UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

MARCELO MENDES DA SILVA

ABORDAGEM DE PONTOS DE FUNÇÃO NO DESENVOLVIMENTO DE SISTEMAS UTILIZANDO PRÁTICAS ÁGEIS

DISSERTAÇÃO

CURITIBA
2019

MARCELO MENDES DA SILVA

ABORDAGEM DE PONTOS DE FUNÇÃO NO DESENVOLVIMENTO DE SISTEMAS UTILIZANDO PRÁTICAS ÁGEIS

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Orientadora: Prof.ª Dr.ª Maria Cláudia Figueiredo Pereira Emer
Coorientador: Prof. Dr. Adolfo Gustavo Serra Seca Neto

CURITIBA
2019

Dados Internacionais de Catalogação na Publicação

Silva, Marcelo Mendes da

Abordagem de pontos de função no desenvolvimento de sistemas utilizando práticas ágeis [recurso eletrônico] / Marcelo Mendes da Silva. -- 2019.

1 arquivo eletrônico (116 f.): PDF; 2,20 MB.

Modo de acesso: World Wide Web.

Texto em português com resumo em inglês.

Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada. Área de Concentração: Engenharia de Sistemas Computacionais. Linha de Pesquisa: Engenharia de Software, Curitiba, 2019.

Bibliografia: f. 93-97.

1. Computação - Dissertações. 2. Software – Desenvolvimento - Metodologia. 3. Medição de software. 4. Desenvolvimento ágil de software. 5. Análise de pontos de função. 6. Custos - Indicadores. 7. Estudos de viabilidade. 8. Arquitetura de Software - Avaliação. 9. Engenharia de software. I. Emer, Maria Cláudia Figueiredo Pereira, orient. II. Seca Neto, Adolfo Gustavo Serra, coorient. III. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada. IV. Título.

CDD: Ed. 23 -- 621.39

Biblioteca Central do Câmpus Curitiba - UTFPR
Bibliotecária: Luiza Aquemi Matsumoto CRB-9/794

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Pesquisa e Pós-Graduação

CAMPUS CURITIBA

TERMO DE APROVAÇÃO DE DISSERTAÇÃO Nº

A Dissertação de Mestrado intitulada **Abordagem de Pontos de Função no Desenvolvimento de Sistemas Utilizando Práticas Ágeis** defendida em sessão pública pelo candidato **Marcelo Mendes da Silva** no dia **12 de dezembro de 2019**, foi julgada aprovada em sua forma final para a obtenção do título de Mestre em Computação Aplicada, Linha de Pesquisa – Engenharia de Software, pelo Programa de Pós-Graduação em Computação Aplicada.

Profª. Drª. Maria Claudia F. Pereira Emer - (UTFPR) - *Orientadora*
Prof. Dr. Adolfo Gustavo Serra Seca Neto - (UTFPR)
Prof. Dr. Laudelino Cordeiro Bastos - (UTFPR)
Profª. Drª. Mariângela de Oliveira Gomes Setti - (UFPR)

Curitiba, 12 de dezembro de 2019.

_____
Carimbo e Assinatura do(a) Coordenador(a) do Programa

Dedico este trabalho à minha Família.

AGRADECIMENTOS

Agradeço a minha família por ter me apoiado durante este período dedicado ao mestrado.

Agradeço a empresa que me cedeu espaço para cursar as disciplinas, permitindo meu estudo sobre o ambiente de trabalho. Agradeço também aos meus amigos que me incentivaram e ajudaram fornecendo informações valiosas que contribuíram com este trabalho.

Por fim, agradeço aos professores da UTFPR que permitiram meu ingresso no programa de mestrado, aos professores das disciplinas cursadas, professores membros dos seminários, à professora orientadora e professor coorientador deste trabalho. Agradeço pelos direcionamentos, orientações, conhecimento e contribuições realizadas que me permitiram novamente voltar a aprender.

Muito Obrigado.

Feliz aquele que transfere o que sabe e aprende o que ensina.
(Cora Coralina)

RESUMO

A análise de pontos de função é uma técnica que permite medir projetos de desenvolvimento de software e, assim estabelecer uma medida de tamanho que é utilizada para o gerenciamento do projeto de software. No cenário atual temos os métodos e práticas ágeis sendo bastante utilizados e ganhando espaço nos projetos de desenvolvimento de software. Como a métrica de pontos de função já está consolidada e projetos que utilizam métodos ágeis não são estimados por meio de pontos de função, este trabalho faz um estudo da utilização de pontos de função com práticas ágeis e propõe uma abordagem híbrida de desenvolvimento de software, denominada HiPA (Híbrida com Práticas Ágeis). Para relatar o estado da arte com relação à utilização de pontos de função em projetos que utilizam práticas ágeis, fizemos um mapeamento sistemático no qual encontramos 182 artigos dos quais 18 foram selecionados. Com acesso a informações de projetos em desenvolvimento de software de uma mesma organização, obtivemos indicadores que permitem uma avaliação destes projetos de software que trabalham em um método tradicional, o ciclo de desenvolvimento cascata (Waterfall), e, projetos que utilizam práticas ágeis. Assim, em um estudo de caso, analisamos estes projetos por meio de indicadores, para atingir o objetivo de verificar a viabilidade da utilização de pontos de função em projetos que utilizam práticas ágeis. Os projetos analisados no estudo de caso totalizaram mais de 24 mil horas e os resultados obtidos fazem a comparação dos 2 modelos de desenvolvimento de software (ciclo de desenvolvimento cascata e um ciclo de desenvolvimento com práticas ágeis). Resultados numéricos sobre a execução dos projetos em termos de tempo e custo são apresentados, sendo o IDC (Índice de desempenho de Custos) melhor para projetos que utilizam práticas ágeis. Por fim, apesar da literatura não informar como devem ser utilizadas práticas ágeis com pontos de função, propomos uma abordagem híbrida de desenvolvimento de software, denominada HiPA e, avaliamos esta abordagem de desenvolvimento por meio de um questionário sendo que a abordagem HiPA foi indicada por 61,5% dos respondentes como viável e por 38,5% como possivelmente viável. Também avaliamos parcialmente a implementação da abordagem HiPA com o auxílio da contagem dos pontos de função, tendo como resultado indicativos da viabilidade de execução da abordagem.

Palavras-chave: Pontos de função; Métricas de Software; Práticas Ágeis.

ABSTRACT

Function point analysis is a technique that enables the measurement of software development projects and, thus, establish a size unit that is used in software project management. Currently, agile methods and practices have been widely used and have been gaining space in software development projects. Given that function point analysis is already consolidated and projects using agile methods are not estimated by function points, this work explores the use of function points with agile practices and proposes a hybrid approach called HiPA (Portuguese abbreviation for Hybrid with Agile Practices). In order to present the current state of the art regarding the user of function points in projects adopting agile practices, a systematic mapping of 182 articles was done and 18 of them were selected for this work. With access to information on software development projects from a single organization, this work obtained indicators that allowed an evaluation of such projects, part of them using the traditional waterfall methodology and others adopting agile practices. Those indicators were analyzed, in a case study, to achieve the goal of verifying the feasibility of using function points in projects that use agile practices. The projects analyzed in the case study totaled more than 24,000 hours and the results obtained compare the two above-mentioned software development models (waterfall and agile). Numeric results from the execution of projects in terms of time and cost are presented, being the IDC (Portuguese abbreviation for Cost Performance Index) better for projects that adopt agile practices. Lastly, although the literature does not inform how agile practices should be used with function points, this work proposes a hybrid approach, called HiPA, to software development and evaluates it through a questionnaire, in which 61.5% of the respondents indicated HiPA as a viable approach while 38.5% as possibly viable. The implementation of the HiPA approach with the support of counting function points was partially evaluated, with results pointing the approach as viable.

Keywords: Function Points; Software Metrics; Agile Practices.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

BACEN	Banco Central
CEF	Caixa Econômica Federal
COCOMO	*COnstructive COst MOdel*
COSMIC	*Common Software Measurement International Consortium*
CR	Custo Real
DSDM	*Dynamic Systems Development Method*
EVM	*Earned Value Management*
EVMS	*Earned Value Management System*
FDD	*Feature Driven Development*
FiSMA	*Finnish Software Metrics Association*
FPA	*Function Point Analysis*
FSM	*Functional Size Measurement*
IEC	*International Electrotechnical Commission*
IFPUG	*International Function Point Users Group*
IID	*Iterative and Incremental Development*
ISO	*International Organization for Standardization*
MPC	*Metrics Practices Committee*
NDIA	Associação Industrial de Defesa Nacional
NESMA	*Netherlands Software Metrics Association*
PMBOK	*Project Management Body of Knowledge*
PMI	*Project Management Institute*
PMO	*Project Management Office*
RUP	*Rational Unified Process*
SDLC	*Software Development Life Cycle*
SLOC	*Source Lines of Code*
SNAP	Software *Non-functional Assessment*
TCU	Tribunal de Contas da União
UCP	*Use Case Point*
UKSMA	*United Kingdom Software Metrics Association*
UML	*Unified Modeling Language*
VA	Valor Agregado
VP	Valor Planejado
XP	*Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

A utilização de técnicas de estimativa de tamanho baseadas em modelos fornece uma alternativa objetiva e independente. O dimensionamento funcional mantém a independência da avaliação do tamanho do sistema, sem a influência de preocupações orçamentárias externas (WILKIE et al., 2006).

Existem algumas métricas para medição do tamanho de *software*, como UCP (*Use Case Point*) (CARROLL, 2005), SLOC (*Source Lines of Code*) (LIND; HELDAL, 2010), COCOMO (*COnstructive COst MOdel*) (AGRAWAL et al., 2016) COSMIC (*Common Software Measurement International Consortium*) (EBERT; SOUBRA, 2014), sendo o foco deste trabalho a análise de pontos de função ou FPA (*Function Point Analysis*), que é uma métrica bastante consolidada e utilizada no de desenvolvimento de *software*.

Análise de pontos de função (FPA) é uma medida de dimensionamento de importância comercial. Esta técnica quantifica as funções contidas no *software* em termos que sejam significativos para os usuários do *software*. Ou seja, a medida está relacionada aos requisitos de negócio associados ao *software* e usa um conjunto padronizado de critérios básicos. Cada uma das funções de negócio é associada a um índice numérico de acordo com seu tipo e complexidade. Estes índices são totalizados para dar uma medida inicial do tamanho que é, em seguida, normalizada, incorporando uma série de fatores relacionados ao *software* como um todo. O resultado é um único número, chamado de pontos de função, índice que mede o tamanho e a complexidade do produto de *software*. A técnica de pontos de função fornece uma medida comparativa que auxilia na avaliação, planejamento, gestão e controle de produção de *software*[1].

Em projetos de desenvolvimento de *software* que utilizam métodos ágeis, as métricas utilizadas para estimar o tamanho do sistema, em geral são diferentes da estimativa de pontos de função, que é uma técnica bastante consolidada.

Para Usman et al. (2014), *Planning poker*, como prática de estimativa, é a métrica mais utilizada na estimativa no desenvolvimento de software que trabalha com métodos ágeis e, tem bastante precisão quando utilizada por profissionais experientes.

Conforme Diebold e Zehler (2016), práticas ágeis podem ser definidas como tarefas, atividades ou algum aspecto técnico centrado em pelo menos um dos valores do manifesto ágil e, conforme Senapathi e Drury-Grogan (2017), as práticas ágeis utilizadas em projetos permitem uma interação entre o time e facilitam a comunicação, sendo assim ferramentas para execução dos projetos. E, segundo Ruparelia (2010) e Rajlich (2006), um dos métodos mais utilizados e tradicionais de desenvolvimento é o ciclo de desenvolvimento cascata.

Assim, inicialmente realizamos um mapeamento sistemático para buscar na literatura projetos de *software* estimados por pontos de função e que utilizaram práticas ágeis. Após, a partir de uma gama de projetos mensurados por pontos de função, fizemos em um estudo de caso contendo uma comparação de projetos executados em um ciclo de desenvolvimento tradicional ou cascata com projetos executados em um ciclo de desenvolvimento que utiliza práticas ágeis.

---

[1] http://www.ifpug.org/about-function-point-analysis/?lang=pt. Acesso em 15/09/2019.

Para finalizar propomos uma abordagem híbrida de desenvolvimento de *software* com a utilização de pontos de função em conjunto com práticas ágeis de desenvolvimento e avaliamos esta abordagem.

1.1 MOTIVAÇÃO E JUSTIFICATIVA

Para Cohen et al. (2004), embora os métodos ágeis apresentem diferenças entre suas práticas, eles compartilham inúmeras características, incluindo o desenvolvimento iterativo, e um foco na comunicação interativa e na redução do esforço empregado em artefatos intermediários. E, para Jiang e Eberlein (2008), os métodos clássicos e ágeis têm origens filosóficas comuns e são tecnicamente compatíveis e complementares.

Buscamos na literatura, por meio do mapeamento sistemático, trabalhos que fornecem informações de projetos de desenvolvimento de *software* híbrido, relacionado a métrica de pontos de função. Assim, trabalhos com projetos de desenvolvimento de *software* com estas características poderiam fornecer um embasamento teórico para implantação de uma abordagem híbrida de desenvolvimento de software em um ambiente real de desenvolvimento.

Os trabalhos encontrados no mapeamento sistemático ajudaram a fortalecer o conceito de desenvolvimento híbrido de *software*. Isto foi relevante para avançarmos no trabalho com o estudo de caso e finalizamos com uma proposta de uma abordagem de desenvolvimento de software.

1.2 OBJETIVOS

O objetivo deste trabalho é avaliar, em projetos de desenvolvimento de software, se a utilização da combinação de pontos de função e práticas ágeis podem influenciar de maneira positiva ou negativa a execução destes projetos de software. E, também, avaliar a combinação de pontos de função com práticas ágeis.

Para isto os objetivos específicos são os seguintes:

- Investigar se existem projetos de desenvolvimento de *software* com o uso de práticas ágeis e mensurados por pontos de função e avaliar se os resultados desta combinação são favoráveis.

- Analisar em ambiente real de desenvolvimento de *software*, por meio de indicadores de custo e prazo de projetos, se a utilização de práticas ágeis em *software* mensurado por pontos de função resulta em uma associação positiva.

- Propor uma abordagem de desenvolvimento de software que utiliza práticas ágeis e estimativas de pontos de função e avaliar esta abordagem.

1.3 CONTRIBUIÇÃO

Esta dissertação contribui para o emprego de pontos de função combinado práticas ágeis no desenvolvimento de projetos de *software*, sendo que podemos destacar as seguintes contribuições:

- Mapeamento sistemático sobre abordagem de desenvolvimento híbrida de software com a utilização de pontos de função;
- Estudo de caso, no qual foi realizada análise de desempenho de projetos que utilizaram o modelo de desenvolvimento cascata e projetos que utilizaram práticas ágeis. Sendo a análise de desempenho feita por meio de índices de desempenho de prazo e custo de projeto;
- Proposição de uma abordagem para híbrida de desenvolvimento de *software* que utiliza práticas ágeis e estimativa por pontos de função durante o desenvolvimento e, por fim, avaliação desta abordagem de desenvolvimento.

1.4 ORGANIZAÇÃO DO TRABALHO

Além do capítulo introdutório, esta dissertação está organizada nos seguintes capítulos.

No Capítulo 2 é apresentada a fundamentação teórica, na qual são abordados a evolução dos métodos de mensuração de *software*, a técnica de pontos de função, métodos ágeis e conceitos de gerenciamento de projeto.

O Capítulo 3 ilustra a metodologia utilizada para fazermos o mapeamento sistemático, o estudo de caso e a elaboração da abordagem híbrida de desenvolvimento de *software*.

No Capítulo 4 são apresentados os resultados encontrados no mapeamento sistemático e no estudo de caso.

O Capítulo 5 apresenta a abordagem híbrida de desenvolvimento de *software* com estimativa por pontos de função e os resultados encontrados na avaliação desta abordagem.

No Capítulo 6 são abordadas as limitações e ameaças a validade dos resultados encontrados.

Por fim, no Capítulo 7 é apresenta a Conclusão deste trabalho, limitações identificadas e sugestões de trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo são abordados conceitos utilizados no decorrer do estudo. Os temas abordados serão: métodos de mensuração, métodos ágeis, projetos e ciclo de desenvolvimento de *software*.

Sobre os métodos de mensuração traçamos um panorama de evolução, com foco principal em pontos de função e apresentamos algumas definições e exemplificações de métodos e práticas ágeis. Para completar mostramos alguns conceitos de gerenciamento para avaliação de projetos e apresentamos dados de pesquisas sobre a utilização de abordagens ágeis, complementando com conceitos de ciclos de desenvolvimento de *software*.

2.1 MÉTODOS DE MENSURAÇÃO

Sobre os métodos de mensuração aprovados pela Organização Internacional de Normalização (ISO), temos que, em 1996, a ISO estabeleceu os princípios comuns dos métodos de medição de *software* (FSM: *Functional Size Measurement*) e a família padrão ISO / IEC 14143, a fim de promover a interpretação consistente dos princípios do FSM.

Entre esses métodos, cinco foram reconhecidos como padrões (CUADRADO-GALLEGO et al., 2014; KAUR; KAURB, 2019):

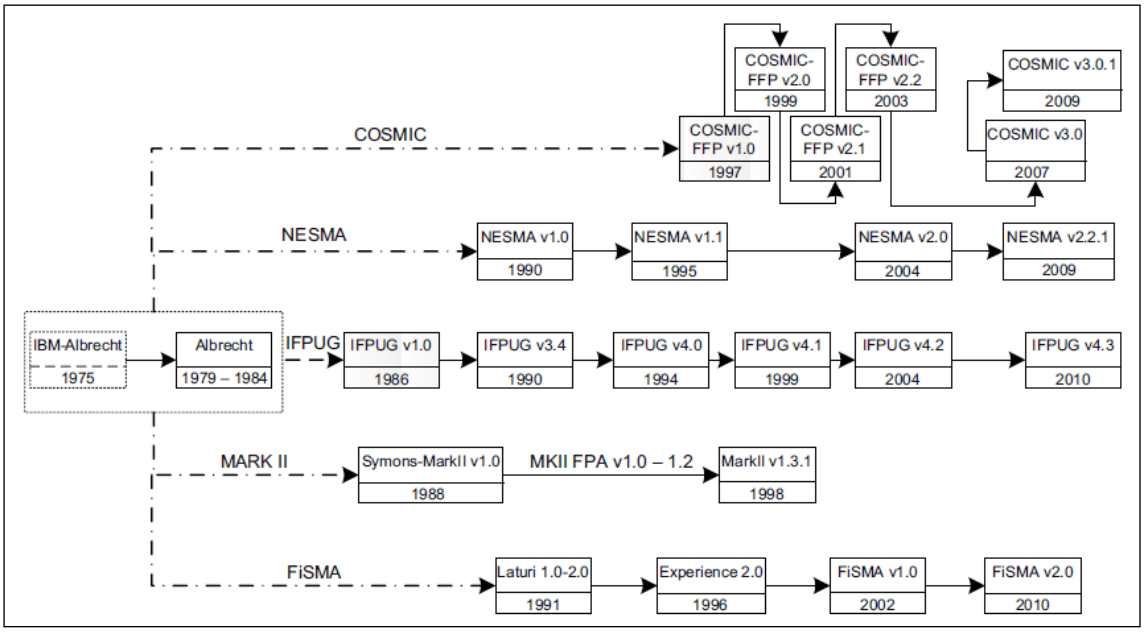
- IFPUG, ISO / IEC 20926: 2009 (ISO / IEC, 2009);
- NESMA, ISO / IEC 24570: 2005 (ISO / IEC, 2005);
- Mk II, ISO / IEC 20968: 2002 (ISO / IEC, 2002);
- COSMIC, ISO / IEC 19761: 2011 (ISO / IEC, 2011);
- FiSMA, ISO / IEC 29881: 2010 (ISO / IEC, 2010).

Em Melo e Caroli (2017), têm-se que a ISO / IEC 14143, que trata de medição funcional, distingue os dois subconjuntos de requisitos do usuário: os Requisitos Funcionais do Usuário e os Requisitos Não Funcionais. O COSMIC, padrão ISO / IEC 19761 é um método jovem no mercado e, embora seja mais utilizado no mercado internacional (México, Holanda, Reino Unido, China, Índia, Colômbia, Itália), no Brasil, neste momento há apenas uma grande organização que o utiliza: uma instituição financeira cooperativa com atuação nacional, com sede no sul do país.

Estes métodos evoluíram significativamente ao longo dos anos (Figura 1). Em 1986, o IFPUG (*International Function Point Users Group*) foi criado para promover o uso do método original da Albrecht e para controlar a evolução da definição padrão de medição. Isso mudou o nome dos pontos de função de Albrecht para IFPUG FPA. Desde então, várias versões do IFPUG FPA foram publicadas (IFPUG, 1990; 1994; 1999; 2000; 2004; 2010) (CUADRADO-

GALLEGO et al., 2014).

Figura 1: Evolução - Tipo de Medidas.



Fonte: Cuadrado-Gallego et al. (2014).

O IFPUG FPA tem popularidade e grandes conjuntos de dados publicamente disponíveis para aqueles que desejam formar seu próprio modelo IFPUG específico da empresa ou comparar suas medidas com outros. Isto foi aceito pela ISO como padrão internacional em 2009 (ISO / IEC, 2009) (CUADRADO-GALLEGO et al., 2014).

O Mk II FPA foi desenvolvido por Symons (1988) para melhorar o método FPA original. Este método trouxe algumas sugestões para refletir a complexidade interna de um sistema. O *Metrics Practices Committee* (MPC) da UK *Software Metrics Association* (UKSMA) tem a autoridade do método. Também foi projetado principalmente para medir sistemas de informações comerciais. O Mk II FPA foi aceito como compatível com ISO / IEC 14143 e tornou-se um padrão ISO internacional em 2002 (ISO / IEC, 2002) (CUADRADO-GALLEGO et al., 2014).

E o NESMA FPA foi introduzido pela *Netherlands Software Metrics Association* (NESMA). Desde então, foram publicadas cinco versões (NESMA, 1990; 1995; 2004; 2005; 2009). Este método tem as mesmas regras que o método IFPUG FPA. No entanto, o manual de medição NESMA fornece diretrizes, sugestões e exemplos diferentes. Foi aceito pelo ISO como padrão internacional em 2005 (ISO / IEC, 2005) (CUADRADO-GALLEGO et al., 2014).

O Consórcio Internacional de Medição de *Software* Comum (COSMIC), conforme Cuadrado-Gallego et al. (2014), introduziu COSMIC FFP (COSMIC, 1997). Foi definido como um método de FSM de 2ª geração como resultado de uma série de inovações, tais como: um ajuste melhor em ambientes de sistema de informação em tempo real, identificação e medição de múltiplas camadas de *software*, diferentes pontos de vista a partir dos quais o *software* pode ser observado e medido. Desde a sua primeira publicação, o interesse desenvolvido tanto na comunidade acadêmica quanto na indústria pela nova unidade tem sido enorme, atingindo uma

grande difusão e utilização com três novas versões publicadas em um curto período de tempo (COSMIC, 1999; 2001; 2003) e duas mais tarde (COSMIC, 2007; 2009). Foi aprovado pelo ISO como um padrão em 2003 como ISO 19761 (ISO / IEC, 2003) e atualizado em 2011 (ISO / IEC, 2011) (CUADRADO-GALLEGO et al., 2014).

O FiSMA FSM foi desenvolvido por um grupo de trabalho da *Finnish Software Metrics Association* (FiSMA). É um método de medição de tamanho parametrizado geral projetado para ser aplicado a todos os tipos de *software*. A diferença entre o FiSMA FSM e outros métodos é a orientação para o serviço e não para processos. Foi recentemente aceito como um padrão internacional FSM em 2010 (ISO / IEC, 2010) (CUADRADO-GALLEGO et al., 2014).

2.2 PONTOS DE FUNÇÃO

A IFPUG, *International Function Point Users Group* (2018), criada em 1986, é a entidade que tem a missão de promover, incentivar e aprimorar a utilização da métrica de pontos de função, sendo seus membros aderentes ao código de ética da entidade.

Pontos de função[2] é uma unidade de medida usada para representar internacionalmente o padrão de tamanho de *software*. O método de medição de tamanho funcional do IFPUG quantifica a funcionalidade do *software* fornecida ao usuário com base unicamente no seu design lógico e requisitos funcionais. O número resultante é chamado de contagem de pontos de função. Os objetivos dos pontos de função são:

- Medir funcionalidades que o usuário solicita e recebe;
- Medir taxas de desenvolvimento e manutenção de software e tamanho, independentemente da tecnologia utilizada para implementação;
- Fornecer uma medida de normalização em projetos e organizações.

SNAP (*Software Non-functional Assessment*) é o processo de avaliação referente a requisitos não funcionais. É uma métrica de *software* que complementa os pontos de função. O SNAP permite a análise dos requisitos não funcionais para além dos requisitos funcionais de dimensionamento medido por pontos de função e não substitui pontos de função[3].

Pontos de função medem o volume (ou tamanho) do fluxo de dados e armazenamento inerente a uma aplicação de *software*. Estes são chamados de requisitos funcionais. SNAP mede o volume (ou tamanho) de outros aspectos do *software* — tais como a configuração de dados, algoritmos, árvores de decisão, validação de dados, colocação de logotipos, etc. Estes são chamados de requisitos não funcionais[4].

A técnica de Pontos de Função é uma abordagem tradicional na qual se tem, no início

[2] https://www.ifpug.org/faqs-2/?lang=pt. Acesso em 15/09/2019.

[3] https://www.ifpug.org/about-snap/?lang=pt. Acesso em 15/09/2019.

[4] https://www.ifpug.org/faqs-2/?lang=pt. Acesso em 15/09/2019.

do trabalho, a totalização do que é necessário ser feito, estabelecendo o escopo e o orçamento. Pontos de função não mede a produtividade, mas pode ser utilizado auxiliando na medição da produtividade. Pontos de função mede o tamanho do que o *software* faz ao invés de como o *software* é desenvolvido. Os conceitos de pontos de função foram iniciados em 1979, sendo os conceitos iniciais refinados e publicados em 1984. Após isto, uma comunidade resolveu efetuar padronizações adicionais nas regras de contagem por pontos de função, formando o Grupo Internacional de Usuários de Pontos de Função (IFPUG).

Segundo Jones (2013), pontos de função é uma das métricas mais precisas e eficazes já desenvolvidas para o dimensionamento de *software* e para estudar a produtividade, os custos, e o valor econômico do *software* e podem ser usados para estudar todas as atividades de *software*, desde o desenvolvimento até a manutenção. Para Lavazza et al. (2016), a análise de pontos de função é amplamente usada para estimar e dimensionar o esforço necessário para desenvolver *softwares* sendo a métrica mais utilizada.

Conforme French (2016), pontos de função podem ser usados para um dimensionamento mais objetivo, por meio de uma técnica para minimizar os riscos de incertezas, sendo este dimensionamento defensável e não dependente da composição da equipe e experiência desta equipe, independentemente da plataforma de tecnologia, e tendo como referência os requisitos sendo estes fatores os mais importantes no custo de um *software*.

Para Ince (1990), os pontos de função são calculados examinando os recursos de uma especificação do sistema que mede a quantidade de funcionalidade do sistema e a explicação por trás da popularidade da análise dos pontos de função é a facilidade com que os pontos de função podem ser calculados manualmente e a criticidade do processo de cálculo de custos.

O uso de pontos de função como uma medida do tamanho funcional do *software* cresceu desde meados dos anos 70, de algumas organizações interessadas até uma impressionante lista de organizações em todo o mundo. Uma vez que houve crescimento no uso de pontos de função, houve maior aplicação e uso da medida. Desde a sua formação, o Grupo Internacional de Usuários de Pontos de Função (IFPUG) aprimorou continuamente o método original de Albrecht para o dimensionamento funcional de *software* (SANTANA et al., 2011).

No Brasil, o Governo Federal, por meio do Ministério do Planejamento, publicou um guia com necessidades consideradas comuns aos órgãos públicos, o Roteiro de Métricas do SISP[5]. Este roteiro é aderente à Instrução Normativa 04/2010, que orienta que os contratos de *software* sejam mantidos com base em critérios objetivos, critérios pelos quais a análise por pontos de função se destaca (MELO; CAROLI, 2017).

Ainda no Brasil, dentre as organizações que utilizam a análise por pontos de função, podemos citar: Banco Central (BACEN), Bradesco, Oi, TAM, Santander, Caixa Econômica Federal (CEF), Embratel, Porto Seguro, SERPRO, Sicredi, DATAPREV, Infraero, Petrobras, Correios, Tribunal de Contas da União (TCU), Aeronáutica, Marinha do Brasil, Centro de Desenvolvimento da Tecnologia Nuclear, Comissão Nacional de Energia Nuclear, INEP, Ministérios da Cultura, Ministério do Trabalho, Ministério da Educação, além de muitas outras empresas dos setores públicos e privado (MELO; CAROLI, 2017).

Com relação às críticas, pontos de função percorre um longo caminho para chegar a estimativa do tamanho do produto, isto é um risco de que estimativas se comportem de maneiras

[5] SISP. Guia de Boas Práticas em Contratação de Soluções de Tecnologia da informação, V 2.0. Brasília: Ministério do Planejamento, Secretaria de Logística e TI, 2014.

inesperadas. Além disto, a contagem de pontos de função envolve julgamento por parte do contador (KITCHENHAM, 1997).

Algumas fragilidades da métrica de pontos de função, conforme Jones (2013):

- A análise do ponto de função é lenta. Devido à baixa velocidade da análise do ponto de função, os pontos de função quase nunca são usados em grandes sistemas.

- A análise do ponto de função é cara. Assumindo uma velocidade de contagem diária de 500 pontos de função, a contagem de um *software* de 10.000 pontos de função exigiria 20 dias.

- O tamanho da aplicação não é constante. Durante o desenvolvimento, as aplicações (*softwares*) aumentam. As regras atuais de contagem não incluem crescimento contínuo.

- Mais de uma dúzia de variações de contagem de pontos de função existem por volta de 2013, incluindo pontos de função COSMIC, pontos de função NESMA. Essas variações produzem totais de pontos de função que diferem dos pontos de função do IFPUG.

Na prática, a contagem de pontos de função é lenta, cara e propensa a uma grande variabilidade. Dados empíricos mostram que diferentes profissionais podem produzir medidas bastante diferentes do mesmo conjunto de requisitos de *software*, mesmo na mesma organização (LAVAZZA et al., 2008).

Em Kitchenham (1997) também é citado que não devemos ignorar os problemas da estimativa e não devemos rejeitar o conceito de pontos de função. Basear essas estimativas no fluxo de dados por meio dos limites do sistema e na quantidade de dados que precisam ser mantidas no sistema é eminentemente razoável para os sistemas de informação, como é feita em pontos de função. O que devemos fazer é entender a limitação de nossas medidas e como usá-las com segurança.

2.3 MÉTODOS ÁGEIS

Na década de 1990, especialmente na segunda metade, conceitos sobre IID (*Iterative and Incremental Development*) no desenvolvimento de *software* estavam crescendo significativamente (LARMAN; BASILI, 2003).

Em fevereiro de 2001, um grupo de 17 especialistas em processos - que representavam DSDM (*Dynamic Systems Development Method*), XP, *Scrum*, FDD e outros - interessados em promover métodos e princípios modernos e simples de IID reuniram-se nos Estados Unidos. A partir desta reunião surgiu a *Agile Alliance* (www.*agile*alliance.org) (LARMAN; BASILI, 2003).

Com relação aos métodos ágeis, o movimento surgiu para incentivar e melhorar a eficiência no desenvolvimento de *software*. Assim, no manifesto ágil, conforme BECK et al. (2001), temos:

- Indivíduos e interações mais que processos e ferramentas.

Processos e ferramentas não são balas de prata. Você pode criar um sistema sem processos e ferramentas, mas não pode criar um sistema sem pessoas. E quanto melhor as pessoas, melhor o resultado. Nada disso é para dizer que o processo e as ferramentas não são importantes (MELLOR, 2005).

- Software em funcionamento mais que documentação abrangente.

Se você pretende dirigir de sua casa para o hospital local, não deve escrever um documento que diga exatamente como fazer isso primeiro. Certamente, você precisa de um mapa que defina a maneira mais eficaz de chegar lá, e você deve obter informações atualizadas sobre se uma determinada rota está bloqueada, mas "documentar" a rota para a posteridade não é o caminho (MELLOR, 2005).

- Colaboração com o cliente mais que negociação de contratos.

Todos nós já ouvimos falar de projetos em que a especificação foi "congelada", muitas vezes com resultados infelizes. Uma especificação, mesmo que informal, é apenas uma estrutura, por isso devemos esperar trabalhar com nossos clientes para garantir a construção do produto certo (MELLOR, 2005).

- Responder a mudanças mais que seguir um plano.

Um plano nos diz como ir de onde estamos para um destino, mas isso não nos diz se aonde estamos indo é o lugar certo para estar. Mesmo que o plano esteja correto, hoje, isso não significa que é o lugar certo amanhã. A tecnologia e o mercado mudam rapidamente demais para isso. Atingir os marcos do plano não equivale necessariamente ao sucesso do cliente. A chave aqui é explorar a mudança para garantir que o produto seja o mais próximo possível do que o cliente precisa (MELLOR, 2005).

Assim têm-se um desenvolvimento de *software* incremental, cooperativo e adaptativo. Incremental refere-se a entregas em ciclos de desenvolvimento. Cooperativo refere-se a uma estreita interação entre clientes e desenvolvedores. Adaptativo refere-se à capacidade de fazer e reagir a mudanças (ABRAHAMSSON et al., 2003).

Com relação aos métodos ágeis, podem-se citar alguns como FDD, XP, DSDM e

Scrum. O método FDD (*Feature Driven Development* ou Desenvolvimento Guiado por Funcionalidades) é um método ágil sendo que seus representantes redigiram o manifesto ágil (BECK et al., 2001) para desenvolvimento de *software*. O método XP (*eXtreme Programming*) foi criado na década de 1990, sendo um método ágil com foco no desenvolvimento de *softwares* com base em três pilares: agilidade no desenvolvimento da solução, economia de recursos e qualidade do produto final. O Desenvolvimento de Sistemas Dinâmicos (*Dynamic Systems Development Method* - DSDM) é um método de desenvolvimento de *software* baseado em "Desenvolvimento Rápido de Aplicação" (RAD). DSDM é um método de desenvolvimento iterativo e incremental que enfatiza o envolvimento constante do usuário.

O *Scrum* como *framework* ágil tem características como adaptabilidade, iteratividade, rapidez, flexibilidade e eficiência que fornecem um valor significativo de forma rápida durante todo o projeto. O *Scrum* garante a transparência na comunicação e cria um ambiente de responsabilidade coletiva e progresso contínuo sendo estruturado de tal forma que apoia o desenvolvimento de produtos e serviços em todos os tipos de indústrias e em qualquer tipo de projeto, independentemente de sua complexidade (SBOK, 2017).

O *framework Scrum* é usado especialmente na indústria de desenvolvimento de *software*. O *Scrum* é escalável e pode envolver práticas que se encaixam nas situações e condições exclusivas do ambiente do projeto (LANDAETA et al, 2011).

Para Jha et al. (2016) o *Scrum* se tornou a escolha de muitas organizações. E tornou-se o principal *framework* utilizado no desenvolvimento de *software* em pequenas e grandes organizações sendo definido para atender a expectativa do Manifesto Ágil e, segundo Paasivaara (2017), muitas organizações de grande porte fizeram a transição dos métodos tradicionais, tipo cascata, para ágeis.

No *Scrum* são utilizadas diversas práticas para o desenvolvimento iterativo e incremental, tendo alguns conceitos como:

- *Sprint*: período para execução de um determinado trabalho. O processo de desenvolvimento do *Scrum* compreende uma série de processos iterativos da *sprint*. Em uma *sprint* existe uma série de atividades de desenvolvimento em tempo limitado, incluindo análise, projeto, codificação, testes. Cada item do *backlog* será organizado em uma *sprint*, então o tempo necessário será estimado pela equipe. Após um *sprint*, deve haver as funções a serem entregues (entregáveis), e o conteúdo completo deve ser demonstrado de acordo com as metas estabelecidas no backlog (ZHI-GEN et al., 2009).

- *Backlog*: o *backlog* é uma lista de tarefas, incluindo o *product backlog* e o *sprint backlog*. O *sprint backlog* é uma lista de todas as tarefas a serem trabalhadas no *sprint* atual. O *sprint* backlog é, na verdade, um subconjunto do *product backlog* do projeto (ZHI-GEN et al., 2009).

- Reunião para planejamento de sprint: momento de definição do objetivo a ser alcançado em uma *sprint*. Com base no *product backlog* são selecionadas as atividades a serem trabalhadas na *sprint* (*sprint backlog*).

- Reunião diária: reunião com periodicidade diária em horário definido para verificar a situação do trabalho e assim sincronizar o trabalho do time.

- Revisão da *sprint*: reunião realizada no final da *sprint*, para verificar os objetivos atingidos e assim atualizar o *product backlog* do projeto com as tarefas concluídas. A atualização do *product backlog* é uma entrada para o planejamento da próxima *sprint*.

O objetivo do *Scrum* é desenvolver *softwares* com o máximo de qualidade possível em várias interações curtas chamadas *sprint*s, acelerando assim o processo de auto-organização, por meio das reuniões diárias. Cada etapa do ciclo de desenvolvimento (Requisitos, Análise, Design, Evolução e Entrega) agora está mapeada para um *sprint* ou uma série de *sprint*s. No *Scrum* não existe um processo predefinido dentro de um *Sprint*. Em vez disso, as reuniões de *Scrum* conduzem a conclusão das atividades alocadas e cada *sprint* opera em uma série de itens de trabalho chamados de *backlog* (BEEDLE; SUTHERLAND, 1999).

O *Scrum* como um *framework* ágil no desenvolvimento de *software* tem como características ser flexível, com *feedback* oportuno e interativo, gestão plana orientada por *release* e participação efetiva dos membros para resolver as dificuldades enfrentadas pelos métodos tradicionais, sendo excelente para entregar o produto de *software* de forma rápida e correta e dá aos membros da equipe as novas responsabilidades de gerenciamento. O processo de gerenciamento de projetos é visível e controlável. Os membros da equipe se tornam mais ativos devido à auto-organização e autogerenciamento (ZHI-GEN et al., 2009).

Os métodos ágeis possuem várias práticas relacionadas ao desenvolvimento de *software*. Estas práticas ágeis estão sendo adotadas por projetos, incluindo grandes projetos com arquiteturas complexas. Na Tabela 1 são identificadas 33 práticas ágeis. As 12 primeiras práticas na Tabela 1 são originárias do *Scrum*. Os restantes 21 elementos são práticas de E*xtreme programming* (XP). Os elementos marcados com um asterisco são práticas do XP que também são práticas recomendadas no *framework Scrum* (SLETHOLT et al., 2012).

Tabela 1. Lista de práticas ágeis.

Número da Prática	Práticas Ágeis
1	*Backlog* do produto – Mantido pelo "*Product owner*"
2	Processo de desenvolvimento e práticas facilitadas pelo *Scrum* master
3	Reunião de planejamento da *Sprint* para criar o *backlog* da *sprint*
4	"Planning poker" para estimar a duração das tarefas durante o planejamento da *sprint*
5	Sprint de tempo fixo (*Time-boxed Sprint*): Para produzir um resultado utilizável
6	*Product Owner* e time tem compromisso com *backlog* do *sprint*
7	Reunião diária para posicionamento e resolução de problemas
8	Equipe auto organizada
9	Monitoramento da sprint por meio de gráficos "*Burn Down*"
10	Reunião de revisão da sprint para apresentar os trabalhos concluídos
11	Revisão de sprint para aprendizado
12	Planejamento de entregas incrementais
13	Escrita de histórias de usuários*
14	Espaço de trabalho aberto para a equipe*
15	Definir um ritmo sustentável para o trabalho*
16	Estimar a velocidade do projeto*
17	Cliente sempre disponível*
18	Códigos e padrões acordados*
19	Codificar o teste unitário primeiro
20	Todo código em produção é feito com programação em par
21	Apenas um codifica por vez
22	Integração frequente
23	Usar propriedade coletiva*
24	Simplicidade no projeto*
25	Escolha uma metáfora para o sistema
26	Usar cartões de responsabilidades nas seções dos projetos
27	Criar soluções para reduzir o custo*
28	Nenhuma funcionalidade adicionada cedo
29	Refatorar sempre que possível
30	Todos os códigos devem ter testes unitários
31	Todos os códigos devem ser testados antes de liberados
32	Testes são criados quando um erro é encontrado
33	Testes de aceitação são frequentes e o resultado é publicado.

Fonte: Sletholt et al.(2012).

Para que características dos métodos ágeis sejam usadas e/ou adaptadas em projetos, é necessário que a estrutura organizacional incentive, apoie e ofereça uma estrutura na qual seja viável utilizar práticas ágeis no desenvolvimento de *software*. Não basta apenas existirem conceitos e uma maneira de realizá-los, isto deve ser apoiado pelo ambiente de desenvolvimento de projeto de sistemas.

O apoio da alta administração, a estrutura organizacional e os desafios culturais são questões muito reais que as organizações enfrentam. Isso implica que a alta administração não apenas aceita o conceito de métodos ágeis, mas compromete-se com o papel que eles têm para desempenhar e sustentar métodos ágeis em suas organizações (SENAPATHI; DRURY-GROGAN, 2017).

As organizações assumem uma transformação com métodos ágeis para resolução de problemas, mas o fato é que a jornada de transformação abre olhos para descobrir os problemas reais que antes não eram percebidos. É fácil colher os benefícios dos métodos ágeis, mas é difícil sustentar e resolver obstáculos sistêmicos, como base de código complexa e arquitetura legada. Embora seja fácil perceber os benefícios iniciais dos métodos ágeis, é difícil melhorar, a menos que haja um impulso da alta administração (HUBLIKAR; HAMPIHOLI, 2016).

Com relação à estimativa utilizada nos métodos ágeis, para Usman et al. (2014) o *planning poker* combina os elementos da opinião de especialistas e, conforme López-Martínez et al. (2017), é uma prática usada principalmente no *Scrum* para estimar o esforço de histórias de usuários por membros da equipe de desenvolvimento responsável pela implementação.

Conforme Dagnino (2013), o *planning poker* é útil nos estágios iniciais do ciclo de desenvolvimento, em que o tamanho do *software* é estimado e o esforço é calculado com base na velocidade da equipe. A velocidade da equipe refere-se à quantidade de tempo que uma equipe de desenvolvimento emprega para implementar as histórias associados à linha de base.

Também com relação às estimativas com métodos ágeis, Ecar et al. (2002) indica que o método COSMIC, que é um método adequado para medir *softwares* que evoluem por meio de iterações, utiliza histórias do usuário como descrições curtas e de alto nível de funcionalidades, escritas em linguagem do cliente durante o estágio inicial de coleta de requisitos e contendo informações suficientes para produzir o esforço de implementação estimado.

2.4 O GERENCIAMENTO DE PROJETO

Segundo o PMBOK, *Project Management Institute* (2013), projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo. A natureza temporária dos projetos indica que eles têm um início e um término definidos. O término é alcançado quando os objetivos do projeto são atingidos ou quando o projeto é encerrado porque os seus objetivos não serão ou não podem ser alcançados, ou quando a necessidade do projeto deixar de existir. Um projeto também poderá ser encerrado se o cliente (cliente, patrocinador ou financiador) desejar encerrá-lo. Temporário não significa necessariamente de curta duração. O termo se refere ao engajamento do projeto e à sua longevidade. O termo temporário normalmente não se aplica ao produto, serviço ou resultado criado pelo projeto; a maioria dos projetos é empreendida para criar um resultado duradouro.

Também de acordo com o PMBOK, *Project Management Institute* (2013), o processo

de monitorar e controlar as comunicações no decorrer de todo o ciclo do projeto é para assegurar que as necessidades de informação das partes interessadas do projeto sejam atendidas. Comunicação eficiente significa fornecer somente as informações que são necessárias. Com relação ao controle de custo e tempo de um projeto, têm-se o gerenciamento de valor agregado ou EVM (*Earned Value Management*) que é uma técnica para fazer a gestão de prazos e custos, medindo estas duas dimensões com relação ao desempenho para verificar a quantidade do montante financeiro em um determinado período de tempo. Assim, alguns conceitos como Custo Real, Valor Planejado e Valor Agregado são verificados durante o projeto para medir o desempenho relacionando o progresso financeiro ao progresso do cronograma do projeto. Estes conceitos são indicados, pois a análise dos projetos deste trabalho tem como dimensões tempo e custo para caracterizar a situação final dos projetos por meio dos indicadores. A seguir, conceitos de Valor Planejado (VP), Valor Agregado (VA) e Custo Real (CR) (Project Management Institute, 2013):

- Valor planejado. Valor planejado (VP) é o orçamento autorizado designado ao trabalho agendado. O valor planejado (VP) é o orçamento designado por fase no decorrer de todo o projeto, mas, em um determinado momento, o valor planejado define o trabalho físico que deveria ter sido executado. O total do VP algumas vezes é chamado de linha de base de medição do desempenho.

- Valor agregado. Valor agregado (VA) é a medida do trabalho executado expressa em termos do orçamento autorizado para tal trabalho. É o orçamento associado ao trabalho autorizado que foi concluído. O VA medido não pode ser maior que o orçamento VP autorizado. O VA é frequentemente usado para calcular a percentagem concluída de um projeto. O VA pode ser monitorado para determinar a situação corrente, e de forma acumulativa para determinar as tendências de desempenho em longo prazo.

- Custo real. Custo real (CR) é o custo realizado incorrido no trabalho executado de uma atividade, durante um período específico. O CR deve corresponder em definição ao que foi orçado para o VP e medido no VA.

Conforme (GARCIE; HIRATA, 2008), a técnica de valor agregado EVM permite acompanhar o desempenho do projeto em termos de custo e tempo em um dado momento durante a execução do projeto. Basicamente, a técnica permite calcular:

i. A diferença entre o valor agregado e o custo real para um determinado escopo de trabalho em uma data específica;

ii. A diferença entre o valor agregado e o valor planejado para um determinado escopo de trabalho em uma data específica.

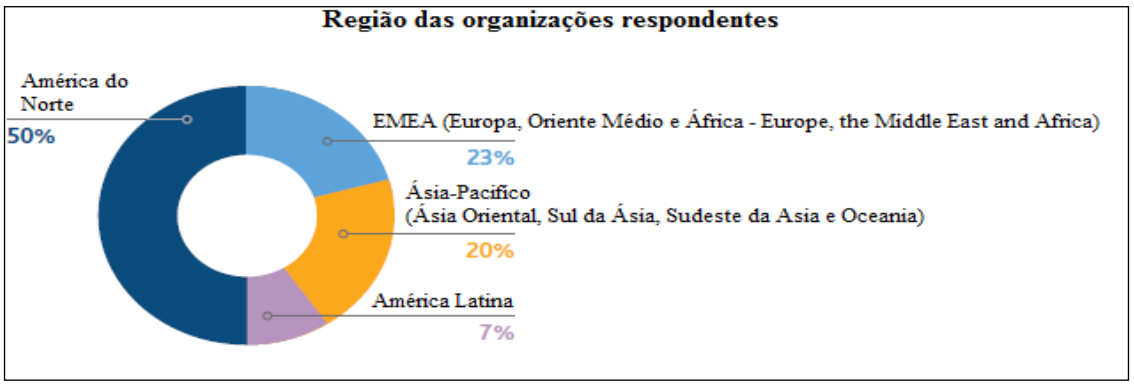
Se a diferença de custo em (i) for positiva, isso indica que o projeto está gastando menos do que o planejado e, em caso de valor negativo, o projeto tem superação de custo. Se a diferença de tempo em (ii) for positiva, isso significa que o projeto está adiantado. As entradas para essa técnica são: o valor agregado, o valor planejado e o custo real do projeto.

O conceito de valor agregado começou na década de 1890, quando os primeiros engenheiros industriais mediram o desempenho nas fábricas americanas. Eles definiram uma "variação de custo" para relacionar os "padrões ganhos" com os "gastos reais" para determinar o desempenho do projeto. Em 1962 os conceitos de valor agregado foram formalmente introduzidos em projetos pela Marinha dos EUA. Em 1996, um novo conjunto de critérios foi produzido para encorajar a adoção no setor privado. A Associação Industrial de Defesa Nacional (NDIA), nos Estados Unidos, desenvolveu critérios e nomeou os critérios de Sistema de Gerenciamento de Valor Agregado (EVMS), atualmente incorporados no ANSI/EIA 748. Por fim, o PMBOK, desenvolvido pelo *Project Management Institute* (PMI), recomenda a utilização de um conjunto semelhante de critérios de valor agregado, como parte do gerenciamento de custos de projetos e de comunicações de projetos (relatórios de desempenho) (CABRI; GRIFFITHS, 2006).

As informações a seguir foram coletadas em uma pesquisa do *Project Management Institute* (2017), que contém informações de 3.234 profissionais de gerenciamento de projetos de diversas indústrias, dentre estes profissionais 200 são executivos seniores e 510 diretores de PMO ("Project Management Office", ou escritório de gerenciamento de projetos. Departamento na empresa que padroniza os processos relativos ao gerenciamento de projeto) de diversas indústrias, e entrevistas com 10 líderes corporativos e 7 diretores e diretores de gerenciamento de projetos do PMO, sendo que a Figura 2 representa as regiões dos respondentes da pesquisa e a Figura 3 a atividade principal das empresas na qual estes profissionais trabalham.

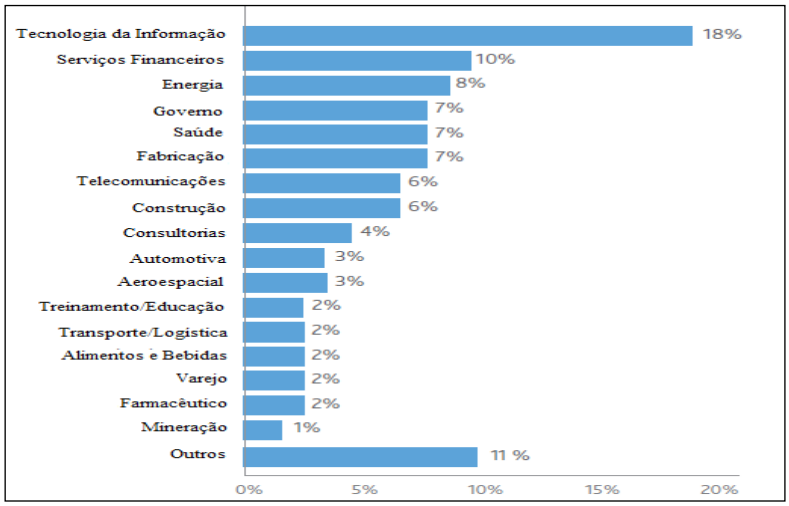
A pesquisa não indica que as informações foram coletadas em projetos de desenvolvimento de sistemas, mas mostram informações pertinentes ao desenvolvimento de *software* e, portanto, foi aqui considerada devido à abrangência e também devido aos métodos indicados nos projetos, como o modelo cascata, métodos ágeis e híbridos (métodos ágeis/cascata).

Figura 2: Regiões da pesquisa.



Fonte: 9th Global Project Management Survey (2017).

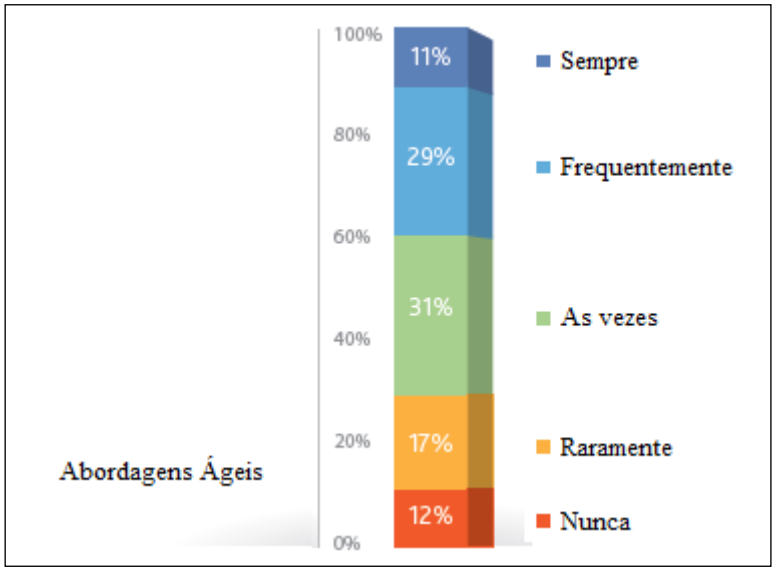
Figura 3: Atividade principal das empresas respondentes.



Fonte: 9th Global Project Management Survey (2017).

As organizações adotam cada vez mais práticas ágeis para gerenciar projetos. Um total de 71% das organizações relata o uso de abordagens ágeis para seus projetos às vezes, muitas vezes ou sempre (Figura 4). Um em cada cinco projetos usou abordagens ágeis, enquanto outro em cada cinco usou abordagens híbridas ou misturadas.

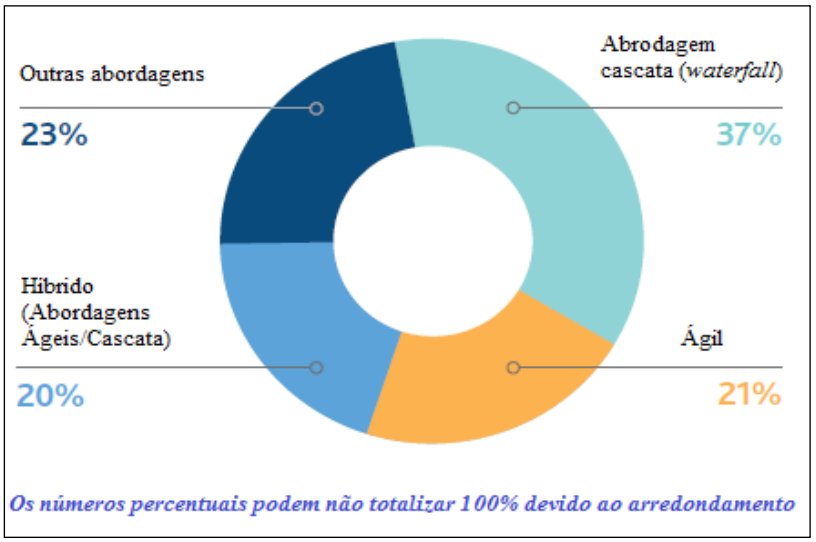
Figura 4: Abordagens Ágeis.



Fonte: 9th Global Project Management Survey (2017).

No cenário apresentado uma das questões foi: “Em sua opinião, qual a porcentagem dos projetos concluídos em sua organização que usou os seguintes tipos de abordagens? ” (Figura 5).

Figura 5: Projetos concluídos X Modelos.

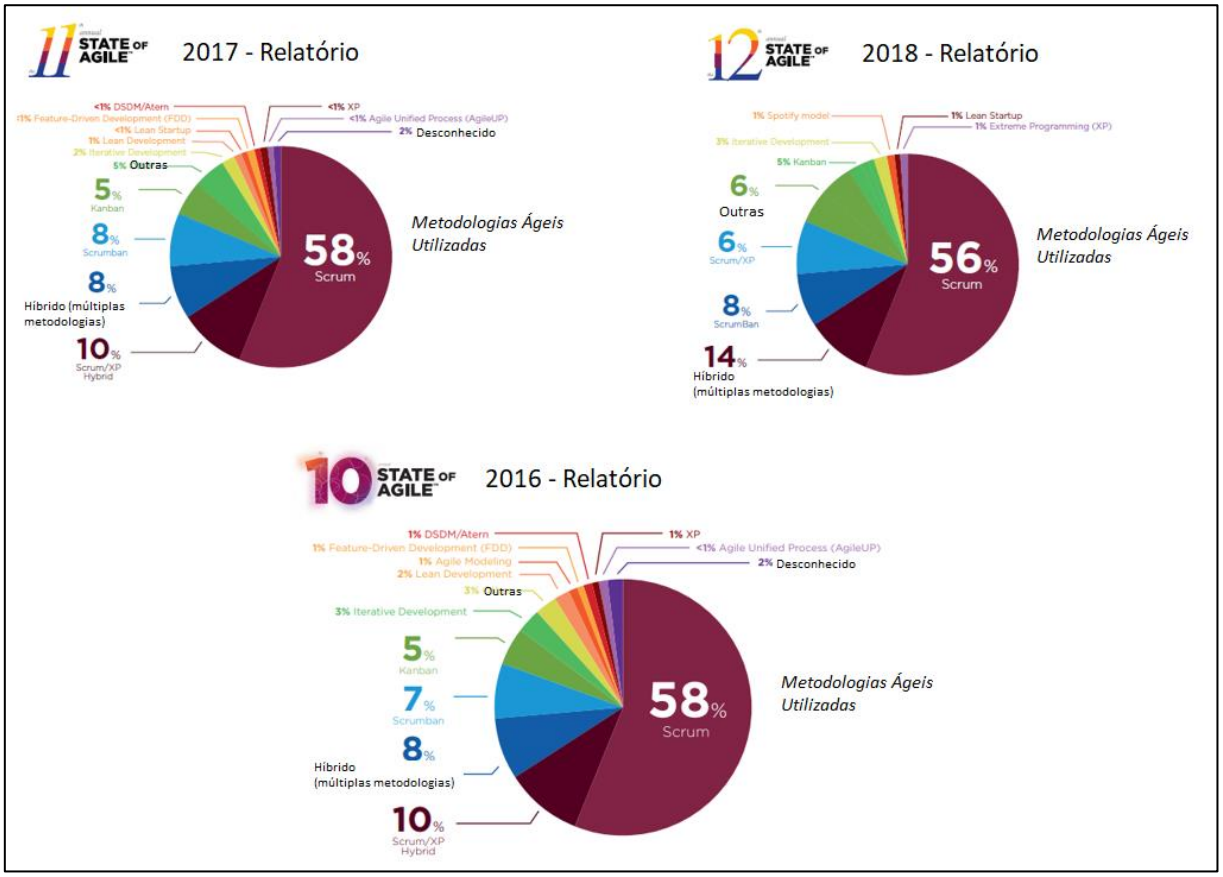


Fonte: 9th Global Project Management Survey (2017).

Assim, têm-se além dos modelos cascata e métodos ágeis um modelo denominado híbrido com uma fatia percentualmente relevante, nas quais características da abordagem cascata e de métodos ágeis são utilizadas em conjunto para fazer uma adaptação de conceitos em uma determinada estrutura organizacional.

Complementando, no levantamento sobre adoção de métodos ágeis, na Figura 6, temos uma comparação de evolução referente os métodos ágeis mais utilizados. Verificamos que o método representado pelo *framework Scrum*, representa o maior percentual, mas os modelos híbridos aparecem como a 2ª fatia mais utilizada no relatório do 12º *State of Agile Report* em 2018, tendo um aumento em relação as pesquisas de 2016 e 2017.

Figura 6: Relatório – Métodos Ágeis Utilizados.

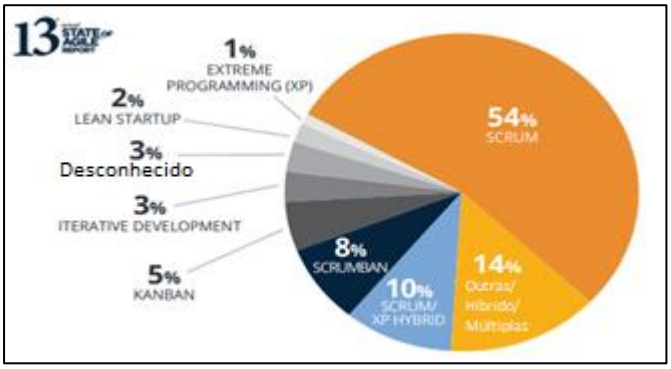


Fonte: Version One. State of Agile Survey (2016, 2017 e 2018).

Na Figura 7, contendo o relatório mais atual sobre o levantamento da adoção de métodos ágeis, 13th *Annual State of Agile Report* (2019), o *framework Scrum* aparece como o método mais utilizado. Em segundo lugar, com 14%, está a categoria "Híbrido". Mas a pesquisa 13th *Annual State of Agile Report* (2019) não deixa explícita a definição desta categoria, logo não podemos afirmar que corresponde à definição de modelo híbrido que utilizamos aqui, baseada em Kuhrmann et al. (2017)[6]. Na seção "Técnicas Ágeis Empregadas" do 13th *Annual State of Agile Report* (2019), os resultados indicam que 61% dos que responderam da pesquisa utilizam "*Planning poker*/estimativa por equipe", o que indica que há espaço para a adoção de outras técnicas de estimativa.

[6] A nossa definição de modelo híbrido inclui, necessariamente, a combinação de um método ágil com um método tradicional. Aparentemente a definição da pesquisa 13th *Annual State of Agile Report* permite que um modelo híbrido seja a combinação de dois ou mais métodos ágeis, como se pode ver na Figura 7 no item "Híbrido *Scrum*/XP".

Figura 7: State of Agile 2019.



Fonte: 13th Annual State of Agile Report (2019).

Também conforme a definição anterior de modelo híbrido, temos em Kuhrmann et al. (2019), Tabela 2, os três principais motivadores para implementar uma abordagem híbrida de desenvolvimento.

Tabela 2. Motivadores de abordagens híbridas.

Motivadores	**Exemplos**
Gerenciamento de projetos ou produtos e comprometimento	Melhora a estabilidade da equipe, o gerenciamento do ciclo de vida do produto, a integração de abordagens "*Waterfall* e ágeis "e a falta de adesão da administração.
Evolução e pragmatismo	Evolução constante, uma transição gradual da empresa para o desenvolvimento ágil.
Operação de projeto e flexibilidade	Flexibilidade de equipes e recursos, selecionando a abordagem mais adequada, evolução rápida dos requisitos do produto, cumprimento dos prazos e melhor controle dos requisitos voláteis.

Fonte: Kuhrmann et al. (2019)

Abordagens híbridas foram vistas como um caminho para equipes de projeto mais estáveis, porém flexíveis. Ao incluir abordagens clássicas conhecidas, as abordagens híbridas ajudam a melhorar o comprometimento de gerenciamento, enquanto os desenvolvedores obtêm flexibilidade usando práticas ágeis. Com relação a evolução e pragmatismo, a abordagem atual não foi planejada, mas sim evoluiu a partir das diferentes práticas de trabalho aplicadas.

Assim, conforme Kuhrmann et al. (2019), os processos tradicionais de desenvolvimento de software, complementados por normas, padrões e regras, devem apoiar o desenvolvimento e a qualidade do produto. Em comparação com os métodos ágeis, essas abordagens são consideradas pesadas, a ponto de as empresas serem obrigadas a procurar alternativas para aumentar a agilidade.

Também em Kuhrmann et al. (2019), temos que abordagens de desenvolvimento híbrido representam uma solução que, independentemente do tipo de empresa e setor da indústria, permite que as empresas se beneficiem dos dois mundos, fornecendo aos clientes e à

gerência um ambiente seguro e desenvolvedores com a flexibilidade exigida. Abordagens de desenvolvimento híbridas surgiram gradualmente, emergindo da experiência e são adaptadas durante os projetos em resposta a determinadas situações.

2.5 CICLO DE DESENVOLVIMENTO DE *SOFTWARE*

O desenvolvimento de um *software* tem por objetivo entregar um produto que atenda as expectativas de qualidade, tempo, orçamento e que contemple os requisitos indicados pelo cliente. O ciclo de desenvolvimento identifica os estágios estabelecidos durante o projeto do *software*.

O ciclo de desenvolvimento de *software* desempenha um papel importante, pois ajuda a definir os requisitos de *software*, modelar o componente de *software*, reduzir o custo de desenvolvimento e manutenção e, finalmente, fornecer software gerenciável (SHAH, 2016).

O ciclo de desenvolvimento nada mais é do que o desenvolvimento de um *software* dividido em etapas, como: estudo da viabilidade, análise do sistema, projeto, implementação, geração do teste de aceite, garantia de qualidade, descrição de procedimentos, conversão de banco de dados e instalação (YOURDON, 1989).

SDLC (*Software Development Life Cycle*) é um acrônimo que é usado para descrever ciclos de vida de desenvolvimento de *software* ou sistemas. A utilização de um SDLC varia de acordo com o ambiente em que o *software* é desenvolvido (RUPARELIA, 2010).

Alguns ciclos de vida são baseados em etapas, como levantamento de requisitos, testes, implantação e com um trabalho de documentação bem detalhado. Dentro destes modelos pode-se citar "RUP (*Rational Unified Process*), modelo Espiral (AZAM et al., 2010) e o modelo Cascata".

Com relação ao ciclo de desenvolvimento cascata, este foi proposto por Winston Royce em 1970 sendo utilizado até os dias atuais. O modelo cascata teve um efeito no desenvolvimento de software e influenciou muitos modelos SDLC prevalentes hoje (RUPARELIA, 2010).

O modelo cascata foi documentado pela primeira vez por Benington em 1956 e modificado por Winston Royce em 1970. Este modelo sustentou todos os outros modelos, pois criou uma base sólida para os requisitos a serem definidos e analisados antes de qualquer projeto ou desenvolvimento (RUPARELIA, 2010). Conforme Larman e Basili, (2003), no conhecido artigo "Managing the Development of Large Software Systems", Royce (1970), Winston Royce compartilhou suas opiniões sobre o que seria conhecido como o modelo Cascata. No modelo original cascata, Bennington recomendou que o software fosse desenvolvido em etapas:

Análise Operacional => Especificação Operacional => Desenho e Especificações de Códigos => Desenvolvimento => Testes => Implantação => Avaliação (RUPARELIA, 2010).

Ao reconhecer que pode haver dificuldades de projeto imprevistas quando uma linha de base é criada no final de cada etapa, Royce aprimorou esse modelo fornecendo um loop de *feedback* para que cada etapa anterior pudesse ser revisada. Esta iteração permitiu que as etapas se sobrepusessem, além da etapa anterior, a ser revisada (Figura 8).

Royce também sentiu que esse arranjo poderia ser inadequado, uma vez que a iteração talvez precise transcender a iteração do par do estágio precedente. Assim, a Figura 8 mostra o loop de *feedback* mais complexo por meio de setas tracejadas. Além disso, Royce sugeriu que uma etapa de projeto preliminar pudesse ser inserida entre os requisitos e os estágios de análise. Isso iria abordar o papel que a fase de projeto desempenha na minimização de riscos com o *feedback* da Figura 8 (RUPARELIA, 2010).

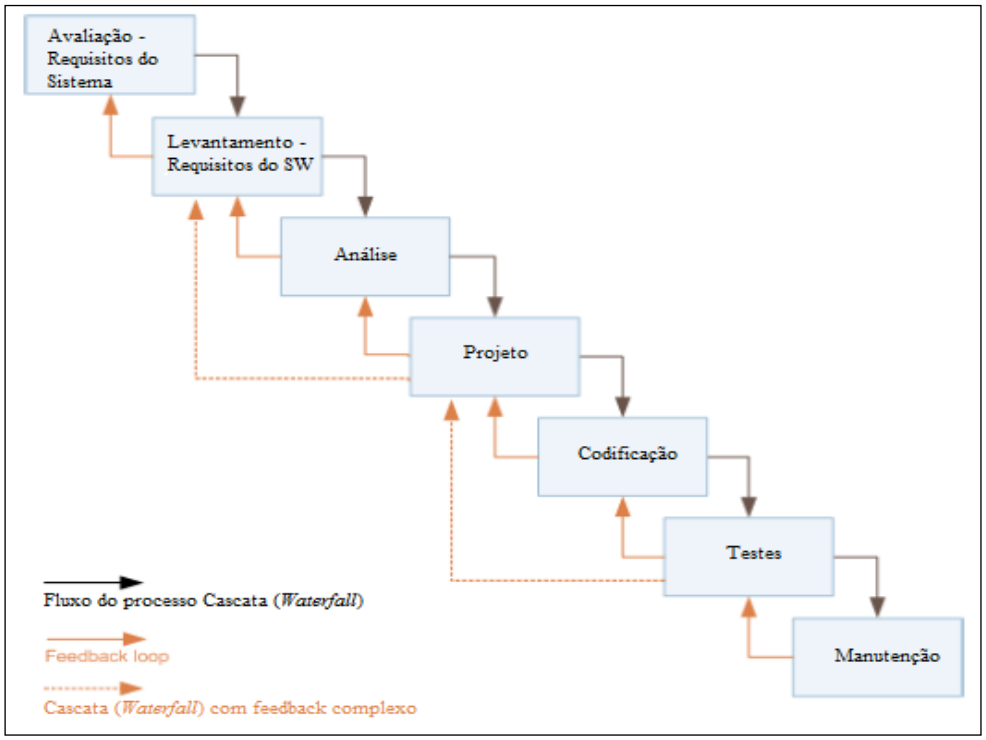
Também em (RUPARELIA, 2010), Royce sugeriu que fossem produzidos pelo menos seis tipos distintos de documentos:

1. Documento de requisitos durante a fase de requisitos.
2. Especificação de projeto preliminar durante a fase de projeto preliminar.
3. Especificação do projeto da interface durante a fase de projeto.
4. Especificação de projeto final que é ativamente revisada e atualizada em cada visita do estágio de projeto. Isso é atualizado ainda mais durante as etapas de desenvolvimento e validação.
5. Plano de teste durante a fase de projeto. Este documento é atualizado posteriormente com os resultados do teste durante a fase de validação ou teste.
6. Manual ou instruções de operação durante o estágio de implantação.

A garantia de qualidade é incorporada no modelo cascata dividindo cada etapa em duas partes: uma parte executa o trabalho e a outra verifica. Por exemplo, o estágio de projeto incorpora verificação (para avaliar se é adequado), o estágio de desenvolvimento possui testes de unidade e integração e a fase de validação contém testes de sistema.

O ciclo de desenvolvimento cascata fornece uma estrutura clara para organizar e controlar as atividades durante todo o ciclo de desenvolvimento de software (PAPADOPOULOSA, 2014) sendo amplamente utilizado na indústria (RAJLICH, 2006). Para este ciclo de desenvolvimento (Figura 8), inicialmente são definidos os requisitos. A seguir tem-se a etapa de projeto (requisitos do sistema agrupados para estabelecer sua arquitetura), após tem-se a fase de codificação e os testes de unitários (isso envolve a verificação de cada unidade se atende a sua real especificação). Após essas etapas, a integração e o teste do sistema (integração de todas as unidades construídas) são realizados. Por último, têm-se a operação e a manutenção (na qual o sistema é colocado em operação e recebe manutenção).

Figura 8: Modelo Cascata com o feedback iterativo de Royce.



Fonte: Ruparelia (2010).

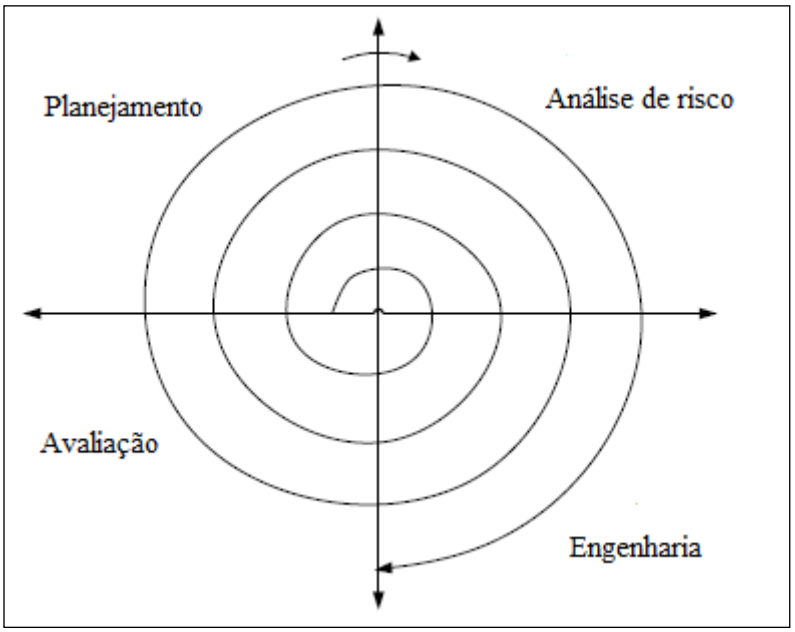
O antigo paradigma Cascata tentou congelar os requisitos durante a duração do desenvolvimento de *software* e criou uma situação em que a volatilidade dos requisitos causou muitas falhas no projeto. Esses tópicos foram negligenciados no passado pelos pesquisadores (RAJLICH, 2006).

O modelo de desenvolvimento de *software* espiral foi inicialmente apresentado por Boehm em 1986, com a ideia básica de minimizar o risco com o uso de protótipos (LUMA-OSMANI; ARIFI, 2014). Conforme a Figura 9, o modelo espiral consiste em quatro fases: Planejamento, Análise de Risco, Engenharia e Avaliação.

As definições das fases do modelo espiral são (LUMA-OSMANI; ARIFI, 2014):

- Planejamento: Envolve a determinação dos propósitos, alternativas e restrições. Além disso, a identificação dos riscos do projeto, os requisitos do sistema e a estimativa de custos, tecnologia e cronograma são feitos nesta fase.
- Análise de risco: é a parte mais importante do modelo em questão, que inclui avaliação de alternativas e identificação dos riscos como principal objetivo desta etapa. Por exemplo, se houver o risco de os requisitos serem inadequados, um sistema protótipo pode ser desenvolvido.
- Engenharia: Execução da fase de desenvolvimento. Nesta fase, desenvolvemos o produto planejado e testamos o mesmo. Para fazer o desenvolvimento, o modelo cascata ou a abordagem incremental podem ser implementadas.
- Avaliação: Tem a ver com a entrega do protótipo para o cliente, a fim de obter suas sugestões e comentários se deseja continuar com o loop adicional e, em seguida, agir de acordo com seus novos requisitos. É semelhante ao teste.

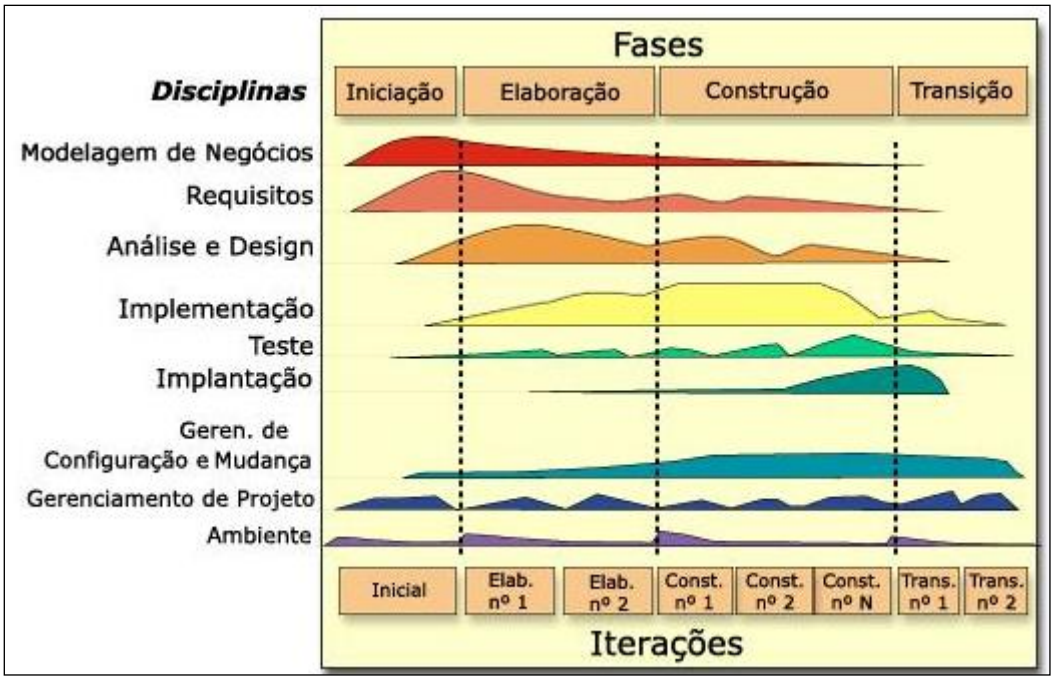
Figura 9: Espiral.



Fonte: Luma-Osmani e Arifi (2014).

O RUP (*Rational Unified Process*) é um processo de engenharia de *software* criado pela *Rational Software* Corporation e usa a abordagem de orientação a objetos sendo documentado utilizando a UML (*Unified Modeling Language*). A Figura 10 mostra as fases do processo RUP, conforme (HIRSCH, 2002).

Figura 10: RUP.



Fonte: Hirsch (2002).

Com relação às disciplinas (Figura 10) temos:

- Modelagem de Negócios: Descreve os processos de negócios e a estrutura interna do processo, a fim de entender melhor os negócios e obter os requisitos adequados para os sistemas de software a serem construídos para os negócios em questão.
- Requisitos: Organização e documentação dos requisitos.
- Análise e Design: Criando a arquitetura do sistema de software.
- Implementação: Escrever o código-fonte, testes unitários e gerenciamento de compilação.
- Teste: Teste de integração, sistema e aceitação.
- Implantação: Empacotar o software, criar scripts de instalação, escrever a documentação do usuário final e outras tarefas necessárias para disponibilizar o software para seus usuários finais.
- Configuração e Gerenciamento de Mudanças: Abrange todas as tarefas relacionadas ao gerenciamento de versão e liberação com gerenciamento de solicitação de mudança.
- Gerenciamento de Projetos: Planejamento e monitoramento de projetos.
- Ambiente: Adaptar o processo às necessidades de um projeto. Foco nas atividades necessárias para configurar o processo para o projeto de software.

Também com relação a Figura 9, no eixo horizontal que representa a evolução de tempo do projeto de *software* têm-se as seguintes definições:

- Iniciação: Definição dos objetivos do projeto.
- Elaboração: Criação e validação da arquitetura do sistema de *software*, capturando os requisitos mais importantes e críticos e planejando e estimando o restante do projeto.
- Construção: Implementação do sistema com base na arquitetura executável criada na elaboração
- Transição: Teste do sistema e preparação para liberação final.

2.5.1 Combinação de Abordagens de Desenvolvimento de Software

Para muitos projetos, o gerenciamento puro de projetos tradicionais não é eficaz, e os modelos híbridos são a solução mais apropriada. O método tradicional de gerenciamento de projetos é melhor para os projetos com um objetivo e uma solução claros. Por outro lado, os projetos que não têm objetivos e soluções claras são gerenciadas mais com métodos ágeis, mas os métodos ágeis puros não são bons o suficiente para muitos desses projetos, sendo ajustada de forma flexível, resultando no método híbrido flexível. Acredita-se que esta combinação de práticas ágeis com o método tradicional que forma os modelos híbridos de gerenciar os projetos de TI está apenas emergindo (RAHMANIAN, 2014).

Todos os ciclos de desenvolvimento são complementares e não competitivos. Existem três grandes categorias. Modelos tradicionais, modelos ágeis e modelos híbridos. A combinação de modelos tradicionais e modelos ágeis pode se beneficiar da vantagem de ambos os modelos,

chamados de modelo híbrido. A principal armadilha dos modelos tradicionais é a menor comunicação, que é a maior vantagem dos modelos ágeis. No modelo híbrido, no estágio inicial os modelos tradicionais são usados para fins de documentação e análise. Os modelos ágeis são usados para fazer a codificação e o teste, considerando a comunicação frequente com os clientes, a análise contínua e a liberação periódica do produto (SHAH, 2016).

Tendo como referências métodos tradicionais como cascata e métodos que permitem interações como XP (*Extreme Programming*) e *Scrum*, têm-se as abordagens híbridas com conceitos destes métodos que estão sendo utilizadas atualmente e indicadas na Seção 2.4, Figura 5, Figura 6 e Figura 7.

O modelo híbrido pode ser qualquer combinação de modelos tradicionais e modelos ágeis se beneficiando das características de cada modelo. Na prática, os desenvolvedores estão se movendo para adotar o modelo híbrido para alcançar os principais benefícios dos modelos tradicionais e ágeis. No entanto, a aplicabilidade ainda depende do ambiente de *software* (SHAH, 2016).

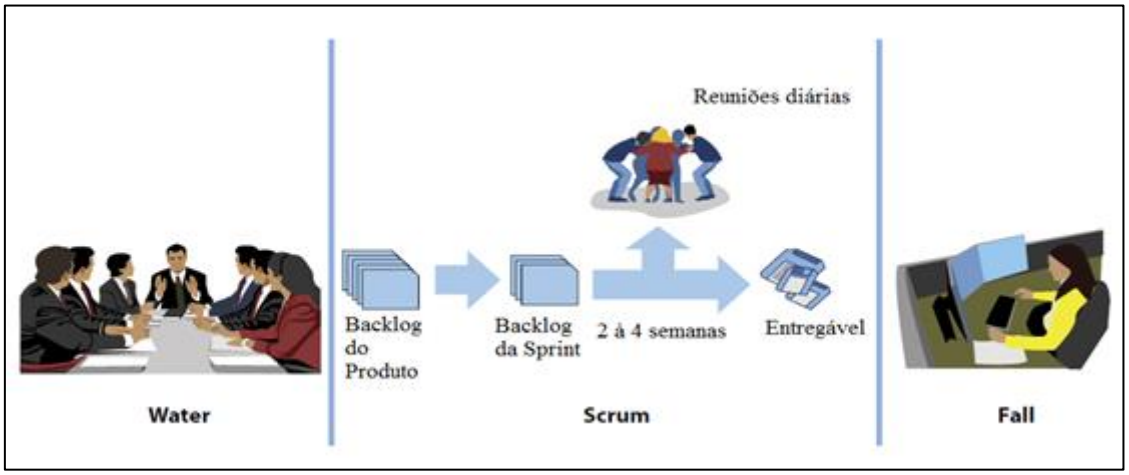
Os modelos híbridos surgiram como uma tentativa para utilizar os modelos ágeis e o modelo cascata integrando um conjunto de práticas ágeis nos processos de desenvolvimento em cascata (HASSANI et al., 2018).

Segundo Kuhrmann et al. (2017), uma abordagem de desenvolvimento de *software* híbrido é qualquer combinação de abordagens ágeis e tradicionais que uma unidade organizacional adote e personalize. Abordagens híbridas são usadas na prática por empresas, independentemente do tamanho da empresa e do setor da indústria, e combinações de abordagens de desenvolvimento seguem um padrão no qual um processo tradicional serve como estrutura reestabelecida por práticas múltiplas.

Conforme West (2011), os métodos híbridos são uma realidade na maioria das implementações ágeis. Isso acontece em parte porque a adoção de métodos ágeis tem sido conduzida por praticantes, levando as equipes a se concentrar em domínios que podem influenciar principalmente a própria equipe. Áreas fora de seu controle, como análise de negócios e gerenciamento de lançamento, continuam a seguir abordagens mais tradicionais, o que significa que a adoção do Scrum é limitada ao nível da equipe de desenvolvimento (Figura 11). Os requisitos de conformidade são outro fator que orientam abordagens híbridas, pois exigem processos de governança sólidos antes e depois do desenvolvimento.

Os métodos ágeis e tradicionais (como cascata) têm características peculiares na qual cada um claramente funciona melhor e o outro terá dificuldades. As abordagens híbridas que combinam ambos os métodos são viáveis e necessários para projetos que combinem características locais e ágeis. A análise de risco das características do seu projeto pode ajudar a determinar o melhor equilíbrio de métodos ágeis e tradicionais (BOEHM, 2002). Embora os métodos ágeis sejam mais prevalentes do que 10 anos atrás, os métodos tradicionais ainda são populares. As organizações também utilizam múltiplos métodos em projetos (VIJAYASARATHY; BUTLER, 2016).

Figura 11: Modelo Híbrido (Water-Scrum-Fall).



Fonte: West (2011).

Desde o final da década de 1960, inúmeros métodos foram desenvolvidos para abordar os vários desafios que ocorrem durante o desenvolvimento de *software*. O advento dos métodos ágeis como resposta a alguns problemas de engenharia de *software* causou um acirrado debate entre desenvolvedores de *software* desde o início de 2000, com proponentes de métodos clássicos, como o modelo cascata, questionando o valor dos métodos ágeis, como XP e *Scrum*. Por um lado, engenheiros de *software* descartam métodos ágeis e defendem fortemente o valor da prática clássica, enquanto outras insistem que os métodos ágeis substituirão os modelos semelhantes ao cascata e se aplicarão a todos os projetos de *software* (Jiang e Eberlein, 2008).

Conforme Theocharis et al. (2015), diferentes abordagens são geralmente usadas em combinação, ou seja, as abordagens misturadas ou híbridas representam o modelo comum de uso, e a combinação inclui abordagens tradicionais e de métodos ágeis. Entre as diferentes combinações, o *Scrum* e o modelo cascata representam a maioria das menções. O *Scrum* tornou-se o método ágil mais popular e os métodos ágeis são adaptados e combinadas com outros processos (tradicionais e ágeis).

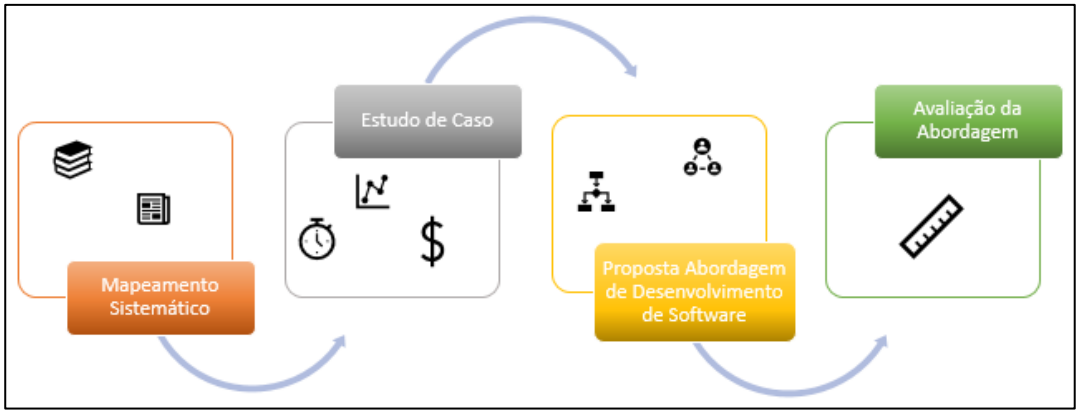
3 METODOLOGIA

Inicialmente, para o desenvolvimento desta dissertação, fizemos um mapeamento sistemático que tem como finalidade ajudar a responder o objetivo deste trabalho e caracterizar o estado da arte.

Em seguida, realizamos um estudo de caso com a finalidade de estudar em um ambiente real a viabilidade da estimativa de *software* por pontos de função. Comparamos os resultados dos indicadores de custo e tempo entre projetos que utilizaram o método cascata com projetos que utilizam práticas ágeis.

Para finalizar, propusemos uma abordagem de desenvolvimento de *software* e avaliamos esta abordagem por meio de um questionário de pesquisa e executando parcialmente esta abordagem de desenvolvimento. Denominamos a abordagem de desenvolvimento de HiPA (Híbrida com Práticas Ágeis).

Figura 12: Metodologia.



Fonte: Autoria própria.

3.1 MAPEAMENTO SISTEMÁTICO

Neste mapeamento sistemático buscamos pesquisas sobre medições de *software* para obter evidências e lacunas que poderiam direcionar novas pesquisas, relacionando os conceitos de medições aos métodos de desenvolvimento de sistemas em projetos. Este mapeamento teve como guia o protocolo apresentado por Kitchenham (2004).

Um mapeamento sistemático possibilita uma revisão dos estudos existentes em um tópico específico para identificação de lacunas e sugestão de pesquisas futuras (Kitchenham et al., 2015). O mapeamento sistemático é adequado para este trabalho, pois a pesquisa tem um escopo amplo e permite estruturar o estudo de maneira sistemática para categorizar as informações e conceitos relevantes em um campo de pesquisa. A seguir é apresentado o protocolo definido para o mapeamento sistemático.

Assim, com o protocolo definido, elaboramos os questionamentos a serem respondidos e definimos uma estratégia de busca e os repositórios utilizados. Conduzimos o mapeamento para extração dos trabalhos e seleção destes trabalhos e, por fim, apresentamos os resultados para responder as questões de pesquisa, tendo como referência os trabalhos selecionados.

3.1.1 Objetivo

O objetivo deste mapeamento sistemático é identificar projetos de desenvolvimento de *software* híbridos com práticas ágeis e estimados por pontos de função.

3.1.2 Questões de Pesquisa

Para caracterizar o panorama de estudos da utilização de pontos de função com práticas ágeis, este mapeamento sistemático pretende responder as seguintes questões no contexto de pesquisa:

> ***QP1****: Pontos de Função são utilizados em projetos que utilizam práticas ágeis de desenvolvimento?*

A resposta a ***QP1*** deve fornecer um panorama da utilização de métricas de estimativas de tamanho de *software*, neste caso específico para pontos de função, juntamente com a utilização de práticas ágeis no desenvolvimento de *softwares*.

> ***QP2****: Existe uma combinação do ciclo de desenvolvimento cascata e de práticas ágeis sendo utilizada no desenvolvimento de software?*

A resposta a ***QP2*** fornecerá informações de abordagens híbridas de desenvolvimento de *software* e assim ambas as questões permitem explorar a utilização de pontos de função com práticas ágeis e buscar uma abordagem híbrida de um ciclo de desenvolvimento de *software*.

Ambas questões podem identificar novas abordagens de desenvolvimento de sistema, nas quais podem ser combinadas características de desenvolvimento que juntas tendem a ser mais produtivas para um determinado contexto, sendo assim uma abordagem híbrida. E

utilizando como parâmetro para medição a técnica de pontos de função.

A identificação de trabalhos que utilizam uma abordagem hibrida de desenvolvimento com a métrica de pontos de função ajudaria a mostrar características pertinentes ou não deste modelo híbrido, e eventuais ganhos ou perdas com a utilização dessas abordagens de desenvolvimento que utilizam práticas ágeis que por sua vez são utilizados em ambientes dinâmicos e propensos a mudanças.

3.1.3 Estratégia de Busca

A estratégia de busca tem como objetivo encontrar pesquisas relacionadas a pontos de função. No desenvolvimento de *software*, a abordagem foi direcionada para o ciclo de desenvolvimento Cascata e para o *framework Scrum, s*endo a palavra-chave *Scrum* usada em combinação com a palavra-chave *Agile*.

Assim, temos as seguintes palavras-chave:

- *Software*
- *Waterfall*
- *Function Point*
- *Agile ou Scrum*

Os termos selecionados estão em inglês porque este é idioma utilizado pelos repositórios de pesquisas indicados a seguir. Assim, as palavras-chave que formaram a *string* de busca, neste mapeamento sistemático, têm por objetivo buscar pesquisas de medições de software para caracterizar o estado atual e buscar evidências e lacunas que poderiam direcionar novas pesquisas, relacionando os conceitos de medições de tamanho de *software* aos métodos de desenvolvimento em projetos de sistemas.

Não utilizamos a palavra “híbrido” para termos foco nos métodos ágeis e tradicionais, buscando assim conceitos intercalados entre esses métodos, direcionando principalmente para o modelo cascata, que é o modelo tradicional mais conhecido e utilizado em desenvolvimento de *software*.

3.1.4 Fontes de Pesquisa

Os repositórios de pesquisas utilizados para o mapeamento sistemático foram:

- *ACM Digital Library*[7]
- *IEEE Xplore Digital Library*[8]

[7] https://dl.acm.org/

[8] https://ieeexplore.ieee.org/Xplore/home.jsp

- *Scopus*[9]
- *Science Direct*[10]
- *Springer Link*[11]

Estes repositórios permitem que o mapeamento tenha um escopo amplo com vários referenciais de pesquisas que podem ser usados para o propósito.

3.1.5 *String* de Busca

Tendo identificado as palavras-chave e identificado os repositórios, elaboramos a seguinte string de busca para realizar o mapeamento:

- *"(Software) AND (Waterfall) AND (Function Point) AND (Scrum OR Agile)"*

Esta *string* direciona a pesquisa para medições de software por meio da métrica de pontos de função e uma combinação dos métodos de desenvolvimento tradicional (cascata) associando também um método ágil (*Agile*) com foco no *framework Scrum*.

O operador lógico “AND” é utilizado para obter trabalhos com todas as palavras não tendo assim combinações parciais que poderiam ser feitas com o operador lógico “OR”. Assim, a calibração foi feita com as palavras-chave para chegarmos a uma sentença de busca com a métrica de ponto de função, práticas ágeis e o modelo cascata.

Desta maneira, a *string* de busca tem por finalidade, encontrar trabalhos que respondam as questões de pesquisa ***QP1*** e ***QP2*** mesclando conceitos por meio das palavras-chave definidas.

3.1.6 Critérios de Seleção

Kitchenham (2004) informa que devem ser seguidos critérios de inclusão e exclusão para os artigos que são retornados pela *string* de busca. Assim, foram definidos alguns critérios de inclusão e exclusão de artigos.

Neste trabalho o período para a busca foi definido de Janeiro de 2000 a Agosto de 2019 para limitar o resultado em trabalhos mais recentes. Os critérios de inclusão e exclusão de artigos foram empregados em um primeiro filtro, com a seleção de trabalhos por meio do título e do abstract e, em um segundo filtro, por meio da leitura completa dos estudos selecionados no primeiro filtro.

Os critérios de inclusão e exclusão de artigos foram definidos como:

---

[9] https://www.scopus.com/standard/marketing.uri

[10] https://www.sciencedirect.com/

[11] https://link.springer.com/

- Estudos que abordavam o uso de pontos de função em processos de desenvolvimento que empregam práticas ágeis foram selecionados.
- Estudos que investigavam a combinação do modelo cascata e de práticas ágeis no desenvolvimento de software foram selecionados.
- Trabalhos disponibilizados pelos repositórios, mas que devem ser pagos para possibilitar a leitura foram excluídos.
- Trabalhos duplicados foram excluídos.

3.2 ESTUDO DE CASO

Conforme Wohlin et al. (2012), estudos de caso são conduzidos para investigar uma única entidade ou fenômeno em seu contexto real, em um espaço de tempo específico, selecionando variáveis que representam uma situação típica e pode ser aplicado como uma estratégia de pesquisa comparativa, comparando os resultados de usar um método ou alguma forma de manipulação, com os resultados de usar outra abordagem. Para evitar preconceitos e garantir a validade interna, é necessário criar uma base sólida para avaliar os resultados do estudo de caso.

Segundo Kitchenham et al. (1995), um exemplo de abordagem de estudo de caso pode ser o uso de um projeto piloto para avaliar os efeitos de uma mudança em comparação com alguma linha de base. Outro exemplo é uma comparação dos resultados da utilização de um novo método em relação a uma linha de base da empresa. A empresa deve coletar dados de projetos padrão e calcular características como produtividade média e taxa de defeitos. Então é possível comparar os resultados do estudo de caso com os números da linha de base.

Neste trabalho, realizamos um estudo de caso para analisar a relação entre projetos, fazendo uma comparação de resultados e explorando os indicadores gerados por estes projetos, que foram estimados por pontos de função e executados com práticas ágeis, alinhados assim ao objetivo de verificar a viabilidade da utilização de pontos de função em projetos de *software* que utilizam práticas ágeis.

A metodologia descrita para a realização do estudo de caso serve para detalhar a exploração de informações de maneira cronológica. Os resultados dos projetos analisados foram compilados em informações para gerar cenários de análise de resultados e mostrar de maneira objetiva, numérica e não tendenciosa as situações pertinentes aos objetivos descritos.

Para o entendimento do problema utilizamos um método dividido nas seguintes etapas, indicado por Kitchenham et al. (2004):

a) Concepção: Etapa de definição do problema;
b) Rastreamento: Verificação da existência de informações para delimitação dos cenários e variáveis estudadas para atingir os objetivos indicados.
c) Condução: Indicação das atividades conforme definido na etapa de rastreamento e obtenção dos valores a serem analisados;

d) Avaliação: Comparação dos dados obtidos e verificação da compatibilidade dos dados;

e) Uso dos resultados: Uso dos resultados para estabelecer conclusões.

Este estudo de caso tem por objetivo comparar as evidências geradas pelos resultados de projetos que utilizam um ciclo de desenvolvimento tradicional, neste caso cascata, com projetos que foram executados com práticas ágeis. Assim, práticas de engenharia de *software* distintas são comparadas por meio da análise dos resultados de índices de desempenho de prazo (IDP) e o índice de desempenho de custo (IDC) dos projetos, sendo que estes índices indicam o resultado final em termos financeiros e de tempo do projeto, seguindo as etapas previstas que foram conceituadas por Kitchenham et al. (2004) como engenharia de *software* baseada em evidências para estabelecer este estudo de caso.

3.2.1 Concepção

Nesta etapa são referenciados projetos de escopo fechado de uma empresa, na área de projetos de sistemas, que aqui não será identificada por motivo de sigilo. Todos os projetos indicados foram feitos somente para uma empresa cliente que também não será identificada.

Todos os projetos foram mensurados via pontos de função e possuem indicativos que podem ser numericamente quantificados para permitir uma avaliação. Estes indicativos numéricos são caracterizados aqui como variáveis que permitem uma análise situacional do projeto. Também com relação aos projetos, estes são compostos de projetos caracterizados no modelo cascata e projetos que utilizam práticas ágeis de desenvolvimento, aqui indicados como projetos híbridos. A escolha dos métodos está associada a certas características organizacionais, de projeto e de equipe.

Nesta etapa de concepção também verificamos os perfis profissionais que compõem os projetos, sendo que estes profissionais têm perfis de Gerente de Projeto, Analista de Sistemas, Analistas de Testes e Analistas Funcionais, e possuem capacitação técnica nivelada para os conhecimentos necessários para os projetos estudados. Isto foi feito devido ao reconhecimento do ambiente e do contexto do trabalho dos profissionais.

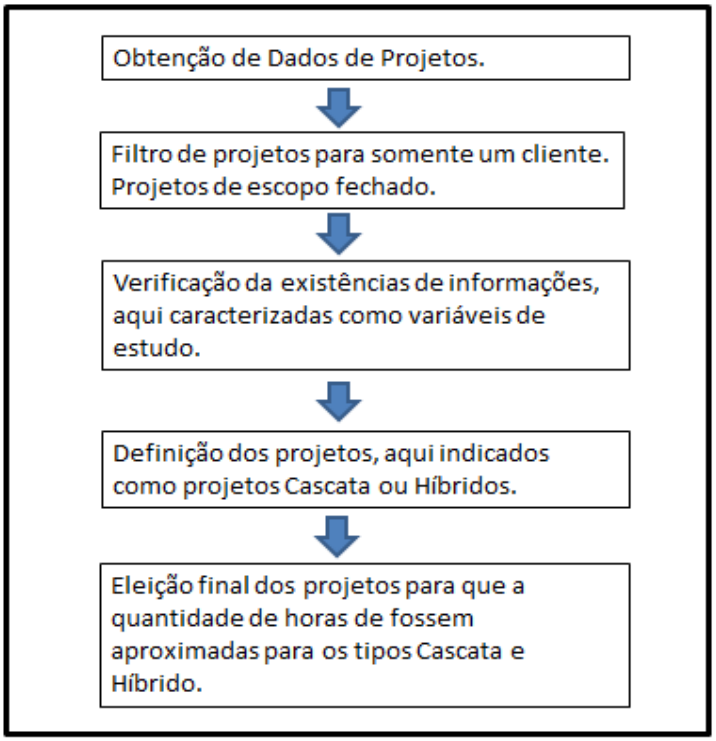
3.2.2 Rastreamento

Para estabelecer qual o domínio de informação a ser estudado, utilizamos os seguintes critérios de seleção de projetos:

- Projetos de escopo fechado;
- Projetos de um mesmo cliente.

O rastreamento foi realizado de acordo com os passos observados na Figura 13.

Figura 13: Rastreamento.



Fonte: Autoria própria.

Os indicadores, associados às variáveis de estudo, empregados no momento de análise dos projetos e de seus resultados são apresentados na Tabela 3.

Tabela 3. Indicadores de Projeto.

Objetivo	**Indicador**
Verificar o impacto da utilização de práticas ágeis em um *software* estimado por pontos de função.	Indicadores de custo dos projetos.
	Indicadores de prazo dos projetos.

Fonte: Autoria própria.

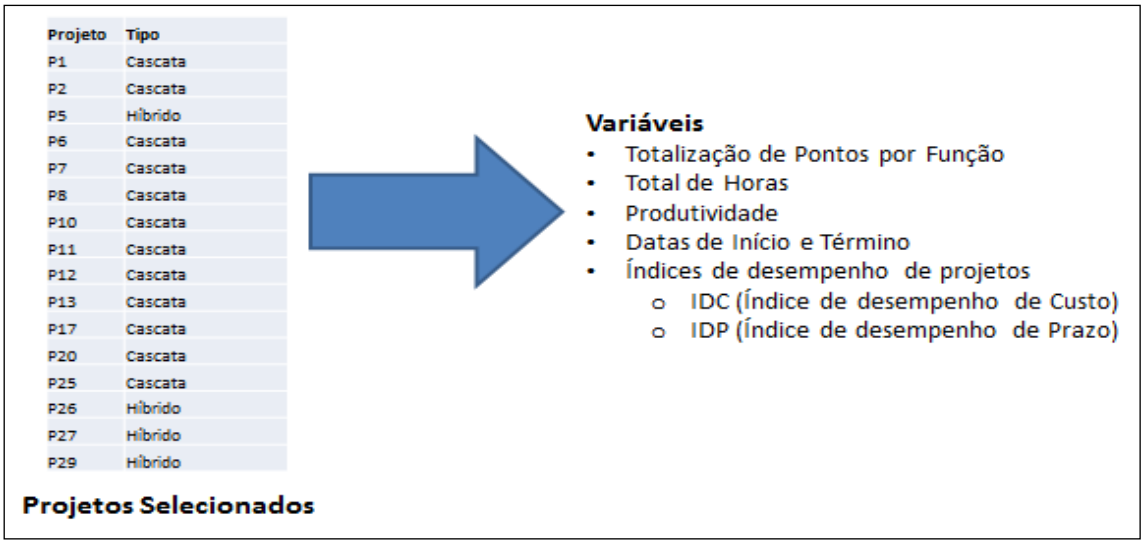
O índice de desempenho de custo (IDC) é definido aqui como sendo a medida de eficiência em termos de custo de projeto, sendo representada pela razão entre o valor do projeto e o custo real do projeto. E o índice de desempenho de prazo (IDP) é definido como sendo a razão entre o tempo real gasto do projeto e o tempo esperado, e representa a medida de eficiência com relação ao tempo do projeto.

3.2.3 Condução

A partir da obtenção dos projetos e das variáveis identificadas, estabelecemos e delimitamos o cenário a ser estudado, conforme já indicado nas fases anteriores de concepção

e rastreamento. Assim, na etapa de condução, analisamos 29 projetos, sendo que obtivemos 16 projetos que foram filtrados conforme a etapa de rastreamento e caracterizamos o ciclo de desenvolvimento. Obtivemos as informações de pontos de função, total de horas, produtividade (quantidade de horas utilizadas para se fazer 1 ponto de função), data de início e término dos projetos, índice de desempenho de custo e índice de desempenho de prazo dos projetos que são definidos aqui como as variáveis dos projetos (Figura 14).

Figura 14: Projetos e variáveis.



Fonte: Autoria própria.

3.2.4 Avaliação

Nesta etapa analisamos as variáveis de pesquisa. Estas análises foram feitas para comparar os projetos com ciclos de vida diferentes e caracterizar se estes ciclos de vida podem influenciar a métrica de pontos de função utilizada para medir os projetos.

3.2.5 Uso dos Resultados

O uso dos resultados serve para estabelecer a conclusão final do trabalho. O referencial para este uso é o objetivo estabelecido inicialmente. Assim, para os resultados obtidos com base na etapa de avaliação, é estabelecida uma conclusão podendo também indicar situações de recomendação e desvios de projetos que podem agregar indicações positivas ou negativas para os modelos de ciclo de desenvolvimento apresentados.

3.3 ELABORAÇÃO DA PROPOSTA DE UMA ABORDAGEM DE DESENVOLVIMENTO DE *SOFTWARE*

Para consolidar o estudo elaborado por meio do mapeamento sistemático e do estudo de caso e atender o objetivo geral de verificar a viabilidade da utilização de pontos de função em projetos que utilizam práticas ágeis, propusemos (Seção 5) uma abordagem de desenvolvimento de *software* híbrido que utiliza práticas ágeis e estimados por pontos de função, que é um dos objetivos específicos deste trabalho.

Esta abordagem é uma representação da execução de tarefas relacionadas ao desenvolvimento de *software* para servir como apoio nas sequências de trabalhos a serem realizados, sejam operacionais ou gerenciais, e assim obter um resultado adequado com indicativos de controle da execução do projeto.

Já existem alguns trabalhos como West (2011), Kuhrmann et al. (2017), Kuhrmann et al. (2019) e State of Agile (2019) que fazem indicações a abordagens híbridas, mas a proposta de abordagem aqui indicada é caracterizada como híbrida, ressaltando principalmente a relação com a métrica de pontos de função.

3.4 AVALIAÇÃO DA PROPOSTA

Para avaliarmos a proposta da abordagem de desenvolvimento de *software*, realizamos os seguintes passos:

- Elaboramos e submetemos um questionário de pesquisa para obter críticas de profissionais que atuaram com projetos de desenvolvimento de *software*;
- Executamos etapas da abordagem de desenvolvimento proposta e analisamos os resultados obtidos.

O questionário (Apêndice C) tem por objetivo, mapear o conhecimento, tempo de experiência e a percepção da aplicabilidade da abordagem de desenvolvimento proposta. O objetivo da avaliação da abordagem de desenvolvimento é verificar a aplicabilidade em um ambiente real de desenvolvimento de *software*.

A avaliação da proposta de abordagem de desenvolvimento de *software* não isenta esta proposta de limitações. Por esta razão estamos submetendo a avaliação apenas a um grupo de profissionais e a um determinado nicho de projetos. Para uma consolidação desta abordagem vemos que é necessário uma avaliação e execução para um número maior de projetos até mesmo para adaptação desta abordagem a mecanismos de contratação de projetos *software*, pois é com um contrato comercial que é firmada a negociação de execução de um projeto de *software*.

3.4.1 Questionário

Questionário é um método de coletar dados no campo, de interagir com o campo, composto por uma série ordenada de questões a respeito de variáveis e situações que o pesquisador deseja investigar. Tais questões são apresentadas a um respondente, por escrito, para que ele responda também dessa forma, independentemente de ser a apresentação e a resposta em papel ou computador. A escolha do meio é sempre do pesquisador (Vergara, 2013).

As questões foram direcionadas para obtenção dos perfis e experiência desses profissionais no mercado para, assim, traçar um perfil de formação, experiência e conhecimento relevantes para a proposta do questionário, que é avaliar a viabilidade da abordagem de desenvolvimento proposta.

Para ter conhecimento dos perfis dos profissionais foram feitas perguntas referentes a formação acadêmica, tempo de experiência com desenvolvimento de software, cargo atual, experiência com *softwares* estimados por pontos de função, conhecimento de práticas ágeis e modelos de ciclos de desenvolvimento de *software*.

Após a obtenção dos perfis, foi feita uma explicação e questionamentos sobre a viabilidade da abordagem de desenvolvimento de *software* apresentada.

O questionário foi disponibilizado por meio do *Google Forms* e não foi solicitada nenhuma identificação. Junto com o questionário foi disponibilizado um termo de consentimento, com o qual os participantes tiveram que concordar para responder ao questionário.

Com relação ao entendimento das questões apresentadas, o questionário foi apresentado primeiramente para 2 pessoas e após, verificado que as perguntas estavam descritas de forma que ambas as pessoas conseguiram compreender sobre os questionamentos. Em seguida, o questionário foi encaminhado por e-mail para os respondentes.

4 RESULTADOS

Nesta seção, apresentamos os resultados referentes ao Mapeamento Sistemático, ao Estudo de Caso e, com relação a abordagem HiPA apresentamos resultados referentes ao questionário e a avaliação parcial de projetos executados nesta abordagem de desenvolvimento.

4.1 RESULTADOS DO MAPEAMENTO SISTEMÁTICO

A Tabela 4 mostra a quantidade de artigos encontrados em cada repositório, no período definido e com a *string* de busca (Seção "3.1.5 *String* de Busca") elaborada para o mapeamento apresentado neste trabalho. Nesta etapa foram encontrados 182 artigos.

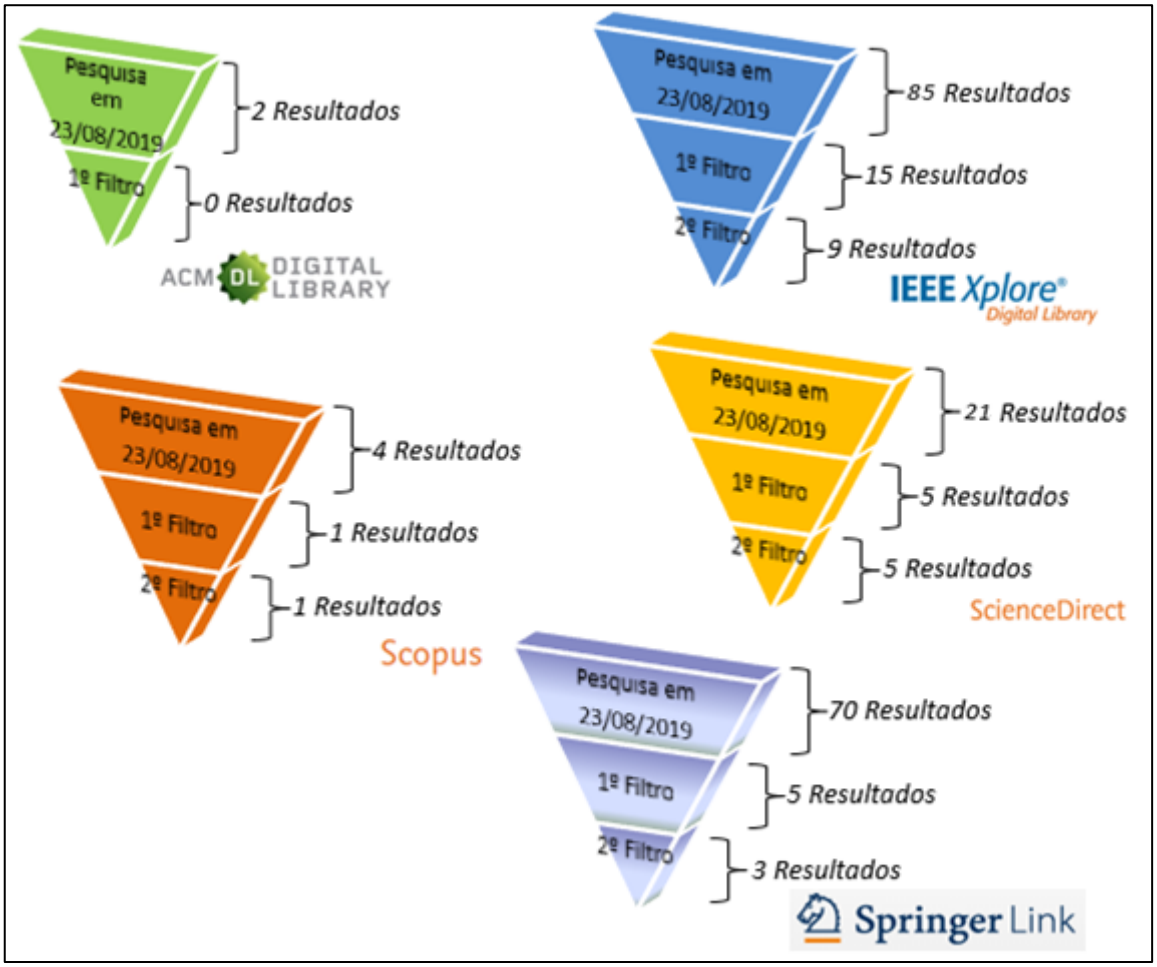
Tabela 4. Resultado do Mapeamento Sistemático.

Base de dados	Data da pesquisa	Número de artigos
ACM Digital Library	23/08/2019	2
IEEE Xplore Digital Library	23/08/2019	85
Scopus	23/08/2019	4
Science Direct	23/08/2019	21
Springer Link	23/08/2019	70

Fonte Autoria própria.

Após a obtenção dos artigos e aplicação dos critérios de inclusão e exclusão foram selecionados 18 artigos. Os filtros mostrados na Figura 15 estão descritos na Seção "3.1.6 Critérios de Seleção" e no Apêndice B são listados todos os artigos obtidos.

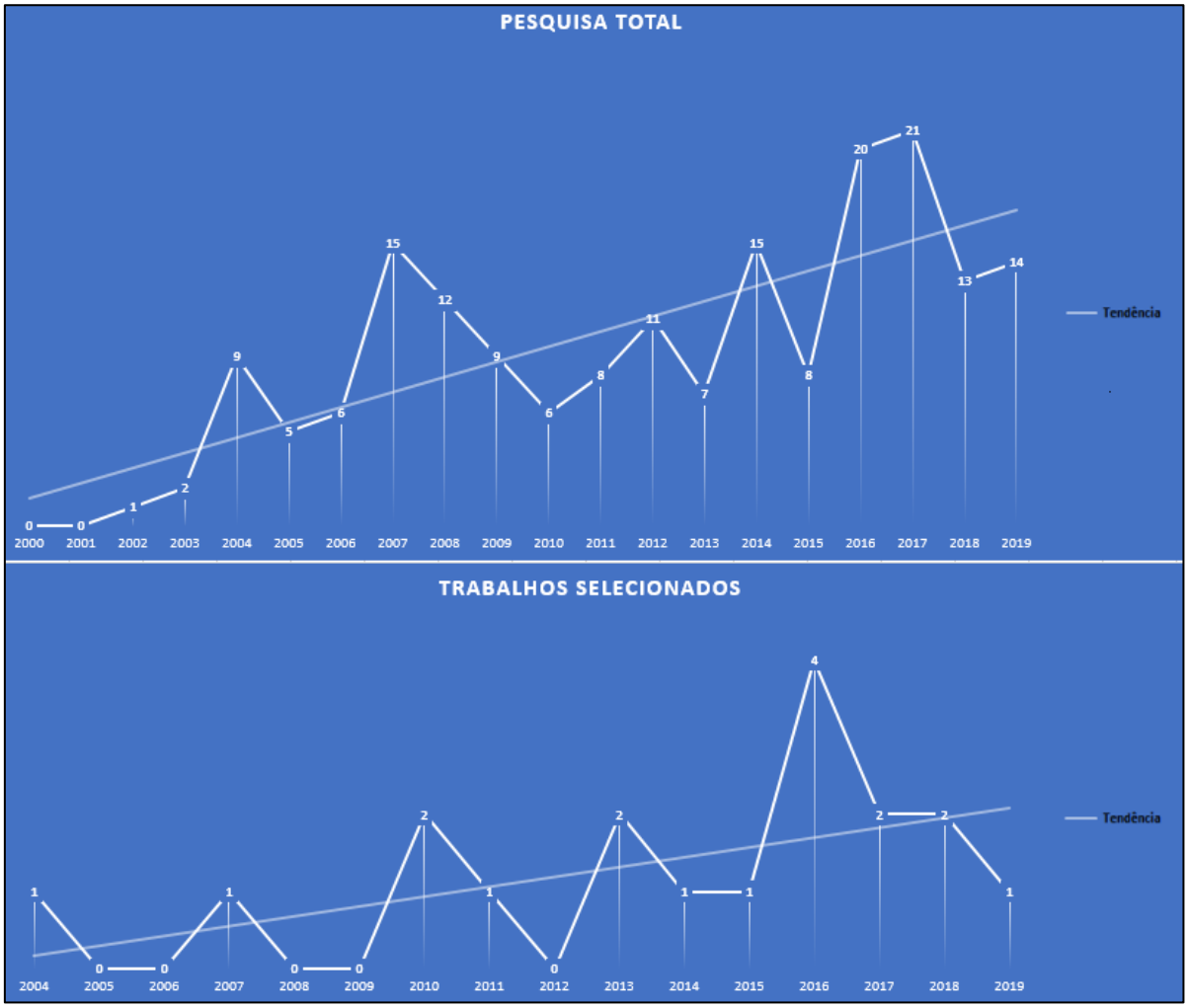
Figura 15: Resultado do Mapeamento Sistemático.



Fonte: Autoria própria.

A Figura 16 apresenta os gráficos com os resultados da pesquisa, sendo a data final da pesquisa indicada na Tabela 4. O primeiro gráfico (Pesquisa Total) representa a distribuição temporal do resultado de busca em todos os repositórios para a *string* de busca definida e, no segundo gráfico (Trabalhos selecionados) são indicados os 18 trabalhos resultantes, após aplicação dos critérios de seleção, mostrando a distribuição temporal dos trabalhos, sendo a linha de tendência deste gráfico com um indicativo leve de crescimento do número de trabalhos da literatura associados ao tema desta pesquisa. As temáticas centrais dos trabalhos selecionados são mostradas a seguir para os 18 trabalhos indicados no Apêndice A, conforme as questões a serem respondidas neste mapeamento.

Figura 16: Distribuição temporal de trabalhos.



Fonte: Autoria própria.

- **Q1**: *Pontos de função são utilizados em projetos que utilizam práticas ágeis de desenvolvimento?*

Existem algumas indicações de que a medição funcional, além de importante para o dimensionamento de *software*, já está sendo direcionada também para uma abordagem de desenvolvimento ágil. Não temos conclusões que nos digam como pontos de função são utilizados com práticas ágeis e quais eventuais problemas e benefícios da utilização de pontos de função em projetos com práticas ágeis.

Com relação aos métodos e medições, segundo Basri et al. (2016), o desenvolvimento de *software* adota 2 tipos de métodos, tradicional e ágeis. Nenhum dos modelos de estimativa de esforço de mudança existentes é comprovado para se adequar ao método tradicional e métodos ágeis. Assim, é proposto um modelo de estimativa de esforço de mudanças que seja aplicável nos métodos tradicionais e ágeis, indicando que em trabalhos futuros pretendem estender o modelo para incluir pontos de função como parte das métricas de tamanho de

software suportadas no modelo proposto.

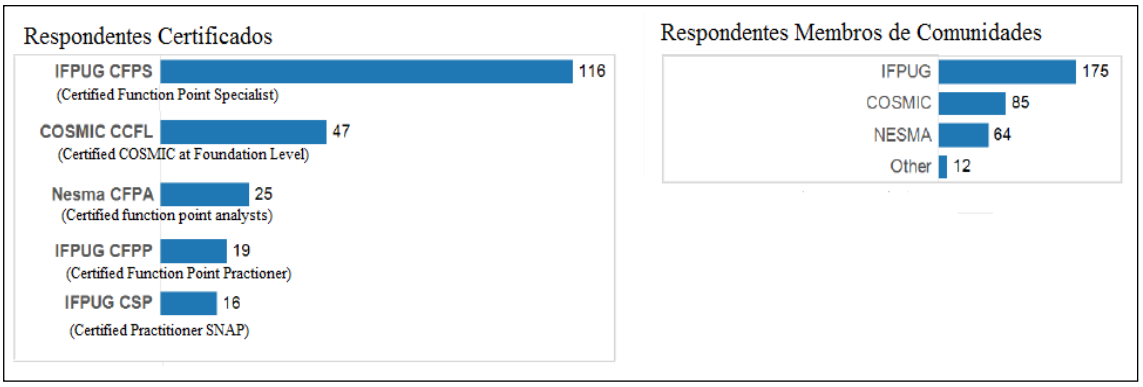
Com relação à medição de tamanho de *software*, para Wilkie et al. (2011) as técnicas para estimar o tamanho de um sistema de *software* tem sido o assunto de muita pesquisa por muitos anos e a adoção de práticas de estimativa baseadas em modelos permite o estabelecimento de métricas de tamanho de *software*. Isto facilita o refinamento das práticas subsequentes de estimativa e pode fornecer uma contribuição mais confiável para o processo de cotação. À medida que os dados de dimensionamento se acumulam a partir de projetos adicionais, a relação entre tamanho e esforço pode ser definida de maneira mais confiável. Isso permite que o esforço e o custo sejam derivados para projetos subsequentes quando o tamanho do sistema for estimado. Assim, estimativas de tamanho podem ajudar de duas formas principais. Primeiro, podem ajudar apoiar as decisões sobre os custos do projeto, fornecendo um referencial para comparação. Em segundo lugar, são úteis para o gerenciamento e acompanhamento do projeto em evolução até o detalhamento em estágios de especificação funcional.

No estudo exploratório Huijgens et al. (2016), feito com 336 especialistas em medições de *software*, a suposição de que os métodos ágeis tendem a dificultar a medição de tamanho funcional não é confirmada pela maioria dos entrevistados. Esta medição de tamanho funcional de um sistema é uma ferramenta importante para a tomada de decisão (Tabela 5). Dos 336 especialistas, os perfis são indicados na Figura 17, sendo que nem todos responderam todas as questões.

Tabela 5. Questões para especialistas FSM.

Questão	Número de respondentes	Percentual de concordância
Medição de tamanho funcional é uma ferramenta importante para tomadores de decisão em projetos de *software*.	245	87%
Abordagens ágeis de desenvolvimento de *software* dificultam a preparação de medições de tamanho funcional boas e confiáveis.	245	22%

Figura 17: Respondentes da pesquisa.



Fonte: Huijens et al. (2016).

Em Tengshe et al. (2007) pontos de função são usados para se ter o tamanho e o custo para priorização do *backlog* do projeto. Isso é feito antes da equipe iniciar o desenvolvimento do projeto e fornece um melhor controle sobre a estimativa inicial e também sobre o tamanho do escopo entregue.

O conceito de gerenciamento de valor agregado (EVM: *Earned Value Management*) apareceu nos artigos encontrados e ressaltamos aqui pois pode ser utilizado para administração de custos e tempo no desenvolvimento de projetos de *software* estimados por pontos de função. Segundo Efe et al. (2013), a análise de EVM não é muito difundida em projetos de *software*, mas neste artigo são indicados os benefícios do uso do EVM em projetos de *software*. No artigo Torrecilla-Salinas et al. (2014) também é relacionado o conceito de EVM, mas neste caso referenciando alguns trabalhos relacionando EVM ao *Scrum* e direcionando o cálculo de EVM a nível de *sprints*. Com relação aos benefícios do EVM, tem-se o acompanhamento das variações de custo e tempo de um projeto, possibilitando ações corretivas dos projetos.

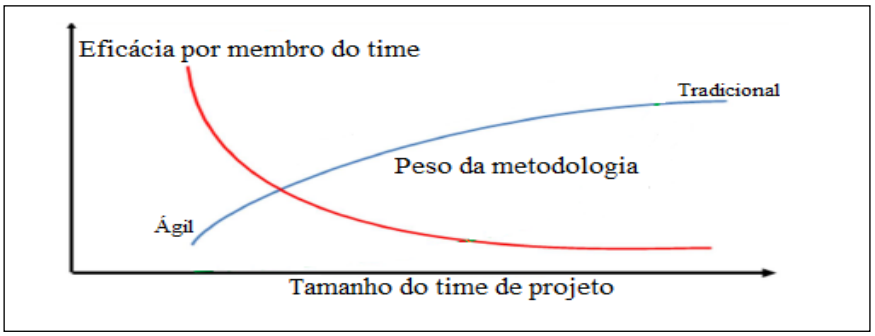
- **Q2**: *Existe uma combinação do ciclo de desenvolvimento cascata e de práticas ágeis sendo utilizada no desenvolvimento de software?*

Existe uma tendência para que métodos e práticas diversas de desenvolvimento de *software* sejam utilizadas em conjunto para uma melhor obtenção de resultados.

A mudança em um método de desenvolvimento de *software* pode refletir em diversas áreas, não somente em uma área executora do projeto de *software*. Em Tengshe et al. (2007) e Olszewska et al. (2015), são feitas indicações da mudança de um PMO ("*Project Management Office*" ou escritório de gerenciamento de projetos) tradicional em um PMO para apoiar diferentes equipes de portfólio de projeto ágeis. Os serviços incluíam a realização de treinamento, o início de novas equipes ágeis, o incentivo ao empoderamento de equipes e a transformação de funções, artefatos e processos existentes para se tornar mais eficiente, a captura de métricas entre equipes e a criação de relatórios de gerenciamento. Estes conceitos estão ligados à transformação entre modelos de desenvolvimento que envolve os perfis de trabalho em um projeto de *software*.

Entre projetos de *software* de métodos ágeis e tradicionais, Keshta et al. (2017) faz algumas comparações e o resultado do artigo ilustra que cada método tem uma área específica na qual se encaixa melhor. As organizações devem considerar todos os fatores e escolher o método de acordo com a situação. A indicação (Figura 18) é que métodos ágeis se encaixam melhor com equipes pequenas. Os métodos tradicionais se adaptam melhor com equipes grandes, para projetos de artefatos previsíveis e reutilizáveis. No entanto, eles podem coexistir.

Figura 18: Tamanho do time de projeto.



Fonte: Keshta et al. (2017).

Aqui pode-se citar que já existe um trabalho de transformação entre modelos de desenvolvimento e que métodos ágeis e clássicos podem coexistir gerando assim um conceito para uma abordagem híbrida de desenvolvimento de *software*. Isto embasa o crescimento da utilização de métodos ágeis, apesar dos métodos tradicionais ainda serem populares, sendo que as organizações também utilizam múltiplos métodos em projetos (VIJAYASARATHY; BUTLER, 2016).

Conforme Garousi et al. (2015) o modelo cascata é mais popular em empresas maiores, enquanto os profissionais empregados por empresas menores favorecem mais os métodos ágeis. Para Chari et al. (2018) os métodos ágeis são mais úteis em ambientes com alta volatilidade de requisitos e para mitigar os riscos do projeto. Os métodos ágeis que usam abordagens de desenvolvimento iterativo estão se tornando cada vez mais populares.

Em Raju et al. (2013) são indicados conceitos dos modelos cascata e métodos ágeis e também são feitas indicações sobre mudanças do modelo cascata para métodos ágeis, sendo que o modelo tradicional cascata é o paradigma dominante de gerenciamento de projetos e os métodos ágeis é a resposta direta ao modelo em cascata.

No documento intitulado “Desenvolvimento tradicional vs Ágil uma comparação usando a teoria do caos”, Shawky et al. (2014) apresenta uma comparação entre métodos ágeis e o modelo cascata usando a teoria do caos. Os métodos ágeis surgiram para superar as desvantagens dos métodos tradicionais de desenvolvimento. No entanto, há um debate em andamento sobre os pontos fortes e fracos desses métodos em comparação com os tradicionais. As abordagens tradicionais de desenvolvimento de *software* em cascata geralmente são consideradas incapazes de lidar com a complexidade do desenvolvimento.

Como resultado, os métodos e práticas ágeis surgiram como uma tentativa explícita de adotar formalmente taxas mais altas de mudança de requisitos, também indicado em Garousi et al. (2015) sendo um dos objetivos dos projetos que usam práticas ágeis é manter um processo, que permite a adaptação às mudanças que fornecem aos clientes e usuários com vantagens competitivas.

Também em Shawky et al. (2014) têm-se que métodos ágeis são usados em sistemas mais caóticos, com elementos variáveis e em constante mudança, executadas mais rapidamente do que aqueles desenvolvidos usando o método em cascata. No modelo cascata, os desenvolvedores têm que esperar até que o todo documento de requisitos tenha sido aprovado, e se o ambiente é caracterizado como altamente dinâmico, métodos ágeis devem ser escolhidos. O modelo de processo não é sempre escolhido do zero, e existem modelos estabelecidos em

empresas que precisam de mais melhorias (PETERSEN; WOHLIN, 2009).

Em Pries-Heje et al. (2010) é mencionado que os princípios ágeis consideram o trabalho em equipe e enfatizam a troca de conhecimento informal, colaboração e experiência como elementos importantes. Esses princípios são essenciais para o gerenciamento de projetos de *software* em ambientes voláteis, nos quais tecnologias e mercados em rápida mudança geram habilidades e conhecimentos que mudam rapidamente e que alguns ambientes de desenvolvimento de sistema de informação já adotaram diferente práticas ágeis associadas ao desenvolvimento tradicional.

Para Chari et al. (2018), algumas variações da abordagem cascata com fases distintas e especificações de requisitos de linha de base bem documentadas, continuam a ser amplamente utilizadas. Em Molokken et al. (2004), a proposta é verificar se o modelo de desenvolvimento afeta a precisão da estimativa. As descobertas não ficam evidentes, e há uma observação de que não houve revisões de estimativas adicionais em projetos incrementais e evolutivos. Outra afirmação não confirmada na pesquisa foi a de que os usos diferenciais de modelos de desenvolvimento incremental e modelos de desenvolvimento em cascata levam a diferenças na funcionalidade entregue nos projetos.

Em Chauhan et al. (2017) são comparados projetos executados com métodos ágeis com projetos executados no modelo cascata. Neste comparativo os esforços dos projetos foram indicados em pontos de função. E, como resultado, é informado que os projetos executados com métodos ágeis são executados com menos esforço e com mais sucesso. Mas aqui também não é indicado nenhum modelo híbrido de desenvolvimento, contudo os pontos de função são usados como comparativo de esforço.

Assim, os documentos selecionados neste mapeamento sistemático, possuem várias abordagens, mas não mostram um direcionamento específico, informando que pontos de função são utilizados em projetos de *software* que utilizam práticas ágeis (***QP1***). Com relação a ***QP2*** existem evidências de que já existe uma tendência de combinação entre os métodos tradicionais e ágeis, mas a combinação principal da utilização de pontos de função em projetos híbridos não fica devidamente esclarecida ao responder ambas as questões.

Finalizando a exploração das questões de pesquisa, temos o trabalho (Russo et al. 2018), que explora a lacuna da utilização de pontos de função com métodos ágeis, informando que a preocupação surge quando se trata do contrato entre um consumidor de *software* e um produtor de *software*. Como contratar a "produção ágil" de *software* para organizações, sendo que é indicado que, na literatura, pouco foi feito, do ponto de vista de formalização de contratos. Nessas situações, as empresas de *software* lutam pela minimização do esforço, enquanto o cliente normalmente espera artefatos de alta qualidade.

Conforme Russo et al. (2018), os contratos no modelo cascata são bem conhecidos, em um sentido em que são fáceis de concordar inicialmente, porque as partes geralmente estão plenamente conscientes e há um histórico de uso. Contratos de *software* tradicionais são muito caros, juntamente com os altos custos de especificação devido a requisitos muito detalhados. Também há a dificuldade de avaliar com objetividade o artefato a ser construído. Tais barreiras têm um impacto direto no custo do contrato e na eficiência do mercado.

Ainda de acordo com Russo et al. (2018), embora existam rotinas estabelecidas sobre como executar um contrato no modelo cascata, existem muito poucas diretrizes sobre os métodos ágeis e a Análise de Pontos de Função (APF) fornece objetividade suficiente no processo de avaliação, independentemente da tecnologia utilizada. Esta é a razão pela qual a

APF pode ser a opção adequada para garantir a flexibilidade adequada dos métodos ágeis. Outro ponto forte a favor dos pontos de função é que eles são conhecidos e já utilizados. Isso significa que seria bastante eficaz escrever um contrato com base na experiência já adquirida.

Em Hariharn et al. (2019), o *framework scrum* é aplicado em projetos de escalabilidade estimados por alguns critérios como por exemplo pontos de função, com base em parâmetros de estimativa de esforço semelhantes ao modelo cascata. Contudo, não é feito nenhum aprofundamento da utilização de pontos de função com métodos ou práticas ágeis.

4.1.1 Discussão - Mapeamento Sistemático

As estimativas de tamanho são consideradas importantes para o desenvolvimento de projetos de *software*. No entanto, o resultado encontrado neste mapeamento sistemático, apesar de prover um direcionamento, não estabelece uma conclusão fortemente embasada em relação ao uso de estimativa por pontos de função em softwares desenvolvidos com métodos ágeis, práticas ágeis ou abordagens híbridas, indicando assim a necessidade de estudos futuros que permitam uma conclusão mais consolidada.

Modelos híbridos são utilizados em ambientes de desenvolvimento de *software* e, segundo Rahmanian (2014), a combinação de práticas ágeis com o modelo tradicional que forma os modelos híbridos de gerenciar os projetos de TI está apenas emergindo.

Conforme Rahmanian (2014), o gerenciamento puro de projetos tradicionais não é eficaz, e os modelos híbridos são a solução mais apropriada. O método tradicional de gerenciamento de projetos é melhor para os projetos com um objetivo e uma solução claros. Por outro lado, os projetos que não têm objetivos e soluções claras são melhores gerenciadas com métodos ágeis, mas um método ágil puro não é bom o suficiente, sendo ajustado de forma flexível, resultando em um modelo híbrido.

Com relação a abordagens de desenvolvimento de software, em Lavazza et al. (2008) todos os ciclos de desenvolvimento são complementares e não competitivos. Existem três grandes categorias. Modelos tradicionais, métodos ágeis e modelos híbridos. A combinação de modelos tradicionais e métodos ágeis pode se beneficiar da vantagem de ambos os modelos, sendo esta combinação chamada de modelo híbrido.

Conforme Lavazza et al. (2008), a principal armadilha dos modelos tradicionais é a menor comunicação, que é a maior vantagem do modelo ágil. No modelo híbrido, no estágio inicial, os modelos tradicionais são usados para fins de documentação e análise; os métodos ágeis são usados para fazer a codificação e o teste, considerando a comunicação frequente com os clientes, a análise contínua e a liberação periódica do produto.

Existem alguns desafios ainda não evidenciados para trabalhar com pontos de função em projetos ágeis ou híbridos, sendo o modelo de contrato comercial uma lacuna indicada. As questões de pesquisa, que abordam a utilização de pontos de função em projetos híbridos, quando relacionadas com a literatura, evidenciam a questão da relação contratual, que conforme Russo et al. (2018), indicam poucas diretrizes de contratos para o modelo com métodos ágeis. Também temos obstáculos para introduzir métodos ou práticas ágeis em um ambiente de trabalho.

Para que características dos métodos ágeis sejam usadas ou adaptadas em projetos, é

necessário que a estrutura organizacional incentive, apoie e ofereça uma estrutura na qual seja viável utilizar práticas ágeis no desenvolvimento de software. Não basta apenas existirem conceitos e uma maneira de realizá-los, isto deve ser apoiado pelo ambiente de desenvolvimento de projeto de software.

O apoio da alta administração, a estrutura organizacional e os desafios culturais são questões importantes que as organizações enfrentam. Isso implica que a alta administração não apenas aceite o conceito de métodos ágeis, mas compromete-se com o papel para desempenhar e sustentar métodos ágeis em suas organizações (SENAPATHI; DRURY-GROGAN, 2017).

4.2 RESULTADOS DO ESTUDO DE CASO

Neste tópico apresentamos as variáveis obtidas em todos os projetos analisados, e a caracterização de trabalhos feitos durante a execução destes projetos para, por fim, analisar os resultados conforme os tipos de ciclo de desenvolvimento dos 16 projetos.

4.2.1 Caracterização dos projetos

Para todos os projetos encontramos a existência das seguintes tarefas:

a) Abertura do projeto: Neste momento é feita a abertura do projeto já alinhando a necessidade de prazo do cliente estabelecendo os canais de comunicação que serão utilizados por todos os componentes da equipe para comunicações do projeto. Participantes: Gerente de Projetos e analistas de ambas as empresas;

b) Planejamento do projeto: Aqui é indicada toda a infraestrutura necessária para o desenvolvimento e teste do sistema. Toda a documentação criada para o projeto deverá ser aprovada pelo cliente, sendo que a documentação pode ser considerada um entregável do projeto, caso definida em contrato comercial. Participantes: Analistas funcionais e de sistemas de ambas as empresas. Gerente de Projeto é responsável pelo acompanhamento e obtenção do aceite de fase;

c) Desenvolvimento do projeto: Nesta etapa, toda a escrita do código do sistema é feita. Podem-se ter vários entregáveis, sendo que isto é detalhado no momento na abertura de projeto com o cliente. O canal de comunicação estabelecido é utilizado para situações de dúvidas e problemas. Semanalmente existe uma reunião de status do projeto com o cliente, na qual é indicado o percentual de conclusão do projeto e eventuais situações pertinentes. O gerente de projeto tem o papel de obter o aceite dos entregáveis de fase ou projeto. A equipe de desenvolvimento atua internamente e com a equipe do cliente.

d) Homologação do projeto ou fase: Neste momento uma empresa terceirizada é contratada pelo cliente ou uma equipe local do cliente é responsável pela homologação do projeto. Devem ser indicados defeitos somente dentro do planejamento feito. Defeitos relevantes fora do planejamento não são indicados

como pertinentes ao projeto, mas podem gerar outra (s) demanda (s) conforme necessidade do cliente.

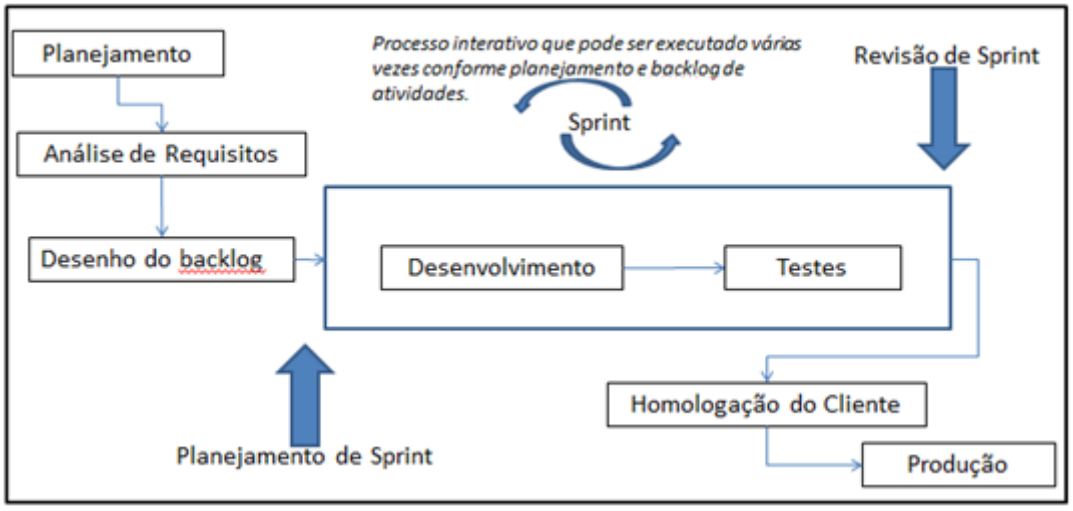
e) Entrega para Produção: Acompanhamento da disponibilização do projeto em ambiente final.

Os projetos do tipo cascata seguiram uma organização sequencial, não havendo interações diárias na equipe feitas de maneira periódica não possibilitando um acompanhamento mais próximo para indicar eventuais distorções de escopo.

Os projetos aqui denominados como híbridos tem o modelo de execução mostrado na Figura 19, no qual é priorizado o *backlog* do projeto para que sejam feitas várias entregas ao cliente, caso sejam solicitadas estas entregas durante o planejamento.

Este modelo no qual podem existir vários *sprints* deve ser adequado e planejado para que não ultrapasse em termos de horas a indicação de tempo feita por meio de pontos de função. Como este modelo é adequado à existência de interações e a uma revisão periódica de escopo, quaisquer divergências de escopo podem gerar uma adequação de projeto e isto indica uma nova tratativa comercial.

Figura 19: Fluxo de execução com práticas ágeis.



Fonte: Autoria própria.

Nos projetos híbridos foram utilizadas algumas práticas ágeis como o estabelecimento e priorização de um *backlog* de atividades, planejamento e revisões de *sprint* e também foi utilizada a ferramenta Jira, na qual foi feito o planejamento dos *sprints* e o acompanhamento de atividades com *Kanban*, sendo a equipe de projeto responsável pela atualização dos estados das atividades nas reuniões periódicas do projeto.

A Figura 27 mostra o ciclo de desenvolvimento dos projetos de modelo híbrido e neste ciclo são indicadas práticas ágeis utilizadas na execução do projeto, sendo utilizados dentro do fluxo de execução, conceitos relacionados à *sprint* e seu planejamento, reuniões periódicas para alinhamento e revisões de objetivos atingidos ou não.

4.2.2 Indicadores dos projetos

Os indicadores mostrados são o IDC (Índice de desempenho de Custo) e IDC (Índice de desempenho de Prazo). Estas informações qualificam o projeto em termos de Custo e Tempo, informando se os andamentos dos projetos estão dentro do esperado para estas duas dimensões, sendo números maiores ou igual a 1, satisfatórios. Projetos que tiveram retornos financeiros satisfatórios terão valor maior ou igual a 1 e projetos feitos dentro do prazo estabelecido também terão valor maior ou igual a 1.

Assim, para realizar o cálculo do IDC é necessário ter o valor do projeto e o custo real do projeto. Do mesmo modo para o cálculo do IDP é necessário ter a duração planejada e a duração realizada dos projetos.

Na Tabela 6 temos o resultado das variáveis obtidas durante a etapa de condução. Para cada projeto identificamos seu tipo (Cascata ou Híbrido), o total de horas do projeto, a totalização de pontos de função (APF e SNAP), a produtividade considerada para o projeto, que é utilizada para o cálculo dos pontos de função e as datas de início e término do projeto. Na Tabela 7 temos o complemento das variáveis nas quais mostramos os índices de desempenho de custo (IDC) e índices de desempenho de prazo (IDP) para os projetos.

Tabela 6. Projetos e valores das variáveis.

Projeto	Tipo	Total de Horas	APF	SNAP	Produtividade	Início	Término
P1	Cascata	185,40	15,45		12,00	06/12/2016	30/12/2016
P2	Cascata	272,40	22,7		12,00	03/10/2016	16/11/2016
P5	Híbrido	3.198,00	42,5	224	12,00	18/01/2017	19/04/2017
P6	Cascata	1.081,60	3	42	24,04	02/09/2016	07/02/2017
P7	Cascata	608,00	38		16,00	07/12/2016	14/03/2017
P8	Cascata	2.220,00	185		12,00	02/06/2014	16/01/2015
P10	Cascata	1.120,00	112		10,00	27/10/2014	24/04/2015
P11	Cascata	630,00	63		10,00	15/10/2014	20/03/2015
P12	Cascata	1.110,00	111		10,00	18/11/2014	19/05/2015
P13	Cascata	715,00	56		12,77	06/01/2015	10/04/2015
P17	Cascata	792,00	66		12,00	20/05/2015	22/09/2015
P20	Cascata	2.178,00	198		11,00	05/08/2015	26/01/2016
P25	Cascata	790,00	79		10,00	15/02/2016	22/06/2016
P26	Híbrido	3.820,02	297	14	12,28	01/04/2016	23/10/2016
P27	Híbrido	1.704,00	124	18	12,00	31/05/2016	08/09/2016
P29	Híbrido	3.942,00	185	95,1	14,07	22/05/2017	28/08/2017

Fonte: autoria própria.

Tabela 7. Projetos e índices de desempenho.

Projeto	Tipo	IDC	IDP
P1	Cascata	1,23	1
P2	Cascata	0,84	1
P5	Híbrido	3,31	1
P6	Cascata	2,15	1
P7	Cascata	1,41	1
P8	Cascata	1,72	1
P10	Cascata	1,92	0,95
P11	Cascata	1,59	1
P12	Cascata	2,19	1
P13	Cascata	1,92	1
P17	Cascata	2,46	1
P20	Cascata	1,34	1
P25	Cascata	1,28	1
P26	Híbrido	1,31	1
P27	Híbrido	2,66	1
P29	Híbrido	1	1

Fonte: Autoria própria.

Na Tabela 8 temos uma sumarização de horas dos projetos cascata e híbridos. Apesar do total de projetos híbridos serem apenas 25% do total, a quantidade de horas destes projetos representa uma fatia de 51,98% do total de horas dos projetos selecionados no estudo.

Tabela 8. Totalização de horas nos projetos.

Total de Projetos	**16 (100%)**
Total de Projetos (Cascata)	12 (75%)
Total de Projetos (Híbrido)	4 (25%)
Total de Horas	**24.366,42 (100%)**
Total de Horas (Cascata)	11.702,40 (48,02%)
Total de Horas (Híbrido)	12.664,02 (51,98%)

Fonte: Autoria própria.

A Tabela 9 apresenta a totalização de pontos de função. Percentualmente os dados das Tabelas 8 e 9 teriam que representar o mesmo valor. Mas a variação indica que a produtividade (quantidade de horas utilizadas para se fazer 1 ponto de função) é diferente.

Tabela 9. Tipos de projetos e métrica.

Tipo de Projetos	PF e SNAP
Projetos Cascata	991,95 (49,81%)
Projetos Híbridos	999,60 (50,19)
Total	1.991,55 (100%)

Fonte: Autoria própria.

A produtividade é uma variável da métrica de pontos de função, sendo que representa a quantidade de horas necessária para a conclusão de 1 ponto de função e, o valor é estabelecido no início do projeto durante a medição do sistema. Este valor de produtividade pode ser alterado devido a históricos de projetos anteriores e não consta neste trabalho a análise de alterações.

Tabela 10. Tipos de projetos e produtividade.

Tipo de Projetos	Produtividade Média
Projetos Cascata	12,65
Projetos Híbridos	12,59

Fonte: Autoria própria.

Na Tabela 10 temos as produtividades para os projetos. As informações indicam que não houve variação significativa.

Vale ressaltar que o projeto 6 (P6), indicado na Tabela 6, tem um valor alto de produtividade. Esta produtividade foi justificada em momento de estabelecimento do contrato de projeto: “Foi utilizada neste projeto a produtividade de: 24,0355, a revisão da produtividade foi solicitada, pois o projeto apresenta muito mais itens de características técnicas, como alterações nos processos batch (contagens SNAP), do que itens que envolvem funcionalidades (por APF), e neste caso, a contagem padrão realizada não cobre os esforços necessários para a realização deste projeto. ”. Retirando o projeto 6 (P6) desta análise, teríamos os dados da Tabela 11.

Tabela 11. Tipos de projetos e produtividade (Sem projeto P6).

Tipo de Projetos	Produtividade Média
Projetos Cascata	11,61
Projetos Híbridos	12,59

Fonte: Autoria própria.

Assim, verificamos que para a medição dos sistemas analisados não houve nenhuma variação de produtividade que indique alguma característica que venha a causar discrepância. A aplicação do método de estimativa foi realista independentemente do tipo de projeto.

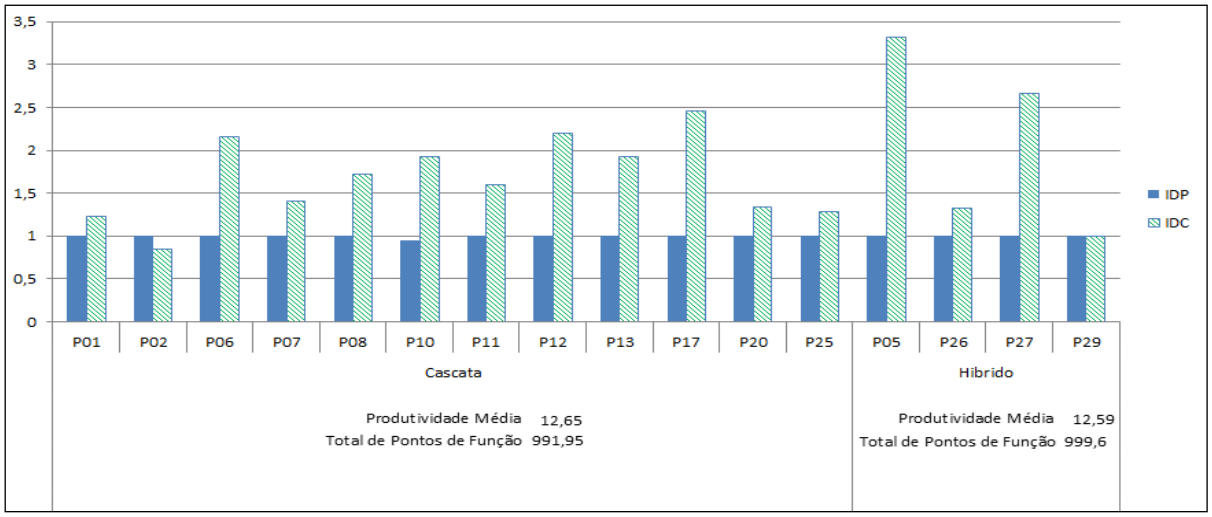
Vale ressaltar que em momento de medição do sistema, não existe uma diferenciação

do modelo do ciclo de desenvolvimento de projeto, pois a técnica de pontos de função não considera os tipos aqui indicados.

Com relação à análise com base nos indicadores de tempo e custo (Figura 20), temos os seguintes índices de desempenho:

- IDP: Índice de desempenho de prazo. Valores abaixo de 1, indicam que o projeto não foi entregue no prazo;
- IDC: Índice de desempenho de custo. Valores igual ou acima de 1, indicam que o projeto teve um desempenho financeiro dentro do esperado. Quanto maior que 1 melhor o retorno de custo.

Figura 20: Índices de desempenho.



Fonte: Autoria própria.

Para o IDP temos:

- Média dos projetos Cascata: 0,995;
- Média dos projetos Híbridos: 1,000.

Para o IDC temos:

- Média dos projetos Cascata: 1,670;
- Média dos projetos Híbridos: 1,820.

Analisando os índices de desempenho de prazo (IDP) e de custo (IDC), verificamos que os projetos híbridos tiveram desempenho melhor, principalmente quanto ao custo de projetos. Percentualmente o IDP ficou 0,5% melhor para projetos híbridos e o IDC ficou em 8,98% melhor para projetos híbridos.

4.2.3 Discussão - Estudo de Caso

As obtenções dos índices de desempenho de prazo e do índice de desempenho de custo permitiram a realização de comparações com relação à abordagem de pontos de função entre os projetos realizados.

O índice de desempenho de custo (IDC) mostra um diferencial positivo para os projetos híbridos em relação aos projetos cascata e o índice de desempenho de prazo (IDP), indica uma conformidade de tempo de execução dos projetos em relação ao valor estimado por pontos de função.

Com relação à abordagem da utilização de pontos de função em projetos que utilizam práticas ágeis, não consideramos mudanças de escopo dentro dos projetos. Para isto seria necessária uma nova contagem para adequação do escopo. Como em projetos ágeis esta troca de escopo pode ser mais frequente, pode-se verificar que não houve mudança de escopo ou mudanças foram absorvidas para uma recontagem fora do contexto de trabalho do projeto corrente.

Tendo como referência o custo, os projetos híbridos tiveram um melhor desempenho, devido à utilização de práticas ágeis. Este indicador embasa inicialmente um incentivo para que projetos futuros sejam executados utilizando práticas ágeis e até mesmo a adoção de todo um *framework* ágil para execução de projetos.

Uma das formas mais eficazes de aprender com a experiência anterior é analisar projetos passados na perspectiva de fatores de sucesso (Lindvall et al., 2002). Assim, para os 16 projetos analisados, 4 utilizaram práticas ágeis em um momento inicial de adoção de um *framework* ágil, sendo estimados por meio de pontos de função e os resultados obtidos são positivos, principalmente com relação ao custo.

Identificamos pontos em comum dos projetos como a abertura e planejamento do projeto, desenvolvimento do projeto seguida da homologação e entrega para ambiente produtivo. As estimativas de pontos de função utilizadas em todos os projetos foram aderentes, pois verificamos com os índices de projeto que houve um retorno positivo ao final.

Toda a comparação entre os projetos foi possível devido ao levantamento das variáveis e análise destas, o que permitiu uma verificação numérica e não tendenciosa dos valores obtidos para estabelecimento desta conclusão. Além disto, os indicativos aqui informados podem ser utilizados para validar o ambiente de desenvolvimento da empresa analisada e as medidas de *software* feitas.

Para termos indicadores mais abrangentes, seria necessário ter um maior número de projetos, mais clientes e avaliar indicativos de qualidade. Isto seria um aprofundamento da análise e consequentemente um passo adiante para implantação deste *framework* ágil.

O modelo adotado para execução dos projetos pode ser conceituado como híbrido, pois utilizou um dimensionamento de pontos de função, com projetos de escopo fechado, não

permitindo interações para dimensionar as atividades, apenas executando atividades de projetos com práticas ágeis utilizadas no *Scrum*, e fazendo um fechamento de projeto após todas as entregas. As técnicas utilizadas permitiram uma maior interação e proximidade com o cliente.

No âmbito deste trabalho, verificamos que a inserção de práticas ágeis em projetos trouxe alguns indicativos de melhorias, mesmo que pequenas. Estes indicativos podem fortalecer o trabalho a ser realizado para a utilização de práticas ágeis em uma maior abrangência nos projetos, bem como indicar conceitos de métodos com características híbridas.

Nestes projetos analisados aqui e caracterizados como projetos híbridos, podemos indicar conceitos, conforme caracterizado como modelo "*Water-Fall-Scrum*" mencionado em Theocharis et al. (2015): "Diferentes abordagens geralmente usadas em combinação, ou seja, as abordagens misturadas ou híbridas representam o modelo comum de uso, e a combinação inclui abordagens 'agilizantes tradicionais' e 'ágeis'. Entre as diferentes combinações, o *Scrum* e o modelo cascata representam a maioria das menções. O *Scrum* tornou-se o método ágil mais popular e os métodos ágeis são adaptados e combinados com outros processos (tradicionais e ágeis). O modelo '*Water-Scrum-Fall'* ou combinações similares se tornaram realidade. O principal motor de desenvolvimento desse tipo é a expectativa de diferentes grupos de partes interessadas em direção à abordagem de desenvolvimento de *software* 'ótima'. Os gerentes de projetos exigem alguma estabilidade para executar as tarefas usuais de gerenciamento de projetos, como estimativa, planejamento ou controle".

Além disso, os gerentes de projetos também têm a responsabilidade de alinhar projetos com a estratégia de empresa respectiva, ou seja, projetos devem interagir com processos adicionais (de negócios), como gerenciamento de recursos humanos, vendas, contratação, e assim por diante. Uma vez que os métodos ágeis geralmente abordam apenas tarefas relacionadas ao sistema ou ao desenvolvimento, essas interfaces estão faltando muitas vezes. Assim, as abordagens tradicionais são usadas para fornecer uma estrutura básica e uma estrutura para a organização do projeto e fornece interfaces para a respectiva empresa (THEOCHARIS et al., 2015).

Esta combinação de técnicas e características indicadas em várias abordagens pode ser explorada conforme o cenário do projeto e a abordagem permitida dentro de uma estrutura organizacional.

5 PROPOSTA DE UMA ABORDAGEM DE DESENVOLVIMENTO DE *SOFTWARE*

Este tópico tem por objetivo propor uma abordagem de desenvolvimento de projetos de *software*, aderente à métrica de estimativa por pontos de função, utilizando características do ciclo de vida cascata com práticas ágeis, e indicar parâmetros de controle do projeto auxiliando assim o gerenciamento.

5.1 EMBASAMENTO DA PROPOSTA

Para propor esta abordagem buscamos definições de conceitos e realizamos um mapeamento sistemático e um estudo de caso. Elaboramos a abordagem de acordo com o processo indicado no estudo de caso, dividindo em etapas.

A abordagem apresentada a seguir nesta dissertação usa o conceito de abordagem híbrida de desenvolvimento que, segundo Kuhrmann et al. (2017), é uma abordagem de desenvolvimento de software que combina abordagens ágeis e tradicionais que uma unidade organizacional adote e personalize, sendo exemplificada em West (2011) e Kuhrmann et al. (2019).

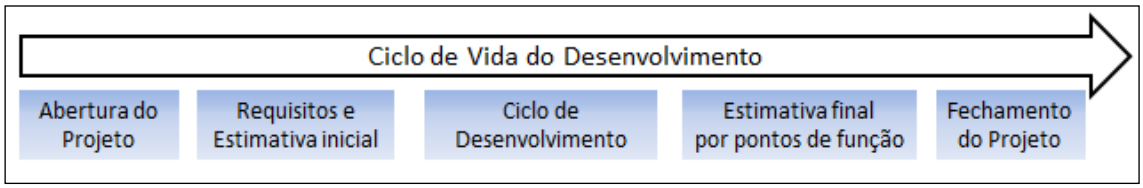
A abordagem proposta visa aumentar a colaboração junto aos clientes com um maior nível de interação entre os envolvidos no projeto, utilizando conceitos do ciclo de vida cascata, práticas ágeis, valor agregado de projeto e pontos de função, sendo estes conceitos citados no referencial teórico deste trabalho. Relações comerciais aqui não serão indicadas, mas é um aspecto relevante no trabalho de desenvolvimento de *software*, principalmente devido à utilização de pontos de função que é uma estimativa comercial para construir o custo do projeto.

Estes procedimentos serviram para a definição da proposta de uma abordagem de desenvolvimento que, por meio da estimativa de pontos de função e a utilização de práticas ágeis, permite a execução de um projeto de *software* e uma avaliação final utilizando indicadores de tempo e custo do projeto.

5.2 ABORDAGEM HiPA

A abordagem proposta está dividida em etapas, conforme a Figura 21, sendo denominada “Abordagem HiPA” ou seja, abordagem Híbrida com Práticas Ágeis.

Figura 21: Etapas da abordagem – HiPA.



Fonte: Autoria própria.

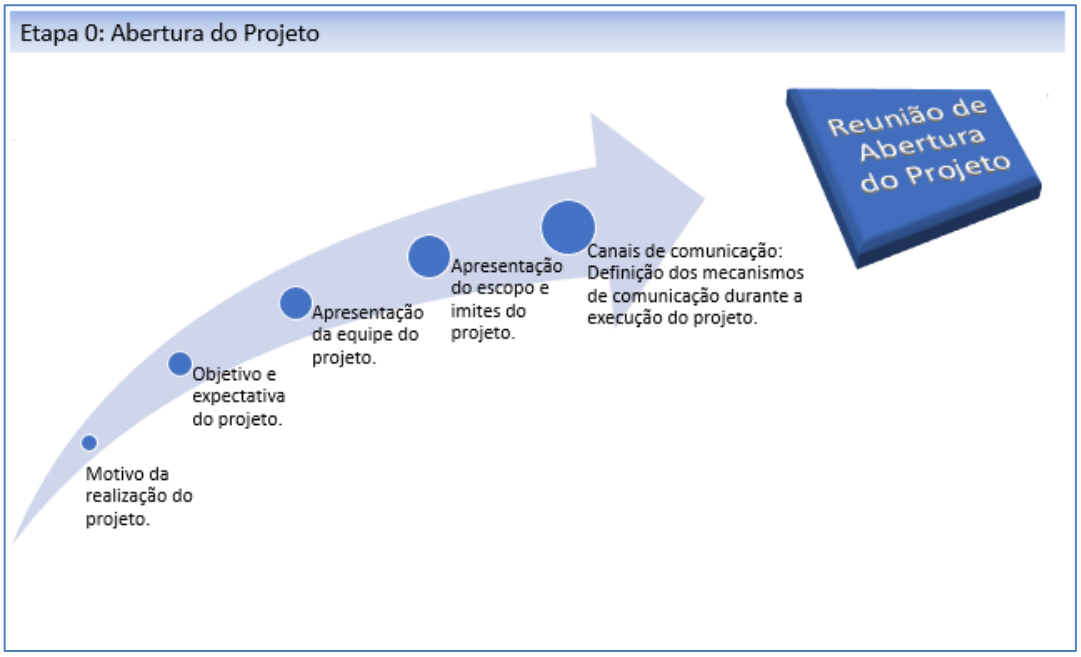
Esta abordagem proporciona uma integração da estimativa de pontos de função com práticas ágeis de desenvolvimento. A integração destes conceitos também permite uma maior resposta a mudanças, que é uma característica dos métodos ágeis, sendo aderente a conceitos de projetos para verificar o retorno gerado pelo projeto. A seguir, explicamos quais as etapas indicadas para a abordagem HiPA de desenvolvimento de *software*.

Etapa 0: Abertura do projeto. (Figura 22)

Etapa inicial para alinhamento de expectativa entre os envolvidos no projeto (*stakeholders*). São apresentadas todas as pessoas envolvidas, seus papeis dentro do cenário de trabalho, os conhecimentos pertinentes ao projeto, o motivo para a realização, os benefícios e resultados esperados do trabalho. O escopo é detalhado e são indicadas eventuais restrições que podem limitar o *software* a ser entregue.

Neste momento também são indicados os canais de comunicação para todos os envolvidos, quem são as pessoas responsáveis por decisões e como devem ser escalonadas estas decisões de projeto, caso necessário.

Figura 22: Abertura do projeto.



Fonte: Autoria própria.

Na etapa 0, Figura 22, temos a abertura do projeto, que é um marco do projeto onde é realizada uma reunião de abertura com todos os envolvidos. Nesta reunião, são mostrados os objetivos do projeto, o motivo da realização do trabalho e apresentada a expectativa de realização do trabalho.

Todo o escopo e limites do projeto são detalhados para que não haja nenhuma dúvida ou percepção diferente que venha a distorcer o objetivo.

Com relação aos canais de comunicação, são referentes à definição para a distribuição de informações às partes interessadas do projeto, no momento oportuno e durante todo o ciclo de desenvolvimento do projeto, mantendo os envolvidos informados sobre o andamento do trabalho. A divulgação do andamento do projeto é importante para manter a expectativa dos envolvidos, e as informações divulgadas devem ser definidas neste momento juntamente com o público que irá receber estas informações e a periodicidade de divulgação.

Segundo Wang e Song (2010), as partes interessadas do projeto referem-se a indivíduos e organizações como aqueles que estão ativamente envolvidos no projeto ou aqueles cujos interesses podem ser positiva ou negativamente afetados pela implementação ou conclusão do projeto, como clientes, patrocinadores, organizações implementadoras e o público. As partes interessadas podem ter influência na entrega do projeto e podem vir de diferentes níveis, dentro e fora da organização. É muito importante para o sucesso do projeto, logo no início, identificar as partes interessadas e analisar seus interesses, expectativas, importância e influência.

Segundo o PMBOK (2013), um plano de comunicação determina as necessidades de informação e comunicação dos envolvidos no projeto. E cabe ao gerente de projetos avaliar como devem ser demonstrados os resultados e situacionais do projeto para influenciar as partes interessadas, de maneira a aumentar o apoio e reduzir eventuais impactos negativos no projeto.

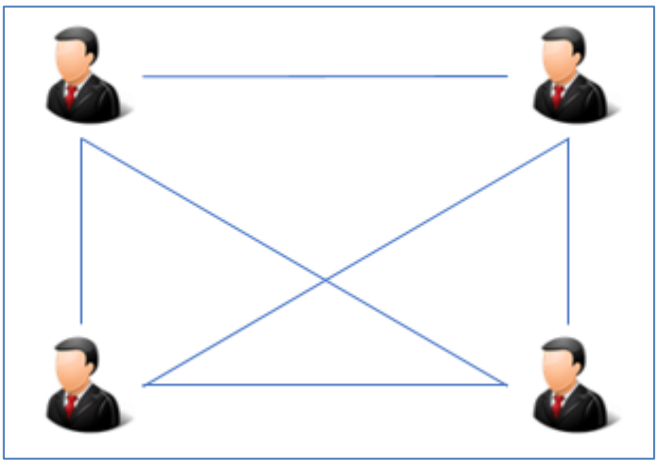
A comunicação eficaz deve fornecer informações no formato correto e no momento certo e trazer o impacto positivo da informação.

Conforme PMBOK (2013), os canais de comunicação são as possibilidades de comunicação que podem ser estabelecidos entre os envolvidos em um projeto. O número total dos canais de comunicação pode ser expresso pela fórmula:

$$Canais\ de\ Comunicação\ = \frac{n(n-1)}{2}$$

Onde *n* representa o número de partes interessadas. Então, conforme a Figura 23, se temos 4 envolvidos em um projeto, temos um total de 6 canais de comunicação.

Figura 23: Canais de Comunicação.



Fonte: Autoria própria

Sendo assim, temos que desenvolver uma abordagem de comunicação apropriada e um plano de comunicações que direcione as informações a todos os envolvidos no projeto e o processo de monitoramento e controle das comunicações deve ser executado durante todo o ciclo de vida do projeto.

Conforme Ofori (2013) o gerenciamento da comunicação tem impacto no sucesso do projeto sendo um dos fatores críticos de sucesso de um projeto, ao lado de "suporte da alta gerência", "clareza de propósito e metas" e "envolvimento das partes interessadas".

É de importância nesta etapa que seja de conhecimento dos envolvidos a abordagem de desenvolvimento *software* utilizada. E principalmente que nesta abordagem a medição do *software* será feita em dois momentos, após a elaboração dos requisitos e após a entrega final do *software*, para que eventuais requisitos não estimados sejam corrigidos na estimativa final, corrigindo principalmente desvios de custos e a produtividade para estimativas futuras de outros projetos.

Etapa 1: Métrica inicial e requisitos. (Figura 24)

Temos aqui um processo sequencial, sendo o escopo do projeto insumo para elaboração dos requisitos, análise da arquitetura do *software* e por fim estabelecimento da

estimativa inicial do tamanho deste *software*, sendo esta estimativa feita por pontos de função. No momento inicial de definição do escopo de um *software* a ser desenvolvido, temos o levantamento dos requisitos que dará origem ao *backlog* do projeto. Logo após este levantamento pode ser feita uma estimativa inicial do sistema.

A contagem inicial por pontos de função é usada para ter a estimativa de tempo a ser utilizada na etapa seguinte, sendo esta estimativa aprovada pelo cliente do projeto. Neste ponto vale salientar que deve existir um profissional qualificado em pontos de função para fazer a estimativa inicial. A saída desta etapa é o *backlog* do projeto, que será utilizado na próxima etapa juntamente com a estimativa de tempo e tamanho do produto.

Os pontos de função são uma métrica que fornece um medidor de tamanho de produtos muito cedo nos ciclos de desenvolvimento (KANG et al., 2010).

A definição da arquitetura do *software* tem por objetivo estabelecer todos os componentes técnicos envolvidos e que irão suportar o *software*, bem como as limitações e fronteiras do *software*. Com referência a esta arquitetura será desenvolvido o produto de *software*.

Caso viável, a prototipação pode ser feita durante a elaboração dos requisitos e fornece ao usuário uma previsão inicial das interfaces a serem elaboradas. Neste caso, temos a prototipação das interfaces fornecendo ao cliente uma percepção inicial do *software* e uma ideia inicial de suas funcionalidades (sem regras de negócio) e navegabilidade. Caso não seja viável a confecção de um protótipo, pois nem todo *software* possui interface, pode-se também indicar o desenho das fronteiras dos sistemas, e neste caso, se necessário, estabelecer e definir eventuais protocolos de comunicação e execução do *software*.

A prototipação tem como principal função validar um conjunto de hipóteses a partir da formulação de questões, criação de protótipos, testes, avaliações e conclusão. E isso proporciona a seleção e refinamentos de ideias, avaliando e validando interativamente. Assim pode-se antecipar a visibilidade sobre eventuais gargalos e problemas, o que tende a contribuir para a redução de riscos e otimização de custos relacionados (MELO; CAROLI, 2017).

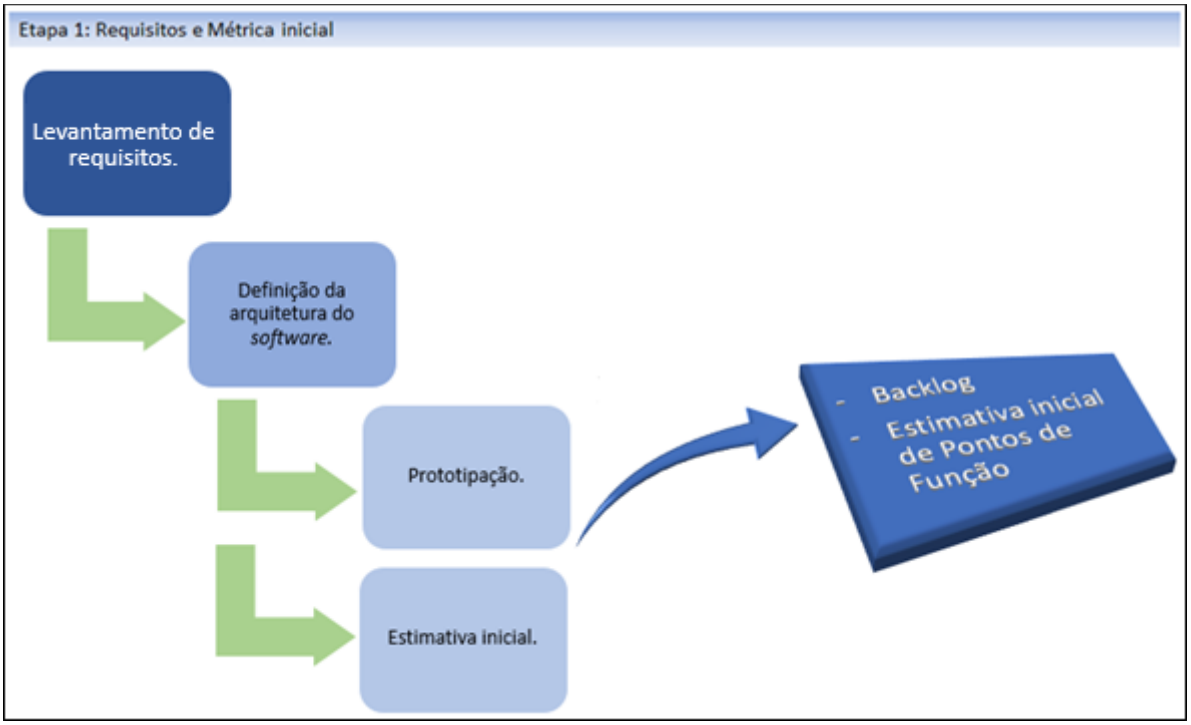
Neste momento de elaboração dos requisitos, é pertinente que também exista um profissional técnico que conheça os limites da arquitetura do *software* para apoiar em decisões técnicas o desenho do sistema.

Na saída desta etapa, juntamente com o *backlog* do projeto e a estimativa inicial em pontos de função, pode-se calcular o valor do projeto, que neste momento é o valor planejado do projeto (Seção 2.4). Este valor planejado do projeto pode ser estabelecido, pois é possível ter o valor do ponto de função, e a quantidade de horas necessárias para concluir um ponto de função. Assim, valores iniciais de custo e tempo podem ser indicados nesta etapa. Nesta etapa, também poderão ser utilizados históricos de projetos anteriores para a estimativa inicial.

Ao contrário das abordagens tradicionais de desenvolvimento de *software*, a mudança é um aspecto fundamental nos métodos ágeis. O dinamismo resultante dos requisitos torna difícil estimar o esforço com precisão. Portanto, uma técnica de estimativa de esforço sistemática, é necessária para apoiar o julgamento especializado e melhorar sua eficácia. No desenvolvimento de *software*, os requisitos normalmente não podem ser completamente especificados antecipadamente e são desenvolvidos conforme o andamento do projeto. Portanto, as estimativas de esforço precisam ser ajustadas para cada *sprint*. Uma estimativa de esforço tem de considerar o impacto de uma alteração em vários artefatos de *software*, e o teste de regressão na unidade e no nível funcional é agora considerado parte da responsabilidade das

equipes de desenvolvimento como parte de um *sprint* (TANVEER, 2016).

Figura 24: Requisitos e Métrica inicial.



Fonte: Autoria própria

Assim, os requisitos do produto são organizados em um *backlog*, ou seja, uma lista dos pequenos pedaços de requisitos. Em seguida, a partir do *backlog*, uma certa quantidade de trabalho é selecionada para uma interação, formando uma porção menor do *backlog*. No *Scrum*, chamamos esses *backlog*s respectivamente de *backlog* do produto e *backlog* da *sprint*.

Etapa 2: Ciclo de desenvolvimento. (Figura 25)

Nesta etapa de trabalho, serão utilizadas algumas práticas e conceitos indicados em métodos ágeis, como:

Sprint: É um ciclo de desenvolvimento com uma duração fixa onde serão implementadas determinadas partes do produto de *software* representadas no *backlog* do projeto.

Backlog: Lista de trabalhos a serem implementadas. Corresponde aos requisitos do projeto de *software*.

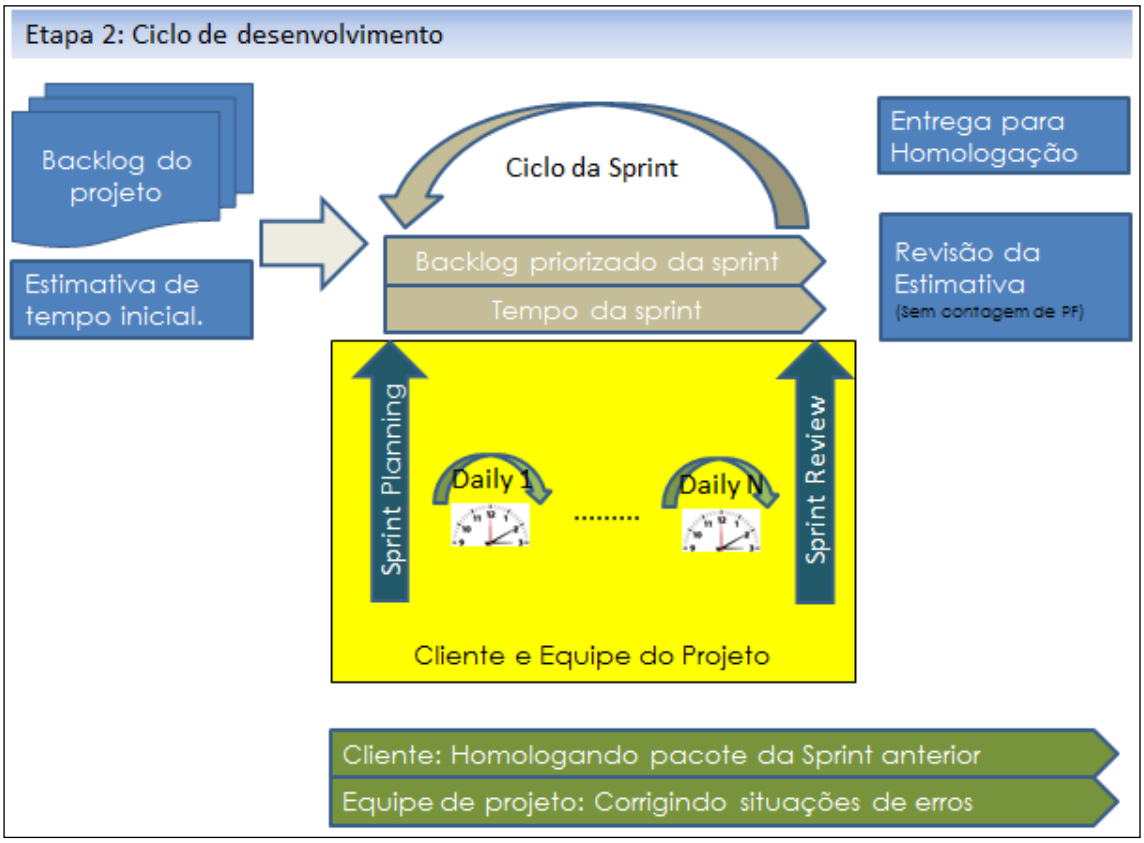
Sprint Planning: Reunião de planejamento da *sprint*. Define o que será feito e como será feito. Tem como entrada o *backlog* do projeto, onde são listados os requisitos do *software*. Nesta reunião de planejamento os requisitos são priorizados e definidos quais serão adicionados na *sprint*, estabelecendo assim os requisitos que serão incluídos na execução da *sprint*.

Sprint Review: No final de cada *sprint* é feita uma reunião para revisão do trabalho feito durante a *sprint*. Esta revisão tem por objetivo definir se os itens planejados foram feitos. Esta atividade ajudará no planejamento da próxima *sprint*, pois o *backlog* futuro poderá ser

afetado por itens não finalizados na *sprint* atual.

Daily: Reunião diária para discutir as atividades executadas, visando manter o alinhamento da equipe, principalmente para manter o objetivo de consumir o *backlog* da *sprint* no período estimado.

Figura 25: Ciclo de desenvolvimento.



Fonte: Autoria própria.

Na Figura 25 podem ser observadas algumas indicações:

- As informações de *backlog* do projeto e a estimativa serão entradas nesta etapa para o planejamento da *sprint*;
- A duração da sprint será definida conforme o *backlog* do projeto sendo indicado o máximo de quatro semanas e dependendo da necessidade e priorização com o cliente. Ao final de cada *sprint* deve ser gerado um relatório de atividades executadas para acompanhamento gerencial do projeto e posterior apresentação destas informações em comitês de projetos fora do âmbito de execução do projeto.
- Ao final do ciclo de todas as *sprints*, todo o *backlog* deverá ter sido trabalhado. Uma revisão da estimativa feita inicialmente deverá ser executada para eventuais correções dos requisitos trabalhados que foram consumidos durante as *sprints*.

- Ao finalizar todas as *sprints* necessárias para consumir o *backlog* do projeto, têm-se a etapa de ciclo de desenvolvimento finalizada, sendo as saídas das *sprints* os entregáveis do projeto.

Profissionais e pesquisadores reconhecem a necessidade de mais conhecimento sobre o dimensionamento de práticas ágeis. O segredo para o gerenciamento de projetos bem-sucedido é aprender com o passado e a retrospectiva é uma maneira de alcançar tal reflexão e aprendizado. O termo "retrospectiva" foi adotado pela comunidade ágil, sendo uma prática chave dentro dos métodos ágeis. A retrospectiva recebeu muita atenção tanto por profissionais quanto por pesquisadores devido ao fato de contribuir para os processos de melhoria de *software* com métodos ágeis (VESTUES, 2016).

Como indicamos que o trabalho de execução de projetos será executado em *sprints*, vale ressaltar que, conforme (Batra e Sin, 2006), para que as equipes tenham sucesso no desenvolvimento com métodos ágeis, espera-se que os membros das equipes sejam especialistas e tenham habilidades em trabalho em equipe e o cliente esteja disponível diariamente para orientar o desenvolvimento.

Logo após a entrega de uma *sprint*, na qual é iniciada a homologação, a equipe do projeto pode estar trabalhando na *sprint* seguinte. Em momento de homologação pode-se encontrar situações de erros que serão priorizadas com a equipe de desenvolvimento. Assim, o trabalho que está sendo executado na *sprint* atual deve ser paralelizado e priorizado com o cliente nas reuniões diárias, devido às situações de erro que possam surgir durante a homologação.

Com o andamento do projeto, ou seja, conforme as *sprints* são entregues, o valor agregado do projeto pode ser atualizado a cada *sprint*, permitindo assim medir a execução do projeto comparando este valor ao valor planejado do projeto. Com os valores de VA (Valor Agregado), VP (Valor Planejado) e CR (Custo Real) (Seção 2.4), é possível sintetizar os resultados do projeto no momento de finalização das *sprints*. Estes valores (VP, VA e CR) comparados a linha de base do projeto, que são os valores iniciais estimados, permitem medir eventuais desvios e assim ter uma rápida resposta às mudanças que é uma das características dos métodos ágeis.

Com os valores numéricos dos projetos é possível enviar às partes interessadas os relatórios de desempenho do projeto, por meio dos canais de comunicação indicados na etapa 0. Assim, deve-se coletar e publicar informações sobre o desempenho, incluindo status e resultados de medições de progresso.

Conforme Wang e Song (2010), o processo de relato de desempenho envolve a coleta, comparação e análise regulares para entender e comunicar o progresso e o desempenho do projeto e prever os resultados do projeto. Os relatórios de desempenho precisam fornecer informações de forma adequada para as partes interessadas. O formato do relatório de desempenho pode variar de simples relatório de status a relatório de descrição detalhada. Como relatório com uma descrição detalhada podemos citar a análise do desempenho passado, o risco atual, situação atual e riscos das tarefas para o próximo período do projeto.

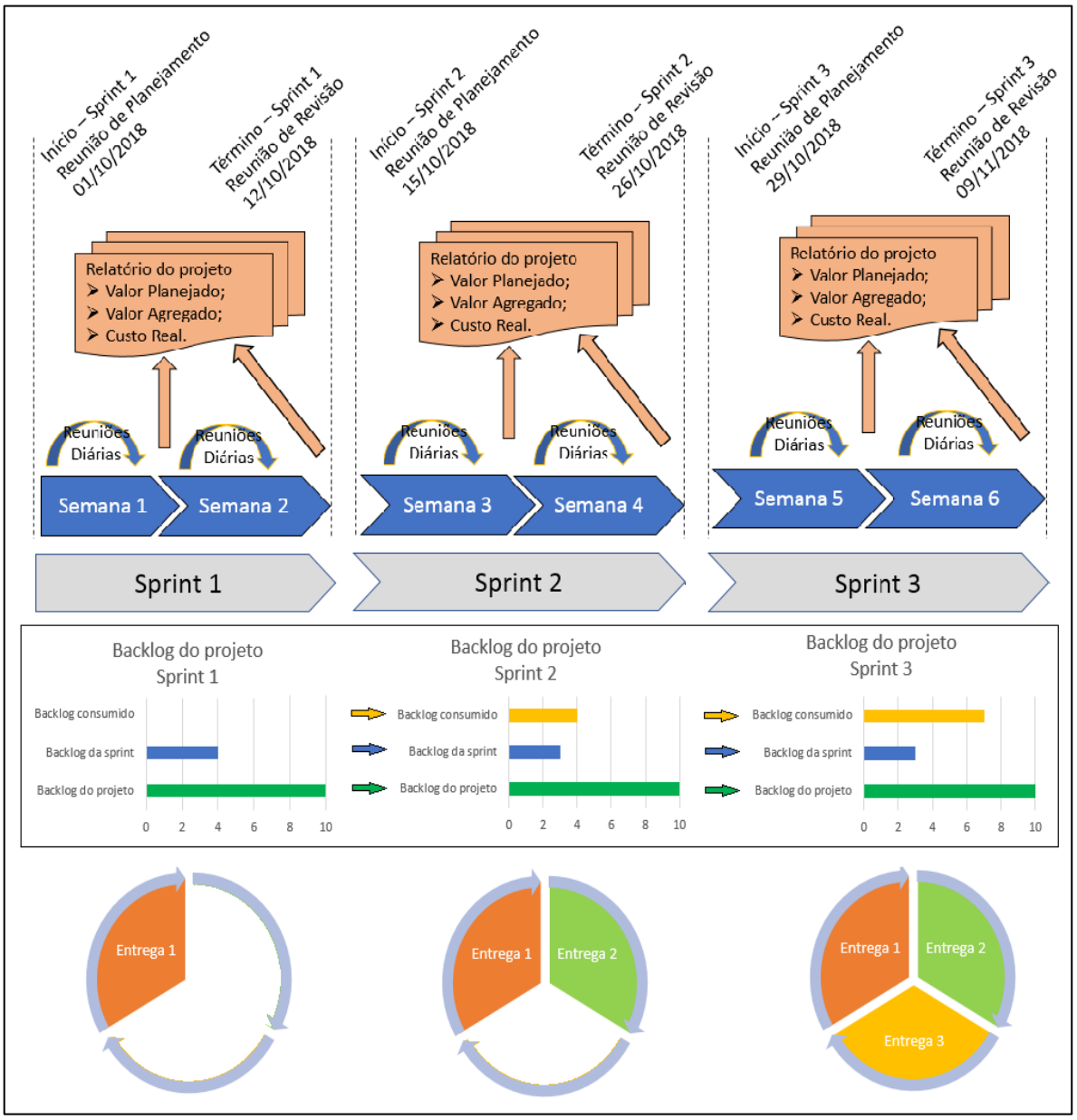
Para execução da abordagem de desenvolvimento será necessário que o time do projeto tenha profissionais com os seguintes perfis:

- Analistas com conhecimentos para elaborar requisitos de *software* com base em uma especificação de negócio;

- Analistas com conhecimento desenvolvimento de *software*;
- Analistas de testes que tem como objetivo definir testes com base em requisitos escritos e após, programados em uma determinada linguagem;
- Profissional com conhecimento em estimativa de pontos de função para estimar com base nesta métrica, valores pertinentes a custo e tempo, que serão gerenciados e acompanhados durante todo o projeto para garantir a viabilidade de execução e eventuais tomadas de decisões que envolvam as variáveis de custo e tempo.

A Figura 26 exemplifica a execução da etapa 2, referente ao ciclo de desenvolvimento.

Figura 26: Exemplo do ciclo de desenvolvimento.



Fonte: Autoria própria.

Conforme a Figura 20, ficam evidentes as entregas parciais, que podem agregar valor ao negócio em uma menor escala inicialmente e com incrementos parciais até chegar à entrega final. Com *feedbacks* parciais a cada entrega temos uma previsibilidade e visibilidade do produto de *software* implementado e, quando comparamos com o modelo cascata, temos no modelo cascata uma única entrega ao final do projeto.

Exemplificando, na Figura 26 temos 3 entregas totalizando com um backlog de 10 requisitos, assim temos:

- Na *sprint* 1 temos o *backlog* da *sprint* de 4 requisitos.
- Na *sprint* 2 temos o *backlog* da *sprint* de 3 requisitos e o *backlog* consumido de 4 requisitos (referente a *sprint* 1).
- Na *sprint* 3 temos o *backlog* da *sprint* de 3 requisitos e o *backlog* consumido de 7 requisitos (referente as *sprints* 1 e 2).
- No final da *sprint* 3, devemos ter o *backlog* consumido de 10 requisitos sendo estes 20 requisitos o total do *backlog* do projeto.

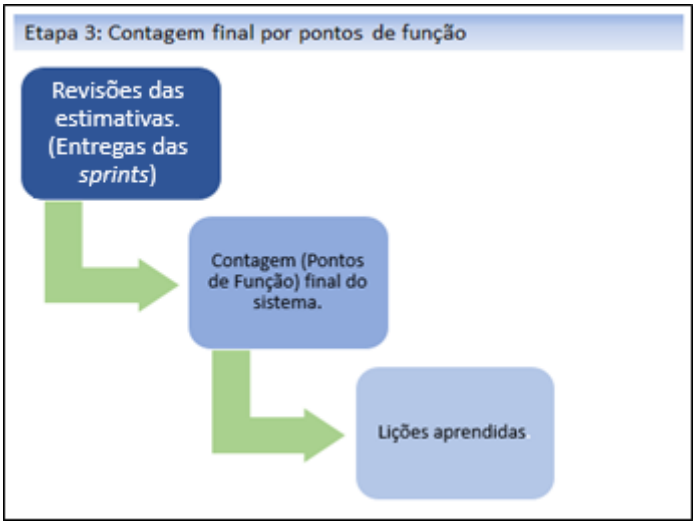
Supondo que para cada requisito tenhamos 1 ponto de função, teríamos um total de 100 pontos de função feitos em 3 *sprints*. Assim, na *sprint* 1 foram feitos 40 pontos de função, na *sprint* 2 foram feitos 30 pontos de função e na s*print* 3 foram feitos 30 pontos de função.

Etapa 3: Contagem final por pontos de função. (Figura 27)

Após a finalização de todo o *backlog* do projeto e entregas realizadas (Etapa 2), devem estar disponíveis todas as informações necessárias para uma recontagem do projeto por pontos de função. Assim, no momento desta recontagem, com os requisitos atualizados (com eventuais mudanças), podem ser realizadas correções acima ou abaixo da contagem inicial, corrigindo a primeira estimativa, caso necessário. Esta correção tem impacto no custo do projeto e em contagens futuras, pois o conhecimento gerado no projeto servirá como lição apreendida para projetos futuros (Figura 22).

Esta nova estimativa também servirá para os cálculos de índices de desempenho de prazo e custo, pois agora, com todas as entregas feitas, temos a contagem final de pontos de função, o tempo de execução do projeto e o custo detalhado. Estes valores podem ajudar em contagem futuras, pois podemos utilizar estes valores para ajustar a produtividade da equipe, que é um parâmetro utilizado na contagem de pontos de função.

Figura 27: Contagem final por pontos de função.

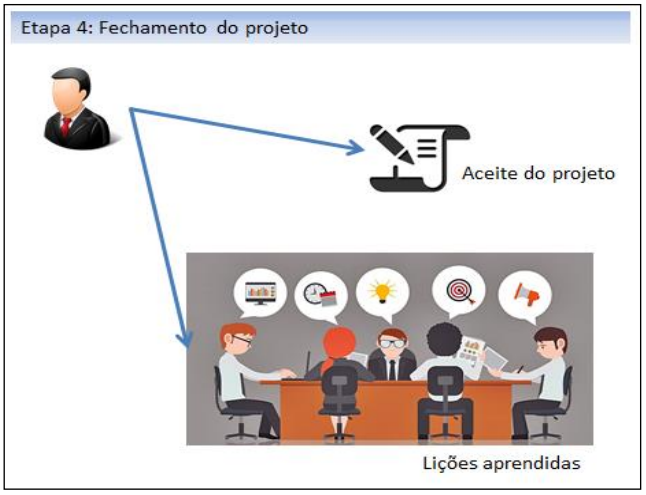


Fonte: Autoria própria.

Etapa 4: Fechamento do projeto. (Figura 28)

Esta é uma etapa na qual é formalizada a entrega final do projeto por meio da assinatura dos documentos que caracterizam o aceite. Não representa nenhuma etapa técnica, mas sim gerencial, que é feita juntamente com o cliente do projeto. Também é solicitada uma avaliação por parte do cliente do trabalho executado, visando melhorar o relacionamento e corrigir situações geradas anteriormente no ciclo de vida do projeto (Figura 28).

Figura 28: Fechamento do projeto.



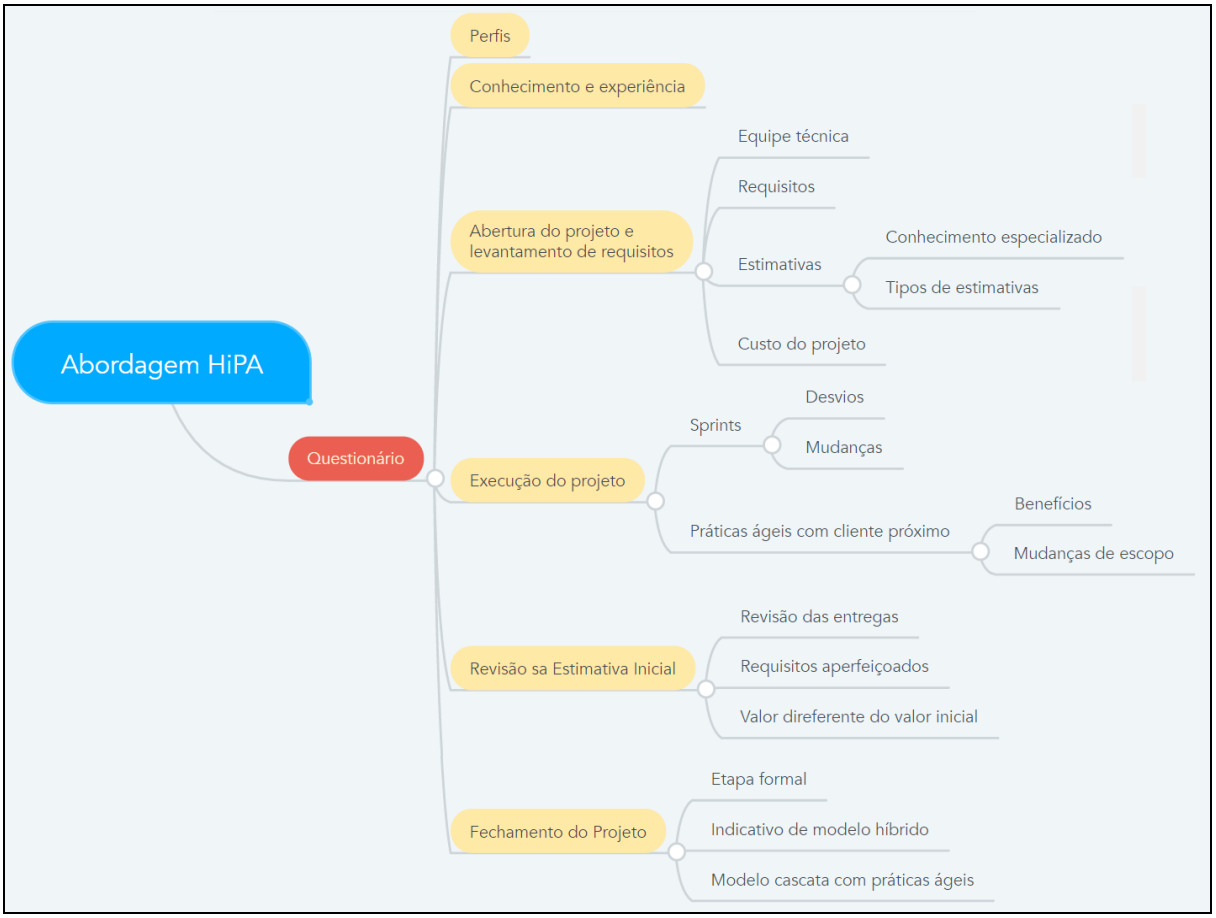
Fonte: Autoria própria.

5.3 AVALIAÇÃO DA ABORDAGEM DE DESENVOLVIMENTO DE *SOFTWARE* (ABORDAGEM HiPA)

A seguir, iremos mostrar os resultados obtidos por meio do questionário de avaliação da abordagem de desenvolvimento de *software* (Abordagem HiPA). O questionário está disponível no Apêndice C.

O mapa mental (Figura 29), mostra algumas informações observadas nas respostas do questionário, indicando tópicos importantes que devem ser observados na abordagem de desenvolvimento. Este mapa mental foi criado tendo como referências as questões indicando na granularidade final os principais tópicos observados das respostas dos participantes da pesquisa.

Figura 29: Mapa Mental.



Fonte: Autoria própria.

5.3.1 Resultados

O questionário composto de 10 questões foi aplicado no período de 08 de Abril de 2019 à 26 de Abril de 2019 e foi aplicado na mesma empresa que forneceu dados para o estudo de caso. Esta empresa não será aqui citada para manter a confidencialidade, mas podemos citar

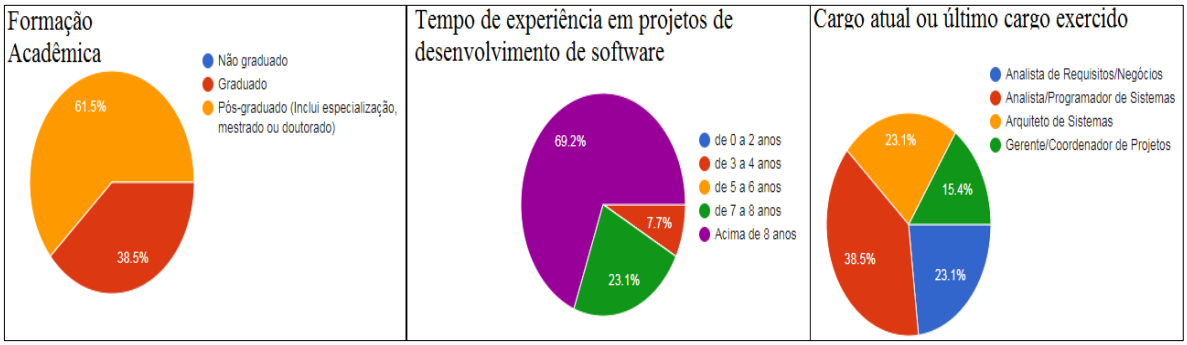
que é multinacional brasileira de soluções digitais, presente no mercado há mais de 20 anos e com operações em países da América do Sul, América Central e Estados Unidos.

O questionário foi encaminhado para 20 profissionais e a Figura 30 mostra o perfil profissional dos 13 respondentes da pesquisa. Todos os profissionais são graduados e 38,5% destes profissionais tem pós-graduação.

A maioria dos profissionais (69,2%) têm mais de 8 anos de experiência em projetos de desenvolvimento de *software*, na qual pode-se constatar um nível sênior de experiência.

Para os cargos exercidos, temos a maioria dos profissionais atuando na frente de análise e desenvolvimento de sistemas (38,5%). Em cargos de coordenação ou gerentes temos 15,4%, que representa 2 dos 13 respondentes.

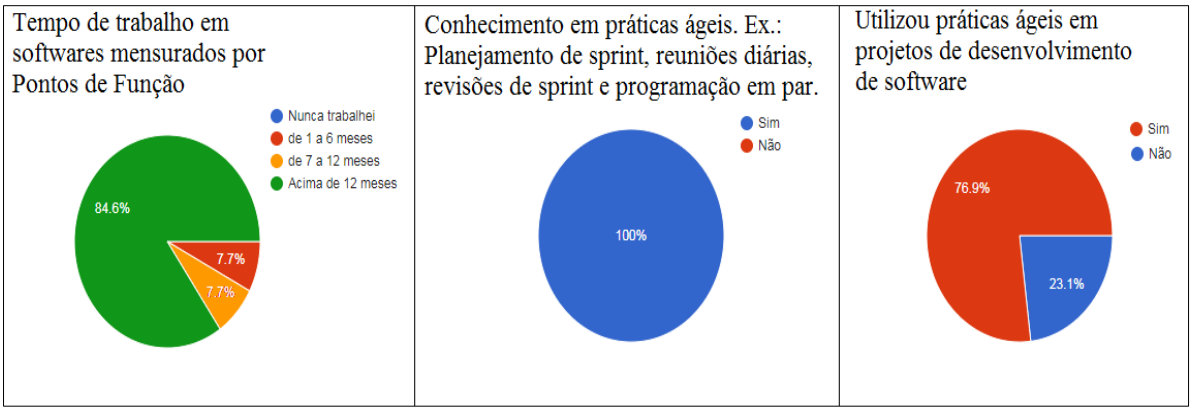
Figura 30: Perfis dos profissionais da pesquisa.



Fonte: Autoria própria.

Dos respondentes da pesquisa, todos já trabalharam com *softwares* mensurados por pontos de função (Figura 31). Este fato não indica que todos os respondentes tenham conhecimento na utilização da métrica, mas sim que têm conhecimento de um modelo de estimativa que foi aplicado para estabelecer o tamanho do *software*.

Figura 31: Experiência em Pontos de Função e práticas ágeis.



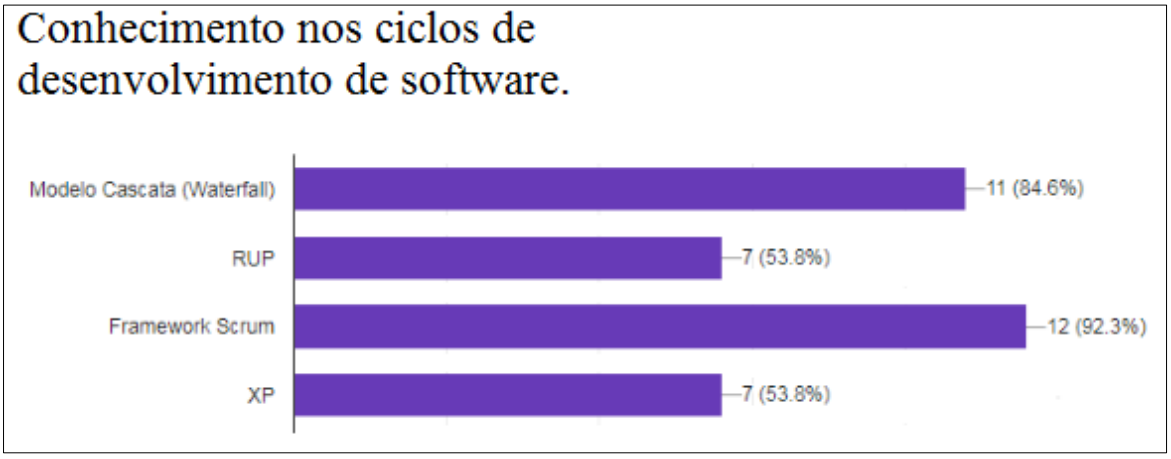
Fonte: Autoria própria.

Também conforme a Figura 31, todos os respondentes indicaram ter conhecimento de práticas ágeis. Aqui vale ressaltar que foram exemplificadas algumas práticas e não um método

ágil, sendo que a maioria dos respondentes indicaram que já utilizaram alguma prática ágil.

A Figura 32 indica o conhecimento que os respondentes indicaram ter com relação aos métodos ágeis, neste caso referente a XP e ao *Scrum* e também com relação aos métodos tradicionais, aqui, para o modelo Cascata e RUP. O modelo cascata e o *framework Scrum* foram os mais indicados se destacando do modelo RUP e XP.

Figura 32: Conhecimento em ciclos de desenvolvimento.



Fonte: Autoria própria.

A questão dissertativa da pesquisa, apresenta a abordagem HiPA de desenvolvimento de *software* e solicita aos respondentes indicar pontos pertinentes à abordagem. A seguir são apresentadas considerações feitas pelos respondentes em relação a abordagem HiPA:

1. Com relação as indicações referentes as etapas de abertura do projeto, levantamento de requisitos e a contagem inicial por pontos de função temos:

- *"No processo de abertura do projeto, a participação da equipe técnica é algo muito importante, afinal quanto maior o conhecimento negocial, maior a qualidade e velocidade da codificação, além de reduzir os riscos de desenvolvimentos incorretos, que podem gerar estornos futuros e até o descontentamento do cliente".*
- *"Os requisitos e métricas iniciais, ajudam a equipe a enxergar o direcionamento do trabalho e torna particularidades de regras de negócios mais detalhadas e documentadas".*
- "*A estimativa do "esforço X funcionalidades" existentes na solução, mostra ao solicitante a visão de esforço e custo para desenvolvimento, fornecendo ao cliente a possibilidade de aprovação ou não do orçamento antes de iniciar projeto. Outro ponto forte é não condicionar o esforço a qualquer tecnologia, estrutura ou linguagem e sim a funcionalidade negocial, que é uma característica do ponto de função*".
- "*Necessário conhecimento especializado para mensurar esforços*".

- "*Estimativa não pode ser contestada, quando for definida por um profissional certificado. Existe algumas garantias pelo órgão certificador sobre essas questões".*
- "*A compensação está no equilíbrio do valor financeiro acordado por cada ponto identificado. Nada adianta ter uma contagem assertiva se o valor de contrato por pontos de função não compensar*".
- "*A estimativa em Pontos de Função deveria ser utilizada como métrica financeira e não como métrica de tempo para entrega do projeto. Vejo que as estimativas iniciais deveriam ser feitas pelo time que estará envolvido no desenvolvimento (analistas, desenvolvedores etc.) utilizando práticas ágeis, como por exemplo, Planning Poker*".
- "*Considero a etapa inicial de suma importância pela clareza na definição dos papéis dos integrantes da equipe. A ciência do papel de cada um evita atritos, cobranças indevidas, insubordinação e outros fatores negativos no relacionamento interpessoal, além de deixar claro o grau de responsabilidade de cada indivíduo no desenrolar do projeto*".
- "*Quanto aos requisitos e métrica inicial, considero o tempo um fator crítico para que não haja uma grande diferença entre a contagem inicial e a contagem final, além da disponibilidade e engajamento do cliente na definição dos requisitos*".
- "*Na minha visão as etapas iniciais, antes de começar o desenvolvimento por meio da execução das sprints, são as etapas principais para que o desenvolvimento e produto final atenda às expectativas do cliente com baixo nível de desvio.*
- *Um bom levantamento e um bom projeto de arquitetura agilizam de forma significativa o desenvolvimento".*
- "*A contagem de pontos inicial pode ser superestimada, fazendo que uma recontagem apenas no final do projeto gere prejuízos, caso todo o planejamento de valores seja realizado com base na contagem inicial*".

2. Com relação as indicações referentes a execução do projeto:

- "*Na execução do projeto, vejo que a proximidade com o cliente nas reuniões diárias torna possível a correção de problemas de forma imediata*".
- "*A interação da equipe com o cliente, acelera o desenvolvimento, reduz riscos de erros e dúvidas relacionadas ao projeto. Mas essa proximidade pode acabar gerando problemas, uma vez que o cliente estar diariamente visualizando o projeto e solicitando "modificações", pode comprometer o prazo e o orçamento, uma vez que o cliente se sente a vontade para modificar o sistema como quiser. O cliente pode exigir alterações que fogem do escopo do projeto, comprometendo toda a entrega. No final o fornecedor pode ser penalizado por esse atraso, além do que fugindo do escopo, corre-se o risco de realizar desenvolvimentos sem custo para o cliente".*

- "*O ponto fraco desse modelo é que em algumas vezes, por decorrência do prazo/urgência do projeto, sprints anteriores pode causar impacto de tempo nas sprints seguintes"*.
- "*O cliente receberá várias entregas e consegue dar melhor feedback do desenvolvimento*".
- "*O sucesso da etapa de execução das sprints depende em grande parte das etapas iniciais. Vejo que nos métodos ágeis, é fundamental a senioridade dos desenvolvedores para a rapidez das entregas e baixo nível de correções*".
- "*É fato que a teoria prega a correção de desvios a cada sprint, mas na prática, este cenário não é bem aceito pelo cliente e nem pela gerência do projeto. Há que se esclarecer este aspecto dos métodos com mais ênfase aos envolvidos"*.
- "*Durante o desenvolvimento sempre existe um grande risco de mudança de escopo, o que acarretaria na estimativa de prazo/valor*".

3. Com relação a revisão da estimativa inicial:

- "*Na revisão dos requisitos e mensuração final do sistema, é necessário levantar tudo que será entregue e tudo que não foi possível entregar*".
- "*A fase de contagem e recontagem não condiz com o conceito de agilidade. A estimativa por pontos de função requer conhecimento especializado (algumas vezes certificado)*".
- "*Concordo com a contagem em Pontos de Função neste momento em que a demanda já está homologada pelo cliente*".
- "*A contagem final poderá distorcer o valor inicial (para mais ou para menos), quando o esforço já foi executado, não podendo voltar atrás. Alguma das partes poderá ficar no prejuízo, de alguma forma*".
- "*Para que a etapa de revisão da contagem traga bons resultados, na prática, os itens que aperfeiçoam a contagem detalhada precisam ser registrados durante todo o projeto, de modo a garantir que nada seja perdido*".

4. Com relação ao fechamento do projeto:

- "*No fechamento do projeto é necessário mostrar tudo que foi desenvolvido de forma clara para ambos os lados, pois nesse momento poderemos ser cobrados, questionados e até ter o projeto recusado, dependendo do cliente*".
- "*Foi uma abordagem cascata que teve a inclusão de algumas cerimonias ágeis. Acredito que pode ser visto como uma iniciativa inicial para o time, visando uma abordagem 100% ágil*".
- "*Acredito que se o escopo do projeto for pequeno e bem definido a abordagem funcione. Aparentemente é aplicado uma abordagem híbrida, onde parece usar o modelo cascata nas etapas iniciais e uma abordagem ágil nas etapas seguintes. Acredito que funcione para projetos de escopo pequeno*".

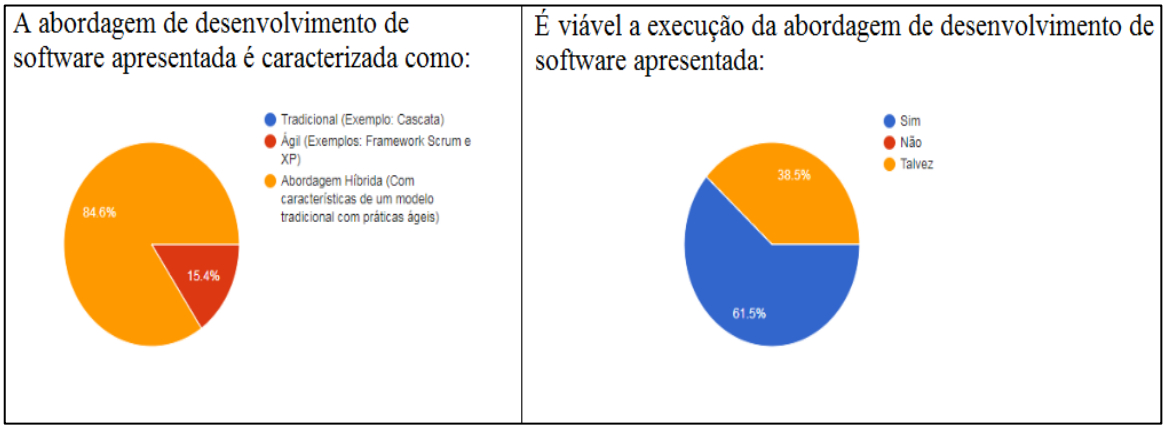
- "*A etapa final é meramente formal, mas de importância pela formalização da entrega*".

5. Com relação a métrica de pontos de função:

- "*Fazendo uso repetido da APF, fazendo análises dos resultados históricos (previsto x realizado) e fazendo os ajustes, ao longo do tempo os indicadores de produtividade e previsões irão se tornar mais confiáveis. Com os devidos ajustes, tende a se tornar uma forma de medição e remuneração objetiva*".
- "*Em termos de planejamento, previsibilidade, pode ser usada para decidir melhor as prioridades considerando "esforço X prazos*"".
- "*Na minha experiência prática, existem diferenças de interpretação de contagem entre o cliente e o fornecedor. Um exemplo é o analista do fornecedor contar seguindo estritamente o padrão de APF e o analista do cliente usar critérios ligeiramente diferentes em alguns pontos. Outro exemplo, itens não funcionais, normalmente são contados com um padrão próprio do cliente. Esses dois exemplos podem gerar divergências que podem impactar em vários aspectos como o financeiro e indicadores de nível de serviço/qualidade*".
- "*Os analistas do fornecedor devem estar treinados/habilitados para uma contagem de pontos de função o mais precisa possível e estar atento as exceções do cliente*".
- "*Existem fatores de ajustes das contagens de APF que nem sempre são revisadas, refinadas e ajustadas para as caraterísticas de projeto com relação a produtividade e arquitetura por exemplo*".
- "*Como sugestão, acredito que além do alinhamento do padrão de contagem entre cliente e fornecedor. Deveria também haver um processo de melhoria contínua em relação ao processo de contagem, revisando e identificando exceções a serem tratadas na contagem e produtividade*".

Finalizando o questionário temos a Figura 33, onde é indicado pela maioria dos respondentes (84,6%) que a abordagem de desenvolvimento é uma abordagem híbrida e, indicada por 15,4% como uma abordagem Ágil. A abordagem HiPA foi indicada como possível de ser executada por 61,5% dos respondentes e não indicada como sendo uma abordagem inviável.

Figura 33: Viabilidade da abordagem.



Fonte: Autoria própria.

5.3.2 Avaliação Preliminar de Viabilidade da Abordagem

Fizemos uma avaliação de viabilidade da abordagem HiPA na execução de projetos. Todos os projetos aqui analisados foram feitos na mesma empresa desenvolvedora que também forneceu os dados para o estudo de caso. E os 3 projetos aqui analisados foram executados para um mesmo cliente, não sendo identificados por motivo de sigilo. Mas podemos informar que o cliente se trata de uma grande empresa do setor bancário que atua a nível nacional.

Considerando a abordagem HiPA (Figura 15) e conforme as etapas a seguir, avaliamos as 3 etapas intermediárias.

1) Etapa 0 - Abertura do Projeto;
2) Etapa 1 - Requisitos e Estimativa Inicial (*Etapa avaliada*);
3) Etapa 2 - Ciclo de Desenvolvimento (*Etapa avaliada*);
4) Etapa 3 - Contagem final por pontos de função (*Etapa avaliada*);
5) Etapa 4 - Fechamento do Projeto.

Observamos projetos que foram executados para termos dados reais para avaliação. Para os 3 projetos (Tabela 12) observamos as etapas de “Requisitos e Estimativa Inicial, Ciclo de Desenvolvimento e Contagem final por pontos de função”, sendo as etapas observadas aderentes a abordagem proposta.

As etapas indicadas nos itens 1 (Abertura do Projeto) e 5 (Fechamento do projeto) não foram observadas nem executadas devido ao modelo de execução destes projetos, o que poderia descaracterizar a forma de execução destes projetos de *software* no modelo real em que estavam sendo observados. Mas as fases intermediárias permitem avaliar a viabilidade de execução da abordagem HiPA.

Todos os 3 projetos foram executados no período de 1 mês, sendo este o tempo da *sprint* única, conforme negociação com o cliente.

Tabela 12. Projetos de avaliação do modelo

Projeto	Etapa 1 – Requisitos e Estimativa Inicial	Etapa 2 – Ciclo de Desenvolvimento (Execução da *Sprint*)	Etapa 3 – Medição final por pontos de função
P1	29 PF	Execução com Práticas Ágeis	40 PF
P2	24 PF	Execução com Práticas Ágeis	27 PF
P3	112 PF	Execução com Práticas Ágeis	149 PF

Fonte: Autoria própria.

Diante dos dados evidenciados na Tabela 12, temos:

1) Na etapa 1, temos uma estimativa inicial de pontos de função e elaboração dos requisitos necessários para o desenvolvimento do sistema.
2) Durante a etapa de execução (Etapa 2), utilizamos práticas ágeis, conforme (Seção 5.2, Figura 25).
3) Após a finalização das *sprint*s, foi feita uma nova contagem com base nos requisitos iniciais, sendo que estes requisitos foram reescritos conforme a execução do projeto.
4) Em todos os projetos notamos que houve uma mudança de valores na contagem. Assim, esta contagem final corrige as necessidades de mudança realizadas nos requisitos iniciais. Com esta correção, temos um novo valor de pontos de função o que vem a resultar em uma correção de custo e tempo do projeto.

Estas correções da contagem dos pontos de função mostram que existe a viabilidade de uma recontagem do tamanho do *software*. Assim, indicamos que a fase de estimativa final é necessária para readequação do tamanho do *software*.

Esta contagem final, corrige desvios de requisitos identificados inicialmente e que passaram por alguma mudança durante o ciclo de desenvolvimento, ficando estes requisitos mais claros e melhor definido após a execução do desenvolvimento.

O índice de desempenho de prazo não foi analisado, mas os 3 projetos foram entregues no prazo da *sprint* de 1 mês. E a recontagem feita serve para ajustar a produtividade que é uma variável de cálculo dos pontos de função.

Deste modo vemos como viável a contagem de tamanho de *software* por meio de pontos de função em 2 etapas de um projeto desenvolvido com a utilização de práticas ágeis. Além de verificar a viabilidade, sugerimos que a contagem seja refeita para corrigir valores de custo, produtividade e gerar lições aprendidas que sejam contribuições futuras em novos projetos com contagem de pontos de função.

6 LIMITAÇÕES E AMEAÇAS À VALIDADE DOS RESULTADOS

Inicialmente iremos fazer algumas colocações sobre aspectos aqui não estudados, mas pertinentes para a situação, pois este é um ponto que pode ser evoluído em outros estudos e assim direcionando o assunto para um viés menos técnico.

No contexto deste trabalho, a análise de diversidade da equipe não foi considerada. Para gerenciar níveis de interação entre profissionais de software. Segundo Yilmaz et al. (2016), é importante observar que indivíduos podem ter diferentes traços de personalidade e a consciência de tal diferença pode melhorar significativamente a motivação da equipe.

Sobre aspectos de diferentes personalidades em uma equipe, o fator do capital social ou o conhecimento que cada profissional retém e utiliza em projetos e os acessos a especialistas nas redes de relacionamentos são cruciais para equipes de trabalhando. A importância da rede de relacionamento é evidente em projetos de grande escala, em que a acumulação de conhecimento leva muito tempo e o conhecimento especializado é distribuído entre diferentes unidades dentro da organização de desenvolvimento (SMITE et al., 2016). E, conforme Nagappan et al. (2008), estudos geralmente ignoram um dos fatores mais influentes no desenvolvimento de software, especificamente "pessoas e estrutura organizacional".

As pessoas são fundamentais para o processo de desenvolvimento de software e os fatores pessoais são um dos elementos que podem afetar as equipes de desenvolvimento software (YILMAZ et al., 2016).

Com relação a adoção e implementação de métodos ágeis, os três desafios mais significativos são relatados como, cultura organizacional em desacordo com os valores ágeis (53%), resistência organizacional geral à mudança (46%) e apoio e patrocínio inadequado à gestão (42%)[12]. Estes pontos são aspectos interessantes quando analisados em conjunto com aspectos da estrutura organizacional de uma empresa, pois pode ser um fator problemático ao tentar incorporar uma nova maneira de trabalho em uma organização.

Assim, existem alguns fatores chave, aqui não estudados, que podem ser considerados limitações do presente trabalho.

O mapeamento sistemático indicou que a temática de mensuração de tamanho de software é um assunto que é abordado em pesquisas. Também foi possível constatar que existe um direcionamento para que métodos de desenvolvimento sejam utilizados parcialmente, tendo assim um modelo híbrido para atender um determinado ambiente de desenvolvimento de software. Apesar disto, não foi encontrado nenhum trabalho que faça o direcionamento do uso de pontos de função em projetos ágeis. Este assunto foi abordado pelo IFPUG[13] em 2017, tendo como título “Como pontos de função ajuda em projetos de métodos ágil”.

No estudo de caso, fizemos uma comparação entre projetos executados com um método cascata e projetos utilizando práticas ágeis, sendo todos os projetos de escopo fechado estimados por pontos de função. Esta comparação de resultados, por meio de indicadores de tempo e custo permite verificar a eficácia da abordagem de utilização de pontos de função em projetos com práticas ágeis, que é uma lacuna de pesquisa aqui explorada.

---

[12] https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report (VersionOne 12th Annual State of *Agile* Report). Acesso em 15/09/2019.

[13] http://www.ifpug.org/how-function-points-help-*agile*-methodology-projects/?lang=pt. Acesso em 15/09/2019.

Com relação ao modelo híbrido, este tema foi apresentado em trabalhos *como Project Management Institute* (2017), West (2011) e Theocharis et al. (2015) e, a abordagem híbrida pode ser vista como uma nova lacuna para explorar os motivos que levam um ambiente de desenvolvimento a adotar e evoluir nesta abordagem.

A abordagem de desenvolvimento proposta é considerada híbrida e foi colocada para avaliação conceitual por meio de um questionário e avaliada parcialmente em projetos reais. Com a iniciação de execução desta abordagem poderão surgir várias indicações de melhorias e correções para adequação ao um determinado cenário de projeto de software. É natural que isto ocorra, e a iniciação total desta abordagem deve provocar ações para viabilizar melhorias no ciclo de desenvolvimento. Com relação à abordagem de desenvolvimento proposta (Abordagem HiPA), a iniciação do modelo deverá criar ações de críticas. Isto poderá ser avaliado e enriquecido no modelo visando a maturidade da implantação dessa abordagem em um cenário real de desenvolvimento de software.

Uma das ameaças a abordagem proposta é a forma de avaliação dos indicadores de projeto. Conforme Cabri e Griffiths (2006), o valor agregado é uma técnica de gerenciamento de projetos que foi iniciada em 1890.

Também em Cabri e Griffiths (2006), o conceito de valor agregado se baseia em um plano de linha de base para medir o progresso em um projeto com escopo bem definido que evolui de maneira sequencial e linear. Com métodos ágeis, os projetos evoluem de forma interativa e não linear, com ciclos de *feedback* que afetam o plano inicial. A mudança é esperada e frequente durante toda a duração do projeto. Portanto, medir o progresso relativo ao plano inicial pode ser enganoso. Embora existam problemas na tentativa de aplicar conceitos de valor agregado aos métodos ágeis, práticas ágeis de gerenciamento de projetos, gráficos "*burndown charts*" (indicando a quantidade de funcionalidade pendente vs. concluída ao longo do tempo, etc.) fornecem informações de progresso e status muito semelhantes ao valor agregado. E, os custos podem ser adicionados aos gráficos dos métodos ágeis para exibir as informações juntamente com a taxa de conclusão do recurso. Tudo isso pode ser mais valioso para o gerenciamento de projeto e para as partes interessadas do projeto no monitoramento de um projeto que utiliza métodos ágeis, em vez de tentar aplicar o valor agregado tradicional.

Em Li et al. (2008), com relação à aplicação da técnica de valor agregado para validação de indicadores de projeto, têm se que existe um problema em como expressar consistentemente as linhas de base do projeto e medir o progresso técnico dos projetos de *software*, principalmente para refletir as mudanças que venham a ocorrer no desenvolvimento de *software*.

Como este trabalho propôs uma abordagem de desenvolvimento híbrida utilizando vários conceitos como pontos de função, análise de desempenho de projeto por meio de valor agregado e práticas ágeis, temos um cenário complexo no desenvolvimento de software.

Podemos dizer que esta abordagem híbrida de desenvolvimento, tem uma principal ameaça, que é termos várias execuções desta abordagem para consolidar os passos descritos. Mas esta consolidação da abordagem pode ser evoluída para se tornar uma oportunidade de melhoria, assim a abordagem de desenvolvimento poderia ser ajustada e melhorada. Neste caso, podemos tratar estas ameaças como uma grande oportunidade para consolidar o trabalho proposto tecnicamente.

7 CONCLUSÃO

Com a realização do mapeamento sistemático, encontramos lacunas de pesquisa referentes a projetos de *software* que utilizam práticas ágeis e estimados por pontos de função. Ao mesmo tempo que encontramos essas lacunas, encontramos alguns trabalhos e pesquisas que indicam a utilização de um modelo ou abordagem híbrida de desenvolvimento de *software*.

No estudo de caso, fizemos uma análise situacional de projetos por meio de indicadores de custo e prazo para verificar os resultados de projetos que foram executados em um modelo tradicional e projetos que foram executados com práticas ágeis. Após obtenção dos resultados dos indicadores, constatamos que os projetos trabalhados com práticas ágeis tiveram um retorno positivo e melhor que os projetos executados no modelo tradicional.

Desta maneira procuramos abordar os 2 primeiros objetivos deste trabalho. Como este trabalho envolve métricas de sistema, e práticas ágeis, sendo que os métodos e práticas ágeis estão ganhando espaço juntamente com modelos híbridos de desenvolvimento, direcionamos o trabalho para propor uma abordagem híbrida de desenvolvimento de *software* com estimativa realizada por pontos de função.

Propusemos a abordagem de desenvolvimento de *software* HiPA, que foi avaliada parcialmente em projeto de software e por meio de um questionário. Então, temos uma abordagem híbrida de desenvolvimento de *software* com uma estimativa bastante utilizada como método de mensuração comercial de *software* em conjunto com práticas ágeis. Esta combinação de pontos de função e práticas ágeis está sendo pouco explorada na literatura atual.

Para execução e avaliação dessa abordagem híbrida em um modelo mais robusto, precisamos amadurecer a abordagem e em uma estrutura de desenvolvimento de *software*. O modelo inicial já temos, o desafio agora é transformar essa abordagem em um modelo forte que seja amparado por profissionais que consigam trabalhar e evoluir nessa proposta inicial.

Assim, conforme os objetivos estabelecidos, vemos que, se tivermos um acerto de produtividade na estimativa de pontos de função, o trabalho em um modelo híbrido utilizando práticas ágeis pode entregar maior valor ao negócio, conforme as previsibilidades que fornecem a utilização de práticas ágeis e, com ajuda de pontos de função como uma medição que ajuda na precificação e relação entre quem vende e compra o produto de *software.*

A implantação de um modelo não é um processo simples dentro de uma estrutura organizacional. Existem vários obstáculos que podem dificultar a implantação desta abordagem como, cultura e resistência organizacional à mudança, falta de apoio e patrocínio da gestão, fatores pessoais e diversidades das pessoas, ou seja, aquilo que não está formalmente expresso em documentos, mas que direcionam as atitudes e hábitos das pessoas dentro de sua função.

A relação comercial entre quem compra e quem vende um desenvolvimento de *software* também é um assunto pouco explorado. A partir de uma abordagem de desenvolvimento de *software* em que mudanças podem ocorrer em uma maior frequência, a relação comercial entre quem compra e quem vende um software é um ponto muito importante.

Então, ameaças pertinentes a fatores pessoais e organizacionais, juntamente com modelos comerciais para compra e venda de *software,* poderiam ser abordadas em trabalhos futuros. Estas ameaças são fatores bastantes relevantes para contribuir com uma abordagem dinâmica de desenvolvimento de *software* mensurado com uma métrica de precificação, sendo

que esta abordagem aqui é indicada como uma contribuição.

O desenvolvimento de *software* tradicional engloba todos os métodos de desenvolvimento de *software* caracterizados por um ciclo de desenvolvimento estruturado do *software*, consistindo em fases predefinidas, documentação extensa de requisitos e estruturas de organização hierárquica. Este tipo de desenvolvimento tem uma baixa frequência de lançamento. Práticas ágeis podem coexistir com o desenvolvimento tradicional. No entanto, deve-se ter em mente o contexto e tomar medidas para mitigar os desafios incorridos (WAARDENBURG; VLIET, 2013).

7.1 CONTRIBUIÇÕES

As principais contribuições deste trabalho foram:

- Caracterização do panorama de estudos da utilização de pontos de função com práticas ágeis por meio da execução de um mapeamento sistemático da literatura.
- Definição de um modelo de comparação de projetos por meio de índices de desempenho. Aqui utilizamos os índices de desempenho de custo e prazo. E a partir destas informações temos indicadores numéricos que podem ser comparados, gerando referências de informação em contextos de projeto de software.
- Proposição de uma abordagem hibrida de desenvolvimento de software utilizando a métrica de pontos de função, que é bastante difundida e utilizada, para estimativa de projeto de software.
- Uso de práticas ágeis em conjunto com indicadores numéricos de custo e prazo para acompanhar e avaliar a execução do projeto.
- Avaliação da abordagem hibrida de desenvolvimento de software proposta por meio da execução parcial em projetos reais, para assim, concluir que temos uma abordagem viável e passível de medições para acompanhamento do projeto de software.

7.2 TRABALHOS FUTUROS

Como trabalhos futuros decorrentes desta dissertação, podemos citar:

- Investigar como fatores de uma estrutura organizacional podem influenciar no desenvolvimento de software empregando a abordagem hibrida proposta por este trabalho de pesquisa, visto que as mudanças no decorrer do desenvolvimento devem ser analisadas e implantadas, pois são bem vistas em uma estrutura dinâmica de desenvolvimento de software.
- Avaliar a abordagem proposta em conjunto com outras formas de obtenção de estimativa do projeto, por exemplo, *planning poker*. Esta avaliação poderia, também, apontar para um outro trabalho de pesquisa, que explore como outros tipos de métricas de estimativa de projeto de software poderiam ser aceitas em uma relação de compra e venda de *software*.
- Analisar como a questão cultural, a diversidade e a resistência as mudanças em um ambiente com várias pessoas, culturas e traços diferentes de personalidades, entre outros fatores, podem interferir em mudanças de paradigmas de desenvolvimento de *software* em uma organização.

REFERÊNCIAS BIBLIOGRÁFICAS

ABRAHAMSSON, Pekka; WARSTA, Juhani; SIPONEN, Mikko; RONKAINEN J. New Directions on *Agile* Methods: A Comparative Analysis. International Conference on *Software* Engineering. 2003.

AGRAWAL, Amit; JAIN, Vaibhav; SHEIKH, Mohsin. Quantitative Estimation of Cost Drivers for Intermediate COCOMO towards Traditional and Cloud Based Software Development. Proceedings of the Ninth Annual ACM India Conference on - ACM Compute 2016.

AZAM, Farooque; GULL, Hina; BIBI, Saeeda; AMJAD, Sameera. Back & Forth (BnF) Software process model. 2nd International Conference on Computer Engineering and Applications, ICCEA. 2010.

BATRA, Dinesh; SIN, Thant. Modified Agile Practices for Outsourced Software Projects. Twelfth Americas Conference on Information Systems. 2006.

CABRI, Anthony; GRIFFITHS, Mike. Earned Value and Agile Reporting. Conference AGILE 2006.

CARROLL, Edward R. Estimating software based on use case points. 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications - OOPSLA. 2005.

DAGNINO, Aldo. Estimating software-intensive projects in the absence of historical data. Proceedings - International Conference on Software Engineering. 2013.

DIEBOLD, Philipp.; ZEHLER, Thomas. The right degree of agility in rich processes. Managing Software Process Evolution: Traditional, Agile and Beyond – How to Handle Process Change. Springer International Publishing, 2016.

BARBALHO, Sanderson César Macêdo; ROZENFELD, Henrique. Análise do processo de desenvolvimento de produtos de uma pequena empresa de alta tecnologia. Anais do XXIV Encontro Nacional de Engenharia de Produção, 2004, Florianópolis. Associação Brasileira de Engenharia de Produção, Novembro, 2004.

BECK, Kent; Et. al. Agile Manifesto. 2001: http://agilemanifesto.org (Acesso em 15/09/2019).

BEEDLE, Mike; SUTHERLAND, Jeff. SCRUM: An extension pattern language for hyperproductive software development. 1999.

BOEHM, Barry. Get Ready for Agile Methods, with Care. IEEE Journals & Magazines, (p. 64-69). 2002.

CHAUHAN, Bishan Dayal; RANA, Ajay; SHARMA, Neeraj. Kumar. Impact of development methodology on cost & risk for development projects. 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO). 2017.

COHEN, David; LINDVALL, Mikael; COSTA, Patricia. An introduction to agile methods. In Advances in Computers. Advances in Computers, (p. 1-66). 2004.

CUADRADO-GALLEGO, Juan J.; ABRAN, Alain; RODRÍGUEZ-SORIA, Pablo; LARA Miguel A. An experimental study on the conversion between IFPUG and UCP

functional size measurement units. Journal of Zhejiang University SCIENCE C, (p. 161–173). 2014.

ECAR, Miguel; KEPLER, Fabio; SILVA, João Pablo S. Cosmic User Story Standard. Agile Processes in Software Engineering and Extreme Programming. 19th International Conference, XP 2018 Porto, Portugal, May 2018.

EBERT, Christof; SOUBRA, Hassan. Functional size estimation technologies for software maintenance. IEEE Software. 2014.

FRENCH, Dan. Agile and Function Points: A Winning Combination. Presented at the 2016 ICEAA Professional Development. http://cobec.com/wp-content/uploads/2016/08/Agile-and-Function-Points-A-Winning-Combination-by-Daniel-French.pdf. (Acesso em 15/09/2019). Atlanta. 2016.

GARCIA, Carlos Augusto Lombardi; HIRATA, Celso Massaki. Integrating functional metrics, COCOMO II and Earned Value Analysis for software projects using PMBoK. ACM symposium on applied computing. 2008.

HARIHARN, S; RENGRAJAN, A; PREM KUMAR, R. Scrum based scaling using agile method to test software projects and its future solutions using in artificial neural networks. International Journal of Innovative Technology and Exploring Engineering (IJITEE). 2019.

HASSANI, Rachida; EL BOUZEKRI, Younes Idrissi; ABOUABDELLAH Abdellah. Digital project management in the era of digital transformation: Hybrid method. ICSIM-International Conference on Software Engineering and Information Management. 2018.

HIRSCH, Michael. Making RUP agile. OOPSLA Practitioners Reports. 2002.

HUBLIKAR, Sandeep; HAMPIHOLI Shrikanth. Pause, Reflect and Act, the Pursuit of Continuous Transformation. Agile Processes in Software Engineering and Extreme Programming. 17th International Conference XP. 2016.

INCE, Darrel. Software metrics: introduction. Information and Software Technology, (p. 297-303). 1990.

INTERNATIONAL FUNCTION POINT USERS GROUP. Disponível: http://www.ifpug.org. (Acesso em Abril/2019).

JHA, Madan Mohan; VILARDELL, Rosa Maria Ferrer; NARAYAN Jai. Scaling Agile Scrum Software Development: Providing Agility and Quality to Platform Development by Reducing Time to Market. IEEE 11th International Conference on Global Software Engineering (ICGSE). Irvine, CA, USA. 2016.

JIANG, Li; EBERLEIN, Aarmin. Towards a Framework for Understanding the Relationships between Classical Software Engineering and Agile Methodologies. International Conference on Software Engineering. Leipzig, Germany. 2008.

JONES, Capers. Function Points as a Universal Software Metric. ACM SIGSOFT Software Engineering Notes. 2013.

KANG, Sungjoo; CHOI, Okjoo; BAIK, Jongmoon. Model-based Dynamic Cost Estimation and Tracking Method for Agile Software Development. 9th IEEE/ACIS International Conference on Computer and Information Science. 2010.

KAUR, Anureet; KAURB Kaur. A COSMIC function points based test effort estimation model for mobile applications. Journal of King Saud University - Computer and Information Sciences. 2019.

KITCHENHAM, Barbara. Procedures for performing systematic reviews. Joint technical report, Department of Computer Science Keele University, United King and Empirical Sofware Engineering, National ICT Australia Ltd., Australia. 2004.

KITCHENHAM, Barbara; DYBA, Tore; JORGENSEN, Magne. Evidence-based Software Engineering. Proceedings of the 26th International Conference on Software Engineering, (p. 273-281). Washington DC. 2004.

KITCHENHAM, Barbara; PICKARD, Lesley M.; PFLEEGER, Shari Lawrence. Case studies for method and tool evaluation. IEEE Software. 1995.

KITCHENHAM, Barbara. The Problem with Function Points. IEEE Software. 1997.

KITCHENHAM, Barbara; BUDGEN, David; BRERETON, Pearl. Evidence based software engineering ans systematic reviews. CRC Press. 2015.

KUHRMANN, Et. al. Hybrid Software and System Development in Practice: Waterfall, Scrum, and Beyond. International Conference on Software and System Processes. Paris. 2017.

KUHRMANN, Marco; Et. al. Hybrid Software Development Approaches in Practice - A European Perspective. IEEE Computer Society. 2019.

LANDAETA, Rafael E.; VISCARDI, Stacia; TOLK Aandreas. Strategic Management of Scrum Projects: An Organizational Learning Perspective. First International Technology Management Conference. San Jose, CA, USA. 2011.

LARMAN, Craig; BASILI, Victor. Iterative and Incremental Development: A Brief History. IEEE Computer Society. 2003.

LAVAZZA, Luigi; LENARDUZZI, Valentina; TAIBI, Davide. Towards Component-Aware Function Point Measurement. Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement. 2016.

LAVAZZA, Luigi; DEL BIANCO, Vieri; GARAVAGLIA, Carla. Model-based Functional Size Measurement. Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM. 2008.

LI, Jinhua; MA, Zhibing; DONG, Huangzhen. Monitoring software projects with earned value analysis and use case point. Seventh IEEE/ACIS International Conference on Computer and Information Science. 2008.

LIND, Kenneth; HELDAL, Rogardt. On the relationship between functional size and software code size. Workshop on Emerging Trends in Software Metrics – WETSoM. Cape Town, South Africa. 2010.

LINDVALL, Mikael; Et. al. Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile, (p. 197-207). 2002.

LÓPEZ-MARTÍNEZ, Janeth; Et. al. Analysis of Planning Poker Factors between University and Enterprise. 5th International Conference in Software Engineering Research and Innovation. 2017.

LUMA-OSMANI, Shkurte; ARIFI, Gjulie; IDRIZI, Florim. Choosing the Most Suitable Model for Developing a Software. Sixth International Conference on Computational Intelligence, Communication Systems and Networks. 2014.

MELO, Ismael; CAROLI Paulo. Enxugando a Máquina. Lean MVP e Pontos de Função. 2017.

MELLOR, Stephen J. Adapting Agile Approaches to Your Project Needs. IEEE Software. 2005. Sixth International Conference on Computational Intelligence, Communication Systems and Networks. 2014

NAGAPPAN, Nachiappan; MURPHY, Brendan; BASILI, Victor. The Influence of Organizational Structure on Software Quality. ACM/IEEE 30th International Conference on Software Engineering, ICSE. Leipzig, Germany. 2008.

OFORI, D. F. Project Management Practices and Critical Success Factors–A Developing Country Perspective. International Journal of Business and Management. 2013.

PAASIVAARA, Maria. Adopting SAFe to Scale Agile in a Globally Distributed Organization. IEEE 12th International Conference on Global Software Engineering (ICGSE). Buenos Aires, Argentina. 2017.

PAPADOPOULOSA, Georgios. Moving from traditional to agile software development. International Conference on Strategic Innovative Marketing, IC-SIM 2014. Madrid, Spain. 2014.

PROJECT MANAGEMENT INSTITUTE. A Guide to the Project Management Body of Knowledge (PMBOK Guide). Fifth Edition 2013.

PROJECT MANAGEMENT INSTITUTE. 9th Global Project Management Survey. Newtown Square, USA. 2017.

RAHMANIAN, Mojdeh. A Comparative Study on Hybrid IT Project Managment. International Journal of Computer and Information Technology. 2014.

RAJLICH, Vaclav. Changing the Paradigm of Software Engineering. Communications of the ACM. 2006.

ROYCE, Winston. Managing the Development of Large Software Systems: Concepts and Techniques. IEEE CS Press, (p 328-339). 1970.

RUPARELIA, Nayan B. Software Development Lifecycle Models. ACM SIGSOFT Software Engineering Notes. 2010.

RUSSO, Daniel; Et. al. Contracting Agile Developments for Mission Critical Systems in the Public Sector. ACM/IEEE 40th International Conference on Software Engineering: Software Engineering in Society. 2018.

SANTANA, Celio; Et. al. Using Function Points in Agile Projects. Agile Processes in Software Engineering and Extreme Programming: 12th International Conference, XP 2011, Madrid, Spain, May 10-13, 2011. Proceedings (pp.176-191).

SBOK. A Guide to the Scrum Body of Knowledge (SBOK™GUIDE). 3rd Edition. 2017.

SHAH, Unnati. An Excursion to Software Development Life Cycle Models. ACM SIGSOFT Software Engineering Notes. 2016.

SENAPATHI, Mali; DRURY-GROGAN, Meghann. Refining a model for sustained usage of agile methodologies. The Journal of Systems and Software, (p. 298-316). 2017.

SLETHOLT, Magnus; Et. al. What Do We Know about Scientific Software Development's Agile Practices? Computing in Science & Engineering (Copublished by the IEEE CS and the AIP). 2012.

SMITE, Darja; MOE, Nils B.; SABLIS, Aivars; WOHLIN, Claes. Software teams and their knowledge networks in large-scale software development. Information and Software Technology, (p. 71-86). 2016.

TANVEER, Binish. Hybrid Effort Estimation of Changes in Agile Software Development. Agile Processes in Software Engineering and Extreme Programming. 17th International Conference XP. 2016.

THEOCHARIS, Georgios; Et. al. Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices. PROFES 2015 Proceedings of the 16th International Conference on Product-Focused Software Process Improvement - Volume 9459, (p. 149-166). Bolzano, Italy. 2015.

USMAN, Muhammad; Et. al. Effort Estimation in Agile Software Development: A Systematic Literature Review. 10th International Conference on Predictive Models in Software Engineering, Italy. 2014.

VERGARA, Sylvia Constant. Métodos de Coleta de Dados no Campo. 2. ed. São Paulo: Atlas, 2013. ISBN 8522470537.

VESTUES, Kathrine. Planned Research: Scaling Agile Practices in Software Development. Agile Processes in Software Engineering and Extreme Programming. 17th International Conference XP. 2016.

VIJAYASARATHY, Leo; BUTLER, Charles. Choice of Software Development Methodologies - Do Organizational, Project, and Team Characteristics Matter? IEEE Computer Society, (p. 86 – 94). 2016.

YILMAZ, Murat; Et. al. An examination of personality traits and how they impact on software development teams. Information and Software Technology, (p. 101-122). 2016.

YOURDON, Edward. Modern Structured Analysis. London: Prentice-Hall International. 1989.

ZHI-GEN, Hu; YUAN, Quan; ZHANG, Xi. Research on Agile Project Management with Scrum method. IITA International Conference on Services Science, Management and Engineering. 2009.

WAARDENBURG, Guus Van; VLIET, Hans Vliet. When agile meets the enterprise. Information and Software Technology. 2013.

WANG, Qian; SONG, Zhen-hua. Research on Multi-Perspective Communication Management of Software Development Project Based on Theory of Project Management. 2nd International Conference on Signal Processing Systems (ICSPS). 2010.

WEST, Dave. Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today. Forrester Research, Inc. 2011.

WILKIE, Frederick; Et. al. The value of software sizing. Information and Software Technology, (p. 654-667). 2006.

WOHLIN, Claes; Et. al. Experimentation in software engineering. 2012.

APÊNDICE A

LISTA DOS 18 ARTIGOS INCLUÍDOS NO MAPEAMENTO SISTEMÁTICO

An Algorithmic-Based Change Effort Estimation Model for Software Development. S. Basri; N. Kama; H. M. Sarkan; S. Adli; F. Haneem. 23rd Asia-Pacific Software Engineering Conference (APSEC). 2016.

An Exploratory Study on Functional Size Measurement Based on Code. H. Huijgens; M. Bruntink; A. van Deursen; T. van der Storm; F. Vogelezang. IEEE/ACM International Conference on Software and System Processes (ICSSP). 2016.

Applying EVM in a Software Company: Benefits and Difficulties. P. Efe; O. Demirors. 39th Euromicro Conference on Software Engineering and Advanced Applications. 2013.

Comparison between traditional plan-based and agile software processes according to team size & project domain. N. Keshta; Y. Morgan. 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). 2017.

Contracting Agile Developments for Mission Critical Systems in the Public Sector. Russo D., Taccogna G., Ciancarini P., Messina A., Succi G. ACM/IEEE 40th International Conference on Software Engineering: Software Engineering in Society. 2018.

Cross-factor analysis of software engineering practices versus practitioner demographics: An exploratory study in Turkey. Vahid Garousi, Ahmet Coskunçay, Onur Demirörs, Ali Yazici. The Journal of Systems and Software. 2015.

Does Use of Development Model Affect Estimation Accuracy and Bias?. Lecture Notes in Computer Science 3009. Kjetil Molokken, Anette C. Lien, Magne Jorgensen, Sinan S. Tanilkan, Hans Gallis, Siw E. Hove. 2004.

Establishing the Agile PMO: Managing variability across Projects and Portfolios. A. Tengshe; S. Noble. Agile 2007 (AGILE 2007). 2007.

Estimating, planning and managing Agile Web development projects under a value-based perspective. C.J. Torrecilla-Salinas, J. Sedeño, M.J. Escalona, M. Mejías. The Journal of Systems and Software. 2014.

Impact of development methodology on cost & risk for development projects.Chauhan B. D., Rana A., Sharma N. K. 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO). 2017.

Impact of incorrect and new requirements on waterfall software project outcomes. Kaushal Chari, Manish Agrawal. Empirical Software Engineering. 2018.

Kanban Pull and Flow - A transparent workflow for improved quality and productivity in software developmet. H. K. Raju; Y. T. Krishnegowda. Fifth International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom 2013). 2013.

Quantitatively measuring a large-scale agile transformation. Marta Olszewska, Jeanette Heidenberg, Max Weijola, Kirsi Mikkonen, Ivan Porres. The Journal of Systems and Software. 2015.

Scrum based scaling using agile method to test software projects and its future solutions using in artificial neural networks. Hariharn S., Rengrajan A., Prem Kumar R. International Journal of Innovative Technology and Exploring Engineering (IJITEE). 2019.

Software process improvement through the Lean Measurement (SPI-LEAM) method. Kai Petersen, Claes Wohlin. The Journal of Systems and Software. 2009.

Traditional vs Agile development a comparison using chaos theory. D. M. Shawky. 9th International Conference on Software Paradigm Trends (ICSOFT-PT). 2014.

The value of software sizing. F.G. Wilkie, I.R. McChesney, P. Morrow, C. Tuxworth, N.G. Lester. Information and Software Technology. 2011.

When Global Process Fails: A Grounded Theory Study of a Case from Agile Engagement to Compulsive Outsourcing. Jan Pries-Heje, Magnus Hansen, and Sofia Bergbäck Knudsen. E-Services and Global Processes. 2010.

APÊNDICE B

LISTA DE TODOS OS ARTIGOS DO MAPEAMENTO SISTEMÁTICO

Repositório ACM Digital Library (2 Artigos)

1. An Approach for Effort Estimation in Incremental Software Development Using Cosmic Function Points. Freddy Paz;Claudia Zapata;José Antonio Pow-Sang. 2014.
2. Automating the Estimation of Project Size from Software Design Tools Using Modified Function Points. Jason Ceddia; Martin Dick. 2004.

Repositório IEEE Xplore Digital Library (85 Artigos)

1. A Cost Model for Software Maintenance & Evolution. 20th IEEE International Conference on Software Maintenance (ICSM'04). Harry M. Sneed. 2004.
2. A Model and System for Applying Lean Six Sigma to Agile Software Development Using Hybrid Simulation. IEEE International Technology Management Conference. Kamran Ghane. 2014.
3. A QFD based model integration in Agile software development. 12th Iberian Conference on Information Systems and Technologies (CISTI). Andreea Ionica; Monica Leba; Raluca Dovleac. 2017.
4. A survey of software testing practices in Alberta. IEEE Journals & Magazines. Adam M. Geras; M.R. Smith; J. Miller. 2004.
5. Achievements and Challenges in Cocomo-Based Software Resource Estimation. IEEE Software. Barry W. Boehm ; Ricardo Valerdi. 2008.
6. An Algorithmic-Based Change Effort Estimation Model for Software Development. 23rd Asia-Pacific Software Engineering Conference. Sufyan Basri; Nazri Kama; Haslina Md Sarkan2; Saiful Adli; Faizura Haneem. 2016.
7. An Autonomic Framework for Quantitative Software Process Improvement. EEE International Conference on Industrial Informatics (INDIN). Huaglory Tianfield. 2003.
8. An Empirical Analysis of Cost Estimation Models on Undergraduate Projects Using COCOMO II. International Conference on Smart Computing and Electronic Enterprise (ICSCEE). Faiza Tahir; Mahum Adil. 2018.
9. An Exploratory Study on Functional Size Measurement Based on Code. IEEE/ACM International Conference on Software and System Processes. Hennie Huijgens; Magiel Bruntink; Arie van Deursen; Tijs van der Storm; Frank Vogelezang. 2016.
10. An integrated approach to security in software development methodologies. Canadian Conference on Electrical and Computer Engineering. Abhay Raman; Steven Muegge. 2008.
11. Appendix A: Acronyms and Glossary of Key Terms. Book Chapter: Software War Stories: Case Studies in Software Management. Donald J. Reifer. 2014.
12. Application of agile method in the enterprise website backstage management system: Practices for extreme programming. 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet). Linghui Liu ; Yao Lu. 2012.
13. Applying EVM in a Software Company: Benefits and Difficulties. 39th Euromicro Conference Series on Software Engineering and Advanced Applications. Pınar Efe; Onur Demirörs. 2013.
14. Architecture of DSM. Book Chapter: Domain-Specific Modeling: Enabling Full Code Generation. Steven Kelly; Juha-Pekka Tolvanen. 2007.

15. Case Study: Project Management Using Cross Project Software Reliability Growth Model. IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). Kiyoshi Honda; Nobuhiro Nakamura; Hironori Washizaki; Yoshiaki Fukazawa. 2016.

16. Comparative Analysis of Requirement Change Management Challenges between In-house and Global Software Development: Findings of Literature and Industry Survey. IEEE Access. Sajid Anwer; Lian Wen; Zhe Wang; Sajjad Mahmood. 2019.

17. Comparison Between Traditional Plan-Based and Agile Software Processes According to Team Size & Project Domain (A systematic literature review). 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON. Nesma Keshta; Yasser Morgan. 2017.

18. Contracting Agile Developments for Mission Critical Systems in the Public Sector. IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS). Daniel Russo; Gerolamo Taccogna; Paolo Ciancarini; Angelo Messina; Giancarlo Succi. 2018.

19. Cost and value analysis of software development method focused on individual function. Eighth International Conference on Intelligent Computing and Information Systems (ICICIS). Atsushi Shimoda; Taro Yabuki. 2017.

20. Distributed Scrum: Agile Project Management with Outsourced Development Teams. 40th Hawaii International Conference on System Sciences. Jeff Sutherland; Anton Viktorov; Jack Blount; Nikolai Puntikov. 2007.

21. Early Phase Cost Models for Agile Software Processes in the US DoD. ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. Wilson Rosa; Raymond Madachy; Bradford Clark. 2017.

22. Early Web size measures and effort prediction for Web costimation. Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry. E. Mendes; N. Mosley; S. Counsell. 2004.

23. Earned Scope Management: A Case of Study of Scope Performance Using COSMIC (ISO 19761) with a Real Project. Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement. Francisco Valdés-Souto. 2016.

24. Effect of Staffing Pattern on Software Project: An Empirical Analysis. 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM. Fei Dong; Mingshu Li; Juan Li; Ye Yang; Qing Wang. 2009.

25. Embedded Agile Project by the Numbers with Newbies. AGILE Conference (AGILE'06). Nancy Van Schooenderwoert. 2006.

26. Embedded Software: Facts, Figures, and Future. IEEE Journals & Magazines. Christof Ebert; Capers Jones. 2009.

27. Empirical-Based Extension of the COSMIC FP Method. Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement. Reiner Dumke; Robert Neumann; Andreas Schmietendorf; Cornelius Wille. 2014.

28. Engineering and Economics Concepts for Understanding Software Process Performance. Book Chapter - Software Project Estimation: The Fundamentals for Providing High Quality Information to Decision Makers. Alain Abran. 2015.

29. EQUITY 2007 Report. IEEE International Conference on Exploring Quantifiable IT Yields. Gijs Hillenius; Laurenz Eveleens; Peter Kampstra; Łukasz Kwiatkowski; Sander Sman. 2007.

30. Establishing the Agile PMO: Managing variability across Projects and Portfolios. AGILE 2007. Ash Tengshe; Scott Noble. 2007.

31. Estimate of the Appropriate Iteration Length in Agile Development by Conducting Simulation. Agile Conference. Ryushi Shiohama; Hironori Washizaki; Shin Kuboaki; Kazunori Sakamoto; Yoshiaki Fukazawa. 2012.

32. Estimating Effort. Book Chapter - Software Measurement and Estimation: A Practical Approach. Linda M. Laird; M. Carol Brennan. 2006.

33. Experiences with an Industrial Long-Term Reengineering Project. 11th Working Conference on Reverse Engineering (WCRE'04). Ralf Kollmann. 2004.

34. Extended Planning and Infrastructure. Book Chapter - Automated Defect Prevention: Best Practices in Software Management. Dorota Huizinga; Adam Kolawa. 2007.

35. Four Basics That Work. Book Chapter - The Software Project Manager's Handbook: Principles That Work at Work. Dwayne Phillips. 2004.

36. FPA and Quality Metrics in Contracts. Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement. Harold van Heeringen Sogeti; Hans Kuijpers; Rini Scholten; Frans Schoot Uiterkamp; Dirk Müller; Jolijn Onvlee; Hans Bernink; Marcel Pereboom. 2014.

37. Frontmatter. Book Chapter - Trustworthy Systems Through Quantitative Software Engineering. Lawrence Bernstein; C. M. Yuhas. 2005.

38. Fully Distributed Scrum: Replicating Local Productivity and Quality with Offshore Teams. 42nd Hawaii International Conference on System Sciences. Jeff Sutherland; Guido Schoonheim; Maurits Rijk. 2009.

39. Glossary of Terms. Book Chapter - Managing and Leading Software Projects. Richard E. Fairley. 2009.

40. Glossary. Book Chapter - Automated Defect Prevention: Best Practices in Software Management. Dorota Huizinga; Adam Kolawa. 2007.

41. Identifying and Managing Risk. Book Chapter - Trustworthy Systems Through Quantitative Software Engineering. Lawrence Bernstein; C. M. Yuhas. 2005.

42. IEEE Approved Draft Recommended Practice on Software Reliability. 2016.

43. IEEE Draft International Standard - Systems and Software Engineering - Life Cycle Management - Part 5: Software Development Planning. 2015.

44. IEEE Draft Recommended Practice on Software Reliability. 2016.

45. IEEE Recommended Practice on Software Reliability. 2017.

46. Impact of development methodology on cost & risk for development projects. 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO). Bishan Dayal Chauhan; Ajay Rana; Neeraj Kumar Sharma. 2017.

47. Impact of Software Resource Estimation Research on Practice: A Preliminary Report on Achievements, Synergies, and Challenges. 33rd International Conference on Software Engineering. Barry Boehm; Ricardo Valerdi. 2011.

48. Improving the Quality of Software Development Process by Introducing a New Methodology. IEEE Access. Muhammad Azeem Akbar; Jun Sang; Arif Ali Khan; Fazal-E-Amin; Nasrullah; Muhammad Shafiq; Shahid Hussain; Haibo Hu; Manzoor Elahi; Hong Xiang. 2018.

49. Index. Book Chapter - Automated Defect Prevention: Best Practices in Software Management. Dorota Huizinga; Adam Kolawa. 2007.

50. Index. Book Chapter - Trustworthy Systems Through Quantitative Software Engineering. Lawrence Bernstein; C. M. Yuhas. 2005.

51. Index. Book Chapter - Real-Time Systems Design and Analysis: Tools for the Practitioner. Phillip A. Laplante; Seppo J. Ovaska. 2012.

52. Index. Book Chapter - Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research. Richard W. Selby. 2007.

53. Introduction and Background. Book Chapter - Software Process Dynamics. Raymond J. Madachy. 2008.

54. ISO/IEC/IEEE Approved Draft International Standard - Systems and Software Engineering – Vocabulary - IEEE Standards. 2017.

55. ISO/IEC/IEEE Draft Systems and Software Engineering – Vocabulary - IEEE Standards. 2016.

56. ISO/IEC/IEEE Draft Systems and Software Engineering – Vocabulary - IEEE Standards. 2017.

57. ISO/IEC/IEEE International Standard - Systems and Software Engineering - Life Cycle Management - Part 5: Software Development Planning - IEEE Standards. 2017.

58. ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary - IEEE Standards. 2017.

59. Kanban Pull and Flow - A transparent workflow for improved quality and productivity in software development. Fifth International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom). H. K. Raju; Y. T. Krishnegowda. 2013.

60. Life Cycle Models. Book Chapter - Software Management. Donald J. Reifer ; Barry Boehm. 2007.

61. Magnitude economic effects of requirements in the development process - A simulation study of workload behavior. Computing Conference. David Kuhlen; Andreas Speck. 2017.

62. Maximizing Software Production & Quality with Minimum Staff using Clarity™ - A Real-World Case Study. IEEE Aerospace Conference. Rob Thorpe. 2019.

63. Measurement of Software Size: Advances Made by the COSMIC Community. Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement. Charles Symons; Alain Abran; Christof Ebert; Frank Vogelezang. 2016.

64. Measuring and Controlling Work Products. Book Chapter - Managing and Leading Software Projects. Richard E. Fairley. 2009.

65. Measuring Best-in-Class Software Releases. Conference of the 23nd International Workshop on Software Measurement (IWSM) and the Eighth International Conference on Software Process and Product Measurement (Mensura). Hennie Huijgens; Rini van Solingen. 2013.

66. Measuring dimensions of software engineering projects´success in Academic context. Federated Conference on Computer Science and Information Systems (FedCSIS). Rafał Włodarski; Aneta Poniszewska-Marańda. 2017.

67. Monitoring Software Projects with Earned Value Analysis and Use Case Point. Seventh IEEE/ACIS International Conference on Computer and Information Science (ICIS). Jinhua Li; Zhibing Ma; Huanzhen Dong. 2008.

68. New direction in project management success: Base on smart methodology selection. International Symposium on Information Technology. Iman Attarzadeh; Siew Hock Ow. 2008.

69. People Applications. Book Chapter - Software Process Dynamics. Autor: Raymond J. Madachy. 2008.

70. Processes for software development within the Public Administration. ICSE Workshop on Software Quality.Andrea Bei; Maurizio Lancia; Flavio Lombardi; Roberto Puccinelli. 2009.

71. Resource/Schedule/Content Test Planning Model. IEEE International Conference on Software Quality, Reliability and Security (QRS). Pete Rotella. 2016.

72. Software Estimating. Book Chapter - Software Management. Donald J. Reifer; Barry Boehm. 2007.

73. Software Industry Performance: What You Measure Is What You Get. IEEE Software. Charles Symons. 2010.

74. (CSDP) Software Engineering Process. Course – IEEE. Rob Oshana. 2011.

75. (CSDP) Software Testing. Course – IEEE. Linda Shafer. 2011.

76. Stability in Software Engineering: Survey of the State-of-the-Art and Research Directions. IEEE Transactions on Software Engineering. Maria Salama; Rami Bahsoon; Patricia Lago. 2019.

77. Success Factors in Managing Legacy System Evolution: A Case Study. IEEE/ACM International Conference on Software and System Processes (ICSSP). Hennie Huijgens; Arie van Deursen; Rini van Solingen. 2016.

78. Systems Engineering: Building Successful Systems. Book. Howard Eisner. 2011.

79. The Influence of COCOMO on Software Engineering Education and Training. 19th Conference on Software Engineering Education & Training (CSEET'06). Richard E. Fairley. 2006.

80. The Modeling Process with System Dynamics. Book Chapter - Software Process Dynamics. Raymond J. Madachy. 2008.

81. Think Like An Engineer - Especially for Software. Book Chapter - Trustworthy Systems Through Quantitative Software Engineering. Lawrence Bernstein; C. M. Yuhas. 2005.

82. Towards an Adaptable Large Scale Project Execution Monitoring. 5th International Symposium on Applied Computational Intelligence and Informatics. Ciprian Stanciu; Dacian Tudor; Vladimir-Ioan Cretu. 2009.

83. Traditional vs Agile Development a Comparison Using Chaos Theory. 9th International Conference on Software Paradigm Trends (ICSOFT-PT). Doaa M. Shawky. 2014.

84. Using Analytics to Guide Improvement During an Agile/DevOps Transformation. IEEE Software. Barry Snyder; Bill Curtis. 2018.

85. Using Static and Dynamic Impact Analysis for Effort Estimation. IET Software. Sufyan Basri; Nazri Kama; Saiful Adli; Faizura Haneem. 2016.

Repositório Scopus (4 Artigos)

1. An approach for effort estimation in incremental software development using cosmic function points. International Symposium on Empirical Software Engineering and Measurement. Paz, F., Zapata, C., Pow-Sang, J.A. 2014.

2. Effort Estimation in Incremental Software Development Projects Using Function Points. Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity. José Antonio Pow-Sang; Ricardo Imbert. 2012.

3. Scrum based scaling using agile method to test software projects and its future solutions using in artificial neural networks. International Journal of Innovative Technology and Exploring Engineering. Hariharan, S., Rengarajan, A., Prem Kumar, R. 2019.

4. Agile Processes in Software Engineering and Extreme Programming - 12th International Conference, XP 2011, Proceedings. Lecture Notes in Business Information Processing. 2011.

Repositório Science Direct (21 Artigos)

1. Agile programming – Introduction and current legal challenges. Computer Law & Security Review. Thomas Hoeren; Stefan Pinelli. 2018.

2. A systematic literature review on crowdsourcing in software engineering. Journal of Systems and Software. Aslı Sarı; Ayşe Tosun; Gülfem Işıklar; Alptekin. 2019.

3. Cross-Factor Analysis of Software Engineering Practices Versus Practitioner Demographics: An Exploratory Study in Turkey. The Journal of Systems and Software. Vahid Garousi; Ahmet Coskunçay; Onur Demirörs; Ali Yazici. 2016.
4. Quantitatively Measuring a Large-Scale Agile Transformation. The Journal of Systems and Software. Marta Olszewska; Jeanette Heidenberg; Max Weijola; Kirsi Mikkonen; Ivan Porres. 2016.
5. Estimating, Planning and Managing Agile Web Development Projects Under a Value-Based Perspective. Information and Software Technology. C.J. Torrecilla-Salinas; J. Sedeño; M.J. Escalona; M. Mejías. 2015.
6. Automated COSMIC Function Point Measurement Using a Requirements Engineering Ontology. Information and Software Technology. Selami Bagriyanika; Adem Karahoca. 2016.
7. The Value of Software Sizing. Information and Software Technology. F.G. Wilkie; I.R. McChesney; P. Morrow; C. Tuxworth; N.G. Lester. 2011.
8. History of Computers, Electronic Commerce and Agile Methods. Advances in Computers. David F. Rico; Hasan H. Sayani; Ralph F. Field. 2008.
9. The Situational Factors that Affect the Software Development Process: Towards a Comprehensive Reference Framework. Information and Software Technology. Paul Clarke; Rory V. O'Connor. 2012.
10. Software Process Improvement Through the Lean Measurement (SPI-LEAM) Method. The Journal of Systems and Software. Kai Petersen; Claes Wohlin. 2010.
11. Standardisation: A Tool for Addressing Market Failure Within the Software Industry. Computer Law & Security Review. Roksana Moore. 2013.
12. Measuring and Predicting Software Productivity: A Systematic Map and Review. Information and Software Technology. Kai Petersen. 2011.
13. Introduction to MBASE (Model-Based (System) Architecting and Software Engineering). Advances in Computers. David Klappholz; Daniel Port. 2004.
14. The influence of COCOMO on software engineering education and training. 19th Conference on Software Engineering Education & Training (CSEET'06). Richard E. Fairley. 2006.
15. Chapter 6 Factors Influencing Software Development Productivity—State-of-the-Art and Industrial Experiences. Advances in Computers. Adam Trendowicz; Jürgen Münch. 2009.
16. A COSMIC function points based test effort estimation model for mobile applications. Journal of King Saud University - Computer and Information Sciences. Anureet Kaur; Kulwant Kaur. 2019.
17. Towards functional change decision support based on COSMIC FSM method. Information and Software Technology. Mariem Haoues; Asma Sellami; Hanêne Ben-Abdallah. 2019.
18. Subject Index. Book Chapter - Building a Scalable Data Warehouse with Data Vault 2.0. 2016.
19. Subject Index. Book Chapter - Advances in Computers. 2006.
20. Investigating Web Size Metrics for Early Web Cost Estimation. The Journal of Systems and Software. Emilia Mendes; Nile Mosley; Steve Counsell. 2005.
21. User Knowledge Transformation Through Design: A Historical Materialism Perspective. Information and Organization. Yutaka Yamauchi. 2014.

Repositório Springer Link (70 Artigos)

1. Improved Agile: A Customized Scrum Process for Project Management in Defense and Security. Software Project Management for Distributed Computing. Luigi Benedicenti; Paolo Ciancarini; Franco Cotugno; Angelo Messina; Alberto Sillitti; Giancarlo Succi. 2017.
2. Develop - Agile Product Development. Book Chapter – ISBN: 978-1-4842-1068-0. Tathagat Varma. 2015.
3. Digital Project Execution. Book Chapter - Complete Guide to Digital Project Management. Shailesh Kumar Shivakumar. 2018.
4. Contracting Agile Developments for the Public Sector: The Italian Case. Proceedings of 5th International Conference in Software Engineering for Defence Applications. Daniel Russo; Gerolamo Taccogna; Paolo Ciancarini; Angelo Messina; Giancarlo Succi. 2016.
5. Product Backlog Rating: A Case Study on Measuring Test Quality in Scrum. Innovations in Systems and Software Engineering. Imrul Kayes; Mithun Sarker; Jacob Chakareski. 2016.
6. Distributed Agile: Project Management in a Global Environment. Empirical Software Engineering. Seiyoung Lee; Hwan-Seung Yong. 2010.
7. A Case Study to Enable and Monitor Real IT Companies Migrating from Waterfall to Agile. International Conference on Computational Science and Its Applications. Antonio Capodieci; Luca Mainetti; Luigi Manco. 2014.
8. Aspects of software quality applied to the process of agile software development: a systematic literature review. International Journal of System Assurance Engineering and Management. Gloria Arcos-Medina; David Mauricio. 2019.
9. Distributed Scrum Process Guide. Book Chapter - Collaboration in Outsourcing. Koen Bos. 2012.
10. Measuring Success in an Agile World: Agile EVMS. Book Chapter - Agile Project Management: Managing for Success. James A. Crowder; Shelli Friess. 2015.
11. Software Reliability Analysis. Book Chapter - Next Generation and Advanced Network Reliability Analysis. Syed Riffat Ali. 2019.
12. Agile Project Controlling. International Conference on Extreme Programming and Agile Processes in Software Engineering. Stefan Roock; Henning Wolf. 2004.
13. When Global Process Fails: A Grounded Theory Study of a Case from Agile Engagement to Compulsive Outsourcing. International Conference on Global Information Systems Processes. Jan Pries-Heje; Magnus Hansen; Sofia Bergbäck Knudsen. 2010.
14. Effort Estimation in Incremental Software Development Projects Using Function Points. Book Chapter - Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity. José Antonio Pow-Sang; Ricardo Imbert. 2012.
15. Function Point Analysis Using NESMA: Simplifying the Sizing Without Simplifying the Size. Software Quality Journal. P. MorrowEmail; F. G. Wilkie; I. R. McChesney. 2014.
16. Preparing the Project. Book Chapter - Agile Development in the Real World. Alan Cline. 2015.
17. Impact of Incorrect and New Requirements on Waterfall Software Project Outcomes. Empirical Software Engineering. Kaushal Chari; Manish Agrawal. 2018.
18. Software Development Productivity in Different Sourcing Arrangements. Book Chapter - Entrepreneurship in Technology for ASEAN. Niharika Dayyala; Kallol Bagchi; Purnendu Mandal. 2016.
19. Mutual Dependency of Function Points and Scope Creep towards the Success of Software Projects: An Investigation. ICT and Critical Infrastructure: Proceedings of the 48th Annual Convention of Computer Society of India. K. Lakshmi Madhuri; V. Suma. 2014.

20. Organizational Maturity: The Elephant Affecting Productivity. Book Chapter - Rethinking Productivity in Software Engineering. Bill Curtis. 2019.
21. Software Vendor Strategies. Book Chapter - The Software Industry. Peter Buxmann; Heiner Diefenbach; Thomas Hess. 2012.
22. State of Practice in Requirements Engineering: Contemporary Data. Innovations in Systems and Software Engineering. Mohamad KassabEmail; Colin Neill; Phillip Laplante. 2014.
23. From Service Innovation to Service Engineering. Book Chapter - International Conference on Cloud Computing and Services Science. Wil Janssen; Marc Lankhorst; Timber Haaker; Henny de Vos. 2011.
24. Digital Project Estimation and Pricing. Book Chapter - Complete Guide to Digital Project Management. Shailesh Kumar Shivakumar. 2018.
25. Proposed Software Development Model for Small Organization and Its Explanation. International Conference on Computer Science and Information Technology. Vinish Kumar; Sachin Gupta. 2012.
26. Agile Processes and Practices. Book Chapter - Business in the Digital Economy. Alan Brown; Jerry Fishenden; Mark Thompson. 2014.
27. Product- and Process- Metrics. Book Chapter - The IT Measurement Compendium. 2008.
28. Project Management. Book Chapter - Software Measurement. Christof Ebert; Reiner Dumke. 2007.
29. Improving Processes and Products. Book Chapter - Software Measurement. Christof Ebert; Reiner Dumke. 2007.
30. Applications of a Generic Work-Test-Rework Component for Software Process Simulation. International Conference on Software Process. Thomas Birkholzer; Dietmar Pfahl; Michael Schuster. 2010.
31. Towards the Measuring Criteria of IT Project Success in University Context. World Conference on Information Systems and Technologies. Rafa Wodarski; Aneta Poniszewska-Maranda. 2018.
32. A Machine Learning Based Model for Software Cost Estimation. Proceedings of SAI Intelligent Systems Conference. Muhammad Raza Tayyab; Muhammad Usman; Waseem Ahmad. 2016.
33. Prescriptive Process Models. Book Chapter - Software Process Definition and Management. Jürgen Münch; Ove Armbrust; Martin Kowalczyk; Martín Soto. 2012.
34. Initial framework for website design and development. International Journal of Information Technology. Jatinder Manhas. 2017.
35. Simplifying Maintenance by Application of Architectural Services. International Conference on Computational Science and Its Applications. Jaroslav Krá; Michal Žemlička. 2014.
36. Impact of Semantic Web and Cloud Computing Platform on Software Engineering. Book Chapter - Software Engineering Frameworks for the Cloud Computing Paradigm. Radha Guha. 2013.
37. Quality in Projects: Achieving Success through Standards and Transparency. Book Chapter - Management for Professionals. Stephan Kasulke; Jasmin Bensch. 2017.
38. Characteristics of Large-Scale Defense Projects and the Dominance of Software and Software Project Management. Book Chapter - Software Project Management for Distributed Computing. Kadir Alpaslan Demir. 2017.
39. Does Use of Development Model Affect Estimation Accuracy and Bias? International Conference on Product Focused Software Process Improvement. Kjetil Moløkken; Anette C. Lien; Magne Jørgensen; Sinan S. Tanilkan; Hans Gallis; Siw E. Hove. 2004.
40. Fundamentals of Software Testing. Book Chapter - Thinking-Driven Testing. Adam Roman. 2018.
41. Planning Activities and Predicting Costs. Book Chapter - Software Quality Assurance. Neil Walkinshaw. 2017.

42. Common Factors Influencing Software Project Effort. Book Chapter - Software Project Effort Estimation. Adam Trendowicz; Ross Jeffery. 2013.

43. Re-estimating software effort using prior phase efforts and data mining techniques. Innovations in Systems and Software Engineering. Pichai Jodpimai, Peraphon Sophatsathit; Chidchanok Lursinsap. 2018.

44. IT Portfolio and Project Management. IT Management. Lionel Pilorget; Thomas Schell. 2018.

45. Characterizing industry-academia collaborations in software engineering: evidence from 101 projects. Empirical Software Engineering. Autores: Vahid Garousi; Dietmar Pfahl; João M. Fernandes; Michael Felderer; Mika V. MÃntyla; David Shepherd; Andrea Arcuri; Ahmet Coskuncay; Bedir Tekinerdogan. 2019.

46. The Importance of Requirements. Book Chapter - Requirements Writing for System Engineering. George Koelsch. 2016.

47. Software Cost Estimation for User-Centered Mobile App Development in Large Enterprises. International Conference on Applied Human Factors and Ergonomics. Maria Lusky; Christoph Powilat; Stephan Böhm. 2017.

48. Systematic Development of Web Information Systems. Book Chapter - Design and Development of Web Information Systems. Klaus-Dieter; ScheweBernhard Thalheim. 2019.

49. Review of Current Software Estimation Techniques. Book Chapter - International Conference on Recent Developments in Science, Engineering and Technology. Bhawna Sharma; Rajendra Purohit. 2017.

50. Using Pattern-Based Architecture Reviews to Detect Quality Attribute Issues - An Exploratory Study. Transactions on Pattern Languages of Programming III. Neil B. Harrison; Paris Avgeriou. 2013.

51. The Name and Nature of Software Engineering. Book Chapter - Advances in Software Engineering. Michael Jackson. 2008.

52. Clustering of Near Duplicate Images Using Bundled Features. Cluster Computing. G. Kalaiarasi; K. K. Thyagharajan. 2017.

53. Hybrid Computational Models for Software Cost Prediction: An Approach Using Artificial Neural Networks and Genetic Algorithms. International Conference on Enterprise Information Systems. Efi Papatheocharous; Andreas S. Andreou. 2009.

54. Towards an Understanding of the Causes and Effects of Software Requirements Change: Two Case Studies. Requirements Engineering. Sharon McGee; Des Greer. 2012.

55. Empirical Laws and Rules of Thumb. Book Chapter - Software Measurement. Christof Ebert; Reiner Dumke. 2007.

56. SPDW+: A Seamless Approach for Capturing Quality Metrics in Software Development Environments. Software Quality Journal. Patrıcia Souza Silveira; Karin Becker; Duncan D. Ruiz. 2010.

57. An Empirical Analysis of Agent Oriented Methodologies by Exploiting the Lifecycle Phases of Each Methodology. Emerging ICT for Bridging the Future - Proceedings of the 49th Annual Convention of the Computer Society of India (CSI) Volume 1. E. Ajith Jubilson; P. M. Joe Prathap; V. Vimal Khanna; P. Dhanavanthini; W. Vinil Dani; A. Gunasekaran. 2015.

58. Software Architecture Quality of Service Analysis Based on Optimization Models. Book Chapter - Intelligent Decision Making in Quality Management. Pasqualina Potena; Ivica Crnkovic; Fabrizio Marinelli; Vittorio Cortellessa. 2015.

59. ProSim/RA - Software Process Simulation in Support of Risk Assessment. Book Chapter - Value-Based Software Engineering. Dietmar Pfahl. 2006.

60. Towards Improving Decision Making and Estimating the Value of Decisions in Value-Based Software Engineering: The VALUE Framework. Software Quality Journal. Emilia Mendes; Pilar Rodriguez; Vitor Freitas; Simon Baker;Mohamed Amine Atoui. 2018.
61. Grundlagen des Software Engineering. Book Chapter - Software Engineering. Reiner Dumke. 2003.
62. Software Cost Estimation Using Similarity Difference Between Software Attributes. Proceedings of the Second International Conference on Soft Computing for Problem Solving. Divya Kashyap; A. K. Misra. 2014.
63. Hybrid Modeling of Test-and-Fix Processes in Incremental Development. International Conference on Software Process. He Zhang; Ross Jeffery; Liming Zhu. 2008.
64. Web Information Systems Engineering. Book Chapter - Design and Development of Web Information Systems. HKlaus-Dieter Schewe; Bernhard Thalheim. 2019.
65. Introducing a Measurement Program. Book Chapter - Software Measurement. Christof Ebert; Reiner Dumke. 2007.
66. RAN-Map: A System for Automatically Producing API Layers from RDF Schemas. Journal of Ambient Intelligence and Humanized Computing. Daniele Toti; Marco Rinelli. 2017.
67. Summary of the 2006 Model Size Metrics Workshop. International Conference on Model Driven Engineering Languages and Systems. Frank Weil; Andrij Neczwid. 2007.
68. On the Effectiveness of Early Life Cycle Defect Prediction with Bayesian Nets. Empirical Software Engineering. Norman Fenton; Martin Neil; William Marsh; Peter Hearty; Łukasz Radliński; Paul Krause. 2008.
69. Making a Method Work for a Project Situation in the Context of CMM. International Conference on Product Focused Software Process Improvement. Mehmet N. Aydin; Frank Harmsen. 2002.
70. System Dynamics Applied to Project Management: A Survey. Encyclopedia of Complexity and Systems Science. Springer, Berlin, Heidelberg. David N. FordJ; ames M. Lyneis. 2019.

APÊNDICE C
QUESTIONÁRIO DE AVALIAÇÃO DA ABORDAGEM HIPA

TERMO DE CONSENTIMENTO LIVRE

Estou sendo convidado a participar de um estudo denominado "Abordagem de Pontos de Função em projetos que utilizam práticas ágeis", cujo objetivo é avaliar uma abordagem de desenvolvimento de sistemas.

Caso aceite participar desta pesquisa, responderei a um questionário de 10 questões sobre o assunto.

Estou ciente de que minha privacidade será respeitada, ou seja, meu nome, ou qualquer outro dado confidencial, será mantido em sigilo. A elaboração final dos dados será feita de maneira codificada, respeitando minha confidencialidade. Estou ciente de que posso me recusar a participar do estudo, ou retirar meu consentimento a qualquer momento, sem precisar justificar, nem sofrer qualquer dano.

Este estudo, é referente ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná, apoiando o mestrando Marcelo Mendes da Silva com as questões pertinentes a dissertação de mestrado.

Portanto, li este termo, e fui orientado quanto ao teor da pesquisa acima mencionada e tenho compreensão quanto ao objetivo do estudo do qual fui convidado a participar.

Concordo, voluntariamente em participar desta pesquisa, sabendo que não receberei nem pagarei nenhum valor econômico por minha participação.

Questão 1: Qual é sua formação acadêmica?

- Não graduado
- Graduado
- Pós-graduado (inclui especialização, mestrado e doutorado)

Questão 2: Quanto tempo de experiência tens em projeto de desenvolvimento de *software*?

- De 0 à 2 anos
- De 3 a 4 anos
- De 5 a 6 anos
- De 7 a 8 anos
- Acima de 8 anos

Questão 3: Qual seu cargo atual ou último cargo? Caso não exista a opção, escolha a que mais se aproxime.

- Analista de Requisitos/Negócios
- Analista/Programador de Sistemas

- Arquiteto de Sistemas
- Gerente/Coordenador de Projetos

Questão 4: Há quanto tempo você trabalha ou trabalhou com *softwares* estimados por pontos de função?

- Nunca trabalhei
- De 1 a 6 meses
- De 7 a 12 meses
- Acima de 12 meses

Questão 5: Possui conhecimento de práticas ágeis utilizadas durante a execução de um projeto de *software*? (Como exemplo, podemos citar algumas como "Planejamento de *Sprint*", "Reuniões diárias", "Revisões de *sprint*" e "Programação em par").

- Sim
- Não

Questão 6: Caso tenha conhecimento dessas práticas ágeis, e esteja em atividade, já utilizou ou utiliza essas práticas no ambiente de seu trabalho em projetos de desenvolvimento de *software*?

- Sim
- Não

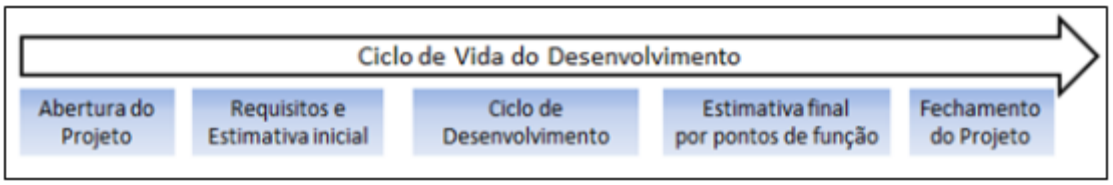
Questão 7: Possui conhecimento de modelos de ciclos de desenvolvimento, processos e frameworks indicados a seguir e utilizados em projetos de *software*?

- Modelo Cascata (*Waterfall*)
- RUP
- *Framework Scrum*
- XP
- Nenhuma das opções acima

Questão 8: Com relação ao seu conhecimento de ciclo de desenvolvimento de *software*, tendo como referência a abordagem de desenvolvimento a seguir, favor indicar as relevâncias das fases mostradas e produzir um comentário final.

Estas são as 5 etapas descritas a seguir para uma abordagem de desenvolvimento de *software* (Figura 34).

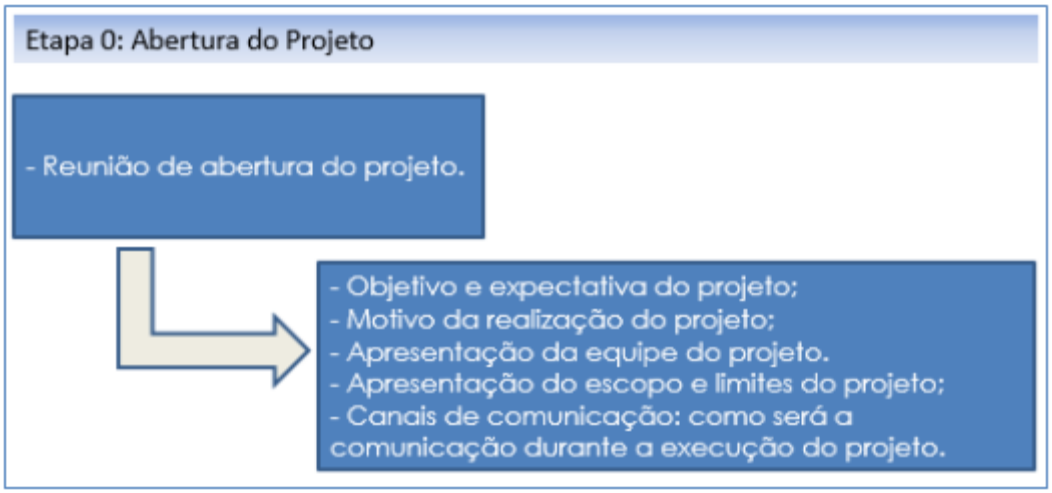
Figura 34: Questionário - Etapas da abordagem de desenvolvimento de software.



Fonte: Autoria própria.

A Etapa 0 (Figura 35) é o início da abordagem para alinhamento de expectativa entre os envolvidos no projeto (stakeholders). São apresentadas todas as pessoas envolvidas, seus papeis dentro do cenário de trabalho, os conhecimentos pertinentes ao projeto, o motivo para a realização, os benefícios e resultados esperados do trabalho. O escopo é detalhado a nível negocial e não tecnicamente. Eventuais restrições que podem limitar o software a ser entregue neste momento são indicadas. Também são indicados os canais de comunicação para todos os envolvidos, quem são as pessoas responsáveis por decisões e como devem ser escalonadas estas decisões de projeto, caso necessário.

Figura 35: Questionário - Abertura do Projeto.



Fonte: Autoria própria.

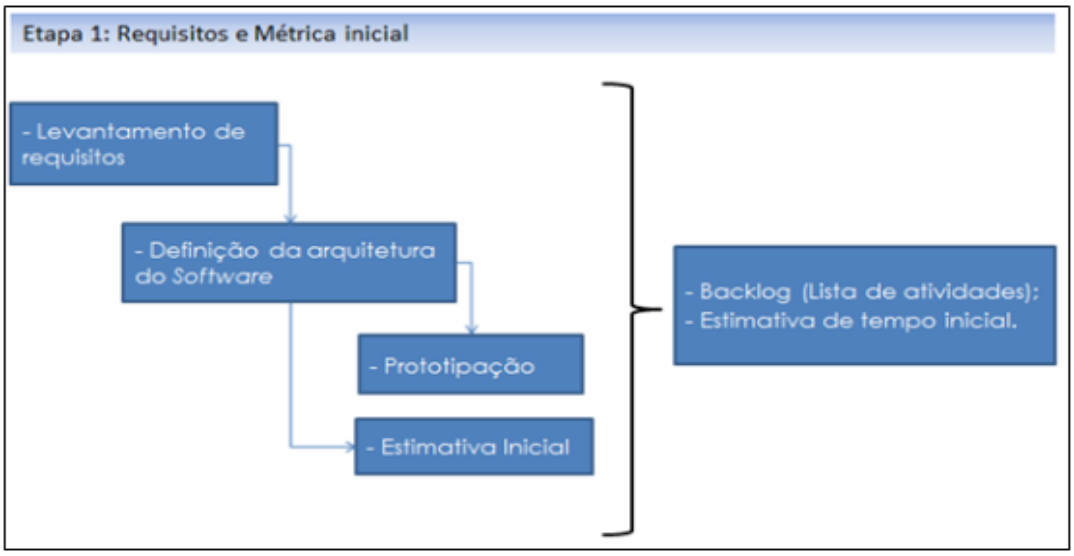
Na etapa 1 (Figura 36) temos um processo sequencial, sendo o escopo do projeto insumo para elaboração dos requisitos, definição da arquitetura do software e por fim estabelecimento da estimativa inicial do tamanho deste software, sendo esta estimativa feita por pontos de função.

A contagem inicial por pontos de função é usada para ter a estimativa de tempo a ser utilizada na etapa seguinte, sendo esta estimativa aprovada pelo cliente do projeto. A saída desta etapa é o *backlog* do projeto, que será utilizado na próxima etapa juntamente com a estimativa de tempo e tamanho do sistema.

Uma prototipação pode ser feita após a elaboração dos requisitos, fornecendo ao usuário uma previsão inicial das interfaces e navegabilidade do sistema, caso necessário, pois

nem todo sistema computacional possui interfaces.

Figura 36: Questionário - Requisitos e Métrica inicial.



Fonte: Autoria própria.

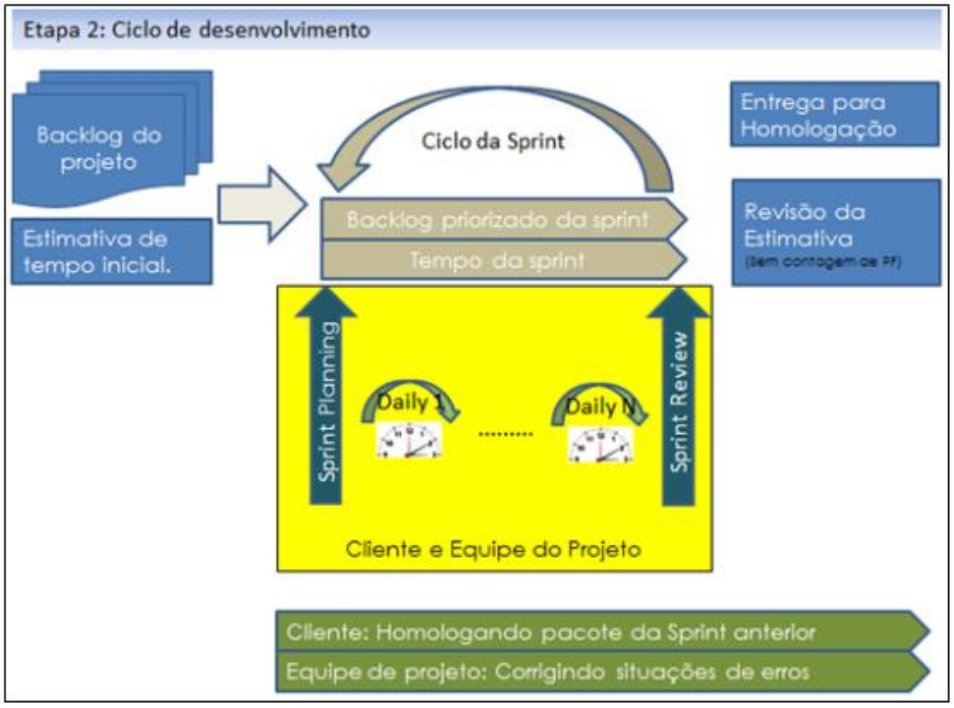
Na etapa 2 (Figura 37) temos a execução do projeto.

As informações de *backlog* do projeto e também a estimativa serão entradas nesta etapa para o planejamento da *sprint*.

A duração da *sprint* será definida conforme o *backlog* do projeto sendo indicado o máximo de quatro semanas e também dependendo da necessidade e priorização com o cliente. Ao final de cada *sprint* deve ser gerado um relatório de atividade executadas para acompanhamento gerencial do projeto e posterior apresentação destas informações em comitês de projetos fora do âmbito de execução do projeto.

Ao final do ciclo de todas as *sprints*, todo o *backlog* deverá ter sido trabalhado. Uma revisão da estimativa feita inicialmente deverá ser executada para eventuais correções dos requisitos trabalhados que foram consumidos durante as *sprints*.

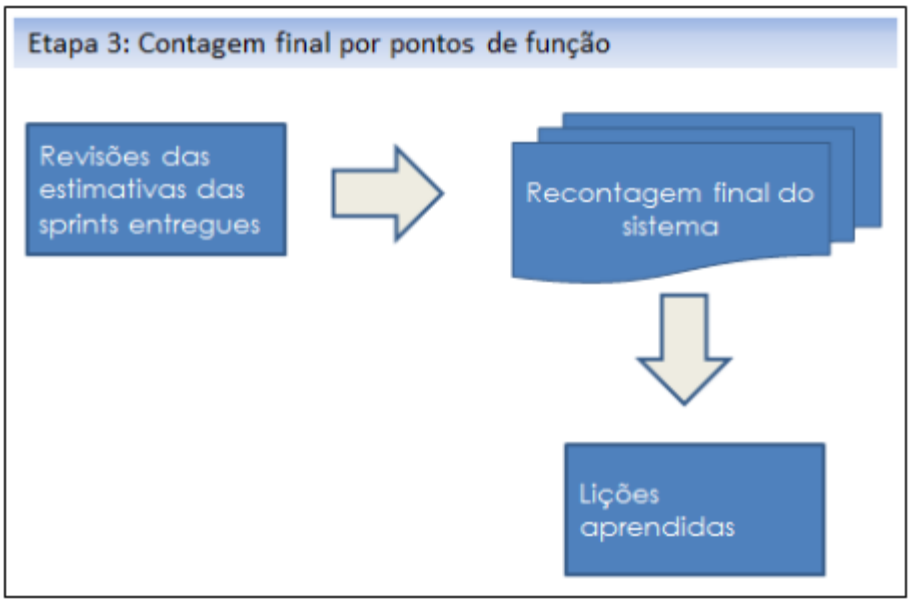
Figura 37: Questionário - Execução do projeto.



Fonte: Autoria própria.

Na etapa 3 (Figura 38), após a finalização de todo o *backlog* do projeto e entregas realizadas, devem estar disponíveis todas as informações necessárias para uma recontagem do projeto por pontos de função. Assim, no momento desta recontagem, com os requisitos atualizados, pode ser realizada correções acima ou abaixo da contagem inicial, corrigindo a primeira estimativa, caso necessário. Esta correção tem impacto no custo do projeto e também em contagens futuras, pois o conhecimento gerado no projeto servirá como lição apreendida para projetos futuros.

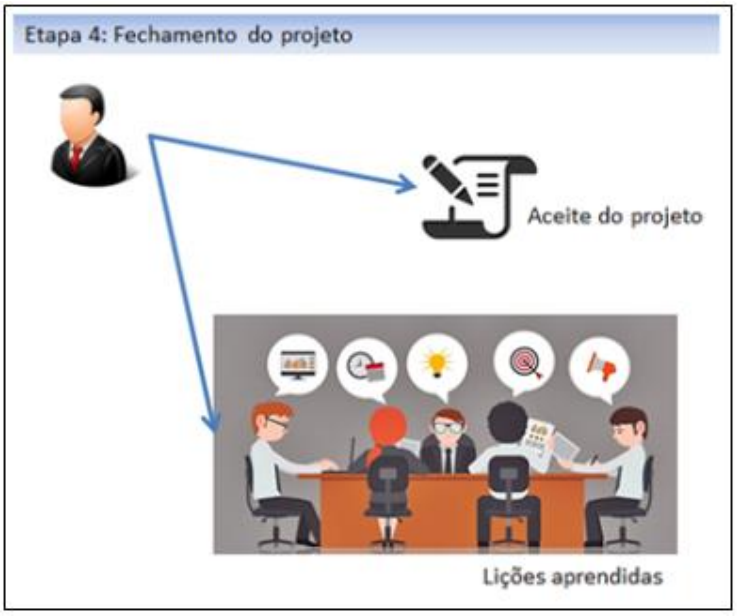
Figura 38: Questionário - Revisão dos requisitos e mensuração final do sistema.



Fonte: Autoria própria.

Na etapa 4 (Figura 39), temos a formalização da entrega final do projeto por meio da assinatura dos documentos que caracterizam o aceite. Não representa nenhuma etapa técnica, mas sim gerencial, que é feita juntamente com o cliente do projeto. Também é solicitada uma avaliação por parte do cliente do trabalho executado, visando aumentar o relacionamento e corrigir situações geradas anteriormente no ciclo de vida do projeto.

Figura 39: Questionário - Fechamento do projeto.



Fonte: Autoria própria.

Com relação a abordagem de desenvolvimento de *software* apresentada anteriormente, favor produzir um comentário, indicando pontos fortes, fracos e críticas pertinentes a abordagem.

Questão 9: Como você caracteriza a abordagem de desenvolvimento apresentado na ***Questão 8***?

- Tradicional (Exemplo: Cascata)
- Ágil (*Framework Scrum* e XP)
- Abordagem Híbrida (Com características de um modelo tradicional com práticas ágeis)

Questão 10: Com referência em seu conhecimento, considera viável a execução desta abordagem, apresentada na ***Questão 8***?

- Sim
- Não
- Talvez