo, como desenvolvedores, gerentes de nível médio e superior e clientes (GANDOMANI; NAFCHI, 2015).

Segundo Gandomani e Nafchi (2015), até o momento, poucos frameworks de transição e adoção Ágeis foram propostas na indústria de software. No entanto, na prática, não é fácil utilizá-los e requer principalmente uma enorme sobrecarga organizacional devido à sua estrutura complexa e não-flexível. Essas desvantagens dificultam a aplicação de tais estruturas em pequenas e médias empresas.

Segundo os autores, para compreender o PTA, é preciso que algumas perguntas sejam respondidas: como que empresas e equipes de software gerenciam a transição para o Ágil? O que é um *framework* eficaz e aplicável para a transição? Quais devem ser suas funções e características? Quais devem ser suas atividades? E por fim, qual deve ser o nível de abstração do *framework* de transição?

Para responder a essas perguntas, os autores utilizaram uma abordagem baseada na metodologia *Grounded Theory* (GT) para realizar um estudo com 49 praticantes de métodos ágeis de 13 países diferentes, sendo que todos haviam experiência prévia em métodos ágeis e já haviam participado em algum papel de um PTA. Foi realizada uma revisão de literatura, examinando as estruturas ágeis propostas anteriormente para se discutir os méritos e falhas relativos de cada um. Ao final deste estudo, os autores propuseram o *Agile Transition and Adoption Framework* ou *framework* para Adoção e Transição Ágil (Figura 6).

Figura 6: Processo do *Agile Transition and Adoption framework*



Fonte: Adaptado de Gandomani e Nafchi (2015)

Segundo Verret (2018), que resumiu o processo, as práticas ágeis que a equipe deseja adotar são mantidas e priorizadas no *Backlog de Práticas*; as equipes selecionam uma ou mais práticas do *backlog* durante a *Seleção de Práticas* e inicia o processo de *Adaptação* das práticas selecionadas para introduzir a nova prática em sua próxima iteração de desenvolvimento. Depois que a iteração for concluída, uma *Avaliação* é realizada para avaliar seus sucessos e desafios na adoção da nova prática ágil. A equipe discute sua experiência na *Retrospectiva* e *Ajustes* são feitos conforme necessário. Depois de concluídas, as *Práticas Adotadas* agora fazem parte da metodologia ágil utilizada pela equipe. Esse processo é repetido em etapas iterativas sempre que necessário. Além disso, os autores afirmam que *Facilitadores de Transição,* como treinamentos, envolvimento da gerência, reuniões e negociações contínuas, mentoria, entre outros, podem ser introduzidos em todos os passos do processo para auxiliar a transição.

Neste capítulo foi realizada a revisão de literatura utilizada como base para a criação do questionário aplicado e análise dos dados obtidos. Nele foi discutido um

breve histórico da prática de desenvolvimento de software, algumas das principais metodologias ágeis em uso atualmente e as dificuldades envolvidas na implementação de uma metodologia encontradas na literatura.

4. ANÁLISE E DISCUSSÃO DOS RESULTADOS

Para identificar os problemas enfrentados pela gestão de projetos de desenvolvimento de software, foi aplicado em 9 empresas de micro e pequeno porte, um questionário com 10 questões. As questões (veja Anexo A) estão alicerçadas em cinco pilares que são: i) a tentativa de identificar se os envolvidos no processo de desenvolvimento de software dentro da empresa tem conhecimento do que é uma metodologia de desenvolvimento; ii) se as equipes possuem um gestor de projeto; iii) quais são as funções desempenhadas pelo gerente de desenvolvimento de software; iv) se os projetos têm a cultura de empregar metodologia ágil de software; v) verificar os problemas enfrentados pelos participantes no processo de desenvolvimento, se em suas organizações

Para tentar compreender o nível de conhecimento dos participantes referente ao conceito de metodologia ágil de desenvolvimento, foi aplicada uma pergunta de resposta livre: "*Para você, o que é uma metodologia de desenvolvimento de software?*". Ao final do questionário, foi dada uma pergunta de múltipla escolha pedindo para o participante avaliar o seu próprio conhecimento sobre metodologias, sendo:

A) Não tenho conhecimento;
B) Conhecimento básico;
C) Conhecimento intermediário;
D) Sou especialista;

De maneira similar, foi criado uma escala de nível de conhecimento para avaliar as respostas, considerando conceitos chaves, técnicas e objetivos das metodologias de desenvolvimento:

A) **Não tem conhecimento** - Possui uma noção extremamente simplista (ou incorreta) do que é uma metodologia, desconhece seus objetivos;
B) **Conhecimento básico** - Possui um entendimento básico do conceito de metodologia, tem uma noção geral do seu objetivo;
C) **Conhecimento Intermediário** - Compreende o conceito geral de

metodologia, conhece algumas de suas práticas e objetivos;

D) **Especialista** - Possui domínio sobre o conceito de metodologias de desenvolvimento e conhece as suas práticas e objetivos;

Utilizando esta ferramenta e, desconsiderando as respostas em branco, levantamos os seguintes dados (Quadro 03):

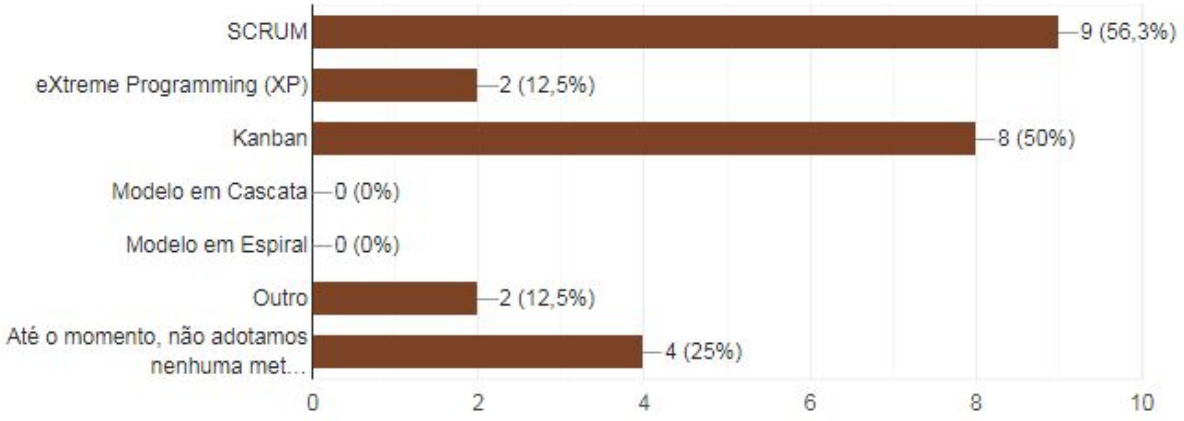
- 75% dos participantes não possuem conhecimento sobre metodologias(nota A) ou possuem um conhecimento rudimentar (nota B);
- Um participante possui conhecimento intermediário sobre o assunto (6,25%) - nota C;
- Um participante apresentou domínio sobre o conceito (nota D) e conhecimento de suas técnicas e objetivos (6,25%) - este participante afirmou ser especialista no assunto;
- Dos participantes, 31,58% ocupam cargos de gestão e 68.42% são desenvolvedores;
- Dentre os gestores, apenas um participante (25%) apresentou domínio sobre o assunto (nota D), o restante não possui conhecimento (50%) ou conhecimento básico (25%);

Quadro 02: Avaliação da compreensão dos participantes quanto ao conceito de Metodologia de Desenvolvimento

Participante	Função	Resposta	Nota	Avaliação Própria
1	Desenvolvedor	"É a forma que deve ser seguida no processo de desenvolvimento. "	A	C
2	Desenvolvedor	"É um padrão de desenvolvimento para ser seguido durante o ciclo de vida do software"	A	C
3	Desenvolvedor	"Conjunto de técnicas e guias com o intuito de evitar problemas comuns e melhorar a qualidade final do software"	B	C
4	Gerente de Produto/Projeto	"Um método, modelo ou framework - do qual já foi aplicado e testado em organizações - que promove o engajamento do time de desenvolvedores de software para que realizem entregas mais assertivas, evoluindo a maturidade, velocidade e melhoria contínua do time."	D	D
5	Gerente de Produto/Projeto	"Uma série de processos e rotinas que organiza e estrutura a forma como pessoas de diferentes papéis (designers, desenvolvedores, PMs, testers, etc) trabalham de forma conjunta em prol de um objetivo (nesse caso, a descoberta, construção e entrega de um software)."	B	C
6	Desenvolvedor	"Forma de trabalho de maneira correta e mais simples, sem ficar amarrado em requisitos desatualizado"	A	B
7	Desenvolvedor	"Um processo organizado visando um resultado específico "	A	B
8	Desenvolvedor	*Não respondeu*	-	B
9	Desenvolvedor	"Conjunto de processos para organizar o trabalho dos desenvolvedores e garantir a qualidade do software produzido."	B	B
10	Gerente de Produto/Projeto e Desenvolvedor	"Uma maneira "padronizada" de se desenvolver um software, com os principais "porquês" respondidos."	A	C
11	Desenvolvedor	"Conjunto de princípios e métodos padronizados entre a equipe de desenvolvimento de software, e integrados com os demais processos da organização."	C	C
12	Desenvolvedor	"São padrões de desenvolvimento pré estabelecidos adotados por equipes de desenvolvimento. Eles visam minimizar problemas comuns em projetos e garantir uma melhoria na criação, implementação e entrega.	B	C
13	Desenvolvedor	*Não respondeu*	-	C
14	Analista de Projetos/Diretor de Empresa	"É um conjunto de práticas e técnicas utilizadas para o desenvolvimento de um software"	A	B
15	Desenvolvedor	"Vejo como um conjunto de práticas que uma equipe tem para manter-se em comunicação sobre o direcionamento, desenvolvimento e dificuldades encontradas para a construção de um software visando mitigar os problemas de uma construção díspar dos módulos e/ou integração com outros sistemas além de manter o cliente informado sobre o desenvolvimento e situações importantes encontradas ao longo do trabalho (transparência)."	A	C
16	Desenvolvedor	"São padrões utilizados para o desenvolvimento mas a eficiente"	A	B

Embora a maioria demonstre conhecimento insuficiente, 75% afirmam utilizar alguma metodologia de desenvolvimento e também 75% utilizam algum método ágil (ou mais de um). A Figura 7 apresenta as metodologias mais utilizadas dentre os participantes. Podemos observar que dentre as metodologias o *Scrum* se destaca com 56% de aplicação.

Figura 7: Metodologias de desenvolvimento utilizadas pelos participantes

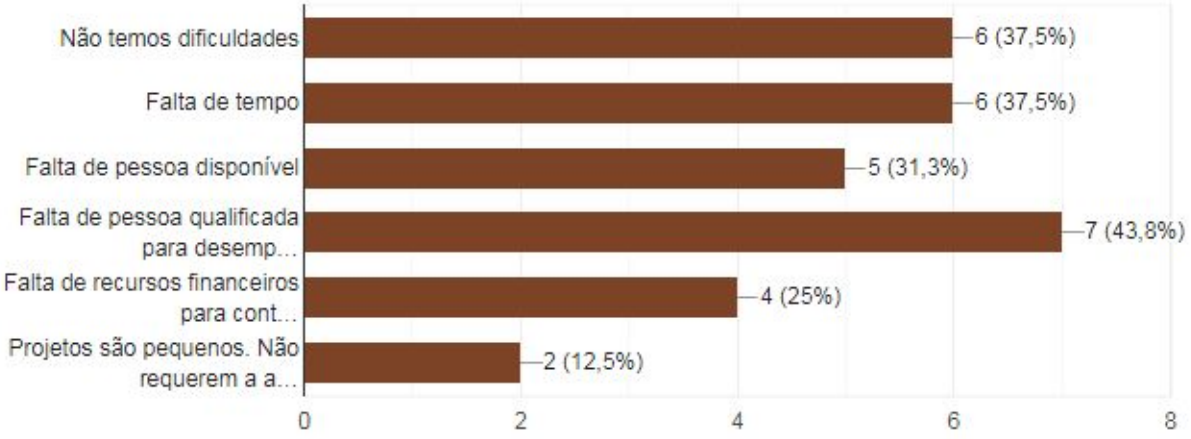


Para verificar as possíveis dificuldades ou desafios referentes à aplicação de uma metodologia de desenvolvimento de software, foi feita a seguinte pergunta de múltipla escolha: "*Dentre as dificuldades para a aplicação de uma metodologia de desenvolvimento de software, você destacaria:"*, com o participante podendo escolher dentre as opções:

- Falta de tempo - 6 respostas (37,5%);
- Falta de pessoal disponível - 5 respostas (31,1%);
- Falta de pessoa qualificada para desempenhar a função - 7 respostas (43,8%);
- Falta de recursos financeiros para contratar um especialista - 4 respostas (25%);
- Projetos são pequenos. Não requerem a aplicação de uma metodologia de desenvolvimento - 2 respostas (12,5%);
- Não temos dificuldades - 6 respostas (37,5%);

Os resultados dessa pergunta (Figura 8) são similares a fatores críticos de sucesso e dificuldades frequentemente mencionadas na literatura (VERRET, 2018).

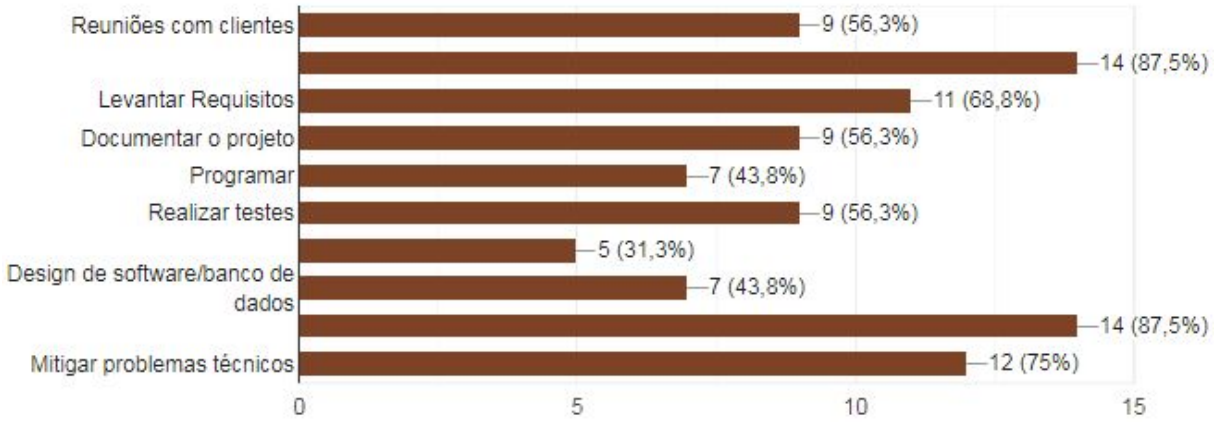
Figura 8: Dificuldades na aplicação de metodologias de desenvolvimento



Procurando entender o papel do gestor de projetos ou principal *stakeholder* do projeto e verificar como o seu papel pode ser um dos fatores que influencia a adoção de metodologias, foi realizada uma pergunta de múltipla escolha solicitando que os participantes marcassem as funções de seu gestor (Figura 9). As opções eram:

- Reuniões com clientes - 9 respostas (56,3%);
- Acompanhar o desenvolvimento do software - 14 respostas (87,5%);
- Levantar Requisitos - 11 respostas (68,8%);
- Programar - 7 respostas (43,8%);
- Realizar testes - 9 respostas (56,3%);
- Apresentar resultados para os clientes - 5 respostas (31,3%);
- Design de software/banco de dados - 7 respostas (43,8%);
- Identificar problemas na equipe - 14 respostas (87,5%);
- Mitigar problemas técnicos - 12 respostas (75%);

Figura 9: Atividades desempenhadas pelo gestor de projetos ou principal *stakeholder*



Com base nas respostas dos participantes, percebe-se que muitos gestores dedicam seu tempo a atividades não relacionadas a gestão, como realização de testes, programação, documentação e design de software/banco de dados. Um dos fatores citados na literatura como um desafio na adoção de uma metodologia é a incapacidade dos gerentes em deixar de microgerenciar os projetos. Outro dado levantado que é interessante para esta pesquisa é: todos os participantes que afirmaram não utilizar uma metodologia de desenvolvimento de software também afirmaram que não há um gerente de software atuando em seus projetos. Sendo assim, é possível perceber a existência de uma correlação entre a presença de um gerente de software e a adoção de metodologias de software.

Além disso, foi feita uma pergunta com o intuito de levantar dificuldades encontradas pelos participantes durante o processo de desenvolvimento, com o objetivo de verificar se há alguma relação com as dificuldades de implantar uma metodologia (Figura 10). As opções eram:

- Estimativas incorretas para com o tempo de desenvolvimento de uma tarefa - 10 respostas (62,5%);
- Requisitos pouco claros - 14 respostas (87,5%);
- Perda de colaboradores - 4 respostas (25%);
- Falta de informações do projeto - 9 respostas (56,3%);
- Integração com outras tecnologias - 11 respostas (68,8%);
- Problemas levantados tardiamente - 10 respostas (62,5%);
- Atraso nas entregas de software - 5 respostas (31,3%);
- Renegociação de prazos - 4 respostas (25%);

- Falta de autonomia para implantar mudanças/tomar decisões - 4 respostas (25%);

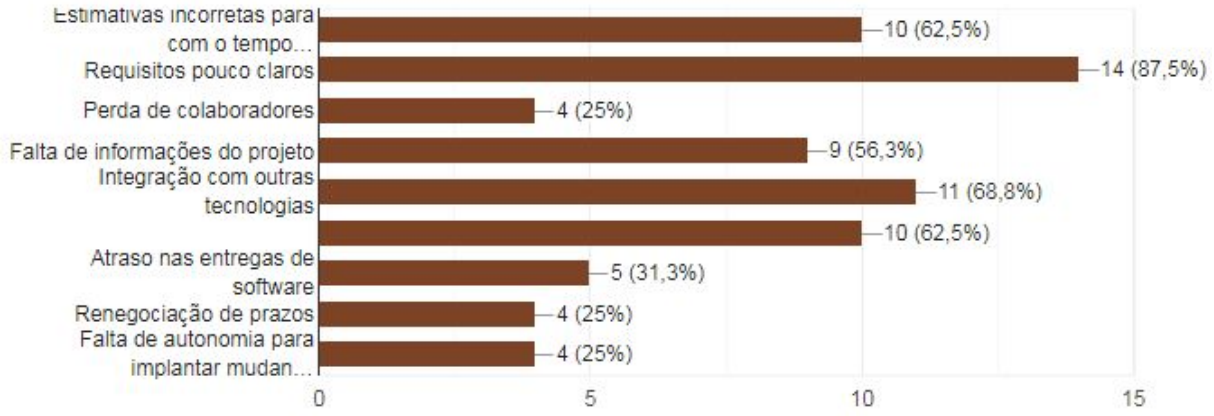
Dentre as dificuldades mais citadas pelos participantes, algumas se destacam justamente como dificuldades que os métodos ágeis buscam resolver.

O maior problema identificado nesta questão foi o de "*Requisitos pouco claros"*. Este é um problema comum, muitas vezes derivado de problemas de comunicação entre o cliente e a empresa. Um dos quatro pilares do Manifesto Ágil está diretamente relacionado a isto: "*colaboração com o cliente* acima de negociação de contratos". Metodologias Ágeis como *Scrum* colocam o cliente como um stakeholder crucial no processo, que irá auxiliar na validação dos requisitos levantados e funcionalidades desenvolvidas. Beck (1999) também estabelece como um dos princípios do XP que a equipe de desenvolvimento deve ter acesso ao cliente a qualquer momento.

As entregas constantes também auxiliam neste quesito, pois além de demonstrar ao cliente que o processo de desenvolvimento está fluindo como programado, dá a ele oportunidade de validar se as funcionalidades cumprem seus requisitos no decorrer do projeto. Vale ressaltar que para que isto ocorra, a colaboração do cliente como um *stakeholder* do processo é de suma importância. O item "*falta de informações do projeto"* (56,3%), também está diretamente ligado com a interação entre o cliente e a equipe de desenvolvimento.

Os itens "*estimativas incorretas para com o tempo de desenvolvimento de uma tarefa"*, marcado por 62,5% dos participantes, e "*atraso nas entregas de software*", marcado por 31,3% dos participantes, são problemas que os métodos ágeis buscam sanar através de ciclos de desenvolvimento curtos de até 30 dias.

Figura 10: Dificuldades encontradas no processo de desenvolvimento de software



Com base nos fatores críticos de sucesso e nos desafios de adoção de métodos Ágeis levantados na revisão de literatura, nos resultados da revisão sistemática da literatura realizada por Verret (2018) referente aos desafios e melhores práticas relacionados à Metodologia Ágil e nos resultados da questionário aplicado neste trabalho, concluímos que os motivos que ainda fazem com que organizações ou gestores não adotem uma metodologia de desenvolvimento de software formalizada, apesar da vasta evidência existente quanto aos benefícios do uso de uma metodologia de desenvolvimento (especialmente no que se refere às metodologias Ágeis), são análogos às dificuldades e fatores de sucesso relatadas neste trabalho.

Considerando as conclusões chegadas por Verret e outros autores citados neste trabalho, levantei abaixo os desafios e problemas que ainda impedem a aplicação de uma metodologia de desenvolvimento de software, separados em duas categorias (semelhantes as forças definidas por Hohl et al. 2016): **inércia organizacional/resistência a mudanças** e **falta de conhecimento sobre metodologias de desenvolvimento ou como aplicá-las**.

Quanto aos motivos relacionados a inércia organizacional/resistência a mudanças, temos a seguinte lista de fatores críticos de sucesso levantadas por Verret:

- “**A inércia organizacional é uma força poderosa, e estratégias**

claras para superar a inércia devem ser desenvolvidas e mantidas" - Organizações temem que mudar seus processos pode influenciar negativamente o produto entregue ao cliente e/ou possuem processos muito rígidos; não acreditam que há valor o suficiente na mudança ou que não é necessário adotar uma metodologia de desenvolvimento, estão satisfeitos com o modo como trabalham.

- "**Investir em treinamentos paga dividendos**" - Aplicar uma metodologia de desenvolvimento é um processo complexo e requer um conhecimento sólido sobre o método em questão e como aplicá-lo. A contratação de um especialista é uma prática frequentemente citada na literatura como um fator crítico de sucesso na adoção de um método Ágil (DIKERT, PAASIVAARA e LASSENIUS, 2016; INAYAT, et al., 2015; SURESHCHANDRA e SHRINIVASAVADHANi, 2008). Uma organização pode considerar o gasto necessário para contratar um especialista ou capacitar um membro de sua equipe (que pode ser considerado alto ou até mesmo impraticável dependendo do tamanho e recursos da organização) como um impedimento para a adoção de um método.
- "**Indivíduos e equipes incentivados/empoderados são mais engajados; o sucesso com o método ágil depende do engajamento indivíduos e equipes envolvidos**" - Verret levanta como um fator crítico de sucesso a presença na organização de indivíduos motivados e engajados, que possuem espaço o suficiente em seu ambiente para direcionar a cultura da organização a uma mudança. Se os principais stakeholders de um projeto não possuirem motivação ou autonomia para aplicar mudanças, não haverá uma força instigadora que puxe a organização em direção a uma metodologia.

Em relação aos motivos relacionados a falta de conhecimento sobre metodologias ou como aplicá-las:

- "**Não existe uma abordagem única que funcione para todas as**

situações e/ou ambientes. As organizações devem decidir sobre uma estratégia ágil que funcione melhor para elas" - A falta de conhecimento sobre quais metodologias (e até mesmo sobre o próprio conceito do que é uma metodologia de software) pode fazer com que uma organização não consiga avaliar qual metodologia ou quais práticas melhor se adequa a sua situação e, por este motivo, não adotar metodologia nenhuma. Além disso, mesmo que uma metodologia seja escolhida (sendo esta compatível com a organização ou não) o processo de adoção de uma metodologia é complexo e intimidador. Se uma organização não tiver conhecimento sobre como navegar esta transição, poderá escolher por não realizar a mudança.

- "**A gerência deve entender os métodos ágeis para oferecer suporte a esses métodos**" - A falta de conhecimento sobre metodologias, seus objetivos e práticas pode levar gestores a terem percepções sem fundamentos em relação a possíveis resultados que a adoção de uma metodologia pode trazer, levando-os a considerar a adoção desnecessária ou não-benéfica a organização.

5. CONCLUSÃO

O objetivo principal deste trabalho foi identificar os desafios e problemas que ainda impedem a aplicação de metodologias de desenvolvimento de software por empresas ou gestores de projetos. Para cumprir este objetivo, um questionário foi aplicado, entre o período de 18 de setembro de 2019 até 10 de outubro de 2019, com profissionais da área de desenvolvimento da Grande Florianópolis, procurando levantar informações quanto ao uso de metodologias nas empresas e, se não são utilizadas, o porquê; as dificuldades envolvidas nos processos de desenvolvimento de software; as dificuldades enfrentadas pelos gestores no desenvolvimento de software; e as dificuldades identificadas pelos participantes quanto à aplicação das metodologias.

Foi realizada uma pesquisa na literatura referente às metodologias de desenvolvimento, buscando elucidar suas principais características, princípios e processos. Para isto, foi apresentado um breve histórico do desenvolvimento de software, contextualizando as motivações e os desafios que levaram a criação de metodologias formalizadas de desenvolvimento de software.

Foi buscado também na literatura pesquisas e estudos de caso que tinham como objetivo levantar dados quanto ao uso das metodologias de desenvolvimento, os benefícios de sua aplicação relatados por profissionais da área e as dificuldades encontradas por profissionais ao aplicar estas metodologias em empresas de TI. Estas pesquisas serviram como base para a avaliação dos dados obtidos com a aplicação do questionário e deram sustentação para formular uma lista hipóteses para responder a pergunta levantada por este trabalho.

5.1. Considerações Finais

Através deste trabalho foi possível levantar e classificar os principais motivos que contribuem para que organizações optem por não aplicar metodologias de desenvolvimento de software em seus projetos. Espera-se que os resultados aqui apresentados sirvam de auxílio para que empresas sejam capazes de identificar os

fatores em seu próprio contexto que as impedem de fazer uso destas importantes ferramentas de desenvolvimento de software.

Consideramos também que a literatura aqui apresentada, em especial no capítulo “Adotando uma Metodologia Ágil”, possa ser de utilidade para empresas que estão enfrentando o processo de aplicação de uma metodologia e não possuem uma estratégia clara de como fazê-lo.

Quanto a trabalhos futuros sobre o tema, pode-se considerar a expansão do questionário aplicado como um ponto de partida. Com as experiências adquiridas neste trabalho, poderia-se refinar a ferramenta de pesquisa e expandir sua aplicação. De um ponto de vista mais prático, poderíamos colocar a disposição os conhecimentos aqui levantados através de *workshops* sobre aplicação de metodologias de desenvolvimento para empresas ou alunos da área.

Por fim, acreditamos que este trabalho está perfeitamente alinhado com o espírito do curso de Gestão da Tecnologia da Informação e espera-se que ele agregue valor e sirva de incentivo para que outros estudantes possam trazer mais conhecimento ao IFSC sobre este tema tão importante da Gestão de TI.

REFERÊNCIAS

ABRAHAMSSON, Pekka; et al. **Agile Software Development Methods: Review and Analysis.**Espoo: Vtt, 2002. 107 p.

ANGELONI, Maria Terezinha. **Organizações do conhecimento: infraestrutura, pessoas e tecnologias.** São Paulo: Saraiva, 2002.

AKBAR, Muhammad Azeem et al. **Statistical Analysis of the Effects of Heavyweight and Lightweight Methodologies on the Six-Pointed Star Model**. Ieee Access, [s.l.], v. 6, p.8066-8079, 2018. Institute of Electrical and Electronics Engineers (IEEE).

AWAD, M. A.. A **Comparison between Agile and Traditional Software Development Methodologies**. 2005. 77 f. Tese (Doutorado) - Curso de Computer Science And Software Engineering, University Of Western Australia, Perth, 2005.

BALLE, Andrea Raymundo et al. **How do knowledge cycles happen in software development methodologies?** Industrial And Commercial Training, [s.l.], v. 50, n. 7/8, p.380-392, 3 set. 2018. Emerald

BECK, Kent et al. **Manifesto para Desenvolvimento Ágil de Software.** 2001. Disponível em: <http://agilemanifesto.org/>. Acesso em: 04 set. 2018.

BECK, Kent. **Embracing Change with Extreme Programming**. Computer, [s.l.], v. 32, n. 10, p.70-77, 1999. Institute of Electrical and Electronics Engineers (IEEE).

BEEDLE, Mike et al. **SCRUM: An extension pattern language for hyperproductive software development.** 1998. Disponível em:

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.15.3987&rep=rep1&type=pdf>. Acesso em: 19 nov. 2019.

BELL, D. (1973), The Coming of Post-Industrial Society: A Venture in Social Forecasting, **Basic Books**, New York, NY.

BOEHM, B. W. (1988). **A spiral model of software development and enhancement**. Computer, 21(5), 61–72.

BOISOT, M. Is your firm a creative destroyer? Competitive learning and knowledge flows in the technological strategies of firms, **Research Policy**, Vol. 24, pp. 489-506, 1995.

CAMPANELLI, Amadeu Silveira; PARREIRAS, Fernando Silva. **Agile methods tailoring – A systematic literature review**. Journal Of Systems And Software, [s.l.], v. 110, p.85-100, dez. 2015. Elsevier BV. http://dx.doi.org/10.1016/j.jss.2015.08.035.

CRESWELL, J.; CLARK, V. P. Designing and Conducting Mixed Methods Research. Thousand Oaks, CA: **Sage**, 2007.

CROWNE, M.. **Why software product startups fail and what to do about it. Evolution of software product development in startup companies**. Ieee International Engineering Management Conference, [s.l.], p.338-343, 2002. IEEE. http://dx.doi.org/10.1109/iemc.2002.1038454.

DAVENPORT, T.H.; PRUSAK, L. Working Knowledge: How Organizations Manage What They Know, **Harvard Business School Press**, Boston, MA, 1998.

DAVENPORT, T.H.; PRUSAK, L. Working Knowledge: How Organizations Manage What They Know, **Harvard Business School Press**, Boston, MA, 1995.

DAVENPORT, T.; PRUSAK, L. **Conhecimento Empresarial**: como as organizações gerenciam o seu capital intelectual. 15º edição. Tradução de Lenke Peres. Rio de Janeiro: Elsevier, 2003.

DESPA, Mihai Liviu. **Comparative study on software development methodologies**. Database Systems Journal, Bucareste, v. 5, n. 3, p.37-56, 2014.

FITZGERALD, Brian. **An empirical investigation into the adoption of systems development methodologies**. Information & Management, [s.l.], v. 34, n. 6, p.317-328, dez. 1998. Elsevier BV. http://dx.doi.org/10.1016/s0378-7206(98)00072-x.

FOWLER, M. **The New Methodology** 2015. Disponível em: <https://www.martinfowler.com/articles/newMethodology.html>. Acesso em: 07 out. 2019.

FONSECA, J. J. S. Metodologia da pesquisa científica. **Apostila Fortaleza: UEC**, 2002.

GAO, F.; LI, M.; CLARKE, S. Knowledge, management, and knowledge management in business operations. **Journal Of Knowledge Management**, v. 12, n. 2, p.3-17, 2008.

GANDOMANI, Taghi Javdani; NAFCHI, Mina Ziaei. A**n empirically-developed framework for Agile transition and adoption: A Grounded Theory approach**. Journal Of Systems And Software, [s.l.], v. 107, p.204-219, set. 2015. Elsevier BV.

GIL, A. C. Como elaborar projetos de pesquisa. São Paulo: **Editora Atlas**, v. 5, p. 61, 2003.

GIL, Antônio Carlos. Métodos e Técnicas de Pesquisa Social. São Paulo: **Editora Atlas**, v. 6, p. 37, 2008.

GREGORY, P., BARROCA, L., SHARP, H., DESHPANDE, A., TAYLOR, K. **The challenges that challenge: Engaging with agile practitioners' concerns.** Information and Software Technology, 77(3), p.92-104, 2015.

HAWRYSH, S.; RUPRECHT, J.. **Light Methodologies: It's Like Déjà Vu All Over Again.** Cutter IT Journal. 13: 4 - 12, 2000.

HIGHSMITH, J.; COCKBURN, A. **Agile Software Development: The Business of Innovation**. Computer 34(9): 120-122, 2001.

HOHL, Philipp et al. **Forces that Prevent Agile Adoption in the Automotive Domain.** Product-focused Software Process Improvement, [s.l.], p.468-476, 2016. Springer International Publishing.

INAYAT, Irum et al. **A systematic literature review on agile requirements engineering practices and challenges**. Computers In Human Behavior, [s.l.], v. 51, p.915-929, out. 2015. Elsevier BV. http://dx.doi.org/10.1016/j.chb.2014.10.046.

KAKKAR, Sandhya. **Implementation Aspects of Software Development Projects**. 2006 Annual Ieee India Conference, [s.l.], p.1-6, set. 2006. IEEE.

KHAN, Asif Irshad; QURESHI, M. Rizwan Jameel; KHAN, Usman Ali. **A Comprehensive Study of Commonly Practiced Heavy & Light Weight Software Methodologies.** International Journal Of Computer Science And Issues, Mahebourg, v. 8, n. 4, p.441-450, jun. 2011.

LAKATOS, E. M.; MARCONI, M. A. Fundamentos de metodologia científica. 5ª ed. São Paulo: **Editora Atlas**, 2003.

LINDVALL, M., MUTHIG, D., DAGNINO, A., WALLIN, C., STUPPERICH, M.,

KIEFER, D., MAY, J., KAHKONEN, T.: **Agile Software Development in Large Organisations**. Computer 37(12), 26–34, 2004.

MARTINS, P. P. **Identificação de Ferramentas e Técnicas da Gestão do Conhecimento para a Promoção do Sucesso de Projetos de Governo Eletrônico**. 2018. 210 f. Dissertação (Mestrado) - Curso de Engenharia e Gestão do Conhecimento, Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis, 2018.

MATHARU, Gurpreet Singh et al. **Empirical Study of Agile Software Development Methodologies**. Acm Sigsoft Software Engineering Notes, [s.l.], v. 40, n. 1, p.1-6, 6 fev. 2015. Association for Computing Machinery (ACM).

NANDHAKUMAR, Joe; AVISON, David E.. **The Fiction of Methodological Development: a field study of information systems development**. Information Technology & People, Southampton, v. 12, n. 2, p.176-191, 1999.

NONAKA, I.; TAKEUCHI, H. The Knowledge-Creating Companies: How Japanese Companies, 1995.

NONAKA, I.; TAKEUCHI, H. Criação de conhecimento na empresa: como as empresas japonesas geram a dinâmica da inovação. In: **Gestão do Conhecimento**. Tradução de Ana Beatriz Rodrigues e Priscilla Martins Celeste. Rio de Janeiro: Campus, 1997.

NONAKA, I.; TAKEUCHI; H. A teoria da criação do conhecimento. In: **Gestão do conhecimento.** Tradução Ana Thorell. Porto Alegre: Bookman, 2008.

O'DONNELL, Michael J.; RICHARDSON, Ita. **Problems Encountered When Implementing Agile Methods in a Very Small Company**. Communications In Computer And Information Science, [s.l.], p.13-24, 2008. Springer Berlin Heidelberg.

PALESTINO, Caroline Munhoz Corrêa. **ESTUDO DE TECNOLOGIAS DE CONTROLE DE VERSÕES DE SOFTWARES.**2015. 72 f. TCC (Graduação) - Curso de Gestão da Informação, Ciências Sociais Aplicadas, Universidade Federal do Paraná, Curitiba, 2015.

PINTO, Sergio Augusto Órfão. **Gerenciamento de Projetos: Análise dos Fatores de Risco que Influenciam o Sucesso de Projetos de Sistemas de Informação**. 2002. 235 f. Dissertação (Mestrado) - Curso de Administração, Universidade de São Paulo, São Paulo, 2002.

QUORA. **What is JIRA? What is it used for?** 2018. Disponível em: <https://www.quora.com/What-is-JIRA-What-is-it-used-for>. Acesso em: 28 ago. 2018.

RAMALINGAM, B. Tools for knowledge and learning: A guide for development and humanitarian organizations. **Overseas Development Institute**, London, 2006.

RAMESH, Balasubramaniam; CAO, Lan; BASKERVILLE, Richard. **Agile requirements engineering practices and challenges: an empirical study**. Information Systems Journal, [s.l.], v. 20, n. 5, p.449-480, 13 nov. 2007. Wiley.

SCHWABER, K. **Scrum Development Process.** OOPSLA'95 Workshop on Business ObjectDesign and Implementation, Springer-Verlag. 1995.

SCHWABER, K.; BEEDLE, M. **Agile Software Development With Scrum.** Upper Saddle, River, NJ, Prentice-Hall, 2002.

SCHWABER, K.; SUTHERLAND, J. **The Scrum Guide.** 2009. Disponível em: <https://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf>. Acesso em: 19 nov. 2019.

SCHWABER, K.; LAGANZA, G.; D'SILVA, D. **The Truth About Agile Processes: Frank Answers to Frequently Asked Questions.** Forrester Report. 2007.

Serviço Brasileiro de Apoio às Micro e Pequenas Empresas, SEBRAE. **Sobrevivência das empresas no Brasil**, Outubro, 2016. Disponível em: <https://m.sebrae.com.br/Sebrae/Portal%20Sebrae/Anexos/sobrevivencia-das-empresas-no-brasil-102016.pdf>. Acesso em: 14 set. 2019.

SERVIN, G.; DE BRUN, C. ABC of knowledge management. **NHS National Library for Health: Specialist Library**, 2005.

SRIVASTAVA, Apoorva; BHARDWAJ, Sukriti; SARASWAT, Shipra. **SCRUM model for agile methodology.** 2017 International Conference On Computing, Communication And Automation (iccca), [s.l.], p.864-869, maio 2017. IEEE

STANDISH. **Chaos Report**. 1995. Disponível em: <https://www.csus.edu/indiv/r/rengstorffj/obe152-spring02/articles/standishchaos.pdf>. Acesso em: 18 nov. 2019.

STANDISH. **Chaos Report**. 2015. Disponível em: <https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf>. Acesso em: 18 nov. 2019.

SURESHCHANDRA, K.; SHRINIVASAVADHANI, J. **Moving from waterfall to agile. Proceedings from Agile**, 2008: AGILE '08 Conference. Toronto, ON, Canada

SVENSSON, M.; HOST, M. **Introducing an agile process in a software maintenance and evolution organization.** Proceedings on European Conference of Maintenance and Engineering. p. 256-264, 2015.

TORTAMIS, Pınar Işıl. **Light vs. Heavy: Which To Choose? Improq 2004: Impact Of Software Process On Quality Workshop,** İzmir, p.6-10, 2004.

VERRET, Jeffrey. **Implementing Agile Methodology: Challenges and Best Practices**. 2018. 70 f. Dissertação (Mestrado) - Curso de Gestão de Informação Aplicada, Universidade do Oregon, Eugene, Oregon, 2018.

VIJAYASARATHY, Leo R.; BUTLER, Charles W.. **Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter?.** Ieee Software, [s.l.], v. 33, n. 5, p.86-94, set. 2016. Institute of Electrical and Electronics Engineers (IEEE).

WILLIAMS, L.. **The XP Programmer: The Few-Minutes Programmer**. Ieee Software, [s.l.], v. 20, n. 3, p.16-20, maio 2003. Institute of Electrical and Electronics Engineers (IEEE).

ANEXO A - Questionário Aplicado

Metodologias de Desenvolvimento de Software

Olá, sou o Bruno Jaime Nascimento. Sou estudante do curso de GTI - Gestão da Tecnologia da Informação no IFSC - Florianópolis. Estou desenvolvendo uma pesquisa para o meu TCC - Trabalho de Conclusão de Curso que visa verificar como as empresas desenvolvem software e quais são os principais problemas enfrentados.

Como contrapartida, ao final desta pesquisa socializaremos o panorama geral (omitindo o nome das empresas) e nos colocaremos a disposição para uma oficina de melhoria do processos de desenvolvimento de software, para as empresas que desejarem.

Desde já agradeço sua colaboração.

Abraço.
Bruno Jaime Nascimento.

*Obrigatório

1. **1. Nome da Empresa**

2. **2. Qual é a sua função na empresa?** *
   Marque todas que se aplicam.
   - [ ] Desenvolvedor
   - [ ] Gerente de Projeto/Produto
   - [ ] Tester de Software
   - [ ] Analista de Projetos
   - [ ] Analista de Requisitos
   - [ ] Diretor da Empresa

3. **3. Para você, o que é uma metodologia de desenvolvimento de software?**

4. **4. Dentro da equipe de desenvolvimento de software existe um gerente de projetos?** *
   Marcar apenas uma oval.
   - ◯ Sim
   - ◯ Não

5. **5. Durante o desenvolvimento de software, nós utilizamos:** *
Marque todas que se aplicam.

- [ ] Reuniões diárias (de no máximo 15 minutos)
- [ ] Reuniões de planejamento
- [ ] Pequenas Implementações: Ciclos de desenvolvimento de no máximo 30 dias
- [ ] "Histórias": Funcionalidades do sistema são definidas através de uma metáfora ou conjunto de metáforas compartilhadas entre o cliente e os programadores
- [ ] Programação em Pares
- [ ] Acesso ao Cliente: O cliente deve estar disponível a equipe de desenvolvimento a qualquer momento
- [ ] Padrões de Programação: Regras para programação existem e devem ser seguidas pelos programadores para garantir consistência e melhorar a comunicação entre a equipe de desenvolvimento
- [ ] Integração Contínua: Um novo trecho de código deve ser integrado ao sistema assim que estiver pronto. Quando integrado, o sistema é reconstruído e todos os testes devem ser realizados para que as mudanças sejam aceitas
- [ ] "Code and fix": Programar e corrigir mediante feedback do cliente/gerente do projeto
- [ ] Teste Unitário

6. **6. Qual(ais) metodologias de desenvolvimento você utiliza na sua empresa?** *
Marque todas que se aplicam.

- [ ] SCRUM
- [ ] eXtreme Programming (XP)
- [ ] Kanban
- [ ] Modelo em Cascata
- [ ] Modelo em Espiral
- [ ] Outro
- [ ] Até o momento, não adotamos nenhuma metodologia de desenvolvimento

7. **7. Dentre as atividades desempenhadas pelo gerente de projeto de software (ou pelo principal stakeholder do projeto, caso não exista um gerente de projetos), estão:** *
Marque todas que se aplicam.

- [ ] Reuniões com clientes
- [ ] Acompanhar o desenvolvimento do software
- [ ] Levantar Requisitos
- [ ] Documentar o projeto
- [ ] Programar
- [ ] Realizar testes
- [ ] Apresentar resultados para os clientes
- [ ] Design de software/banco de dados
- [ ] Identificar problemas na equipe
- [ ] Mitigar problemas técnicos

8. **8. Dentre as dificuldades para a aplicação de uma metodologia de desenvolvimento de software, você destacaria:** *
   Marque todas que se aplicam.

   - [ ] Não temos dificuldades
   - [ ] Falta de tempo
   - [ ] Falta de pessoa disponível
   - [ ] Falta de pessoa qualificada para desempenhar a função
   - [ ] Falta de recursos financeiros para contratar um especialista
   - [ ] Projetos são pequenos. Não requerem a aplicação de uma metodologia de desenvolvimento

9. **9. No desenvolvimento de software é comum encontramos/enfrentarmos:** *
   Marque todas que se aplicam.

   - [ ] Estimativas incorretas para com o tempo de desenvolvimento de uma tarefa
   - [ ] Requisitos pouco claros
   - [ ] Perda de colaboradores
   - [ ] Falta de informações do projeto
   - [ ] Integração com outras tecnologias
   - [ ] Problemas levantados tardiamente
   - [ ] Atraso nas entregas de software
   - [ ] Renegociação de prazos
   - [ ] Falta de autonomia para implantar mudanças/tomar decisões

10. **10. Em relação as metodologias de desenvolvimento de software, você considera que o seu conhecimento é:** *
    Marcar apenas uma oval.

    - ◯ Não tenho conhecimento
    - ◯ Conhecimento básico
    - ◯ Conhecimento Intermediário
    - ◯ Sou especialista

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA
INSTITUTO FEDERAL
DEPARTAMENTO ACADÊMICO DE SAÚDE E SERVIÇOS - DASS

DECLARAÇÃO

Declaro para os devidos fins e efeitos que **Bruno Jaime Nascimento** apresentou o Trabalho de Conclusão do Curso no Curso Superior de Tecnologia em Gestão da Tecnologia da Informação com o título: **"Desenvolvimento de Software: Fatores e Dificuldades que Influenciam a Adoção de Metodologias de Desenvolvimento"**, e teve como Banca de Defesa os seguintes integrantes.

Composição da Banca:

Professor(a) Presidente da Banca: Underléa Cabreira Corrêa, Dra

Professor(a) Orientador(a): Underléa Cabreira Corrêa, Dra

Coorientação: Cleverson Tabajara Vianna, Ma

Membro da Banca: Prof. Fernando Ferreira Aguiar, Msc

Membro da Banca: Prof. Marcos André Pisching, Dr

Prof. Felipe Cantório Soares, Meng
Coordenador do CST GTI

Florianópolis, 05/11/2019