Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia de Software

Estado da Prática da Manutenção de Software no Contexto de Startups de Software em *Early Stage*

Autor: Filipe Toyoshima Silva

Orientador: Mestre Cristiane Soares Ramos

Brasília, DF

2021

Filipe Toyoshima Silva

Estado da Prática da Manutenção de Software no Contexto de Startups de Software em *Early Stage*

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Mestre Cristiane Soares Ramos

Coorientador: Mestre Ricardo Ajax Dias Kosloski

Brasília, DF

2021

Filipe Toyoshima Silva

Estado da Prática da Manutenção de Software no Contexto de Startups de Software em *Early Stage*/ Filipe Toyoshima Silva. – Brasília, DF, 2021-

69 p. : il. (algumas color.) ; 30 cm.

Orientador: Mestre Cristiane Soares Ramos

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2021.

1. manutenção de software. 2. startup. I. Mestre Cristiane Soares Ramos. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Estado da Prática da Manutenção de Software no Contexto de Startups de Software em *Early Stage*

CDU 02:141:005.6

Filipe Toyoshima Silva

Estado da Prática da Manutenção de Software no Contexto de Startups de Software em *Early Stage*

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 16 de novembro de 2021:

Mestre Cristiane Soares Ramos
Orientadora

Mestre Ricardo Ajax Dias Kosloski
Coorientador

Dra. Carla Rocha Aguiar
Convidada

Brasília, DF
2021

Este trabalho é dedicado à inovação e àqueles que assumem o risco de criá-la

Agradecimentos

Tamanho é o caos em que vivemos que é impossível dizer a quem realmente devemos agradecimentos por estar onde estamos. Mas existem alguns nomes específicos que merecem atenção em razão da clara intenção de me proporcionar a oportunidade de poder gerar a contribuição que consta nas páginas a seguir.

Os primeiros responsáveis por qualquer sucesso que eu obtiver em vida, ou até mesmo post-mortem, caso haja, serão meus familiares. A estes, devo o apoio incondicional e o zelo pelo melhor caminho que eu possa tomar. Sem nada a esperar em troca, a não ser a gratidão que minimamente aqui expresso.

Em termos de desenvolvimento profissional, que é relacionado com a motivação do tema deste trabalho, devo reconhecimento à equipe Probono Digital, que me ofereceu mais do que uma primeira experiência no mercado de trabalho, mas uma ampla visão sobre como se pode impactar a sociedade de modo rentável e sustentável. Devo uma menção especial ao atual *CEO*, Rogério, por ter sido para mim uma grande inspiração profissional; alguém que enxerga a recompensa financeira como um reflexo do benefício que esforços realizados corretamente podem trazer à sociedade; alguém que busca desenvolver melhores soluções sem medo de errar no processo e com a certeza de que o erro significa aprendizagem. Graças a esse pensamento, a engenharia de desenvolver inovação é pesquisada aqui.

E é através dessa forma de pensar que a Cotidiano Aceleradora ajuda a semear a inovação no Brasil, cujo trabalho e desenvolvimento de comunidade não só contribuíram para a inspiração do tema aqui abordado, mas também prestou auxílio direto na execução da pesquisa. Por isso eu agradeço toda a comunidade Cotidiano, com menção honrosa a Thaís "Tete"Oliveira.

Não poderia deixar também de agradecer os bons professores que visivelmente têm como preocupação diária (e não apenas em dias úteis!) o sucesso e avanço de seus alunos, pois eles representam muito mais do que o mero produto de seu ofício, mas sim vidas humanas que definirão muitos dos avanços que toda a espécie fará. Não citarei muitos nomes, pois minha memória certamente deixaria alguns importantes de fora, e os realmente bons se reconhecerão nessa leitura. Mas um nome que não pode deixar de ser citado é o de minha orientadora Profª Cristiane Soares Ramos, que mostra uma abertura para os meios de aprendizagem que deveria ser obrigatória para qualquer profissional da educação. E isso além de uma compreensão e personalidade que a tornam um ser humano incrível.

Resumo

As empresas denominadas startups têm despertado cada vez mais interesse do mercado e representam potencial de inovação e novas tecnologias. Em seu estágio inicial, denominado *early stage*, startups são marcadas por baixa disponibilidade de recursos e intenso processo de experimentação, que acarreta em um grande volume de atividades de manutenção para adaptar o software às descobertas de mercado. Este trabalho tem por objetivo identificar um conjunto de práticas de manutenção evolutiva de software utilizadas em early startups. Nessa investigação é usada a técnica de entrevista semi-estruturada, através da qual foram coletados seis depoimentos de programadores e gestores de tecnologia que trabalham em startups *early stage*. Os depoimentos contém diferentes pontos de vista sobre quais práticas foram mais benéficas para a manutenção do software em suas respectivas empresas e, além disso, confirmam a priorização às regras de negócio e a eventual negligência deliberada a várias práticas da Engenharia de Software, mas também revelam que práticas menos complexas e menos custosas em termos de tempo são comumente proveitosas e bem-vindas.

Palavras-chave: manutenção de software; startup; early stage

Abstract

Companies called startups have attracted more and more interest from the market and represent potential for innovation and new technologies. In its initial stage, called *early stage*, startups are marked by low availability of resources and intense experimentation process, which entails a large volume of maintenance activities to adapt the software to market discoveries. This work aims to identify a set of evolutionary software maintenance practices used in early startups. In this investigation, the semi-structured interview technique is used, through which six testimonies of programmers and technology managers who work in *early stage* startups were collected. The testimonies contain different points of view on which practices were most beneficial for the maintenance of the software in their respective companies and, in addition, confirm the prioritization of business rules and the possible deliberate neglect of various Software Engineering practices, but also reveal that less complex and less time-consuming practices are commonly beneficial and welcome.

Key-words: software maintenance; startup; early-stage

Lista de ilustrações

Lista de abreviaturas e siglas

TPS	*Toyota Production System* ou Sistema Toyota de Produção
SEBRAE	Serviço Brasileiro de Apoio às Micro e Pequenas Empresas
VC	*Venture Capital*
PMF	*Product-Market Fit* ou Encaixe Produto-Mercado
MVP	*Minimum Viable Product* ou Produto Mínimo Viável
CTO	*Chief Technology Officer* ou Chefe de Tecnologia
CEO	*Chief Executive Officer* ou Chefe Executivo
PO	*Product Owner*
API	*Application Program Interface* ou Interface de Programação de Aplicações

Sumário

1 INTRODUÇÃO

1.1 Contextualização

Uma startup é uma empresa em modelo de negócio definido como escalar, repetível e lucrativo, que deve atacar um setor novo de mercado (BLANK, 2012). Normalmente fundadas com pouco recurso e operando em déficit, sustentadas apenas pelo aporte financeiro de investidores de risco (REIFF, 2020), o que significa uma condição de escassez de tempo e recursos financeiros, o que consequentemente pode significar a falta de pessoal qualificado em um cenário de incerteza. Tendo em vista todas essas peculiaridades, pode-se considerar que os desafios de startups que desenvolvem softwares são bastante específicos quando comparados a outras empresas que desenvolvem softwares com o suporte de processos bem estruturados e apoiados por recursos e conhecimento de setor.

Ries (2011) defende um modelo de experimentação a ser utilizado por startups que estejam em um estágio inicial, onde ainda há incerteza sobre o mercado a ser explorado. Esse processo, chamado de Lean, depende de uma série de testes realizados com clientes reais utilizando um produto de software em produção. Esse processo de experimentação prevê uma série de alterações nesse produto, que deve se manter operacional ao longo da disponibilização das várias versões que devem ser disponibilizadas, respeitando os limites de tempo e de recursos financeiros da organização.

Neste caso, os processos de manutenção de software são cruciais para que o software possa se adaptar à instabilidade do cenário onde se encontra, já que se espera que novas descobertas acerca do que o produto deveria ser sejam feitas após disponibilização desse produto para os clientes, cuja experiência deve ser o foco nos processos de experimentação citados.

1.2 Problema e justificativa

Startups de software têm tomado cada vez mais espaço nos mais variados mercados que atendem a sociedade, e isso tem acontecido cada vez mais rapidamente (SIEGELE, 2014). Segundo pesquisadores da PitchBook (STANFILL; STANFORD; CHAO, 2021), o mercado de investimentos em startups movimentou 166 bilhões de dólares no ano de 2020 e tem projeções para resultados parecidos em 2021.

E não só se destacam pela crescente presença na economia global, como também pela quantidade de inovação criada de modo rentável com uma baixa disponibilidade de recursos, quando comparadas às empresas mais robustas e tradicionais (UNTERKALMS-

TEINER et al., 2016). Justamente pela sua diferença entre companhias cujas metologias sejam mais conhecidas e amplamente estudadas, e também pelos seus resultados notáveis, este trabalho tem por objetivos fazer uma análise desse nicho tão diferente do usual.

Para as startups onde software seja um elemento crítico para o sucesso da empresa e seja parte necessária do produto ou serviço oferecido, é esperado, já pelo próprio modelo de desenvolvimento exposto anteriormente, que manutenções evolutivas e mudanças de requisitos aconteçam muito frequentemente. A cada iteração do ciclo de ideação, novas informações devem ser levadas em consideração e o software que compõe a proposta de valor da empresa deve se adaptar constantemente a essas descobertas.

Atualizar o código que compõe o produto ou serviço oferecido não é uma tarefa trivial, principalmente quando se leva em consideração um dos princípios do já citado Lean, que é o foco no encurtamento das iterações. É necessário a cada iteração entregar um software, que já era funcional, com novas funcionalidades ou com funcionalidades já existentes, mas que devem ser alteradas para respeitar uma regra de negócio recém levantada ou alterada. E é importante ressaltar que falhas de implementação podem quebrar o ciclo de desenvolvimento, já que as métricas obtidas a partir do comportamento do usuário para com o software desenvolvido serão impactadas pelas falhas entregues a estes usuários.

Neste contexto, a questão de pesquisa deste trabalho é: "Quais são as práticas de manutenção evolutiva de software utilizadas em startups em estágios iniciais (*early startups*)?".

1.3 Objetivos

Neste trabalho, o objetivo geral da pesquisa é: **"identificar um conjunto de práticas de manutenção evolutiva de software utilizadas em *early startups***".

Os objetivos específicos para que se atinja o objetivo geral são:

OE1: Prospectar os desafios da engenharia de software que são enfrentados em *early startups* para executar manutenções evolutivas (pesquisa bibliográfica);

OE2: Elaborar, com base nas práticas encontradas, um instrumento de coleta de dados para identificar as práticas de engenharia de software utilizadas em *early startups*;

OE3: Aplicação do instrumento de coleta de dados em *early startups*;

OE4: Mapear, com base na aplicação do instrumento, quais são as práticas de fato mais utilizadas pelas *early startups* e como essas práticas dialogam com o ambiente em que essas empresas se encontram.

1.4 Organização do trabalho

Além desta desta introdução (Capítulo 1), este trabalho é composto por mais seis capítulos.

O Capítulo 2, Referencial Teórico, constitui o referencial teórico do trabalho. Trata-se de reunir as definições dos conceitos chaves trabalhados ao longo desta pesquisa. Abordará em mais detalhes a visão de diferentes autores sobre o conceito de startup e suas peculiaridades em relação aos modelos de negócio mais tradicionais que envolvem o desenvolvimento e a manutenção de software. Definirá também, de modos generalista e específico, o conceito de manutenção de software e como diferentes autores da Engenharia de Software percebem esse problema, tanto em modelos tradicionais, quanto como nos cenários de escopo aberto e outros semelhantes ao de contexto de startup.

Durante o Capítulo 3, Metodologia de Pesquisa, serão definidas as técnicas e metodologias utilizadas para responder a questão de pesquisa. Serão citados autores que definem essas metodologias e suas particularidades, bem como onde serão defendidas para o contexto deste trabalho. Também abordará os detalhes estratégicos de coleta de dados e os cuidados que serão tomados para garantir o bom aproveitamentos dos esforços aplicados no processo de pesquisa. A partir do resultado esperado, também já será definido o procedimento de análise dos dados coletados.

Dando sequência ao a isso, o Capítulo 4, Roteiro de Entrevista, define em detalhes a ferramenta de coleta de dados: a entrevista semi-estruturada. Justifica a escolha de cada tema abordado na entrevista e define como será a abordagem a cada um desses temas durante a entrevista. Também justifica e descreve a aplicação de um piloto.

O Capítulo 5, Resultados, reúne os dados que foram coletados e aplica as ferramentas de análise descritas anteriormente. As respostas serão analisadas em caráter de tópico isoladamente, descrevendo o que pensam os entrevistados sobre cada um dos tópicos; e também em termos de relacionamento entre conceitos, mostrando o impacto de cada disciplina sobre a outra.

Por fim, o Capítulo 6, Conclusão, conclui a pesquisa, expondo os principais achados durante tudo o que foi descrito neste trabalho, relacionando-os aos objetivos definidos anteriormente e expondo as limitações e possíveis continuidades.

2 Referencial Teórico

Aqui serão discutidos conceitos que cercam o tema deste trabalho. A começar pelo conceito de startup, um breve histórico de como surgiu esse modelo de negócio e o como ele é caracterizado atualmente. É importante dizer que existe pouco material acadêmico sobre o assunto e que, portanto, literatura cinza foi utilizada na construção do conceito utilizado nesse trabalho.

Depois, são levantados conceitos relacionados à Manutenção de Software, tema central a ser investigado posteriormente. Esses conceitos ajudarão a entender a quais pontos deve ser dada a atenção quando startups forem estudadas.

Por fim, é estressada a relação entre esses dois grandes temas: startups e manutenção de software. É constatado, inclusive, que há pouca literatura que descreva essa relação, o que corrobora com a justificativa deste trabalho.

2.1 Startup

2.1.1 O que são Startups?

No Japão, em meio à crise petrolífera de 1973, estava sendo criado o Sistema Toyota de Produção (*Toyota Production System* ou TPS) (MONDEN, 2012). Em um cenário de economia não favorável e escassez de insumos, a empresa criadora do sistema se via em uma condição onde desperdícios eram ainda menos toleráveis do que o normal. Desperdícios estes que não se referem apenas aos insumos materiais, como o petróleo, mas que abarcava também esforços logísticos, como transporte e estoque de material.

O TPS foi criado para melhorar os resultados da empresa e aumentar os lucros através de alguns objetivos específicos. Por exemplo: aumento do lucro através da redução de custos; eliminação de superprodução; controle de qualidade; produção Just-in-Time (JIT) e outros.

O termo "Lean"foi utilizado por Jim Womack na década de 80 para descrever o TPS e seus esforços na diminuição dos desperdícios. Este termo foi utilizado depois em outras metodologias e ferramentas, como o Lean Six Sigma, que também tem como foco a otimização de processos utilizando as mesmas bases do TPS. Existe até mesmo o "Lean Development", uma subcultura derivada do movimento ágil para aplicação da cultura ágil em soma às filosofias defendidas pelo TPS.

Posteriormente, foi popularizado o termo "Lean Startup"pelo livro de mesmo nome (traduzido no Brasil para "A Startup Enxuta") que foi escrito por Ries (2011), considerado

desde então um dos grandes representantes do movimento startup.

Para Ries, "startups são instituições humanas projetadas para entregar um novo produto ou serviço em condições de extrema incerteza". Quando o autor menciona "um novo produto ou serviço em condições de extrema incerteza", ele se refere a um mercado necessariamente novo, onde não se sabe ao certo como o mercado reagirá à existência desse produto, e portanto, tenha condições de negócio consideravelmente incertas.

Posteriormente, Blank (2012) define startups como "uma organização temporária em busca de um modelo de negócio escalável, repetível e lucrativo", além de não ter inicialmente nenhum cliente e ter pouca informação sobre como os clientes se comportam. O Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE) (MORAES, 2020) define startup de modo semelhante, como um negócio embrionário, inovador, de alto risco e que tenha as características de escalabilidade e repetibilidade.

Escalabilidade diz respeito a viabilidade que o modelo de negócio tem de atender um número de clientes que pode crescer exponencialmente, sem a necessidade do aumento proporcional (ou maior) dos custos de operação. Já a repetibilidade diz respeito a possibilidade de ampla aplicação do modelo de negócio, sem a necessidade de grandes alterações para que seja atingido um grande número de clientes.

2.1.2 A Startup em *Early-Stage*

No mercado financeiro, existe o que se chama de "*Venture Capital*"(VC). Esse tipo de investimento consiste, também, em realizar aportes em organizações pequenas que tenham um alto potencial de crescimento e, em troca, obter quotas que representem uma fração dessas organizações (HAYES, 2021). O retorno de investimento vem pela valorização dessas quotas, uma vez que a organização que recebeu esse aporte passe a valer consideravelmente mais do que no momento do aporte. Espera-se que essas organizações sejam valorizadas de modo drasticamente rápido. Um ponto negativo do VC é a alta taxa de falência dessas organizações antes que a esperada valorização ocorra, por isso muitas vezes é também chamado de "Capital de Risco".

Startups normalmente são fundadas em VC e operam inicialmente apenas com os recursos financeiros obtidos dos investimentos (REIFF, 2020). Ou seja, operam em *deficit*, sem que a receita obtida (quando há receita) pague os custos de operação da empresa. Por isso, é necessário que os gastos sejam minimizados ao limite do possível, pois menos recurso gasto a cada mês significa mais tempo de operação para a empresa.

A uma startup que se encontre nesse estágio, se dá o nome de *Early Stage*, e o objetivo defendido por muitas aceleradoras ou incubadoras de startups para esse estágio é o encontro do *Product-Market Fit* (PMF ou Encaixe Produto-Mercado), que é a adequação da inovação desenvolvida a um volume estável e fiel de clientes. Não existe consenso sobre

quando exatamente uma empresa atinge o PMF, mas é acordado e defendido por vários autores da área, incluindo Ries e Blank em seus respectivos livros, que este é um processo de intensa experimentação e aprendizado através de dados. Portanto, um dos grandes focos de startup *Early Stage* é a experimentação.

2.1.3 Processo de experimentação do *Lean Startup*

Em razão das condições de incerteza presentes na definição de uma startup, é esperado que o produto ou serviço desenvolvido inicialmente não seja aquele que vai conquistar uma fatia satisfatória do mercado. Por isso o, Lean Startup defende a ideia de ciclos de desenvolvimento que, eventualmente, se concluam com *pivots*, que são alterações consideráveis no produto ou serviço desenvolvido. Esses ciclos de desenvolvimento são constituídos de três partes e atividades de transição entre cada uma dessas partes. A primeira parte do ciclo é a ideação, onde o insumo disponível são as ideias que guiarão os planos de negócio. Para chegar na segunda etapa, é dada a construção, que terá como resultado um produto (ou estruturação de um serviço). A terceira etapa é atingida através da medição, onde o comportamento o usuário do produto ou serviço, que já deverá estar disponível e usável, será estudado e mensurado. Os dados coletados são então analisados para produção de novas ideias, dando início a uma nova iteração do ciclo, que deverá contar com ideias mais bem fundamentadas no comportamento constatado dos clientes, a fim de corrigir erros e otimizar o produto desenvolvido na iteração anterior, o que pode considerar mudanças bruscas (*pivot*).

Ries, defende a ideia de que as iterações desse ciclo devem ocorrer o mais rápido o possível, pois a cada iteração se aprende mais sobre o mercado que ainda está sendo explorado, o que aumentará as chances de uma empresa de sucesso. Ou, no pior dos cenários, criará evidências de uma não sustentação do modelo de negócio almejado, o que ainda significa uma economia de recursos que não deverão mais ser investidos nesse empreendimento sem retorno.

Como os ciclos devem ser rápidos, não se espera que a cada iteração se construa completamente um produto extremamente complexo. Nesse contexto que surge o conceito de *Minimum Viable Product* (MVP), que se refere a um produto (ou fração dele) que deve conter apenas o suficiente para que a etapa de medição seja realizada com resultados relevantes. Esse conceito existe para evitar desperdícios e facilitar a medição do produto, já que existirão variáveis mais isoladas.

Ao fim de cada iteração do ciclo, algumas decisões podem ser tomadas, como a alteração completa do comportamento de determinada funcionalidade do produto, ou então a adição de uma funcionalidade antes não considerada. Em todo caso, refere-se a alteração de um produto que, previamente, já tinha sido considerado pronto para uso. No caso de startups cujo produto ou serviço seja um software, isso significa alteração

constante no que já tinha sido desenvolvido ao ponto de ser considerado pronto para o uso.

2.2 Manutenção de Software

2.2.1 O que é Manutenção

Segundo Pigoski (1997), existe uma grande confusão sobre as definições acerca do que exatamente é Manutenção de Software, e o mesmo traz consigo várias definições. Uma delas é a utilizada pela IEEE... (1998) que segue como "Modificação de um produto de software depois da entrega para corrigir falhas, melhorar a performance ou outros atributos, ou para adaptar o produto a um ambiente modificado".

Parte da confusão que Pigoski relata diz respeito à diferenciação entre as etapas de desenvolvimento e de manutenção do software, pois ambas consistem em escrita de código que cumpra com requisitos levantados anteriormente. Se tomarmos a definição da IEEE, destaca-se o trecho "depois da entrega", como se houvesse um ponto de ruptura no tempo que separasse as etapas de desenvolvimento e de manutenção do software.

Contudo, o próprio Pigoski relata existirem diversos modelos de ciclo de vida de um produto, onde em alguns se segue como o esperado trazido pela definição da IEEE sobre manutenção, havendo uma clara divisão entre as fases do produto, que passam, entre desenvolvimento, produção, implantação, operação e só então manutenção (modelo Waterfall descrito pelo autor); porém, em outros não existe uma única entrega a partir da qual todo o trabalho sobre o produto de software pode ser considerado "manutenção".

Além do Waterfall, Pigoski descreve outros dois tipos de ciclo de vida de produto, o Incremental e o Evolucionário, onde a etapa de desenvolvimento se torna paralela à de manutenção, pois o produto de software é entregue em versões operacionais antes do cumprimento de todos os requisitos levantados. No modelo Evolucionário, em particular, é comum que o usuário final já tenha contato com o produto de software antes do término da etapa de desenvolvimento.

2.2.2 Tipos de Manutenção

Pigoski também cita diferentes tipos de manutenção, que ocorrem em diferentes ocasiões. A mais completa é trazida pela ISO/IEC/IEEE... (2006) que conta com os seguintes tipos:

- **Manutenção Adaptativa**: são as modificações em um produto de software depois da entrega para mantê-lo usável após ou durante uma mudança de ambiente;

- **Manutenção Corretiva**: é a alteração reativa em um produto de software depois da entrega para resolver problemas descobertos;
- **Manutenção Perfectiva**: modificação de um produto de software depois da entrega para corrigir falhas latentes antes que se tornem erros perceptíveis pelo usuário;
- **Manutenção Preventiva**: modificações em um produto de software depois da entrega para corrigir falhas latentes antes que se tornem falhas operacionais.

O glossário da IEEE ainda trás um conceito que não foi documentado por Pigoski como um tipo de manutenção, mas que se encaixa nas definições mais genéricas de manutenção. Se trata da **manutenção de melhoria**, que é definida como a alteração de um produto de software existente para satisfazer um novo requisito.

Em seu livro, Pigoski traz algumas pesquisas acerca do estado-da-prática de manutenção, como quais categorias consomem mais recursos, porém são pesquisas das décadas de 80 e 90. Além de serem datadas, também são do contexto de empresas grandes e tradicionais, cujas diferenças do escopo tratado aqui já foram elucidadas.

Além do mais, os estudos desse período não só diferem em contexto, mas também na própria metodologia adotada para desenvolver e manter software. Quando se segue uma metodologia ágil, as atividades diferem, e portanto os desafio de manutenção também diferem, que foi a conclusão de Tordrup e Rose (2015), que também propuseram nove heurísticas a respeito de manutenção dentro do processo ágil de desenvolvimento, mais especificamente o SCRUM. São elas:

1. Use *sprints* para organizar o trabalho de manutenção;
2. Permita pedidos urgentes e inesperados de clientes
3. Estruture a aprendizagem da equipe onde os membros da equipe trabalham em tarefas distintas
4. Estruture múltiplos relacionamentos com o cliente
5. Equilibre documentação e comunicação cara-a-cara apropriadamente
6. Escreva (minimamente) casos de uso estruturados para comunicação
7. Fortaleça a propriedade e o aprendizado coletivos testando o código um do outro
8. Encontre motivadores de equipe para substituir a motivação natural de chegar ao final de uma *sprint* ou projeto
9. Mantenha o planejamento de reuniões curtas e efetivas

Além deste, alguns outros estudos (FEHLMANN; FALKNER, 2015; JUNIOR; DANTAS, 2019) demonstraram bons resultados na adoção de práticas ágeis voltadas ao SCRUM. E, embora em um contexto de grandes organizações ainda diferentes do escopo aqui estudado, os estudos citados têm como característica em comum a aplicação em um produto de manutenção intensa.

2.3 Manutenção de Software no Contexto de Startups

Por todas as diferenças já mencionadas de estratégias e modelo de negócio que as startups têm em relação às grandes empresas, os processos de Engenharia de Software também são consideravelmente diferentes, e especificamente a Manutenção de Software não é diferente. Nguyen-Duc, Kemell e Abrahamsson (2021) investigaram 40 startups a fim de entender o modo como as equipes de desenvolvimento de software lidavam com os desafios deste tipo de organização. O estudo se desenvolveu através de várias dimensões da Engenharia de Software, incluindo Engenharia de Requisitos, *Design* de Produto, Teste de Software, Manutenção e outros.

Como já mencionado, o processo de construção de um produto de software em startups paraleliza as etapas de construção e manutenção, fazendo com que seja até difícil categorizar o que exatamente é manutenção e o que é construção. E o estudo confirma isso. Mais especificamente em startups cujo produto esteja sendo utilizado pelos primeiros usuários, que é o caso das startups em *early stage*, é muito comum que novas funcionalidades, ou até mesmo alterações em funcionalidades já existentes, não sejam planejadas, e sim reativas às percepções e respostas dos usuários. Esse é um reflexo do desenvolvimento de produto centrado no usuário e baseado em experimentação.

Em conclusão, o estudo define a manutenção de software como "oportunista". Diz ainda que aceitar o débito técnico faz parte da natureza do processo, e que comumente se descarta um sistema que não funciona adequadamente e se desenvolve um novo sistema, ao invés de realizar engenharia reversa em um produto com falhas.

Ainda sobre gestão de débito técnico, não só é comum que as startups aceitem débito técnico em suas fases iniciais, onde a falta de recursos e o excesso de experimentação e alteração de código em um tempo limitado são fatores contribuintes para essa aceitação, mas também é comum que essas empresas continuem aceitando esse débito mesmo quando prosperam até a etapa de escala (CICO et al., 2020), onde o desafio de encontrar o PMF já teria sido concluído, e portanto a intensidade de experimentação cai.

3 METODOLOGIA DE PESQUISA

Neste capítulo serão abordadas as informações que caracterizam os métodos escolhidos para atingir os objetivos de pesquisa. Será discutida, em detalhes, a classificação da metodologia escolhida.

Em seguida, é detalhado o plano metodológico, descrevendo cada etapa do procedimento, o que inclui o planejamento, a coleta dos dados, a análise e redação dos resultados. E isso seguido da caracterização cronológica de cada etapa.

3.1 Classificação da metodologia de pesquisa

A metodologia de pesquisa adotada neste trabalho está classificada quanto à natureza, à abordagem, à tipologia, aos procedimentos técnicos e às técnicas de coleta de dados.

Quanto à natureza, esta pesquisa é aplicada, pois, tem como objetivo a geração de conhecimentos dirigidos à identificação de práticas de engenharia de software aplicadas em startups de desenvolvimento de software.

Segundo Gil (2002) as pesquisas descritivas tem como objetivo a descrição de determinada população ou fenômeno ou, então, o estabelecimento de relações entre variáveis. Uma de suas características significativas está no uso de técnicas padronizadas de coleta de dados, como o questionário. Sendo assim, quanto à tipologia, esta é uma pesquisa descritiva que tem como objetivo identificar práticas de engenharia de software que vêm sendo usadas por startups em *Early Stage* para fazer manutenção evolutiva de software.

Quanto à abordagem, é uma pesquisa qualitativa, pois é característico desse trabalho a análise em profundidade sobre a realidade de algumas empresas estudadas, sem necessariamente o uso de métricas, mas utilizando como dados a percepção refletida nos depoimentos dos participantes da pesquisa.

Quanto aos procedimentos técnicos, os meios de investigação adotado foram: pesquisa bibliográfica, pesquisa documental (literatura cinza). Já com relação à técnica de coleta de dados, foi adotada a observação direta intensiva (aplicação de entrevista).

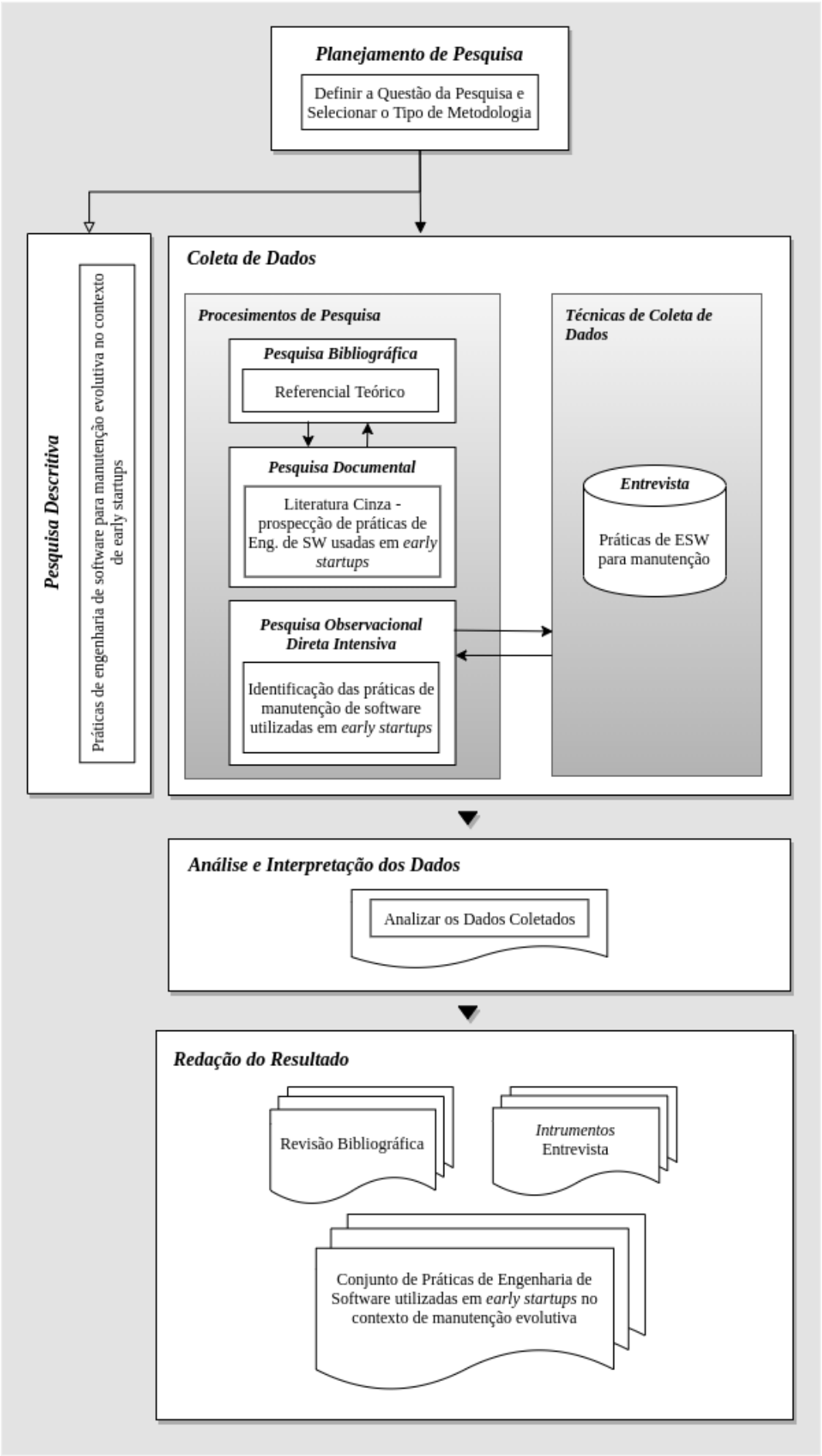

Figura 1 – Planejamento da pesquisa

3.2 Plano metodológico adotado

3.2.1 Planejamento do trabalho

Esta é a primeira fase do trabalho, que por sua vez tem como objetivo estabelecer a questão de pesquisa, os objetivos geral e específicos a serem atingidos, a seleção metodológica com: as fases e as sub-fases da pesquisa, o planejamento dos procedimentos de coleta de dados a serem executados.

3.2.2 Coleta de dados

3.2.2.1 Procedimentos de pesquisa

A *Pesquisa bibliográfica* diz respeito ao levantamento da literatura para construção do referencial teórico com o objetivo de apoiar a o entendimento de conceitos associados a startups e manutenção de software e prospectar práticas de engenharia de software relevantes para startups em estágios iniciais.

Como existe pouco conteúdo bibliográfico que caracterize as startups de estágio inicial, a *Pesquisa na Literatura Cinza* servirá de apoio para melhor parametrização do que deve ser referenciado para garantir a fundamentação desse trabalho. Como literatura cinza serão utilizados livros relevantes entre empreendedores da área e conteúdos *online*, como blogs, cuja autoria seja de relevância para a comunidade de startups.

Todo esse levantamento que caminhará entre *Pesquisa bibliográfica* e *Pesquisa na Literatura Cinza* tem dois principais objetivos:

- A compreensão da caracterização de negócio sobre o que é uma startup em estágio inicial. Essa caracterização ajuda a definir quais são as restrições e prioridades que evidenciam as diferenças entre este tipo de empresa para as mais tradicionais.

- A prospecção sobre práticas de engenharia de software que são relevantes para a realidade deste tipo de empresa, seja porque são práticas relacionadas a disciplinas problemáticas neste tipo de empreendimento, seja porque existe um alinhamento para com as prioridades de negócio.

A *Pesquisa observacional direta intensiva* será realizada através de uma entrevista, que, segundo Lakatos e Marconi (2010), "tem como objetivo principal a obtenção de informações do entrevistado, sobre determinado assunto ou problema". Foi o método escolhido para entender quais são as práticas de engenharia de software que de fato são utilizadas por startups em estágio inicial.

3.2.2.2 Técnica de coleta de dados

A elaboração da entrevista será baseada nos conceitos fundamentos em Flick (2009). O tipo de entrevista escolhido foi o de *Entrevista Semipadronizada*. Também foram utilizados conceitos relacionados à *Entrevista Semi-Estruturada* (QU; DUMAY, 2011).

Essa entrevista será conduzida em tópicos que guiem o diálogo entre entrevistador e entrevistado, que podem ser explorados em ordem pré-definida ou, caso seja soe mais adequado sob o ponto de vista do entrevistador, em uma ordem que siga o fluxo da conversa, pois o entrevistado pode tocar em assuntos relacionados a mais de um tópico pré-definido.

Esse tipo de entrevista foi escolhido por permitir ao pesquisador maior flexibilidade durante a coleta dos dados, o que é importante, visto que existe pouco conhecimento solidificado na literatura que garanta a elaboração de um formulário mais fundamentado. Essa flexibilidade é útil para conduzir a entrevista de modo que o respondente possa contribuir mais ativamente com o seu ponto de vista, pois tem a liberdade de falar sobre os tópicos levantados sob o seu ponto de vista e não sob a rigidez de uma pergunta direta de um formulário. E, ao mesmo tempo que permite essa flexibilidade, é de caráter da entrevista semi-estruturada que exista uma base de tópicos que guiem as perguntas, o que é bem adequado à realidade dessa pesquisa, já que podem ser utilizadas as disciplinas da Engenharia de Software como tópicos.

Entretanto, entrevista como um todo é um método de coleta de dados que tem seus riscos. É necessário cuidado ao elaborar as questões e também ao questionar o entrevistado para que se evite a inserção de viés do pesquisador. Sobre isso, no escopo desse trabalho, serão utilizadas principalmente (FLICK, 2009) as *questões abertas*, onde se pergunta ao entrevistado seu ponto de vista sobre determinado tópico ("O que você acha sobre...?"; "Como é ... para você?") e *perguntas controladas pela teoria e direcionadas para hipóteses*, que confirmam ou confrontam hipóteses previamente levantadas nas etapas de pesquisa que antecedem a elaboração das questões.

Os dados coletados podem também ser considerados semi-estruturados, já que, apesar de organizados por tópicos, ainda serão coletados na forma de uma resposta dada de modo subjetivo, o que pode representar um risco de inserção de viés em momento de análise. A análise será discutida mais adiante nesse trabalho.

3.2.2.2.1 Planejamento da Entrevista

Como este trabalho está sendo realizado em período de pandemia, é necessário manter o distanciamento social, o que impossibilita a realização de uma entrevista presencial. Portanto, as entrevistas serão realizadas através de videoconferências.

De qualquer forma, o canal escolhido oferece suas vantagens. A não necessidade do encontro presencial evita deslocamento, o que pode significar em uma redução de custos para o entrevistador ou uma maior flexibilidade de disponibilidade para o entrevistado. Além de uma maior facilidade em entrevistar um maior número de participantes em um menor período de tempo, já que não é necessário tempo de deslocamento entre uma entrevista e outra.

Outra vantagem da entrevista por videoconferência é a possibilidade de registro audiovisual, que deverá ser permitido expressamente pelo entrevistado. Esse registro permite maior uma maior atenção do entrevistador, que não precisará se preocupar em anotações paralelas à condução da entrevista, e também reduz o risco de viés durante a análise, pois os dados poderão ser analisados na sua forma mais bruta durante o tempo que for necessário.

Dito isso, será levantada uma lista de contatos elegíveis como entrevistados, ou seja, pessoas que tenham trabalhado com desenvolvimento de software em startups *early stage*. Essas pessoas serão apresentadas brevemente à ideia da pesquisa e, caso demonstrem interesse em contribuir, receberão por e-mail uma carta formal que inclui o convite e termos de aceitação de participação da pesquisa. O entrevistado deverá responder por e-mail a confirmação de que leu e aceita com os termos antes que a entrevista seja realizada. A carta utilizada se encontra no Apêndice A.

3.2.2.2.2 Aplicação do Piloto

Para uma melhor garantia de sucesso das metodologia de coleta de dados utilizada, será aplicada uma entrevista piloto, cujos dados não serão utilizados em análise. Essa entrevista piloto seguirá exatamente o plano proposto, mas passará por uma etapa de revisão antes que o plano seja executado para as demais entrevistas.

A análise desse piloto servirá para verificação de eventuais falhas na metodologia utilizada e seus dados não serão utilizados na compilação final dos resultados.

3.2.3 Análise e interpretação dos dados

Dado o planejamento anterior, serão insumos da coleta de dados uma série de entrevistas gravadas. Essas entrevistas serão transcritas a fim de organização e facilitação da consulta posterior.

Realizadas as transcrições, será possível identificar termos e temas comuns entre as respostas coletadas, cumprindo assim com o *OE4*, que se refere ao mapeamento das práticas mais utilizadas dentro do contexto do objeto de estudo.

A análise se dará por meio da Síntese Narrativa, que, segundo Kitchenham (2016), é considerada o método mais comum para pesquisas qualitativas em engenharia de software.

Ainda segundo Kitchenham, "a Síntese Narrativa reporta os resultados de uma revisão sistemática em termos de texto e palavras", quase como "uma forma de contação de história", similar ao que defende Rainer (2021), que menciona o *storytelling* como um método que, apesar dos riscos, pode ser muito útil para análise dos resultados em pesquisas na Engenharia de Software.

O processo de narrativa envolve a organização das informações obtidas em um sentido lógico, de modo que seja explorada a relação entre os dados obtidos. Portanto, além da síntese escrita, a análise contará também com a elaboração de mapa conceitual que relacione os temas abordados, possibilitando uma análise mais profunda sobre como as várias disciplinas de engenharia de software se relacionam nesse contexto, em especial a manutenção, o que é bastante oportuno dada a natureza das entrevistas semi-estruturadas.

3.2.4 Redação dos resultados

A redação dos resultados desta pesquisa é composta por: Referencial teórico obtido a partir da pesquisa bibliográfica; conceitualização da realidade de uma startup através das informações levantadas na pesquisa na literatura cinza; elaboração e validação do roteiro de entrevista para identificar práticas de manutenção de software usadas em startups; e finalmente o guia de práticas para manutenção de software aplicadas em startups em estágios iniciais.

4 ROTEIRO DE ENTREVISTA

Aqui é detalhada a construção do roteiro da entrevista, que teve como base pesquisa prévia. A partir da idealização dos tópicos encontrados em pesquisa, são definidas as perguntas que serão realizadas para obter os dados condizentes com os objetivos de pesquisa. Também é elaborado a aplicação da entrevista piloto.

4.1 Escolha dos Tópicos

Como foi definido anteriormente na Metodologia de Pesquisa, a metodologia escolhida foi a de entrevista semi-estruturada, que tem por característica o uso de tópicos (QU; DUMAY, 2011) que guiem o dialogo entre entrevistador e entrevistado.

Para sustentar a escolha de tópicos a serem abordados pela entrevista, foram escolhidos quatro artigos que elucidem temas relacionados ao desse trabalho. São eles:

- "*Optimising agile development practices for the maintenance operation: nine heuristics*", de Tordrup e Rose (2015), por sua abordagem no tema de manutenção sob a luz de metodologias ágeis, as quais se assemelham com o que foi levantado em pesquisa sobre a realidade das startups. Doravante *Artigo 1*.
- "*The Emergence of Agile Maintenance: A Preliminary Study*", de Ibrahim et al. (2019), pelo mesmo motivo do artigo anterior. Doravante *Artigo 2*.
- "*The entrepreneurial logic of startup software development: A study of 40 software startups*", de Nguyen-Duc, Kemell e Abrahamsson (2021), por se tratar de um amplo estudo que analisa em múltiplas disciplinas a realidade de startups de software. Doravante *Artigo 3*.
- "*Software Development in Startup Companies: A Systematic Mapping Study*", de Paternoster et al. (2014), pelo mesmo motivo do artigo anterior. Doravante *Artigo 4*.

Após leitura e investigação das fontes escolhida, percebeu-se a presença de seis diferentes tópicos que sempre aparecem em evidência em, pelo menos, 3 deles. São eles:

- **Metodologia de Desenvolvimento**: Se refere a quais metodologias os times de desenvolvimento de software seguem, quais rituais ou princípios guiam o cotidiano de trabalho da equipe.

	Artigo 1	Artigo 2	Artigo 3	Artigo 4
Metodologia de Desenvolvimento	✓	✓	✓	✓
Manutenção e Evolução de Software	✓	✗	✓	✓
Proximidade do Usuário no Processo de Manutenção	✓	✓	✓	✓
Engenharia de Requisitos	✓	✗	✓	✓
Documentação de Software	✓	✓	✗	✓
Testes e Controle de Qualidade	✓	✓	✓	✓

Tabela 1 – Tabela de disciplinas e referências

- **Manutenção e Evolução do Software**: Se refere a como os times de desenvolvimento ou manutenção de software trabalham em um produto que já está em contato com o usuário. Também está relacionado com a gestão de débito técnico.
- **Proximidade do Usuário no Processo de Manutenção**: Como foi levantado anteriormente, dados fornecidos pelos usuários tendem a ser relevantes dentro do contexto de startups early stage, e este tópico aborda a relação entre essa característica e seus impactos na manutenção do software.
- **Engenharia de Requisitos**: Estando em uma realidade tão incerta, este tópico aborda quais práticas de gestão de requisitos são adequados para lidar com a mudança frequente de requisitos.
- **Documentação de Software**: Se refere a tudo o que é documentado, de qualquer forma possível, acerca do software que está sendo desenvolvido ou mantido.
- **Testes de Controle de Qualidade**: Se refere aos esforços de garantir que o produto que chegue ao usuário final esteja funcionando como esperado. Também está relacionado com questões de qualidade inerentes ao código, como análise estática.

A relação entre as disciplinas e os artigos lidos se encontra na Tabela 1, onde estão marcados com ✓os temas que foram abordados e com ✗os temas que não foram encontrados.

4.2 Elaboração das Perguntas

Levantadas as disciplinas que serão abordadas, basta transformá-las em perguntas, buscando modelos de perguntas defendidos por Flick (2009), como proposto em Metodologia.

Como os assuntos investigados são complexos e para cada tópico levantado existe um universo de detalhes a ser explorado, foram levantadas várias perguntas para cada

tópico. Esse roteiro com as várias perguntas servirá de guia para que o entrevistador possa lê-las enquanto conduz a entrevista, e assim garantir que o máximo de detalhes sobre determinado tópico possa ser extraído, de modo que se uma resposta simples não traz informação suficiente em primeiro momento, existem outras questões sobre aquele mesmo tópico para auxiliar.

- **Metodologia de Desenvolvimento**: Descreva como é o seu trabalho. Qual é o seu cotidiano? Quais são as atividades? Alguém disse que tem que ser assim, uma diretriz ou uma metodologia? Essa metodologia envolve algum planejamento, como deadlines e entregas?

  Essa deve ser inicialmente uma questão aberta, de modo que o entrevistado possa se sentir confortável para contar tudo o que vier à mente sobre seu cotidiano. É importante conduzir respostas que abordem questões de planejamento e acompanhamento dos trabalhos do time de desenvolvimento de software, caso existam. A existência de reuniões também deve ser investigada, sejam técnicas ou administrativas.

- **Manutenção e Evolução do Software**: Como a metodologia adotada dialoga com a manutenção do software? Existem interrupções não planejadas em razão de demandas urgentes?

  Aqui busca-se investigar também como as startups lidam com riscos, como quebra de cronograma. A gestão de débito técnico também é um ponto a ser investigado, bem como essa gestão também dialoga com a metodologia utilizada pela equipe.

- **Proximidade do Usuário no Processo de Manutenção**: Quão próximo o usuário é do processo de manutenção e evolução do software? Existe algum tipo de contato com os usuários ou alguma influência deles no processo decisório? São recebidas solicitações de manutenção dos clientes, como um bug reportado. Se sim, como acontece o *report*?

  É importante mencionar que usuário nem sempre é o cliente do modelo de negócio, mas ainda assim, ambos (usuário e cliente) podem entrar no escopo da pergunta, pois são ambos considerados *stakeholders* importantes.

- **Engenharia de Requisitos**: Como é a gestão de requisitos? Como decidem o que desenvolver / alterar / consertar em cada momento?

  É importante também abordar questões relativas a pré e pós rastreabilidade desses requisitos, caso seja feita.

- **Documentação de Software**: Como é a documentação do software e como isso afeta a manutenção? O que é documentado? A documentação é útil?

É possível que não exista documentação para ser comentada pelo entrevistado. Nesse caso, é importante entender por que não existe e se já existiu em algum momento. Enfim, qual é a visão do entrevistado sobre documentação para o contexto de startup.

- **Testes e Controle de Qualidade**: Existe algum tipo de procedimento de teste de software manual ou automático? Se sim, como é? Quais são os tipos de testes e ferramentas utilizadas? Existe planejamento desses testes?

  O mesmo vale sobre não aplicação dessa disciplina, caso não seja: Por que não é? Já existiu alguma tentativa? Qual a sua opinião sobre o custo-benefício da realização de testes ou outras formas de controle de qualidade?

4.3 Aplicação do piloto

Antes de aplicar o roteiro de entrevista e obter os dados da pesquisa, foi feita uma verificação da adequação do roteiro através de uma entrevista piloto.

Todos os detalhes da entrevista piloto foram feitos de acordo com o planejado para as demais entrevistas, ou seja, o entrevistado se enquadrava no objetivo de pesquisa, a abordagem e o convite foram feitos da mesma forma. A única exceção sobre a condução da entrevista foi um breve questionamento ao final sobre a própria entrevista para verificar se todas as perguntas foram claras e se o entendimento foi fácil. Depois disso, as respostas do piloto foram verificadas para aferir se as informações obtidas eram condizentes com os objetivos de pesquisa.

5 RESULTADOS

Aqui serão apresentados os resultados da aplicação do que foi previamente planejado. As entrevistas seguiram exatamente planejado, não havendo imprevistos ou contratempos, e a primeira seção descreve o cumprimento desse planejamento em detalhes. Em seguida, a segunda seção descreve os dados demográficos do universo de dados que está sendo estudado neste trabalho, caracterizando cada entrevistado e as empresas sobre as quais relataram.

Por fim, as respostas serão respondidas sob dois diferentes pontos de vistas, um em cada uma das seguintes seções, que são a análise por tópico e a análise de relação entre os assuntos. Na primeira, busca-se entender quais foram os consensos e discordâncias entre os entrevistados para cada um dos assuntos levantados na entrevista. No último, as respostas são modeladas em um mapa conceitual que relaciona os conceitos levantados nas entrevistas e agrupa esses conceitos em categorias.

5.1 Aplicação das Entrevistas

As entrevistas ocorreram como planejadas. A abordagem inicial foi feita através de aplicativos de mensagens, por razões de praticidade, e neles foi dada uma breve explicação sobre a pesquisa e verificado o interesse dos potenciais participantes. Nas mensagens, foi explicado que se tratava de uma pesquisa para um trabalho de conclusão de custo e que o objetivo era entender detalhes do cotidiano de desenvolvimento de software em uma startup *early stage*. Além disso, foi mencionado também que essa entrevista ocorreria por vídeo chamada e que deveria demorar cerca de 30 minutos.

De todos os abordados, apenas um negou a participação por questões de disponibilidade. Todos os demais aceitaram participar, totalizando 6 participações.

Para aqueles que aceitaram participar, foi enviado um e-mail, no qual estava anexado o no Apêndice A, que contém um convite formal para a participação na pesquisa e o termo de aceite para participação. Foi pedido aos interessados que respondessem à mensagem de e-mail como uma demonstração de aceitação aos termos apresentados.

Após confirmação, a comunicação volta a ser pelo mesmo aplicativo de mensagens, restando apenas marcar um dia e horário para a realização da entrevista. Todas as entrevistas foram feitas através da plataforma *Teams* e foram gravadas para posterior consulta durante a etapa de consolidação dos resultados. Todas as gravações foram feitas mediante consentimento explícito do participante, que foi também gravado.

As entrevistadas duraram entre 25 a 35 minutos cada.

5.2 Os Entrevistados

Foram 6 entrevistados ao todo, alguns com experiência em mais de uma startup. Em respeito aos termos acordados com os entrevistados, suas identidades serão anonimizadas e seus nomes substituídos por pseudônimos. Segue uma lista das identidades anonimizadas com uma breve caracterização

- **Professor:** *CTO* de uma startup de educação (chamada doravante de ***Edtech***, também para fins de anonimização) desde sua fundação em 2016, onde já começou com alguma experiência no mundo das startups. Apesar do cargo ligado diretamente a tecnologia, também já assumiu outras responsabilidades, como apoio ao Marketing.

  A ***Edtech*** surgiu em 2016 e passou por um grande processo de pivot em 2020. Em todo o seu histórico, já recebeu um total 2 milhões de reais em investimento. Ainda não atingiu *break-even*, mas tem uma receita bastante alta. Segundo o próprio **Professor**, pode ser considerada um startup em *early stage* tanto por renda quanto por maturidade de produto.

- **Corretor:** Já tinhas experiência acumulada de cerca de 5 anos em startups antes de entrar na que atua desde 2016 (doravante chamada de **Imobiliária**). Assumiu durante quase toda a carreira o papel de programador, mas hoje se enquadra em um perfil mais comercial e gestor de produto.

  A **Imobiliária** atua no ramo imobiliário, onde fornece ferramentas de Big Data. Foi fundada no final de 2016 e já recebeu investimento de 100 mil reais. Passou por *pivots* de produto, mudando consideravelmente a forma como o usuário interage com a plataforma.

- **Chef:** mesclava sua rotina de trabalho entre os papéis de *CEO* e *CTO* em razão do quadro mínimo de membros da startup (doravante chamada **Restaurante**).

  O **Restaurante** já encerrou suas operações, que duraram 4 anos. É do ramo da alimentação e *food service* e já recebeu 100 mil em investimentos. Passou por 2 grandes *pivots* de estratégia de negócio.

- **Filantropo:** atua como *CTO* desde a fundação de sua startup (doravante chamada de **Fundação**). Pouco antes de co-fundar a startup onde trabalha no momento, teve uma breve experiência em outras iniciativas em startup *early stage*.

  A **Fundação** já recebeu, nos seus 5 anos, investimento de mais de 1 milhão e 200 mil reais e um outro de 500 mil dólares. Atua no mercado da filantropia. Já passou por vários *pivots* nesse ano, que envolviam grandes alterações técnicas e de negócio.

- **Garçom:** Trabalhou como programador no **Restaurante** durante a maior parte da sua operação, e brevemente também na **Fundação** entre 2019 e 2020. Hoje trabalha

em uma startup já não mais considerada *early stage*, e por isso todas as suas respostas farão referência à sua experiência nas duas empresas citadas anteriormente.

- **Atleta:** ocupa há cerca de 3 anos e meio o cargo de *CTO* na atual empresa (doravante chamada de **Estádio**). Antes disso, estagiou em uma outra startup.

  O **Estádio** existe há certa de 6 anos, nos quais já passou por dois grandes episódios de *pivots*. Atua no mercado de esportes e já recebeu 100 mil reais em investimento.

5.3 Análise das Respostas por Tópico

Para fins de organização e compreensão em cada tema, foi feita uma análise por questão da entrevista. Como cada questão ilustra uma disciplina da Engenharia de Software, esse tipo de análise proporcionará um entendimento sobre como os entrevistados, de maneira geral, enxergam esse tema. É interessante também destacar as peculiaridades de cada entrevistado em relação aos demais. Também é uma oportunidade para verificar se existe consenso sobre cada um dos temas entre as visões dos entrevistados.

5.3.1 Metodologia de Desenvolvimento

Existe uma variação sobre quanto de metodologia os entrevistados alegaram usar. As respostas variaram entre o completo AdHoc, isto é, sem nenhum tipo de metodologia estabelecida; e, o que foi mais comum, o uso de princípios ágeis.

Um detalhe importante sobre o escopo da pesquisa é que ela foi realizada durante a pandemia de COVID-19 que acontece desde o início de 2020. Por isso, todas as empresas que responderam sobre como atuam hoje, responderam em um contexto de trabalho remoto. Portanto, todas as atividades de trabalho se davam dessa forma, não tendo nenhuma atividade presencial. A única exceção é o **Restaurante**, que encerrou suas operações antes da pandemia, e os entrevistados (**Chef** e **Garçom**) respondem com base na experiência que tiveram durante seu período de atividade da startup.

5.3.1.1 Organização das Entregas em Função do Tempo

Metade dos respondentes alegaram o uso de *sprints* semanais para organização de entregas, que eram separadas por reuniões de planejamento semanal. Essas reuniões foram sempre de nível decisório e quase nunca técnicas, com apenas uma exceção que será discutida posteriormente.

As duas empresas que alegaram não utilizar a divisão das entregas em *sprints* são o **Restaurante** e o **Estádio**.

No caso no **Estádio**, o entrevistado disse estar em um cenário de muita incerteza, tanto em relação às demandas de negócio, quanto em relação à natureza técnica das

implementações, pois metade do time de desenvolvimento é considerado "júnior"e tem pouca autonomia para assumir grandes desafios técnicos. E mesmo sem esse risco da falta de experiência, o entrevistado afirma que dentro de um período de 14 ou até mesmo de 7 dias as prioridades podem mudar, e por isso o uso de *sprints* não faz muito sentido.

Os dois entrevistados que relataram sobre o **Restaurante** colocaram uma situação semelhante. No caso desses dois, a prioridade máxima é sempre entregar o mais rápido possível, por isso a única organização em termos de tempo é a estimativa de *deadlines*.

A estratégia de estimativa de *deadlines* também foi usada no começo da **Fundação**, que tinha apenas o **Filantropo** como desenvolvedor. Citou que, na época, a prioridade também era entregar o mais rápido possível. E, embora tenha funcionado razoavelmente bem com um time de apenas uma pessoa, com o passar do tempo e aumento do time e da complexidade das entregas, o uso de *sprints* se mostrou bastante proveitoso.

Hoje, a estratégia da **Fundação** é um pouco diferente da seguida pelas demais que utilizam *sprints*. O **Filantropo** mencionou a ideia de um "prazo definido para um escopo variável", isto é, se estabelecem metas para um período de tempo, mas não necessariamente um escopo de entregáveis técnicos. A ideia é atrelar esses entregáveis técnicos (como uma *feature*, por exemplo) a uma meta, mas não fixá-los de modo que sem esse entregável a meta não poderia ser batida. O entregável técnico é hipótese sobre como essa meta pode ser atingida, mas pode ser que se demonstre uma hipótese incorreta, e isso pode ser ajustado dentro do período de tempo definido.

5.3.1.2 Reuniões e Rituais

Todas as empresas entrevistadas podem ser consideradas empresas pequenas, não tendo muito mais do que 10 funcionários cada, e um time de desenvolvedores que raramente passa de 5 membros. Dada a natureza enxuta de membros das startups, todas incluíram reuniões decisórias ou administrativas onde desenvolvedores também participavam. Não necessariamente com voz ativa de decisão, mas pelo menos como um ouvinte direto das camadas de maior poder de decisão. Para as empresas que dividiam suas entregas em *sprints*, essa é a reunião que marca a transição entre uma *sprint* e outra.

Novamente o **Estádio** e o **Restaurante** surgem como exceções. No caso do **Restaurante**, o **Chef**, que também assumia papel de *CEO*, passava demandas ao time de desenvolvimento sempre que julgava adequado, e também participava das atividades de desenvolvimento quando necessário. Já no **Estádio**, o **Atleta** disse que as decisões da empresa seguiam uma estrutura *top-down*, isto é, as camadas decisórias da empresa decidiam entre si, e o **Atleta**, na posição de *CTO*, passa as demandas para o time de desenvolvimento quando julga adequado. Essa lógica *top-doown* também foi encontrada na **Imobiliária**.

Um outro ponto comum entre vários entrevistados foi a prática de *daily meetings*. Alguns dos entrevistados mencionaram *daily meetings* síncronas, por videoconferência ou audioconferência. Um fator comum é que, na transição do regime de trabalho *in loco* para o remoto, as *daily meetings* perderam um pouco do rigor de *time boxes* e de pauta pré-estabelecida, e passaram a ser um ponto de contato importante para o time de desenvolvimento socializar um pouco.

Alguns outros mencionaram que não utilizam *daily meetings* por já existir um contato muito frequente entre os membros do time de desenvolvimento. O **Imobiliário** citou uma ferramenta chamada *Discord*, onde é possível criar salas de audioconferência ou videoconferência permanentes, e onde o time se conecta quando trabalha. Dessa forma, é possível manter o contato frequente e o alinhamento entre os membros de forma orgânica.

A única exceção encontrada de uma cultura de reuniões exclusivamente técnicas foi encontrada na **Fundação**, em um cenário de mais maturidade e complexidade do produto. Essas reuniões são feitas para discutir detalhes técnicos das implementações e são realizadas em razão de uma dificuldade que o time encontrou de conseguir estimar carga de trabalho que coubesse em uma *sprint*. O **Filantropo**, buscando soluções com colegas programadores de outras empresas, recebeu o conselho de que ele planejava pouco as alterações, e que um melhor planejamento daria uma melhor dimensão da complexidade das implementações que devem ser feitas. A partir disso, o **Filantropo** passou a gastar algumas horas por semana dedicadas apenas ao planejamento técnico junto à equipe, e disse que isso foi bastante produtivo, tanto para estimativas que auxiliem o planejamento a nível macro da empresa, quanto para melhor direcionar o time para um melhor caminho para implementar a demanda.

5.3.1.3 A Preferência pelo *AdHoc*

Em todo caso, não se nota nenhum rigor sobre como é o cotidiano de trabalho dos times de desenvolvimento investigados, e isso parece uma decisão consciente.

No caso do **Professor**, ele disse "partir sempre do princípio de trabalhar em cima de problemas". Isto é, resolver pontos dentro da engenharia de software da **EdTech** apenas quando necessário. Mais sobre isso será discutido sob a luz da engenharia de requisitos.

O **Filantropo** trouxe uma visão geral sobre o histórico da **Fundação**, que começou com apenas ele como programador e aumentou gradualmente a complexidade do time e das tarefas entregues. É possível notar a implementação de novas metodologias e forma de organizar o trabalho à medida que essa complexidade aumenta, corroborando assim com a ideia trazida pelo **Professor** de trabalhar sempre sobre as demandas atuais.

O caso mais extremo de *AdHoc* com certeza foi o do **Restaurante**, que definiu sua estratégia de engenharia de software como "sentar a bunda na cadeira e programar".

E essa é uma postura deliberada, e não ocasionada como um problema causada por incapacidade do time. O **Chef**, que também assumia a posição de *CEO*, explicou que sua principal prioridade era a validação de que o produto a ser desenvolvido seria bem aceito pelo mercado, e que isso deveria ser testado o mais rápido possível. Portanto, não fazia sentido implementar qualquer rigor que enrijecesse o processo de desenvolvimento de software. Mais consequências dessa decisão serão discutidas a seguir.

5.3.2 Manutenção e Evolução do Software

Todos os entrevistados relataram algum nível de dificuldade com a manutenção do software, como era esperado pela investigação prévia. Quando perguntados sobre como a metodologia de desenvolvimento e manutenção dialogavam, isto é, se uma poderia representar um problema para a outra, unanimemente a resposta foi sim: a metodologia com pouco rigor e feita de modo *AdHoc* é um problema para manutenção. Mas esse não é o único problema, muitos mencionaram também as peculiaridades de regras de negócio que também afetam negativamente a realização de manutenção do software.

Em resposta a isso, muitos dos entrevistados seguem o mesmo raciocínio apresentado pelo **Professor** anteriormente, inclusive o próprio, que consiste em buscar resolver estas questões apenas quando elas representam um problema a nível de empresa. Isto é, a manutenção é negligenciada deliberadamente até que atinja um nível crítico, que prejudique os resultados da empresa.

Segundo o que foi coletado em entrevista, a maior prioridade no começo de uma startup é *time-to-market*, isto é, colocar o produto em produção o mais rápido o possível. Essa é a maior característica de negócio que representa uma dificuldade para práticas de manutenção, por isso existe a aceitação da não realização dessas práticas, mesmo que se reconheça a dificuldade que essa decisão possa representar no longo prazo.

5.3.2.1 Estratégias de Manutenção

Foram encontradas algumas estratégias para reduzir o problema da manutenção de modo barato. O **Professor** disse manter uma cultura de *Clean Code*, e sempre que um desenvolvedor toca um parte de código que por ele mesmo não for considerada bem escrita, que se faça uma rápida refatoração seguindo os princípios do que a empresa considera como código limpo. Essa é uma proposta de solução a nível cultural e não apresenta rigor metodológico, o que é consistente com tudo o que foi apresentado até o momento.

Uma outra prática utilizada pelo **Professor** é o uso intensivo de código *serverless* e *cloud functions*, que são tipos de estrutura lógica, semelhante a um micro-serviço, que não depende diretamente de nenhuma outra parte do sistema e tem pouco acopla-

mento com qualquer dependência externa, inclusive com o ambiente onde é implantada. Esse desacoplamento e, principalmente, isolamento do código, facilita a compreensão e manutenção desse código.

E mesmo com essa facilitação, na **EdTech** ainda se trabalha com lógica de "código descartável". O **Professor** citou vários exemplos de partes do sistema que tiveram que ser inteiramente trocadas em razão de mudanças nas regras de negócio. Portanto, faz sentido aceitar débitos técnicos em partes de código que estão isoladas e podem ser trocadas a qualquer momento, pois isso reduz o risco de esse débito se tornar um problema a longo prazo.

O **Filantropo**, por ser o único programador que está na empresa desde sua abertura, durante muito tempo foi o que tinha maior conhecimento sobre as várias partes do produto de software. Por isso, durante muito tempo julgou que seria a pessoa mais indicada para resolver qualquer problema de manutenção. Principalmente quando era um problema urgente, como um *bug* em produção. Porém, essa prática passou a onerá-lo muito, o que foi sendo um empecilho para que pudesse exercer plenamente sua função de *CTO*. Para resolver isso, passou a alocar um membro rotativo por *sprint* para se dedicar exclusivamente às práticas de manutenção. Disse que a estratégia foi muito positiva, e chegou ao ponto que hoje existem outros programadores que conhecem o sistema melhor do que ele próprio, o que é bom, já que sua função hoje é mais administrativa e estratégica do que técnica e operacional.

Para o **Corretor**, interrupções também são um problema, mas julga que o uso de *sprints*, quando bem organizado, pode ser bastante proveitoso, pois auxilia a decidir quais problemas ligados à manutenção vão ser resolvidos e quando vão ser resolvidos.

5.3.2.2 Agravantes do Problema de Manutenção

Como foi mencionado pelo **Professor**, código bem escrito ajuda a evitar problemas relacionados à manutenção do software. Em contrapartida, código que não apresenta um bom nível de qualidade relacionada à manutenibilidade pode ser um agravante.

Um fator relacionado a esse tipo de código é a experiência dos membros da equipe de desenvolvimento. O **Garçom** conta que sua experiência no **Restaurante** foi uma das suas primeiras experiências profissionais e que, mesmo tendo a vontade de aplicar as boas práticas que já tinha aprendido na teoria, não havia tempo para aprender a executar essas boas práticas dada a pressão do *time-to-market*.

Outro fator contribuinte pode ser a própria natureza de complexidade do produto desenvolvido. O **Atleta** conta que seu software é bastante distribuído, com vários diferentes *front-ends* consumindo um mesmo *back-end*. Essa característica gera acomplamento entre as diferentes partes do código, e muitas demandas de manutenção ou evolução

incluem mudança na estrutura do banco de dados, que tem potencial de desencadear mudança nos vários *front-ends* mantidos.

5.3.2.3 Impactos da Negligência

Segundo a literatura consultada, a prioridade inicial de uma startup é *time-to-market*, e isso foi comprovado pelas entrevistas, e todos julgam que a negligência à manutenção faz sentido para priorização da velocidade de entrega. Portanto, existe um impacto positivo da negligência.

Entretanto, eventualmente a startup vai se preocupar com a escalabilidade e com a qualidade do produto desenvolvido. Nesse ponto, os débitos técnicos gerados anteriormente passam a causar impacto negativo. *Time-to-market* não deixa de ser uma prioridade, e o acúmulo dos débitos começam a impactar na velocidade com a qual o time de desenvolvimento entregava anteriormente. Isso se encaixa com o que foi dito pelo **Professor** de trabalhar em questões técnicas apenas quando se tornam um problema. Enquanto a manutenção não representar um risco imediato para os objetivos da empresa, não há porque despender recursos, que poderiam ser aplicados em outras áreas, na tentativa de consertar algo que nem mesmo é considerado um problema atualmente.

E apesar disso, o modo flexível com o qual é levada a manutenção ainda se demonstra bom, senão necessário às demandas específicas de uma startup. O **Chef**, da única startup que foi desligada dentre as estudadas, disse que a manutenção era um problema bastante sério, mas que se fosse resolvido, mais tempo e dinheiro teria sido gasto antes que a empresa tivesse exatamente o mesmo fim que teve, pois o real motivo do encerramento da empresa não estava na engenharia de software. Ou seja, o ato de não resolver esse problema foi na verdade uma economia de recursos.

5.3.3 Proximidade do Usuário no Processo de Manutenção

Todas as empresas estudadas afirmaram realizar ou ter realizado algum processo de descoberta com usuários ou potenciais usuários do produto a ser desenvolvido. Esse processo de descoberta consiste em estudar o problema que o produto se propõe a resolver e descobrir quais as melhores maneiras de abordar esse problema. Na **EdTech**, inclusive os desenvolvedores participam do processo que chamou de *cust-dev*, uma abreviação de *Customer Development*, que é uma metodologia desenvolvida por Blank (2012). Nas demais empresas, esse processo também era empregado, mas apenas por membros de camadas decisórias, que se viam na necessidade de entender profundamente o mercado em que a empresa estava inserida, ou então por membros da empresa cuja função era a de entender com profundidade as necessidades dos clientes. Em todo caso, esse entendimento sempre serve de insumo para priorização das demandas da empresa, que serão discutidas adiante, em engenharia de requisitos.

Em alguns casos, a relação próxima com o cliente usuário serve inclusive de monitoramento do surgimento de *bugs* no sistema. O **Atleta** conta que a natureza de customização do produto do **Estádio** torna quase impossível uma grande cobertura de casos de teste, qualquer que seja a estratégia definida, e que o cliente assume um papel fundamental em reportar essas falhas. O **Chef** conta que no primeiro dia de operação em produção do **Restaurante**, "haviam 3 clientes e 30 *bugs*".

O **Corretor** também relata que mantém um contato direto com os clientes, que por vezes é usado para não só relatar eventuais *bugs*, como também para cobrar a sua resolução. Nesse ponto, a divisão das tarefas em *sprints* mostra seu ônus, pois na maioria das vezes, quando se segue essa divisão de tempo, é preferível esperar até a próxima *sprint* para resolver o *bug* relatado.

Na **Fundação**, existe uma cultura "**data driven**", onde boa parte das decisões são fundamentadas em dados coletados sobre o uso do produto oferecido. Portanto, é importante mencionar que, mesmo indiretamente, os usuários do produto têm um peso muito alto nas decisões de priorização do sistema, decisões estas que esbarram na área de manutenção.

5.3.4 Engenharia de Requisitos

A metodologia de rastreabilidade de requisitos usada unanimemente foi *Kanban*, que é aplicada em várias ferramentas citadas pelos entrevistados, como *Trello*, *Asana*, *GitHub Projects*. Fora isso, semelhante a tudo o que já foi exposto até o momento, raramente existe rigor metodológico sobre como lidar com os requisitos de software.

5.3.4.1 Priorização

A priorização será discutida antes da elicitação em razão do modo como as startups estudadas fazem esse processo. Como mencionando anteriormente, existem processos que aproximam as camadas decisoras da empresa à realidade do cliente ou usuário, e é através dessa aproximação que se faz a priorização. A elicitação costuma acontecer seguindo uma ordem de priorização pré-estabelecida.

Cada startup tem o seu modo de priorizar, mas nenhuma segue uma metodologia formal. No geral é um conjunto de critérios de alguma forma objetivos, mas que ainda carecem de alguma intuição, muitas vezes chamada de "*feeling*"pelos entrevistados. A relação custo-benefício, por exemplo, sempre está presente entre esses critérios, a medida tanto do custo quanto do benefício são tomadas no chamado "*feeling*". As camadas decisoras técnicas, como *CTO* e *PO*, (quanto há) são os responsáveis por estimar esses critérios.

Alguns exemplos de critérios citados pelo **Corretor** são: a frequência com que determinado pedido é feito pelos clientes; o impacto de determinada alteração na experiência

do usuário.

O **Estádio** passou por momentos financeiros muito conturbados, e por isso, por algum tempo, seu único critério de priorização foi retorno financeiro imediato, para que a empresa não se visse obrigada a fechar as portas.

5.3.4.2 Elicitação e Rastreabilidade

Como Kanban foi uma metodologia amplamente citada, essa também se demonstrou uma ferramenta útil no que se refere ao registro e especificações das tarefas que devem ser realizadas. Isto é, das ferramentas citadas para uso de Kanban, todas permitem campos de descrição, onde é possível detalhar em longos textos quaisquer informações necessárias para a execução da tarefa em questão. Para algumas startups, em um maior grau de detalhe, para outras, quase nenhum.

Um exemplo de tarefa descrita como muito documentada e detalhadamente elicitada foi o do **Estádio**. Levando em consideração que o produto desenvolvido é de grande complexidade e apresenta desafios de acoplamento já descritos aqui, existe a necessidade de documentar muito bem quais partes do sistema serão afetados em cada alteração proposta. Além disso, como não há nenhum tipo de reunião de planejamento técnico, a documentação em cartão de Kanban é bem vinda para assegurar alinhamento entre quem define as tarefas e quem as executas, pois a lógica nesse caso é a de "*top-down*".

Em vários outros casos, foi encontrada uma inversão a essa lógica *top-down*. Embora as tarefas ainda se mantenham priorizadas pelas camadas decisoras da empresa, detalhes técnicos da implementação eram decididos por quem seria responsável por implementar, ou seja, pelos próprios membros do time de desenvolvimento. As descrições das tarefas são entregues em alto nível de abstração, apenas com detalhes do que deve acontecer no ponto de vista do usuário e critérios de aprovação, mas sem entregar instruções de como atingir esses critérios.

5.3.4.3 Pós-Rastreabilidade

O processo de "aceitar uma tarefa"como realizada varia, com diferentes implementações que podem auxiliar a empresa a garantir que uma entrega realmente foi feita adequadamente.

Peer-review é uma prática comum, normalmente executada pelo *CTO* ou algum outro membro do time de desenvolvimento que tenha mais experiência. Essa revisão pode acontecer de dois modos: técnica, averiguando a qualidade do código escrito e da solução proposta; ou funcional, executando o software manualmente e averiguando detalhes da implementação sob o ponto de vista de um usuário. A revisão técnica é mais comum, enquanto que as funcionais são feitas geralmente para grandes entregas antes de elas

entrarem em produção. É muito comum, inclusive, que membros não técnicos da empresa possam testar a solução proposta em ambiente de homologação. Por exemplo, o **Professor** conta que normalmente as demandas técnicas são trazidas por outras partes da empresa, como por exemplo de designers, e que, nesses casos, é o próprio autor da demanda que tem a responsabilidade de averiguar se essa tarefa foi realizada como esperado.

Na **Fundação** existe um processo um pouco mais robusto. Como mencionado anteriormente, muitas das demandas técnicas se baseiam em hipóteses de impacto representado em dados que descrevem o uso do produto. Ou seja, se as hipóteses estiverem corretas, cumprir uma tarefa com sucesso significa uma mudança nos dados coletados. Por isso, esses dados acabam servindo como uma fonte de pós-rastreabilidade das hipóteses, que atuam como requisitos no contexto da **Fundação**.

5.3.5 Documentação de Software

Documentação é um assunto sem muito consenso entre os entrevistados. Quando se fala do quanto se deve documentar, as respostas variam de nada a quase tudo, passando pelos vários pontos intermediários entre os dois extremos. A única convergência, novamente, é a falta de rigor e formalidade nas documentações, pois documentações orgânicas ao invés de engessadas, desde que compreensíveis, são mais fluídas e se adéquam melhor à variabilidade à qual se submete uma startup.

5.3.5.1 Por que Não Documentar

Documentação implica em adição ao trabalho de implementar determinada tarefa, e é um trabalho que não gera valor direto ao cliente ou usuário. Lembrando que a prioridade das startups é *time-to-market*, adicionar mais trabalho sem a adição proporcional de entrega de valor ao mercado não faz sentido.

Tanto o **Chef** quanto o **Garçom** concordam que a documentação pode representar um risco ao *time-to-market* e, portanto, à vida de uma startup no estágio em que estava o **Restaurante**.

Em adição a isso, lembrando a lógica de "código descartável"trazida pelo **Professor** no assunto de manutenção, é possível que o código mude com muita frequência e de maneira abrupta, fazendo com que a opção mais adequada em alguns momentos seja simplesmente refazer determinadas partes do sistema, invalidando assim toda a documentação que foi escrita para descrever essa parte do sistema.

5.3.5.2 Por que Documentar

Ainda trazendo *time-to-market* como uma prioridade, a documentação pode ser uma importante ferramenta na agilização da entrega de software, pois pode ser muito mais

fácil para um desenvolvedor que tenha uma documentação de qualidade à mão entender o que deve ser feito para entregar determinada tarefa, e quais partes do sistema devem ser alteradas para a conclusão dessa tarefa. O **Corretor** foi o que apresentou maior esforço e variedade de documentação e julga bastante positivo seu uso.

No caso da já descrita grande complexidade de produto do **Estádio**, a documentação se fez indispensável para que os desenvolvedores, principalmente os considerados mais juniores, pudessem realizar com mais agilidade as tarefas passadas sem que isso representasse um grande risco para a integridade do sistema. E isso sem uma metodologia de testes bem implementada, que será discutida adiante.

Foi mencionado anteriormente um grande esforço do **Filantropo** em manter o planejamento técnico das entregas, que foi bastante proveitoso para a empresa como um todo. E ele julga que documentar as decisões tomadas nessas reuniões técnicas, isto é, detalhar e expor em documento as decisões para que ficassem de fácil acesso posteriormente, foi um processo essencial para a boa execução e bons resultados desse esforço de planejamento.

5.3.5.3 O que é Documentado

Nos exemplos citados acima, onde a documentação visava cobrir desafios de cunho técnico, naturalmente a documentação pendeu para um perfil mais técnico, portanto se trata de documentação arquitetural, que descreve como cada parte do produto se relaciona com as demais, o que inclui principalmente interações com API's desenvolvidas pela startup.

O **Estádio** documenta também *codestyle*, na tentativa de manter um alinhamento entre os membros da equipe em relação a como deve ser o código que compõe o sistema. O **Corretor** citou uma vasta gama de diferentes documentações que são usadas: justificativas das decisões de engenharia de software; guia de ingresso no projeto para inclusão de novos desenvolvedores no time (que citou ser útil inclusive para que já está no time há algum tempo); funcionamento de API e tratamento de erros.

Já o **Professor**, assumindo uma postura mais enxuta sobre documentação, defende que código bem escrito já é uma boa documentação, o que é coerente com usa postura de tentar manter na **EdTech** uma cultura de código limpo. Isso somado à prática de *cloud functions* permite documentação bem minimalista, no próprio código, especificando o que cada função faz e descrevendo seus *inputs* e *outputs* sem grande esforço. Além disso, ele concorda com outros entrevistados que falaram sobre a importância de documentar decisões e regras de negócio.

5.3.6 Testes e Controle de Qualidade

Entre os entrevistados que relataram uso de teste, foram identificadas apenas duas práticas comuns: testes unitários automatizados, e testes funcionais manuais.

Lembrando mais uma vez que as startups priorizam o *time-to-market*, e teste é uma outra carga de trabalho muitas vezes julgada como não essencial para a entrega de valor ao mercado, e por isso pode ser negligenciada. Somando isso à instabilidade do cenário das startups, que a qualquer momento podem descartar determinada parte do sistema, faz com que simplesmente não valha a pena utilizar *frameworks* robustas de testes automatizados. Problema semelhante ao encontrado quando se fala em documentação. O **Atleta**, relembrando o cenário de complexidade do produto e inexperiência do time, estima que entregar uma *feature* testada corretamente pode consumir o triplo do tempo que uma testada de modo manual.

Mas teste manual não necessariamente é feito de modo completamente *AdHoc*. Algumas startups citam procedimentos e ferramentas que auxiliam na execução desses testes. *Storybook* foi uma ferramenta citada por vários entrevistados, que ajuda a verificar como componentes visuais do sistema se comportam isoladamente e permite testá-las com vários diferentes *inputs* para checar se o comportamento desses componentes é como esperado. Além disso, algumas startups citaram estratégias para documentar o comportamento esperado de determinada alteração no código, mesmo que seja uma planilha básica que liste pontos de aceitação daquela alteração.

Porém, semelhante ao que foi encontrado quando se falou de manutenção, testes é uma disciplina cuja negligência pode ser punida rapidamente dentro de uma startup. O **Garçom**, que trabalho no cenário mais inicial entre as startups estudadas, afirma que a resolução desesperada de *bugs* foi um problema que poderia ter sido evitado com testes, e que testar teria sido mais barato do que resolver os 30 *bugs* que apareceram em produção no primeiro dia de operação do sistema. Julga que testes unitários automatizados, pelo menos nas partes mais críticas do sistema, são o mínimo que deve ser feito.

O **Filantropo** seguiu essa lógica de testes unitários apenas nas partes críticas no início da **Fundação**, mas não necessariamente por uma negligência deliberada. Além da pressão de ter que entregar rapidamente, também existia o fator da inexperiência na disciplina de testes. Esse foi um problema que pôde ser resolvido conforme crescia, pois quanto maior era a complexidade dos problemas de software enfrentados pela empresa, maior era o estágio de maturidade que ela atingia e melhor era a disponibilidade de recursos financeiros, que pôde ser convertida na contratação de mão de obra que já trouxesse consigo experiência e uma cultura mais bem definida de testes. Gradualmente foram aumentando a cobertura e qualidade dos testes realizados, e isso diminuiu drasticamente a quantidade de *bugs* encontrados em produção.

O **Corretor** também relatou boas experiências com a aplicação de testes. Como seu negócio lida com *Big Data*, isto é, volumes muito grandes de dados, o que potencialmente leva a gargalos de performance em sistemas não preparados para esses volumes, testes que garantissem o bom funcionamento do sistemas nessas condições foram essenciais. Citou o uso de testes de integração, performance e carga, todos automatizados. Relata também que trabalhar com hardware em nuvem foi essencial para a boa execução destes testes, pois até mesmo os testes locais com volumes reduzidos não performam bem em máquinas locais.

5.4 Relações entre os Assuntos

Como é de se esperar ao discutir qualquer nicho de aplicação da Engenharia de Software, os diferentes assuntos e disciplinas que a compõem se inter-relacionam. A metodologia de desenvolvimento escolhido pode, por exemplo, representar vantagens ou riscos para a manutenção do software, que pro sua vez pode aumentar ou diminuir a necessidade de testes. E o contexto da aplicação da Engenharia de Software em startups *early stage* não é diferente. A Figura 2 mapeia os achados nas entrevistas.

Se trata de um mapa conceitual, onde foram trazidos termos comuns presentes nos relatos dos entrevistados, e relacionados entre si. Para facilitar a compreensão, o mapa foi dividido em 5 seções: Metodologia de Desenvolvimento, em azul; Regras de Negócio, em rosa; Manutenção, em verde; Testes, em amarelo; e Documentação, em laranja. Muitas dessas seções têm relação direta com um dos tópicos abordados em entrevistas, ficam de fora desse relacionamento direto a Participação do Usuário e a Engenharia de Requisitos, que ainda são abordados no mapa, com seus tópicos espalhados nas seções de Regras de Negócio e Metodologia.

5.4.1 Regras de Negócio

É interessante notar que, mesmo não sendo um tópico abordado diretamente nas entrevistas, as informações sobre Regras de Negócio foram as que mais se destacaram e, ao olhar o mapa, é a seção central, tendo conexões diretas com quase todas as outras. Não surpreendentemente, pois já tinha sido um hipótese levantada em etapa de revisão da literatura, tanto bibliográfica como cinza, as Regras de Negócio são as principais característica de uma startup *early stage* que guiarão as decisões sobre como o software deve ser construído e mantido.

Em relação direta ao conceito de Regras de Negócio, temos *time-to-market* como uma prioridade, termo que apareceu inúmeras vezes ao discutir o que motiva as decisões de diferentes áreas da engenharia de software nas startups. E essa mesma priorização é um potencial prejudicial para fatores de manutenção e teste.

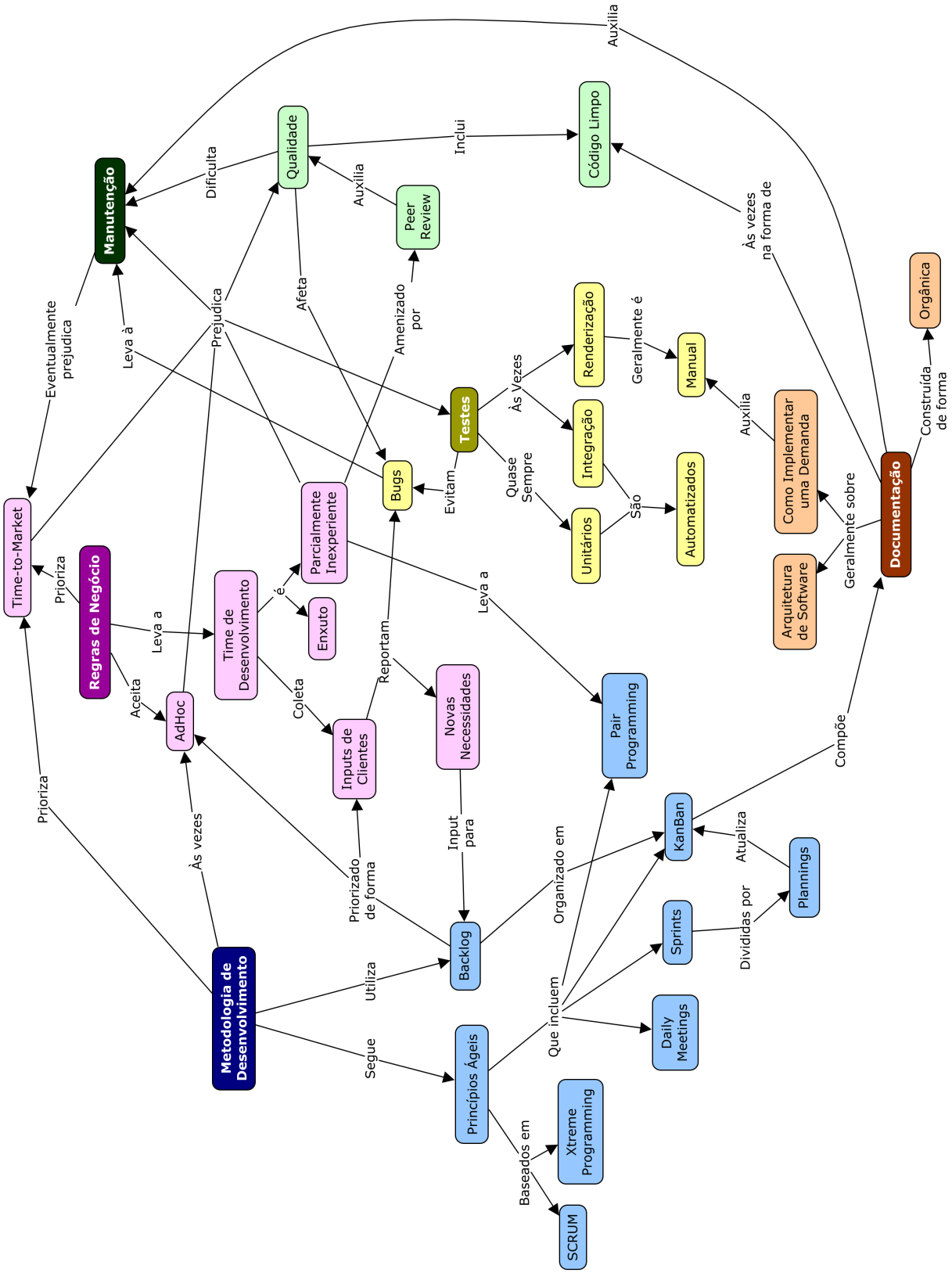

Figura 2 – Mapa Conceitual

Um outro fator potencialmente prejudicial para vários outros conceitos abordados no mapa é maneira *AdHoc* que é permitida, senão preferível, pelas particularidades de regras de negócio. Em nenhum dos casos estudados existiu nenhum tipo de pressão externa para que houvesse alguma formalidade no modo em que era executada qualquer atividade de engenharia de software.

As Regras de Negócio também imputam algumas características aos times de desenvolvimento referentes à disponibilidade de recursos. Os times frequentemente são pequenos e compostos, pelo menos parcialmente, de membros inexperientes, o que pode ser uma característica que potencializa o prejuízo descrito anteriormente.

Uma outra característica particular desse nicho é que não é incomum que membros da empresa ligadas às atividades de engenharia de software tenham também que se preocupar com a realidade dos clientes ou usuários, trabalhando assim na coleta de *inputs* desses clientes ou usuários. Essa coleta pode ser feita através de processos de descoberta, ou *Customer Development*, ou então da criação de algum sistema de coleta automatizada de dados que auxiliem o entendimento do uso do produto desenvolvido. Esses inputs dos clientes são utilizados principalmente para elicitar e priorizar requisitos que servirão como *input* para *backlog*.

5.4.2 Metodologia de Desenvolvimento

Como já dito anteriormente, tem como sua maior prioridade o *time-to-market*, podendo utilizar-se de *AdHoc* em diversas disciplinas da engenharia de software.

Das práticas não consideradas *AdHoc*, a maioria delas se baseia nos princípios ágeis, sendo as duas filosofias citadas o *Scrum* e o *Extreme Programming*. O uso de *backlog* também foi mencionado várias vezes.

Algumas das práticas usadas são: *Pair Programming*, o que inclusive ajuda a conter alguns dos riscos da inexperiência do time; *daily meetings*, que ajuda a suprir a falta de reuniões exclusivamente técnicas; organização do tempo *sprints*, geralmente semanais, e que são separadas por reuniões de *plannings*, que na maioria das vezes são reuniões de caráter administrativo e não técnico; e *Kanban*, que também serve como uma ferramenta de gestão de *backlog*, e compõe parte da documentação utilizada, pois a maioria das ferramentas de *Kanban* relatadas permitem e são usadas para documentar as definições das demandas.

5.4.3 Documentação

Não há muito para falar sobre pois não há muito consenso entre os participantes a respeito desse tema. Uns disseram ser de extrema importância e relevância, outros tratam como um risco aos focos mais importantes de uma startup.

Nos casos onde havia documentação, quase sempre eram documentados os detalhes a respeito das demandas que devem ser implementadas. Esse tipo de documentação é inclusive útil para o procedimento de qualquer tipo de teste manual.

Outro tipo de documentação achado mais de uma vez foi a de arquitetura. Mesmo não seguindo nenhum tipo de padrão rígido, ou seja, sendo construída de forma mais orgânica e intuitiva, documentar quais são os componentes do sistema mantido pela startup ajuda o time de desenvolvimento a entender onde estão as partes que são importantes em cada momento. Serve também como uma boa forma de guiar membros recém chegados aos times de desenvolvimento, que se tornarão produtivos mais rapidamente.

Uma estratégia encontrada para diminuir a necessidade de manutenção sem perder os benefícios dela é a escrita de bom código, que representa um ponto positivo para Manutenção.

5.4.4 Manutenção de software

Embora tenha sido ilustrada com poucos conceitos no mapa (Figura 2), possui muitas conexões com quase todas as outras disciplinas levantadas ao longo da análise, o que ilustra como a manutenção pode ser crítica para quase tudo o que se refere à engenharia de software no contexto estudado.

Quando se falou em Regras de Negócio, foram listados vários riscos e potenciais prejudiciais às outras disciplinas. A Manutenção é a disciplina que mais sofre com esses riscos e prejudiciais. A começar pela negligência à qualidade do código entregue. Essa qualidade representa um pilar essencial para a boa realização das atividades de manutenção, e apesar de ser prejudicada pelas regras de negócio, tem suas estratégias. Uma delas é o uso de *Peer Review*, pois, apesar de o time ser parcialmente inexperiente, é comum ter uma disparidade de experiência entre os membros do time, e o *Peer Review* será um fator de evolução dos membros menos experiente e de mantenimento da qualidade do código. Quando isso acontece, a escrita de código limpo é possibilitada, o que auxilia nas práticas de documentação.

Um detalhe que pode ser interpretado como uma contradição nas priorizações de uma startup, é que o *time-to-market* é sempre a maior prioridade, e por isso a manutenção é negligenciada. Mas isso adiciona um risco ao próprio *time-to-market*, pois o time pode despender muito tempo com impedimentos gerados pelo acúmulo de débitos técnicos e problemas ocasionados pela falta de qualidade no código, como a ocorrência *bugs*, e esse tempo não será utilizado para entregar mais valor, como seria esperado da priorização inicial.

5.4.5 Testes

Testes é uma outra disciplina que pode ser prejudicada pelas peculiaridades das regras de negócio. Entre os vários fatores que podem ser prejudiciais, a inexperiência do time é um dos vetores mais determinantes para negligência ou não às práticas de teste de software. Isso porque, lembrando da priorização de *time-to-market*, um time que não tem experiência nessa disciplina vai tomar muito mais tempo para finalizar de entregar determinada *feature* caso decida implementar alguma prática de teste. Isso pode levar a uma negligência, que leva a aparição de *bugs*, que, como já foi discutido, representará uma ameaça ao *time-to-market*.

Mas, na maioria das empresas que foram abordadas na pesquisa, pelo menos algum tipo de teste é realizado. Quando são manuais, são a nível funcional e de renderização, isto é, verificação manual das funcionalidades do software, que muitas vezes é assistida por documentação.

Quando os testes são automatizados, os achados mais comuns são testes unitários e, com um pouco menos de frequência, os de integração.

6 CONCLUSÃO

O tema da pesquisa foi motivado pela aparente disparidade entre o estado da arte da Engenharia de Software e a realidade de uma startup. Em etapa de pesquisa bibliográfica e literatura cinza, onde se buscava entender de maneira sistemática quais eram as características que definiam uma startup *early stage*, essa disparidade pareceu ainda mais evidente. Os robustos processos e rigorosos padrões de qualidade não parecem se encaixar com a instabilidade e urgência inerentes da própria definição de uma startup: uma empresa com recursos escassos, que não gera receita em primeiro momento e que, por isso, tem um tempo limitado para atingir um objetivo a ser alcançado através de ciclos de experimentação de produto, que deverão ser realizados o mais rápido o possível e provavelmente trarão uma série de alterações a um software que já se encontra em uso.

Diante disso, o presente trabalho teve como objetivo geral "identificar um conjunto de práticas de manutenção evolutiva de software utilizadas em *early startups*". Para atender a este objetivo foram definidos os seguintes objetivos específicos:

OE1: Prospectar os desafios da engenharia de software que são enfrentados em *early startups* para executar manutenções evolutivas (pesquisa bibliográfica);

OE2: Elaborar, com base nas práticas encontradas, um instrumento de coleta de dados para identificar as práticas de engenharia de software utilizadas em *early startups*;

OE3: Aplicação do instrumento de coleta de dados em *early startups*;

OE4: Mapear, com base na aplicação do instrumento, quais são as práticas de fato mais utilizadas pelas *early startups* e como essas práticas dialogam com o ambiente em que essas empresas se encontram.

Quanto ao Objetivo Específico 1 foi encontrada pouquíssima literatura que relacione práticas de manutenção de software com a realidade de uma startup *early stage*, e por isso foram escolhidas outras estratégias para levantamento de bibliografia que sirva como base. Definir, através de literatura cinza, o que é uma startup em *early stage* foi bastante positivo, pois a partir desta base, foi possível procurar literatura relacionada à manutenção que fizesse referência às práticas que foram encontradas na definição de startup. Essa busca possibilitou o cumprimento do próximo objetivo, pois serviu de base para uma melhor definição do que seria buscado no estado da prática.

Fica evidenciado que os resultados da pesquisa bibliográficos foram fidedignos quando os compara com a caracterização dos participantes da pesquisa. Todos mencionaram desafios de negócio condizentes com o que foi levantado, como será discutido adiante, nos achados.

Em relação ao Objetivo Específico 2, a pesquisa bibliográfica foi bem usada como base e bem adequada ao instrumento de pesquisa escolhido: a entrevista semi-estruturada. Tudo o que se sabia a partir da pesquisa bibliográfica é que certas disciplinas da Engenharia de Software mereciam atenção quando o assunto era manter e evoluir software dentro de startups, mas é difícil definir com precisão o que deve ser investigado a respeito dessas disciplinas. A entrevista semi-estruturada foi muito adequada porque permitiu que cada disciplina pudesse ser investigada como um tópico, sobre o qual os entrevistados poderiam mencionar tudo o que julgassem pertinente. Essa forma também permitiu que pudessem ser feitas várias perguntas que se relacionassem com cada tema, mas sem a necessidade de rigidez para a ordem dessas perguntas, fazendo com que aquelas que fizessem mais sentido para o entrevistado tivessem mais atenção, levando assim a um resultado mais orientado à realidade do entrevistado.

Para o cumprimento do Objetivo Específico 3, apesar de este trabalho ter sido aplicado em período de pandemia, onde distanciamento social é priorizado, as tecnologias de apoio que foram escolhidas para aplicação da entrevista resolveram qualquer impeditivo nesse sentido. As entrevistas por vídeo-conferência foram até vantajosas em relação ao que seria presencialmente, pois permitiram a gravação da entrevista em sua integridade, deu mais flexibilidade de disponibilidade de horários e dispensou a necessidade de entrevistado e entrevistadores estarem no mesmo local. É interessante mencionar que todos os entrevistados estavam trabalhando em regime remoto, o que provavelmente foi um fator positivo na familiarização com o modelo de vídeo-conferência.

Ao fim do processo de pesquisa, no Objetivo Específico 4, Síntese Narrativa e Mapa Conceitual foram dois métodos escolhidos para um melhor entendimento dos dados que foram coletados. A Síntese Narrativa foi bastante adequada à forma como os dados coletados, pois permitiu compilar pontos comuns e as divergências do que foi encontrado em pesquisa. Essa análise contribuiu para o que depois veio a se tornar o mapa conceitual apresentado para relacionar os vários assuntos que foram abordados ao longo das entrevistas.

Após o cumprimento de todos os objetivos específicos listados anteriormente, não necessariamente é possível dizer que o objetivo geral foi alcançado em sua completude. Não porque os objetivos específicos não estavam bem orientados, mas sim porque as demandas e limitações das startups estudadas variaram muito, fazendo com que cada uma delas adotasse o seu próprio conjunto de estratégias e práticas para manter e evoluir seu software.

Mas, mesmo não podendo definir um conjunto de práticas sobre o qual todas as startups estudadas tenham entrado em consenso, foi possível identificar as várias justificativas para cada uma das decisões de cada startup. Essa foi uma vantagem da pesquisa qualitativa em profundidade, que, apesar dos contras discutidos ao fim do capítulo, permi-

tiu um entendimento amplo sobre o que leva uma startup a seguir determinados processos e quais práticas da engenharia de software tiveram sucesso na percepção dos entrevistados.

6.1 Achados

A primeira conclusão após a análise das respostas é que a pesquisa bibliográfica para buscar definir a problemática e a realidade de uma startup estava de acordo com a realidade. Foram encontradas startups que depois de anos de operação ainda não apresentam um balanço positivo no caixa, e isso não necessariamente é um problema. Pelo menos não um problema inesperado. Todas as startups também comentaram sobre pivotagens, que representam alterações significativas no produto já oferecido a usuários e que foram motivadas por ciclos de experimentação.

Essa realidade confirma a disparidade que motivou este trabalho, e que serve como segunda conclusão da pesquisa: as práticas robustas da Engenharia de Software na maioria das vezes não se aplicam à realidade de uma startup *early stage*, principalmente no que se refere às práticas manutenção, e isso foi evidenciado por todos os relatos. Foram encontrados diferentes níveis de controle do processo, alguns podendo ser definidos como completamente *AdHoc*, mas nenhum deles apresentando qualquer grau formal de rigidez. Isso acontece não só pela prioridade ao tempo de entrega, mas também pela incerteza atrelada ao produto desenvolvido, pois não se sabe se o software entregue realmente vai atender as necessidades de negócio. Essa descoberta representa um bom cumprimento do primeiro objetivo de pesquisa (OE1) que diz respeito à prospecção de desafios da engenharia de software na literatura.

Mas é importante apontar um detalhe da conclusão anterior: as práticas robustas da Engenharia de Software não se aplicam *na maioria das vezes*, isto é, algumas são sim aplicáveis e muito bem-vindas. E isso define a terceira conclusão: práticas da Engenharia de Software que tenham baixo custo de implementação costumam ser benéficas a curto e longo prazo para as startups em *early stage*. Esse custo de implementação, geralmente definido por uma desaceleração do ritmo de entrega de novas funcionalidades, pode variar de acordo com a experiência que o time tenha em cada tipo de prática. Por exemplo, a aplicação de princípios de código limpo não adiciona nenhuma dificuldade ou tempo de implementação aos programadores que estejam familiarizados com esse conceito, e previnem problemas de manutenção que podem surgir posteriormente.

Conclui-se, também, a importância da necessária relação entre as regras de negócio e a adequação às diferentes práticas da Engenharia de Software que podem ser aplicadas em cada contexto. Por mais que existam vários problemas comuns a todas as startups estudadas, é difícil dizer que existe um conjunto de práticas que se aplicaria a todas elas. Mesmo nichando apenas startups que possam ser consideradas *early stage*, as startups

estudadas variam em nível de maturidade, e por isso todas elas têm seus próprios desafios. Enquanto uma delas mencionou testes e documentação como um prática mandatória para a boa execução das atividades, outra mencionou que documentar e testar demais é um risco ao sucesso da empresa. Analisar os recursos disponíveis e os desafios prioritários é o que define a tarefa da Engenharia de Software dentro de uma startup.

6.2 Contribuições e Perspectivas Futuras

Pode-se destacar como contribuições deste trabalho:

- Registro dos principais desafios das startups *early stage* em aplicar práticas de Engenharia de Software.
- Registro de práticas de Engenharia de Software na prática no contexto das seguintes disciplinas:
  - Metodologia de Desenvolvimento, abordando práticas de controle de processo de desenvolvimento e gestão de projeto
  - Manutenção e Evolução de Software, abordando práticas referentes às atividades de manutenção do software e aplicação de alterações em produção
  - Proximidade do Usuário, abordando práticas que façam o usuário mais próximo do processo de desenvolvimento e manutenção do software
  - Engenharia de Requisitos, abordando práticas referentes à elicitação, priorização e rastreabilidade dos requisitos de software
  - Documentação de Software, abordando práticas que sejam pertinentes a qualquer tipo de registro referente ao software desenvolvido e mantido pela startup
  - Testes e Controle de Qualidade, abordando práticas que auxiliem o controle da qualidade, verificação e validação do software

Os resultados dessa pesquisa podem ter sido limitados pelo tamanho do universo de dados disponível, mas essa foi uma limitação considerada necessária, pois foi escolhida uma abordagem de análise em profundidade ao invés de outra que, mesmo cobrindo um maior universo de dados, abordasse o contexto de startups de modo muito raso. Essa escolha foi feita em razão da pouca base que foi encontrada sobre o assunto, o que fez com que esse trabalho seguisse o caminho de construir essa base.

Uma vez que esta pesquisa trouxe várias contribuições caracterizando quais os desafios e limitações de uma startup, além de um conjunto de opções de processos e práticas de engenharia de software, pode-se dizer que os resultados encontrados abrem portas para outras pesquisas de diferentes naturezas. Por exemplo, podem ser pesquisadas

com muito mais enfoque cada uma das disciplinas abordadas nesse trabalho; podem ser pesquisadas várias das práticas encontradas, mas em caráter mais quantitativo, testando-as contra um maior universo de dados.

As limitações aqui mencionadas, somadas à justificativa apresentada no início dessa pesquisa, demonstram a grandeza e a importância de continuar estudando este tipo de negócio, aproximando cada vez mais o Estado da Arte da engenharia de software de soluções que têm potencial para quebrar paradigmas do mercado e mudarem drasticamente vários aspectos da sociedade.

Referências

BLANK, S. *The startup owner's manual : the step-by-step guide for building a great company*. Pescadero, Calif: K & S Ranch, 2012. ISBN 978-0-9849993-7-8. Citado 3 vezes nas páginas 19, 24 e 46.

CICO, O. et al. Startups transitioning from early to growth phase -a pilot study of technical debt perception. In: . [S.l.: s.n.], 2020. Citado na página 28.

FEHLMANN, S.; FALKNER, K. A case study in agility and evolving the long-lived software system. In: *ACM International Conference Proceeding Series*. [s.n.], 2015. v. 28-September-2015, p. 33–37. Disponível em: <www.scopus.com>. Citado na página 28.

FLICK, U. *Uma introducao a pesquisa qualitativa*. Porto Alegre: Bookman, 2009. ISBN 978-85-363-1711-3. Citado 2 vezes nas páginas 32 e 36.

GIL, A. *Como elaborar projetos de pesquisa*. Sao Paulo: Atlas, 2002. ISBN 85-224-3169-8. Citado na página 29.

HAYES, A. *Venture Capital*. 2021. Disponível em: <https://www.investopedia.com/terms/v/venturecapital.asp>. Citado na página 24.

IBRAHIM, K. S. K. et al. The emergence of agile maintenance: A preliminary study. In: *Proceedings of the International Conference on Electrical Engineering and Informatics*. [s.n.], 2019. v. 2019-July, p. 146–151. Cited By :1. Disponível em: <www.scopus.com>. Citado na página 35.

IEEE Standard for Software Maintenance. *IEEE Std 1219-1998*, p. 1–56, 1998. Citado na página 26.

ISO/IEC/IEEE International Standard for Software Engineering - Software Life Cycle Processes - Maintenance. *ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998)*, p. 1–58, 2006. Citado na página 26.

JUNIOR, G. S. D. A.; DANTAS, A. M. An agile approach applied to intense maintenance projects. In: *ACM International Conference Proceeding Series*. [s.n.], 2019. Disponível em: <www.scopus.com>. Citado na página 28.

KITCHENHAM, B. A. *Evidence-based software engineering and systematic reviews*. Boca Raton: CRC Press/Taylor & Francis Group, 2016. ISBN 978-1-4822-2865-6. Citado na página 33.

LAKATOS, E. M.; MARCONI, M. d. A. *Fundamentos de Metodologia Cientifica*. 7. ed. São Paulo: Atlas, 2010. Citado na página 31.

MONDEN, Y. *Toyota Production System - An Integrated Approach to Just-In-Time*. 4. ed. [S.l.]: CRC Press, 2012. Citado na página 23.

MORAES, S. *O que é uma startup e o que ela faz?* 2020. Disponível em: <https://www.sebrae.com.br/sites/PortalSebrae/ufs/pi/artigos/voce-sabe-o-que-e-uma-startup-e-o-que-ela-faz,e15ca719a0ea1710VgnVCM1000004c00210aRCRD>. Citado na página 24.

NGUYEN-DUC, A.; KEMELL, K.-K.; ABRAHAMSSON, P. *The entrepreneurial logic of startup software development: A study of 40 software startups*. 2021. Citado 2 vezes nas páginas 28 e 35.

PATERNOSTER, N. et al. Software development in startup companies: A systematic mapping study. *Information and Software Technology*, v. 56, n. 10, p. 1200–1218, 2014. Cited By :201. Disponível em: <www.scopus.com>. Citado na página 35.

PIGOSKI, T. M. *Practical Software Maintenance — Best Practices for Managing Your Software Investment*. [S.l.]: Katherine Schowalter, 1997. ISBN 0-471-17001-1. Citado na página 26.

QU, S. Q.; DUMAY, J. The qualitative research interview. *Qualitative Research in Accounting and Management*, v. 8, n. 3, p. 238–264, 2011. Cited By :469. Disponível em: <www.scopus.com>. Citado 2 vezes nas páginas 32 e 35.

RAINER, A. Storytelling in human-centric software engineering research. In: *ACM International Conference Proceeding Series*. [s.n.], 2021. p. 241–246. Disponível em: <www.scopus.com>. Citado na página 34.

REIFF, N. *Series A, B, C Funding: How It Works*. 2020. Disponível em: <https://www.investopedia.com/articles/personal-finance/102015/series-b-c-funding-what-it-all-means-and-how-it-works.asp>. Citado 2 vezes nas páginas 19 e 24.

RIES, E. *A Startup Enxuta*. 1st. ed. [S.l.]: Sextante, 2011. ISBN 0307887898. Citado 2 vezes nas páginas 19 e 23.

SIEGELE, L. A cambrian moment. *The Economist*, p. 1, 2014. Citado na página 19.

STANFILL, C.; STANFORD, K.; CHAO, J. *Venture Monitor Q1 2021*. [S.l.], 2021. Citado na página 19.

TORDRUP, L.; ROSE, J. Optimising agile development practices for the maintenance operation: nine heuristics. *Empirical Software Engineering*, v. 20, 12 2015. Citado 2 vezes nas páginas 27 e 35.

UNTERKALMSTEINER, M. et al. Software startups-a research agenda. *E-Informatica Software Engineering Journal*, v. 10, n. 1, p. 89–123, 2016. Citado na página 20.

Apêndices

APÊNDICE A – Convite para Participação em Pesquisa

O convite formal para entrevista foi enviado por e-mail a todos os que demonstraram interesse em participar. Esse convite contém um resumo do que se trata a pesquisa. Além disso, contém também o termo de aceite da participação.

A resposta oficial desse convite se dá por e-mail, para registro da aceitação do respondente.

Universidade de Brasília Faculdade UnB Gama

Convite Para Entrevista

Agosto de 2021

INTRODUÇÃO

Você está sendo convidado(a) para participar da pesquisa sobre *Estado da Prática da Manutenção de Software em Startups Early-Stage*, que se trata de um trabalho de conclusão de curso para a graduação em Engenharia de Software na Universidade de Brasília. O objetivo da pesquisa é identificar quais são os problemas relacionados à manutenção de software em startups que se encontram em estágio inicial. São buscadas startups que ainda não atingiram o *break-even* ou ainda não alcançaram o *product-market-fit*, ou seja, startups que ainda passam ou passaram recentemente por momentos de intensa experimentação.

Sua contribuição será muito importante para que a Academia se aprofunde mais nos assuntos que cercam a inovação e a disrupção do mercado no contexto de Engenharia de Software.

TERMO DE ACEITE

Caso aceite participar, sua contribuição será dada em formato de entrevista semi-estruturada, isto é, será um diálogo aberto e guiado por tópicos. Ao final do trabalho, espera-se produzir um relatório que reúna os tópicos mais sensíveis para as startups estudadas. Nesse relatório final, que será publicado e disponibilizado aos respondentes, todas as respostas serão anonimizadas, tanto a nível de quem foram os respondentes quanto a nível de quais empresas participaram da pesquisa, garantindo assim o sigilo absoluto dos respondentes.

É importante ressaltar também que a participação é completamente voluntária, podendo o respondente desistir de sua participação a qualquer momento sem que isso acarrete em qualquer forma de prejuízo.

Não há nenhum tipo de transação financeira envolvida na participação da pesquisa, ou seja, o respondente não será remunerado tampouco cobrado pela sua participação.

Para declarar seu interesse em participar da pesquisa e aceite de todos os termos, envie um e-mail para o endereço filipetoyoshima@aluno.unb.br, declarando, na mensagem, o interesse e conformidade com os termos e aguarde confirmação do recebimento. Após isso, novas instruções serão enviadas por esse mesmo canal de comunicação.

Em caso de dúvidas, basta entrar em contato por qualquer um dos canais listados abaixo:

- Email: filipetoyoshima@gmail.com
- Telefone ou Whatsapp: +55 (61) 98649-6244
- Telegram: @filipetoyoshima