Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Desenvolvimento de software em Startups: Estudo de caso na empresa Rua Dois

Autor: Iasmin Santos Mendes
Orientador: Dr. Renato Coral Sampaio

Brasília, DF
2019

Iasmin Santos Mendes

Desenvolvimento de software em Startups: Estudo de caso na empresa Rua Dois

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: Dr. Renato Coral Sampaio

Brasília, DF

2019

Iasmin Santos Mendes

Desenvolvimento de software em Startups: Estudo de caso na empresa Rua Dois/ Iasmin Santos Mendes. – Brasília, DF, 2019-

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

Orientador: Dr. Renato Coral Sampaio

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2019.

1. *Startups*. 2. Desenvolvimento de Software. I. Dr. Renato Coral Sampaio. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Desenvolvimento de software em Startups: Estudo de caso na empresa Rua Dois

CDU 02:141:005.6

Iasmin Santos Mendes

Desenvolvimento de software em Startups: Estudo de caso na empresa Rua Dois

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, 11 de dezembro de 2019:

Dr. Renato Coral Sampaio
Orientador

Dr. Fernando William Cruz
Convidado 1

Dr. John Lenon Cardoso Gardenghi
Convidado 2

Brasília, DF
2019

Dedico este trabalho a minha mãe, por sonhar e me incentivar a chegar até aqui. E ao meu pai, por todo o apoio e suporte dado durante esta caminhada.

Acreditamos no sonho e
construímos a realidade.

Roberto Marinho

Resumo

Nos últimos anos houve um crescente surgimento de empresas denominadas *startups*. Essas empresas possuem como objetivo buscar, no menor tempo possível, um modelo de negócios repetitível e escalável que gere inovação no mercado. Nesse contexto nasceu a *startup* Rua Dois com o propósito de inovar o mercado imobiliário. Contudo, durante a sua busca por inovação e escalabilidade, o processo de desenvolvimento de software foi conturbado. Levando a algumas decisões técnicas que dificultaram a evolução do software na mesma velocidade que era desejada pela a empresa. Consequentemente, foi necessário simplificar toda a arquitetura do software desenvolvido com o intuito de adaptar-se melhor ao atual contexto da *startup*. O presente trabaho realizará uma análise comparativa entre as duas arquiteturas adotadas na empresa com o intuito de avaliá-las e definir possíveis diretrizes que orientem a Rua Dois em relação as suas expectativas de desenvolvimento de software escalável.

Palavras-chave: Desenvolvimento de software. Escalabilidade. Estudo de caso. *Startup*.

Abstract

In the last years there was a growing emergence of companies called startups. These companies aim to seek, as soon as possible, a repeatable and scalable business model that creates market innovation. In this context was born the startup Rua Dois with the purpose of innovating the real estate market. However, during their quest for innovation and scalability the software development process was disturbed. Leading to some technical decisions that made it difficult for software to evolve at the same speed as the company wanted. Consequently, it was necessary to simplify the entire architecture of the software developed in order to better adapt to the current startup context. The present work will perform a comparative analysis between the two architectures adopted in the company in order to evaluate them and define possible guidelines that guide the Rua Dois regarding their expectations of scalable software development.

Key-words: Case study. Scalability. Software development. Startup.

Lista de ilustrações

Lista de quadros

Lista de abreviaturas e siglas

ACID Atomicidade, Consistência, Isolamento e Durabilidade.

API Application Programming Interface.

AWS Amazon Web Services.

BASE Basicamente Disponível, Estado suave, Eventualmente consistentes.

CRC Class, Responsibilities, and Collaboration.

CRM Customer Relationship Management.

NoSQL Not Only SQL.

TCC 2 Trabalho de Conclusão de Curso 2.

TCC 1 Trabalho de Conclusão de Curso 1.

TI Tecnologia da Informação.

XP Extreme Programming.

Sumário

1 Introdução

Existe no mundo um novo cenário emergente no qual, empresas nascentes, denominadas *startups*, produzem inovação ao ponto de serem capazes de competir com empresas já estabelecidas no mercado (DULLIUS; SCHAEFFER, 2016). O ambiente de negócios está mudando devido a fatores como globalização, política e tecnologia. Neste novo contexto, tudo que se conhece dos negócios são hipóteses e um constante alto grau de risco (NAGAMATSU; BARBOSA; REBECCHI, 2013).

Nesse cenário, surgiu a *startup* Rua Dois Tecnologias. Com uma série de hipóteses sobre o mercado imobiliário, a empresa visa inovar esse ramo que é ainda muito burocrático e carente de melhorias (XU, 2019). E nessa euforia por inovação, a *startup* tem em seu modelo de negócios a constante busca pela validação rápida de ideias mas de uma forma que ela esteja preparada pra crescer e se expandir rapidamente, conforme os princípios de repetibilidade e escalabilidade[1] de Blank (2010) para *startups*.

Nessa busca por escalabilidade a empresa tomou uma série de decisões tecnológicas baseadas no que seria escalável, mas sem uma análise coerente com o sistema proposto e com o contexto de validação de ideias que a empresa vivenciava. A exemplo está a escolha de utilizar uma arquitetura de microsserviços sem conhecer de fato a demanda dos serviços que seriam prestados, ocasionando dificuldades para a empresa de modificar e evoluir esse sistema com a rapidez que é desejada para uma *startup*.

Assim surgem as indagações do presente trabalho. Neste contexto das *startups* o software produzido é incessantemente alterado em busca da melhor solução para um determinado problema, é possível preparar esse software de forma que ele possa ser escalável? Se for possível, quais práticas e recomendações podem ser aplicadas para alcançar este objetivo?

Com foco nessas questões, este trabalho almeja realizar um estudo de caso sobre o desenvolvimento de software na *startup* Rua Dois, visando avaliar as questões apresentadas e direcionar esta empresa para um caminho mais consolidade em relação aos seus objetivos de crescimento do sistema.

[1] Os princípios de Steve Blank sobre *startups* serão abordados com maior profundidade na subseção 2.1.1.

1.1 Justificativa

As metodologias ágeis[2] visam diminuir os custos com retrabalho mediante as constantes mudanças de requisitos no processo de desenvolvimento de software, não apenas acomodando essas mudanças, mas abraçando-as. Assim, o design do software é construído de forma contínua (HIGHSMITH; COCKBURN, 2001). Mas como garantir qualidade na construção desse design sendo que esses requisitos na verdade são hipóteses a serem comprovadas sobre o software? E em paralelo a validação dessas hipóteses, preparar esse software para atender a grande demanda que é almejada para as *startups*? Ries (2011) em seu livro *A Startup Enxuta*, traz a seguinte reflexão sobre esta realidade:

> Como sociedade, dispomos de um conjunto comprovado de técnicas para administrar grandes empresas, e conhecemos as melhores práticas para construir produtos físicos. No entanto, quando se trata de *startups* e inovação, ainda estamos atirando no escuro.

Portanto, este trabalho justifica-se por sua contribuição em buscar, neste cenário citado por Ries como obscuro, compreender as necessidades do desenvolvimento de software e como essas necessidades podem ser alinhadas ao ciclo de vida de uma *startup*.

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho tem como objetivo geral realizar uma pesquisa exploratória acerca de escalabilidade de software e como esse fator se reflete no ciclo de vida de uma *startup*. Por meio de uma análise comparativa baseada em revisão bibliográfica pretende-se construir uma hipótese sobre como o desenvolvimento de software escalável deve ser conduzido dentro de cada fase do ciclo de vida de uma *startup*.

1.2.2 Objetivos Específicos

As seguintes metas foram levantadas visando atingir o objetivo geral do presente trabalho:

1. Compreender o contexto e o ciclo de vida de uma *startup*;
2. Compreender as necessidades de uma *startup* em relação ao desenvolvimento de software;

[2] Métodos iterativos para desenvolvimento de software, que visam a reação a mudanças conforme sejam as necessidades do cliente. Vide <http://agilemanifesto.org/>

3. Compreender e elencar técnicas e boas práticas voltadas o desenvolvimento de software escalável;

4. Definir um método capaz de correlacionar as técnicas de escalabilidade com as fases do ciclo de vida de uma *startup*.

5. Aplicar o método definido e identificar os pontos de intercessão entre as técnicas de escalabilidade e as fases de uma *startup*, afim de construir a hipótese citada na subseção 1.2.1.

2 Fundamentação Teórica

Neste capítulo será apresentado as bases teóricas que permeiam o escopo do presente trabalho. Partindo do entendimento sobre o contexto de *startups* até as questões relacionadas a escalabilidade de softwares. As seções estão dispostas em:

1. ***Startups*:** contexto, organização e propósitos de uma *startup*;
2. **Desenvolvimento de Software:** abordagem sobre boas práticas de desenvolvimento de software e como estas estão sendo abordadas no contexto de *startups*;
3. **Escalabilidade de Software:** definição de Escalabilidade e possíveis formas de aplicação.

2.1 *Startups*

No livro *A Revolução das Startups* de Perin (2016), o autor descreve uma Geração Y, nascida entre 1980 e 2000, detentora de um espírito jovem em busca de ser feliz fazendo algo que seja realmente impactante para a sociedade. Diferente das gerações passadas, a Geração Y possui em suas mãos a Internet e o conhecimento para utilizar esta ferramenta em todo o seu potencial.

Essa geração leva a sociedade a um contexto no qual as pessoas estão sempre se questionando se existe outra forma de fazer, se podemos fazer melhor e se somos capazes de resolver algum problema (PERIN, 2016). A partir dessas indagações nascem *startups* revoluniciárias, como o Waze[1], lançado em 2009 com a proposta de mudar a forma das pessoas se locomoverem, deixando de lado a maneira tradicional de se usarem mapas e fazendo com que as pessoas interajam e contribuam para o próprio mapeamento (NEPOMUCENO, 2013).

2.1.1 O que é uma *Startup*?

> Uma *startup* é uma organização formada para buscar um modelo de negócios repetível e escalável.[2] (BLANK, 2010, tradução nossa)

A definição apresentada por Steve Blank traz características fundamentais para entender o conceito de uma *startup*. Começando pela organização que representa um grupo

[1] Empresa que conecta motoristas visando melhorar o tráfego dentro das cidades. Saiba mais em: <https://www.waze.com>

[2] Texto original: *A startup is an organization formed to search for a repeatable and scalable business model.*

de pessoas alinhadas em prol de um mesmo objetivo. Esse grupo de pessoas juntamente com seus ideais são a essência de uma *startup*, afinal, são elas que fazem o negócio fluir somando as habilidades individuais de cada um (PERIN, 2016).

O modelo de negócio é a descrição de como a empresa cria, entrega e captura valor, exibindo todos os fluxos entre as diferentes partes da empresa, como o produto é distribuído para os clientes, como o dinheiro retorna, a estruturas de custos e a interação com outras empresas parceiras. Uma *startup* consiste essencialmente em uma organização criada para procurar um modelo de negócios, iniciando com uma visão sobre o produto e uma série de hipóteses sobre o modelo de negócios (BLANK, 2010).

Segundo Perin (2016), ser repetível significa que a ideia deve ser facilmente replicável em outras regiões, países ou até mesmo outros setores, visando diversificar os ganhos. Ou seja, quanto mais pessoas utilizam o serviço, melhor para as *startups*.

Atingir o objetivo de ser repetível carrega junto a missão de ser também escalável. Afinal, reproduzir o modelo de negócio em outras regiões e países também envolve a capacidade da empresa de crescer, preferencialmente de uma forma saudável. Para tal é necessário que os serviços sejam prestados sem demandar recursos na mesma proporção que o seu crescimento, usando somente uma estrutura básica comum a todos (SPINA, 2012).

No livro *A Startup Exuta*, Ries (2011) traz outra definição de *startup* a qual diz que "Startup é uma instituição humana projetada para criar novos produtos e serviços sob condições de extrema incerteza" . Um contexto no qual toda aprendizagem é válida e deve ser reutilizada em um ciclo constante de coleta de *feedbacks* dos clientes.

Essa segunda definição, adiciona uma nova característica ao conceito de *startup* que é o ambiente extremamente incerto no qual essas empresas estão inseridas. Com base em Rueda-Manazares, Aragón-Correa e Sharma (2008 apud GARDELIN; ROSSETTO; VERDINELLI, 2013), esta incerteza ambiental é perceptível em uma empresa quando há dúvidas, por parte dos gerentes, quanto a uma série de fatores, entre eles: a viabilidade de futuras tecnologias, as expectativas de mudanças de consumo e preferências sociais para os produtos e serviços e possíveis mudanças na legislação.

2.1.2 Fases de uma Startup

De acordo com Blank (2013) no seu livro *The four steps to the Epiphany*, o modelo de desenvolvimento que melhor se adapta ao contexto de uma *startup* é o Modelo de Desenvolvimento do Cliente, apresentado na Figura 1. Blank (2013) defende que este modelo não é um substituto para o Modelo de desenvolvimento do Produto tradicionalmente adotado nas grandes empresas, mas sim um complemento que se concentra na compreensão dos problemas e nas necessidades do cliente. Este modelo é caracterizado por quatro fases,

sendo elas:

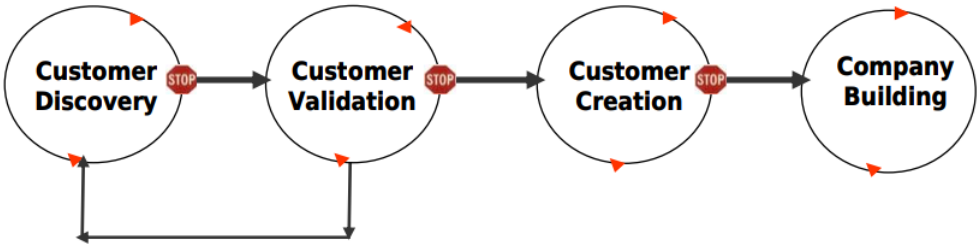

Figura 1 – Modelo de desenvolvimento do Cliente (BLANK, 2013)

Passo 1: Descoberta do cliente O objetivo desta etapa é descobrir quem são os clientes para o produto proposto e se o problema que está sendo solucionado é de fato importante para eles. Nesse sentindo, a primeira ideia e requisitos sobre o produto originam-se na verdade a partir dos fundadores da *startup*, e é papel do Time de Desenvolvimento do Cliente verificar se existe clientes e um mercado para tal visão.

Passo 2: Validação do cliente Esta etapa tem por objetivo montar um roteiro de vendas repetível para as equipes de marketing e vendas seguirem adiante. O propósito desse roteiro é guiar a equipe com base em experiências de campo nas quais a venda foi bem sucedida para os primeiros clientes. Esta validação mostra que foi encontrado um conjunto de clientes interessados e um mercado que reage positivamente ao produto.

Passo 3: Criação do cliente Com base no sucesso das vendas iniciais, esta etapa dedicasse a criar uma demanda no usuário final. Nesta etapa o produto já foi validado e agora os investimentos em marketing podem ser altos, permitindo que a empresa controle sua taxa de queima de caixa e proteja seus ativos.

Passo 4: Construção da Empresa É a etapa na qual a empresa faz a transição da sua equipe de Desenvolvimento do Cliente, orientada para aprendizado e descoberta, para departamentos formais de vendas, marketing, etc. Esses departamentos, por sua vez, devem se concentrar em explorar o sucesso inicial da *startup* no mercado.

2.2 Desenvolvimento de Software

Segundo Despa (2014) o processo de construção de um software consiste em vários estágios distintos, os quais são caracterizados por suas entregas em um período de tempo específico. Podemos classificar esses estágios nas seguintes atividades:

Pesquisa é o estágio destinado a formulação dos requisitos, na qual os envolvidos trocam informações a respeito do sistema que está sendo proposto, definem metas que

devem ser alcançadas e buscam características sobre o mercado, o comportamento do usuário e soluções técnicas;

Planejamento é o estágio destinado a definição dos fluxos que a aplicação terá que executar, com base nesses fluxos definisse também as tecnologias que serão adotadas, como será organizado o banco de dados e a metodologia que será aplicada durante o processo de construção;

Design é o estágio no qual a interface da aplicação é definida com base no seu respectivo contexto. Esta etapa permite uma validação prévia com o cliente, na qual costumam surgir novos requisitos e mudanças no sistema. Em geral, é possível realizar essa etapa em paralelo a programação;

Desenvolvimento é a etapa na qual o software é de fato construído. Inicia-se com a configuração do ambiente de desenvolvimento, o qual deve estar integrado com o ambiente de teste. Nesta fase os desenvolvedores devem estar preocupados em não inserir erros no código e deixar comentários que facilitem a manutenção depois;

Teste é o estágio destinado a identificação de erros tanto de implementação quanto de design. Nessa etapa deve ser verificado se o software está de acordo com os requisitos planejados, se não está sujeito a falhas de segurança e se existe algum problema de usabilidade;

Implantação é o estágio no qual são realizadas todas as configurações e testes necessários para disponibilizar a aplicação em um ambiente ativo para o usuário final. Nesta fase, o sistema deve ser integrado com aplicativos de terceiros, caso seja necessário, e configurado rotinas de backup;

Manutenção é o estágio subsequente a parte de implantação, e refere-se a etapa na qual são identificados novos erros que possam ter passado despercebidos, adicionadas novas funcionalidades e realizado o monitoramento do *log* para a identificação de erros e o monitoramento do tráfego, com o intuito de prever possíveis problemas de de desempenho que a aplicação possa vir a sofrer.

Os estágios mencionados acima são geralmente aceitos pela comunidade de software como os pilares do desenvolvimento de software. Estando, em sua maioria, presentes nas mais variadas metodologias de desenvolvimento, mesmo que com nomenclaturas diferentes (DESPA, 2014).

2.2.1 Manifesto Ágil

O Manifesto Ágil é um conjunto de valores que visam auxiliar a descobrir como fazer as coisas certas dentro do seu contexto de projeto. Uma das características que

destacam o Agile em relação a outras abordagens de desenvolvimento de software é foco nas pessoas que realizam o trabalho e como elas trabalham juntas. Abaixo segue os quatros valores declarados por Beck et al. (2001) no Manifesto Ágil:

1. Indivíduos e interações mais que processos e ferramentas;
2. Software em funcionamento mais que documentação abrangente;
3. Colaboração com o cliente mais que negociação de contrato;
4. Responder a mudanças mais que seguir um plano.

2.2.2 Os 12 princípios do software ágil

Junto com os valores expressos no Manifesto Ágil, o desenvolvimento ágil de software é guiado pelos doze princípios apresentados a seguir:

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

12. Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

2.2.3 Scrum

O Scrum pertence a família dos métodos ágeis e é uma metodologia de desenvolvimento de software iterativa e incremental baseada em *timboxes* e voltada para o gerenciamento do projeto. Esta metodologia parte da premissa de que o desenvolvimento de software é muito complexo e imprevisível para ser planejado exatamente com antecedência. Neste caso, deve ser aplicado o controle empírico do processo afim de garantir transparência, inspeção e adaptação (MAHNIC; DRNOVSCEK, 2005). Para tal, essa metodologia utiliza de papéis, eventos, artefatos e regras que os unem. A seguir, serão explicados cada um desses tópicos individualmente de acordo com as informações dispostas em *The Scrum Guide* escrito por Schwaber e Sutherland (2017).

2.2.3.1 Papéis

Product Owner responsável por representar os interesses de todos os envolvidos no projeto e em seu sistema resultante. Deve manter o *Product backlog* do projeto priorizado conforme as necessidades do cliente;

Time de desenvolvimento é uma equipe autogerenciada, auto-organizada e multifuncional, responsável por implementar o *Product backlog*. Os membros da equipe são coletivamente responsáveis pelo sucesso de cada iteração e do projeto como um todo;

Scrum master é o responsável por garantir a execução do processo Scrum. Deve ensinar as práticas do Scrum aos envolvidos e incentivá-los a seguir essas práticas afim de que a metodologia tenha sucesso na organização.

2.2.3.2 Artefatos

Os artefatos do Scrum são produzidos com o objetivo de fornecer transparência e permitir inspeção e adaptação, visando maximizar a transparência das informações principais com o intuito de que toda a equipe tenha o mesmo entendimento a respeito do artefato.

Product Backlog é uma lista ordenada com todas as funcionalidades que se sabe que o produto deve contemplar. Esta lista nunca está completa e é evoluída ao longo da execução do projeto com a finalidade de atender as necessidades do cliente;

Sprint Backlog é conjunto de itens oriundos do *product backlog*, com o propósito de serem finalizados durante a execução da respectiva *sprint*;

Incremento é a soma de todos os itens do *product backlog* entregues durante a *sprint* e o valor dos incrementos de todas as *sprints* anteriores.

2.2.3.3 Eventos

O objetivo dos eventos no Scrum é criar uma regularidade e minimizar a necessidade de reuniões durante o projeto. Cada evento tem um tempo determinado para acontecer, não podendo ultrapassar esse limite visando evitar desperdício de tempo durante o processo. Tais eventos são apresentados adiante:

Sprint é o *timebox* principal do Scrum com duração de 1 mês ou menos e o objetivo de criar um produto utilizável e potencialmente liberável. Durante a *sprint* são incluídos os outros eventos apresentados adiante;

Sprint Planning evento no qual o Time de Desenvolvimento, colaborativamente, planeja o que será realizado durante a *sprint* que se inicia. São levantados aspectos sobre quais atividades do *backlog* podem ser entregues ao final da iteração e como será o trabalho necessário para alcançá-las;

Daily são reuniões diárias de no máximo 15 minutos. O objetivo destas reuniões é verificar o andamento da *sprint* e planejar o que será executado nas próximas 24 horas. Dessa forma visa-se acompanhar e otimizar o andamento da *sprint*;

Sprint Review realizado ao final da *sprint* com o intuito de inspecionar e adaptar o *product backlog* com base no que foi entregue;

Sprint Retrospective realizado antes do próximo *sprint planning*, é uma reunião para a equipe se auto inspecionar e planejar melhorias para a próxima *sprint*.

A Figura 2 visa ilustrar o ciclo de vida do processo Scrum.

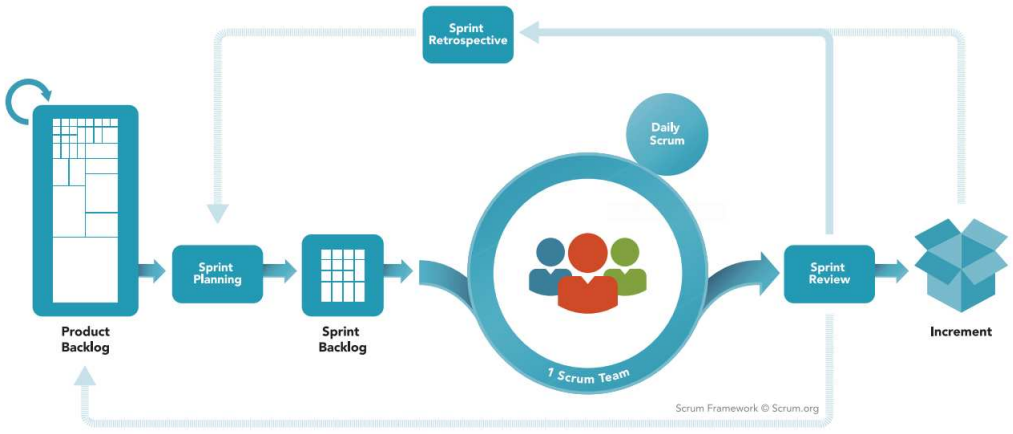

Figura 2 – Scrum Framework (SCRUM.ORG, 2019)

2.2.4 Extreme Programming

A metodologia Extreme Programming (XP) é um processo ágil bastante popular que enfatiza a satisfação do cliente incentivando os desenvolvedores a responder com confiança as mudanças nos requisitos, ainda que no final do ciclo de vida do projeto. Ela divide o processo convencional de desenvolvimento de software em blocos menores e mais gerenciáveis visando minimizar posteriores custos de alteração (WELLS, 2000; DESPA, 2014). Esta metodologia é baseada nos seguintes cinco princípios:

Simplicidade: fazer somente o que é necessário e nada a mais que isso. Assim, o valor entregue será maximizado e o projeto poderá ser mantido a longo prazo por custos razoáveis;

Comunicação: todos fazem parte da equipe e devem se comunicar diariamente sobre todas as áreas, desde requisitos a código, para que todos possam juntos construir a melhor solução;

Feedback: o software deve ser demonstrado cedo e por muitas vezes, sendo importante ouvir atentamente as observações e realizar as alterações necessárias. O processo da equipe deve ser adaptado as necessidades do projeto, e não o contrário;

Respeito: todos dão e sentem o respeito que merecem como um membro valioso da equipe. Toda contribuição é bem vinda e tem igual valor, independente se ela vem do cliente, da gerência ou da equipe de desenvolvimento;

Coragem: deve-se sempre dizer a verdade sobre o progresso e as estimativas e não ter medo de se adaptar as mudanças sempre que elas ocorrerem.

Nos sub tópicos adiante, serão apresentadas as regras que guiam a metodologia XP em cada área do processo de desenvolvimento de acordo com Wells (2000).

2.2.4.1 Planejamento

Histórias de usuário devem ser escritas. Histórias de usuário são frases escritas pelo próprio cliente, sem nenhum linguajar técnico, que visam descrever coisas que o sistema deve fazer. Muitas vezes são utilizadas no lugar de um grande documento de requisitos e servem para criar estimativas quanto ao tempo durante a reunião de planejamento;

O planejamento do lançamento cria a programação de lançamento. Deve ser realizada uma reunião para traçar um plano de lançamento que estabeleça o projeto geral. Neste planejamento são tomadas tanto decisões técnicas quanto decisões de

negócios afim de definir uma agenda com a qual todos os envolvidos podem se comprometer;

Faça lançamentos pequenos e frequentes. A equipe de desenvolvimento deve liberar versões iterativas do sistema frenquentemente, pelo menos a cada duas semanas. Assim no final de cada de iteração você terá testado e demonstrado a funcionalidade para o cliente

O projeto é dividido em iterações. O cronograma deve ser dividido em várias iterações de 1 a 3 semanas com o intuito de realizar constantemente pequenas entregas;

O planejamento da iteração inicia cada iteração. Cada iteração deve iniciar com o seu planejamento, no qual as histórias de usuários são selecionadas para serem implementadas de acordo com a prioridade definida pelo cliente.

2.2.4.2 Gerenciamento

Dê ao time um espaço de trabalho aberto e dedicado. A comunicação é um fator muito importante dentro do Extreme Programming. Aproximar a equipe fisicamente e remover barreiras físicas entre eles incentiva-os a trabalhar colaborativamente como um conjunto no qual todos tem igual valor e contribuições;

Defina um ritmo sustentável. O objetivo do XP é conseguir um software mais completo, testado, integrado e pronto para produção a cada iteração. Software incompleto ou com erros representa uma quantidade desconhecida de esforços futuros que não podem ser mensurados. Nessa perspectiva, a metodologia prega que se as histórias planejadas para a iteração não puderem ser concluídas, o ideal é fazer um novo planejamento da iteração visando definir o que será entregue até o final. Mesmo que falte somente um dia para terminar a iteração, o melhor é focar toda a equipe em uma única tarefa completa, do que terminar a iteração com várias tarefas incompletas;

Comece todos os dias com uma reunião em pé. De acordo com Wells (2000), uma reunião longa para apresentar os resultados é um desperdício de tempo dos desenvolvedores. O ideal é fazer reuniões diárias para comunicar problemas, discutir soluções e promover o foco da equipe. Essas reuniões devem ser feitas em círculos, de forma que todos se sintam a vontade para contribuir, e em pé para evitar que a reunião se prolongue por muito tempo;

Deve-se medir a velocidade do projeto. A velocidade do projeto deve ser medida com o propósito de auxiliar no planejamento das próximas iterações. A proposta do XP é estimar de forma simples a dificuldade de implementação das histórias e planejar a iteração de acordo com o ritmo que a equipe consegue produzir;

Mova as pessoas. Se apenas uma pessoa da sua equipe é capaz de trabalhar em determinada área e essa pessoa sair ou estiver sobrecarregada, o progresso do seu projeto será consideravelmente reduzido. É importante a realização de treinamentos cruzados afim de evitar ilhas de conhecimento. Os desenvolvedores devem ser incentivados a trabalhar em seção diferente do código pelo menos em uma parte da iteração;

Conserte o XP quando ele não funcionar. Algumas alterações serão necessárias na metodologia para atender as particularidades do seu projeto. O recomendado é começar seguindo as regras do XP e realizar reuniões de retrospectiva para conversar com a equipe e discutir o que não está funcionando e como o processo pode ser melhorado.

2.2.4.3 Design

Simplicidade. Faça sempre a coisa mais fácil que possa funcionar a seguir. Optar por um design simples significa menos tempo de implementação e é sempre mais barato substituir um código complexo por um mais simples agora, antes que seja perdido muito mais tempo. É subjetivo medir a simplicidade, mas o recomendado pela metodologia é que o código seja testável, compreensível, navegável e explicável. Testável significa que você pode escrever testes de unidade e aceitação para identificar rapidamente algum problema. Navegável é a qualidade de poder encontrar facilmente o que você precisa, isso é refletido em bons nomes, no uso correto de heranças, etc. A definição de compreensível é óbvia, mas altamente subjetiva. Por isso o código também deve ser explicável, ou seja, compreensível de uma forma que todos que já trabalham com o código consigam entender, e explicável de forma que novos desenvolvedores também possam entender o código;

Escolha uma metáfora do sistema. A metáfora do sistema é algo simples que possa facilmente explicar como é o design do sistema sem precisar ler várias documentações para tal. O intuito dela é orientar o desenvolvimento e facilitar a inserção de novos membros;

Use cartões Class, Responsibilities, and Collaboration (CRC) para o design. O maior do valor dos cartões é fazer com que a equipe se afaste do pensamento processual e enfoque mais nos objetos. Permitir que toda a equipe trabalhe na construção dos cartões promove um melhor um melhor projeto do sistema e um maior entendimento da equipe;

Crie soluções de pico para reduzir o risco. Quando existe dificuldade de tomar decisões sobre problemas técnicos ou designs difíceis, o ideal é criar algo muito simples que explore potenciais soluções, mesmo que este não atenda todos os problemas e resolva todas as preocupações. Muito provavelmente, este programa de teste não

será suficientemente bom para manter e será jogado fora. No entanto, o objetivo dessa ação é reduzir o risco de um problema técnico e aumentar a confiabilidade sobre a história de usuário;

Nenhuma funcionalidade deve ser adicionada cedo. Muitos desenvolvedores são tentados adicionar funcionalidades visando que é fácil adicioná-las agora e que estas poderão ser utilizadas futuramente, ou que a arquitetura se tornará muito mais flexível. No entanto, a realidade é de utilizar muito pouco essas funcionalidades extras e deixar a arquitetura mais complexa do que o necessário para o problema. Dessa forma, o XP recomenda olhar sempre para as necessidades atuais e não para as necessidades futuras;

Refatorar sempre e sempre que possível. O XP prega que você deve refatorar o código sempre que possível, remover duplicações, códigos obsoletos, alterar códigos que sejam difíceis de manejar, etc. Manter o código limpo e conciso economiza tempo ao longo da vida do software.

2.2.4.4 Codificação

O cliente está sempre disponível. O ideal é que o cliente esteja sempre no mesmo espaço que o time de desenvolvimento participando de todas as fases do projeto. É importante que esse cliente seja um especialista da organização afim de ajudar a garantir que as funcionalidades sejam implementadas conforme o desejado;

O código deve ser escrito de acordo com os padrões acordados. Manter o código em um padrão acordado pela equipe facilita o entendimento e a refatoração do mesmo. Além de incentivar a propriedade coletiva do código;

Escreva os testes unitários primeiro. Criar os testes antes do código torna mais rápido e fácil a criação do próprio código. Ajuda o desenvolvedor a considerar o que realmente precisa ser feito e a estabelecer os requisitos por meio dos testes;

Todo o código de produção é programado em pares. A programação em pares aumenta a qualidade do código sem afetar o tempo de entrega. A ideia é contra-intuitiva, mas duas pessoas que trabalham em um único computador adicionam tanta funcionalidade quanto duas que trabalham separadamente, com o diferencial da qualidade do código ser melhor;

Apenas um par integra o código por vez. A integração paralela do código significa que dois ou mais códigos que ainda não foram testados juntos serão integrados acreditando-se que está tudo bem. Contudo, essa integração se aplica também a suíte de testes e, se não existe um conjunto de testes consolidados, problemas de

integração aconteceram sem detecção. Por isso a importância de integrar um código por vez, afim de gerar sempre uma versão mais estável e clara do sistema;

Integre-se frequentemente. Os desenvolvedores devem estar alinhados e trabalhando sempre na versão mais atualizada do código. Isso evita futuros problemas de integração e reduz o tempo de desenvolvimento uma vez que evita a alteração em uma linha de código que está obsoleta;

Configure um computador de integração dedicado. O propósito é que todos os lançamentos do software para produção sejam feitos pelo mesmo computador. Assim, todos os envolvidos podem acompanhar o que está sendo integrado e a garantia de que o código está sempre atualizado;

Propriedade coletiva. Qualquer desenvolvedor deve ser livre pra alterar qualquer linha do código, seja para corrigir um *bug*, refatorar ou implementar uma nova funcionalidade. Toda a equipe é responsável pelo design da aplicação, e não cabe a uma única mente - o arquiteto chefe, por exemplo - definir como será feito esse design.

2.2.4.5 Teste

Todo o código deve ter testes unitários. No XP um código só pode ser lançado se todos os testes unitários estiverem juntos a nova versão. Caso seja identificado algum código sem teste, ele deve ser implementado imediatamente. Assim, você protege o código contra danos acidentais;

Todo o código deve passar por todos os testes unitários antes de ser lançado. Dessa forma visa-se garantir que todas as funcionalidades estarão funcionando no sistema disponibilizado para produção;

Quando um *bug* é encontrado, testes são criados. O teste deve ser criado com o intuito de sinalizar a existência do *bug* e fazer os programadores se moverem o mais rápido possível para repará-lo;

Os testes de aceitação são executados constatemente e a pontuação publicada. Os testes de aceitação são testes de caixa preta com o intuito de verificar que o sistema está cumprindo as exigências definidas nas histórias de usuários. Dessa forma o XP exige que o desenvolvimento de software tenha um relacionamento muito mais próximo com o controle de qualidade.

A Figura 3 visa ilustrar o ciclo de vida da metodologia Extreme Programming.

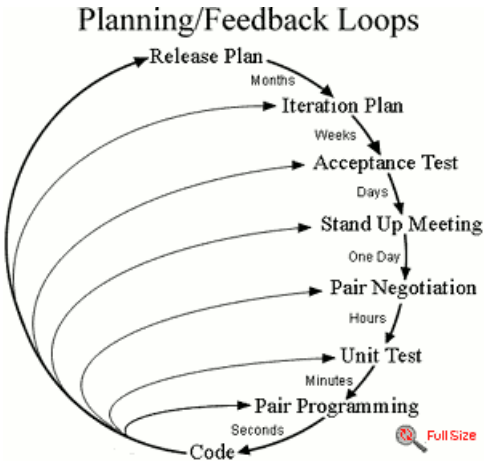

Figura 3 – Ciclo de vida da metodologia Extreme Programming (WELLS, 2000)

2.2.5 Desenvolvimento de software em *startups*

Do ponto de vista da engenharia, o desenvolvimento de software em *startups* trabalha em um contexto difícil para os processos de software seguirem uma metodologia prescritiva. Diante dos diversos fatores associados, o contexto específico de cada *startup* torna o processo de desenvolvimento único e um grande desafio para a Engenharia de Software (GIARDINO et al., 2016). Por isso, são necessárias pesquisas para investigar e apoiar as atividades da engenharia em *startups*, afim de orientar os profissionais na tomada de decisões visando minimizar os riscos do fracasso comercial. Contudo, apesar do tamanho impressionante do ecossistema de *startups*, a pesquisa sobre Engenharia de Software em *startups* apresenta uma lacuna(GIARDINO et al., 2016).

Os processos de desenvolvimento ágil são focados na solução e respondem bem a "como"criar produtos rapidamente, no entanto, eles não respondem bem a "quais"produtos construir (BOSCH et al., 2013). Nesse sentido, eles se aplicam bem a situações em que o problema é bem compreendido e a solução não é. Mas para uma *startup* nem o problema nem a solução são muito bem compreendidos (BOSCH et al., 2013).

Dessa forma, as *startups* costumam optar por práticas voltadas ao produto que lhe permitam ter uma equipe flexível com fluxos de trabalho facilmente adaptáveis a novas direções de acordo com o público-alvo (PATERNOSTER et al., 2014). Nesse contexto, a literatura indica práticas de desenvolvimento de software orientadas ao mercado, nas quais é enfatizado a importância de dedicar tempo a objetivos estratégicos específicos (PATERNOSTER et al., 2014). Assim, as *startups* enfrentam diariamente um troca entre engenharia de alta velocidade e de alta qualidade, não apenas em aspectos como design da arquitetura mas em aspectos multifacetados como gerenciamento de projetos, testes, etc. (GIARDINO et al., 2016)

2.3 Escalabilidade

Segundo Tiwana (2014), escalabilidade refere-se ao grau em que o desempenho funcional e financeiro de um subsistema é independente de seu tamanho. Podendo manter seu desempenho e função retendo todas as propriedades desejadas sem um aumento correspondente em sua complexidade interna (WECK; ROOS; MAGEE, 2011 apud TIWANA, 2014). Para Tiwana, isso implica em duas proposições pertinentes sobre a escalabilidade: a primeira é de que escalabilidade não se refere exclusivamente a aumentar a escala de um sistema, como habitualmente pensamos, mas também a capacidade deste sistema de se contrair conforme a necessidade. A segunda proposição diz que o desempenho de um sistema escalável pode significar tanto a sua capacidade técnica de desempenho quanto a sua capacidade financeira.

Assim, a escalabilidade reflete a capacidade do software em crescer e evoluir conforme as demandas do usuário. Dessa forma, é importante analisar se o modelo de negócios da empresa depende do crescimento desse sistema. Uma vez que não existe essa dependência, a escalabilidade não se torna um requisito básico, podendo um software não escalável funcionar bem com uma capacidade limitada (CONCEPTANEXT, 2017).

No texto *The Importance of Scalability In Software Design* da Conceptanext (2017, tradução nossa), a empresa Concepta [3] diz que "A escalabilidade é um componente essencial do software corporativo. Priorizá-lo desde o início leva a menores custos de manutenção, melhor experiência do usuário e maior agilidade." [4] Nessa abordagem apresentada pela Concepta, a escalabilidade vai além de atender a um grande volume de demandas de requisições, e passa a fazer parte da experiência do usuário além de influenciar positivamente a manutenção desse software. Seguindo essa visão de escalabilidade apresentada pela Concepta, existem três fatores que afetam significantemente a escalabilidade de softwares:

1. A capacidade de usuários simultâneos ou conexões possíveis. Aumentar essas capacidade deve ser tão simples quanto disponibilizar mais recursos ao software;

2. A capacidade de armazenamento, principalmente para aplicações que apresentam muitos dados não estruturados. A escolha do tipo do banco de dados que será usado e o uso de uma indexação adequada são fatores que podem influenciar diretamente a escalabilidade;

3. O próprio código. Desenvolvedores inexperientes tendem a ignorar o fato de que o código deve ser escrito para que possa ser facilmente adicionado ou modificado, sem

[3] Saiba mais em: <https://conceptainc.com>

[4] Texto original: *Scalability is an essential component of enterprise software. Prioritizing it from the start leads to lower maintenance costs, better user experience, and higher agility.*

a necessidade de refatorar o código antigo. Bons desenvolvedores evitam a duplicação de esforços, reduzindo o tamanho e a complexidade geral da base do código.

Priorizar esses pontos no início da construção do software, traz agilidade e benefícios que garantem o crescimento saudável do sistema.

2.3.1 Escalabilidade vertical

A primeira e mais direta forma de lhe dar com o aumento da demanda de um sistema é por meio da escalabilidade vertical. Esta refere-se a maximização dos recursos de uma única unidade visando expandir sua capacidade de lhe dar com o aumento de carga. Pensando em hardware, isto significa aumentar o poder de processamento e memória do servidor no qual o sistema é executado. Em termos de software esse tipo de escalabilidade está relacionado a otimização dos código e algoritmos utilizados (KHARE et al., 2012).

O escalamento vertical costuma ser bem compreendido e arquiteturalmente direto, com o auxílio da engenharia ele pode ajudar o sistema a atender seus requisitos de carga de trabalho (INTERSYSTEMS, 2019). No entanto, ele possui as seguintes limitações:

1. Mesmo com a utilização de servidores poderosos com mais de cem núcleos e terabytes de memória, existe um limite para a quantidade de conexões simultâneas que um sistema pode manter;

2. Maior capacidade de hardware implica em custos mais elevados para adquiri-los sempre que for necessário expandir a capacidade, resultando em um crescimento exponencial sobre o investimento necessário para manter tal escalabilidade;

3. O escalonamento vertical exige um planejamento cuidadoso antes de realizar o mesmo, o que pode ser feito para contextos relativamente estáticos, contudo, para contextos dinâmicos é difícil prever o futuro;

4. Não há elasticidade na escala vertical, uma vez que se aumente a capacidade do sistema, não é possível diminuí-la caso a sua demanda permita. Isto acaba gerando maiores custos por excesso de capacidade;

5. A escalabilidade vertical exige que o seu software seja adaptado para tal. Por exemplo, escalar 128 núcleos é pouca utilidade se a aplicação pode lhe dar com no máximo 32 processos.

2.3.2 Escalabilidade Horizontal

A escalabilidade horizontal refere-se ao incremento de recursos pela adição de unidades ao sistema. Em comparação a escalabilidade vertical, a escalabilidade horizontal

significa adicionar várias unidades de menor capacidade ao invés de uma única unidade com altíssima capacidade. Isso implica na distribuição da carga de trabalho entre as várias unidades existentes (KHARE et al., 2012).

Essa abordagem de escalonamento permite um crescimento gradual do sistema, em contrapartida a substituição abrupta realizada no escalonamento vertical. Dessa forma ela se torna financeiramente mais vantajosa. Além de se adaptar muito bem a infraestruturas virtuais e de nuvem, fornecendo facilmente provisionamento a medida que a carga de trabalho aumenta ou desativando unidades caso a demanda diminua (INTERSYSTEMS, 2019).

2.3.3 Bancos NoSQL

Atualmente, as aplicações têm trabalhado com um volume de dados cada vez maior. E nesse contexto, as propriedades Atomicidade, Consistência, Isolamento e Durabilidade (ACID) implementadas pelos bancos de dados relacionais, tornam-se muito onerosas mediante as necessidades específicas desses aplicativos. Assim, o custo associado ao dimensionamento dos bancos de dados tradicionais em frente ao crescente volume de dados se torna muito caro (GAJENDRAN, 2012).

Diante desta situação, desenvolvedores buscam soluções alternativas para o armazenamento de dados, dentre elas estão os bancos não-relacionais. O termo Not Only SQL (NoSQL) refere-se a um conjunto de banco de dados que não possuem as características tradicionais de um banco de dados relacional (GAJENDRAN, 2012). Estes bancos são projetados para atender uma larga escala de armazenamento de dados e processamento paralelo massivo em cima desses dados (MONIRUZZAMAN; HOSSAIN, 2013).

Para obter essa escalabilidade, os bancos NoSQL *"abriram mão"* de manter a consistências dos dados, como ocorre nos bancos de dados relacionais, resultando em sistemas conhecidos como Basicamente Disponível, Estado suave, Eventualmente consistentes (BASE). Estes sistemas não possuem transações no sentido clássico e introduzem restrições no modelo de dados para permitir melhores esquemas de partição (HAN et al., 2011 apud MONIRUZZAMAN; HOSSAIN, 2013).

Assim eles oferecem um armazenamento relativamente barato e altamente escalonável para pequenos pacotes de registros, como *logs*, leitura de medidores, *snapshots* e para dados semi-estruturados ou não-estruturados, como arquivos de e-mail, *xml* e documentos. A estrutura distribuída desses bancos também os tornam ideias para processamento massivo de dados em lote (MONIRUZZAMAN; HOSSAIN, 2013).

2.3.4 Sistemas monolíticos

A Arquitetura Monolítica é um padrão de desenvolvimento de software no qual um aplicativo é criado com uma única base de código, um único sistema de compilação, um único binário executável e vários módulos para recursos técnicos ou de negócios. Seus componentes trabalham compartilhando mesmo espaço de memória e recursos formando uma unidade de código coesa (SAKOVICH, 2017).

Esse padrão arquitetural permite um desenvolvimento fácil por ser de conhecimento da maioria dos desenvolvedores de software e apresentar baixa complexidade para a execução de tarefas como *deploy*, confecção de testes e compartilhamento do código. Contudo, a longo prazo começam a surgir dificuldades para manter essa arquitetura, entre elas estão:

1. Dificuldade em entender e alterar o código que ao longo do tempo se torna muito extenso e coeso;
2. Limitação na agilidade de atualização do software, uma vez que para cada pequena alteração é necessário reimplantar o código por completo;
3. Necessita de muito esforço para adotar uma nova tecnologia, sendo necessário adaptar todo o código da aplicação para a nova ferramenta;
4. O sistema perde a confiabilidade pois a medida que cresce um *bug* em uma única parte do código pode interromper todo o software já que todos os componentes estão conectados.

2.3.5 Microsserviços

Segundo Newman (2015), microsserviços são pequenos serviços autônomos que trabalham juntos. O objetivo desses serviços é tornar o código coeso e focado em resolver as regras de negócio dentro de um limite específico. As vantagens de adotar essa arquitetura consistem em:

1. Uso de tecnologias heterogêneas;
2. Maior resiliência do sistema;
3. Facilidade em escalar pequenas partes do sistema, ao invés do sistema por completo;
4. Facilidade e maior estabilidade no processo de *deploy*;
5. Maior produtividade da equipe, uma vez que se passa a adotar equipes menores com bases de código menores;

6. Reusabilidade dos serviços;

7. Otimização da substituibilidade: organizações de médio e grande porte costumam possuir sistemas legados, com uma base de código enorme funcionando com tecnologias antigas tanto de software quanto de hardware. A substituição desses códigos é um processo custoso e arriscado. Com serviços pequenos realizar essa substituição se torna um processo mais fácil de gerenciar.

Para o desenvolvimento de microsserviços é necessário pensar sobre o que deve ser exposto e o que deve ser ocultado. Se houver muito compartilhamento os serviços de consumo se acoplam às nossas representações internas, diminuindo a nossa autonomia e consequentemente a nossa capacidade de realizar alterações (NEWMAN, 2015).

2.3.6 Escalabilidade de software em *startups*

Segundo Stampfl, Prügl e Osterloh (2013 apud HUNGARY; US, 2017), o fator mais crucial para o potencial de crescimento de uma *startup*, tanto operacionalmente quanto financeiramente, é a escalabilidade. *Startups* que crescem rapidamente e respondem bem as mudanças provavelmente superarão as grandes empresas. A exemplo, está a *startup* Airbnb que foi fundada no ano de 2008, com sede em San Francisco - Califórnia, e rapidamente se expandiu para outros países. Dentre outros fatores envolvidos, a escalabilidade foi provavelmente um dos ativos mais decisivos no crescimento da empresa, uma vez que permitiu a ela aumentar sua capacidade em proporções superiores ao aumento do custo.

A tecnologia é frequentemente considerada um bom investimento porque tem a capacidade de permitir a escalabilidade. No entanto, se um empresário quer escalar rapidamente pode cair na armadilha da escalabilidade prematura (HUNGARY; US, 2017). Esta escalabilidade prematura pode estar associada ao produto, ao cliente, à equipe, ao modelo de negócios ou às finanças (MARMER et al., 2011).

O relatório *Startup Genome Report Extra on Premature Scaling*[5] confeccionado pelo projeto Startup Genome Project apresenta alguns fatores de inconsistência que sinalizam o escalamento prematuro, dentre eles está o fato de muitas *startups* investirem na escalabilidade da tecnologia antes mesmo deste produto está ajustado ao mercado (MARMER et al., 2011).

Brikman (2015) defende que a equipe de Tecnologia da Informação (TI) dentro de uma *startup* deve estar preocupada com a escalabilidade em dois aspectos. O primeiro refere-se a escalabilidade do código, ou seja, dimensionar as práticas de codificação para lidar com mais desenvolvedores, mais código e maior complexidade. O segundo aspecto é

[5] O relatório está disponível neste link.

dimensionar o desempenho do código para lidar com mais usuários, mais tráfego e mais dados.

> Escalar uma *startup* é um pouco como mudar de marcha em um carro com câmbio manual. Escalar muito cedo é como mudar para uma marcha alta enquanto o carro esta se movendo em baixa velocidade: as marchas vão ranger e você pode parar completamente. Escalar tarde demais é como pisar no pedal do acelerador enquanto se mantém em marcha baixa: o motor será bastante pressionado, empurrado para o vermelho, e se continuar por muito tempo ele superaquecerá sem atingir a velocidade máxima. Você precisa escalar e mudar de marcha no momento certo para manter as coisas funcionando sem problema.[6] (BRIKMAN, 2015, tradução nossa)

Seguindo a analogia apresentada por Brikman, as *startups* em suas primeiras fases do ciclo de vida devem se preocupar com a escalabilidade do código, que seria o equivalente as marchas baixas, ajustá-las antes de se preocupar com as marchas mais altas pois estas, por sua vez, podem nunca nem ser atingidas.

[6] Texto original: *Scaling a startup is a bit like shifting gears in a car with a manual transmission. Scaling too early is like shifting into a high gear while the car is moving at a low speed: the gears will grind and you may stall out completely. Scaling too late is like flooring the gas pedal while staying in low gear: you put a lot of stress on the engine, push it into the red, and if you keep it up too long, you'll overheat without ever hitting top speed. You have to scale and you have to shift gears at the right time to keep things moving smoothly.*

3 Metodologia

3.1 Levantamento bibliográfico

Para referenciamento deste trabalho foram realizados estudos acerca de *startups*, desenvolvimento de software e aspectos relacionados a escalabilidade de um sistema por meio de livros, Google Scholar, dentre outras fontes. Os tópicos abordados dentro de cada tema foram selecionados com base na sua respectiva importância dentro do assunto e no seu envolvimento com o contexto aprensentado. Individualmente, os temas citados são bem explorados e abordados pela comunidade acadêmica, contudo, quando relacionados os artigos e estudos a respeito tendem a ser mais escassos.

Publicações e palestras de empresas de referencia voltadas ao desenvolvimento de software no contexto de *startups* também foram utilizadas com o intuito de contribuir com perspectivas mais atuais a respeito do tema.

3.2 Metodologia de pesquisa

A proposta deste trabalho é correlacionar técnicas de desenvolvimento de software escalável com as fases de vida de uma *startup*. Para tal, foi aplicado o METODO X, visando estabelecer essa correlação. Esta seção visa descrever os objetos de análise, o método aplicado e as etapas envolvidas no desenvolvimento de toda a pesquisa.

3.2.1 Objetos de análise

Os objetos de análise do presente estudo consistem em:

Ciclo de vida de uma *startup* Fases pelas quais uma *startup* passa desde o seu nascimento até a sua estabilização no mercado.

Técnicas de escalabilidade Técnicas e boas práticas aplicadas no mercado em busca da construção de sistemas escaláveis e/ou com a capacidade para tal.

3.2.2 Método de correlação

3.2.3 Etapas do desenvolvimento

A etapas adotadas na construção da seguinte pesquisa foram:

Etapa 1 Definição do método de análise a ser aplicado;

Etapa 2 Coleta de dados a respeito dos objetos de análise;

Etapa 3 Aplicação do método de análise sobre os dados coletados;

Etapa 4 Análise dos resultados obtidos;

Etapa 5 Classificação das técnicas de escalabilidade;

Etapa 6 Validação ??.

A seguir segue a descrição de cada etapa.

3.2.3.1 Etapa 1: Definição do método de análise a ser aplicado

Nesta etapa, foi definido o método de análise utilizado, o qual tem como objetivo de estabelecer a correlação entre entre as técnicas de escalabilidade elencadas e as fases do cicla de vida de uma *startup*.

3.2.3.2 Etapa 2: Coleta de dados a respeito dos objetos de análise

Mediante o método de análise escolhido, foram coletados por meio de pesquisa bibliográfica as informações necessárias acerca dos objetos de análise definidos na subseção 3.2.1. Dessa forma obteve-se as características sobre os objetos de análise pertinentes a aplicação do método.

3.2.3.3 Etapa 3: Aplicação do método de análise sobre os dados coletados

Com os dados em mãos, nessa etapa aplicou-se o méotodo de análise proposto visando obter a correlação entre os objetos analisados.

3.2.3.4 Etapa 4: Análise dos resultados obtidos

A partir dos resultados obtidos na etapa anterior, realizou-se uma análise afim de compreender os mesmos e analisar a coerência com o que era esperado.

3.2.3.5 Etapa 5: Classificação das técnicas de escalabilidade

Por fim, classificou-se as técnicas de escalabilidade elencadas em relação as fases da *startup* usando como base os resultados obtidos nas etapas anteriores. Obtendo assim, as recomendações almejadas no objetivo geral deste trabalho, o qual está disposto na subseção 1.2.1.

3.3 Planejamento das atividades

Esta seção destina-se a descrever o planejamento realizado visando atingir as etapas definidas na subseção 3.2.3 e com base na metodologia de pesquisa descrita na seção 3.2.

Mediante o escopo da pesquisa a ser realizada, planejou-se a execução para um total de 7 semanas, iniciando na metade do mês de Março/2020 e indo até a primeira semana de Maio/2020. No Quadro 1 estão dispostas as atividades que pretende-se cumprir ao longo deste período.

Quadro 1 – Cronograma das atividades

Semana	**Atividades**
1	Definição do método de análise a ser aplicado
2	Coleta de dados a respeito dos objetos de análise
3	Aplicação do método de análise sobre os dados coletados
4	Análise dos resultados obtidos e classificação das técnicas de escalabilidade
6	Revisão e escrita do documento
7	Apresentação

3.4 Ferramentas

As ferramentas que pretende-se usar para a execução deste trabalho são:

Draw.io editor gráfico online que permite a construção de processos e desenhos relevantes ao conteúdo apresentado;

Google Planilhas ferramenta para construção de tabelas e gráficos. A qual pretende-se usar para realizar análises sobre os dados coletados.

Trello quadro interativo que auxilia na organização de projetos. O mesmo foi utilizado para gerenciar as etapas e tarefas alocadas para cada semana de execução do projeto, permitindo acompanhar o andamento do mesmo.

4 Desenvolvimento

Esta seção tem por propósito apresentar o estudo realizado sobre a *startup* Rua Dois Tecnologias, sendo esta o objeto de análise central do presente trabalho. O escopo deste capítulo abrangerá desde o entendimento sobre o contexto da empresa e sua proposta de valor até aspectos mais técnicos como a arquitetura do sistema, sua evolução durante o primeiro ano da empresa e as dificuldades enfrentadas pela *startup* nessa arquitetura.

As seções estão dispostas em:

1. **Rua Dois Tecnologias:** breve apresentação sobre a empresa, seus propósitos e os serviços oferecidos;

2. **Arquitetura do sistema:** descrição da arquitetura de software adotada dentro da Rua Dois, no seu primeiro ano de desenvolvimento, e possíveis soluções discutidas dentro da empresa a respeito da escalabilidade.

4.1 Rua Dois Tecnologias

No artigo *Modernizing Real Estate: The Property Tech Opportunity* [1], a autora Xu (2019) aborda uma visão histórica sobre o desenvolvimento de tecnologias para o mercado imobiliário fazendo um comparativo com a realidade atual. Nesse histórico de desenvolvimento, ela relata que desde 1980 são empregados diversos esforços na área buscando melhorarias por meio de software, mas mesmo assim, continuamos em uma ramo altamente burocrático e carente de melhorias.

Mediante essas limitações, surgiu em outubro de 2018 a Rua Dois Tecnologias, com o propósito de solucionar as dificuldades enfrentadas tanto por parte das imobiliárias quanto por parte dos locatários [2] no processo de locação de imóveis.

A proposta[3] da empresa consiste em atuar como um facilitador para as imobiliárias ingressarem no meio digital e consequentemente propiciar seu crescimento. Para tal, eles buscam aumentar a eficiência do processo de locação por meio da digitalização, proporcionando melhor integração entre operação e tecnologias. Assim, com um processo mais digital, os recursos humanos podem ser melhores empregados no atendimento do cliente favorencendo a experiência do mesmo nesse mercado tão burocrático.

[1] Disponível nest link.
[2] Aquele que mora em um imóvel que não lhe pertence, mediante um contrato de locação.
[3] Mais informações a respeito da proposta e visão da Rua Dois podem ser encontradas neste *podcast*.

4.1.1 Serviços prestados

Visando o processo de locação de imóveis, a Rua Dois conta com quatro serviços sendo desenvolvidos dentro da empresa, afim de agregar valor aos seus clientes, sendo eles:

1. **Captação de Imóveis:** serviço destinado à atrair proprietários de imóveis que desejam divulgar o imóvel para locação;
2. **Visitas:** serviço de visitas aos imóveis, no qual os interessados em locar o imóvel agendam um horário para conhecer o mesmo;
3. **Propostas:** serviço no qual os interessados, após passarem pela etapa de visitação, avaliam o imóvel e fazem uma proposta para o proprietário do imóvel referente ao valor a ser pago pela locação e possíveis ajustes que desejam que sejam feitos na propriedade;
4. **Contrato:** etapa final, destinada a automatizar a avaliação por parte das seguradoras, envio de documentos e assinatura do contrato.

Cada serviço é vendido de forma separada para as imobiliárias, de forma que cada uma adapte ao seu contexto os serviços que melhor se encaixam. Atualmente, o serviço mais desenvolvido é o de visitas e a equipe vem trabalhando com o intuito de estabilizar os demais serviços.

4.2 Arquitetura do sistema

O início do desenvolvimento do software da Rua Dois se deu em outubro de 2018 com base na proposta de criar um sistema usando de uma arquitetura de microsserviços, de forma que esse sistema já fosse preparado pra escalar desde seu início. Contudo, a empresa enfrentou uma série de dificuldades[4] em relação a esta arquitetura e ao seu contexto no respectivo momento. Mediante essas dificuldades a Rua Dois optou em seu processo de desenvolvimento migrar essa arquitetura de microsserviços para um sistema monolítico.

Esta seção visa descrever essas duas fases arquiteturais dentro da empresa com suas respectivas motivações e desafios encontrados. Assim, visando facilitar o entendimento, cada fase será nomeada da seguinte forma:

Fase 1 referente a fase inicial de desenvolvimento na qual foi adotada uma arquitetura de microsserviços;

Fase 2 referente a segunda fase de desenvolvimento na qual foi adotada uma arquitetura monolítica.

[4] Estas dificuldades serão abordadas com maior clareza na subseção 4.2.1

4.2.1 Arquitetura do sistema: Fase 1

A primeira fase de desenvolvimento da *startup* foi marcada pela premissa de que o sistema deveria ser escalável, e com base nessa premissa foram tomadas uma série de decisões visando preparar o sistema para tal. Assim o sistema foi dividido em dois microsserviços principais e em uma série de microsserviços auxiliares distribuídos nos lambdas da Amazon Web Services (AWS). Os dois serviços principais são:

r2service responsável por gerir o cadastro de imóveis e as propostas de locação do imóvel;

r2visit responsável por gerir a lógica de agendamento de visitas a um imóvel.

A Figura 4 visa ilustrar a organização desse sistema. Nela os dois serviços principais estão representados pelos círculos maiores roxos e as demais representações são outros serviços que auxiliam no ciclo de vida do produto. Adiante segue descrito o fluxo com o qual todo o sistema é acionado, seguindo a enumeração disposta na imagem:

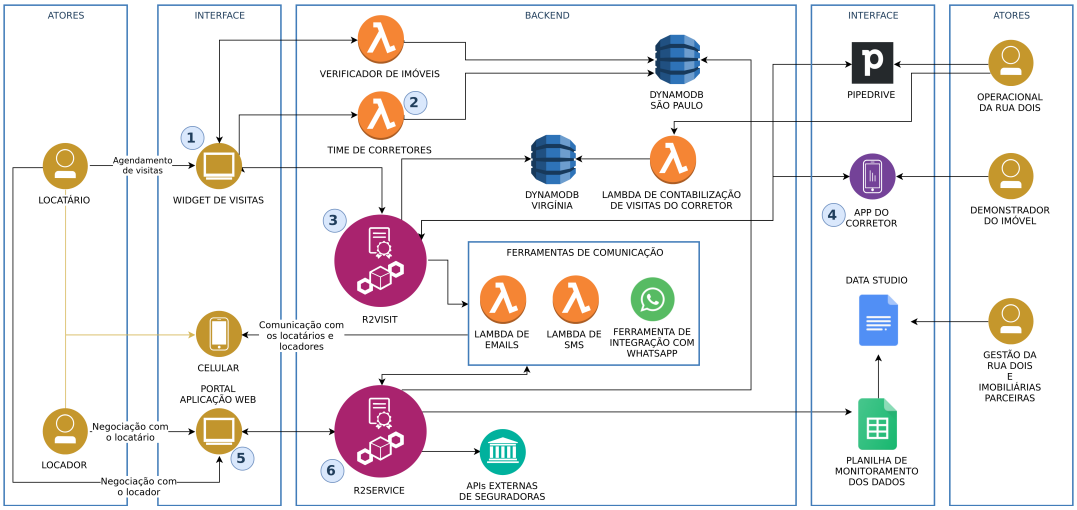

Figura 4 – Arquitetura do sistema da Rua Dois na Fase 1

1. O *Widget* é um componente utilizado pela Rua Dois que roda no *frontend* dos sites das imobiliárias que aderem ao serviço de visitas. Ele se comunica com um lambda responsável por verificar se o imóvel apresentado no site da imobiliária está disponível no banco de dados da Rua Dois. Uma vez que esteja disponível, o *widget* exibe um botão que disponibiliza o agendamento da visita *online* para o interessado;

2. O locatário ao clicar nesse botão, dispara uma requisição ao lambda que identifica o time de corretores responsável por atender as visitas ao imóvel selecionado. E em seguida, uma nova requisição é disparada ao serviço *r2visit* solicitando o calendário com os horários disponíveis do respectivo time;

3. Após o locatário ter selecionado o horário da visita, uma nova requisição é lançada para registrar a visita no *r2visit*, que por sua vez dispara uma série de mensagens por meio das ferramentas de comunicação e atualiza o Pipedrive[5];

4. Após a visita ser realizada, o corretor sinaliza por meio do aplicativo a conclusão da visita, disparando novamente no *r2visit* os eventos de atualização do Pipedrive e as ferramentas de comunicação, para que se possa dar início a nova fase de Negociação, referente ao produto de propostas;

5. Nessa nova fase, locatário e locador trocam mensagens por meio do portal da Rua Dois, o qual é gerido pelo *r2service*. Ao final da negociação, os clientes entram na fase de contrato, na qual seu dados são recolhidos e enviados por meio do *r2service* para Application Programming Interfaces (APIs) externas de seguradoras as quais realizam uma avaliação de crédito do locatário. Quando concedida a avaliação de crédito, o contrato é gerado automaticamente com os dados já recolhidos do locatário, locador, da imobiliária e do próprio imóvel;

6. Por fim, o *r2service* é responsável por atualizar uma planilha do Google Sheets[6] que serve de insumo para geração de relatórios dentro do Data Studio [7].

Nesse fluxo nota-se que existe uma grande diversidade de elementos interagindo entre si e a forma como isto foi organizado gerou as seguintes dificuldades dentro da empresa:

1. Dificuldade de organização dos dados da empresa uma vez que não houve um bom planejamento em relação a interação entre os serviços *r2service* e *r2visit*, o que ocasionou a duplicação da tabela referente aos dados dos imóveis - uma tabela no banco de dados da Virgínia e a outra tabela no banco de dados de São Paulo;

2. Dificuldade de gerir as associações necessárias entre os dados armazenados no banco. A empresa optou pelo uso do banco não relacional DynamoDB, visando novamente a escalabilidade, mas sem planejar corretamente o armazenamento desses dados nessa estrutura não relacional. Isso impactou diretamente o desenvolvimento, visto que existe uma necessidade da empresa de tratar esse dado como relacional mas fez isso em cima de uma tecnologia inadequada para tal;

3. Dificuldade de manter essa infraestrutura, tanto pela complexidade na forma como o sistema está organizado quanto pela escolha de utilizar a infraestrutura da AWS, que

[5] O Pipedrive é uma ferramenta de Customer Relationship Management (CRM) utilizada pela empresa para realizar o gerenciamento operacional e monitoramento dos resultados obtidos. Saiba mais em: <https://www.pipedrive.com>

[6] Saiba mais em: <https://www.google.com/sheets/about/>

[7] Saiba mais em: <https://datastudio.google.com>

é mais árdua de trabalhar do que outras soluções disponíveis no mercado (KAVYA, 2019);

4. Dificuldade de inserção de novos membros na equipe de desenvolvimento, visto que entender como esses vários serviços se relacionam não é uma tarefa simples;

5. Dificuldade de mudança e inserção de novas funcionalidades. Nessa primeira fase a Rua Dois encontrava-se fortemente no período de validação da ideia dentro de uma *startup* o que exigia uma rápida reação da equipe em relação aos *feedbacks* recebidos. Contudo, nessa arquitetura se tornou exaustivo a realização de constantes mudanças vistos todos os problemas relatados nos tópicos anteriores.

4.2.2 Arquitetura do sistema: Fase 2

Mediante as dificuldades enfrentadas na arquitetura adotada na Fase 1 e a incerteza sobre a escalabilidade desse sistema, em junho de 2019 a Rua Dois optou por migrar para uma arquitetura monolítica com a perspectiva de tornar o sistema mais estável e mais apto a realização de mudanças. Nessa mudança os serviços *r2service* e *r2visit* se tornaram um único sistema, agora chamado de *novo r2service*, alguns lambdas foram incorporados por esse novo sistema ou simplesmente foram descartados juntamente com alguns serviços auxiliares que não seriam mais usados. A Figura 5 exemplifica como está o sistema nessa nova configuração.

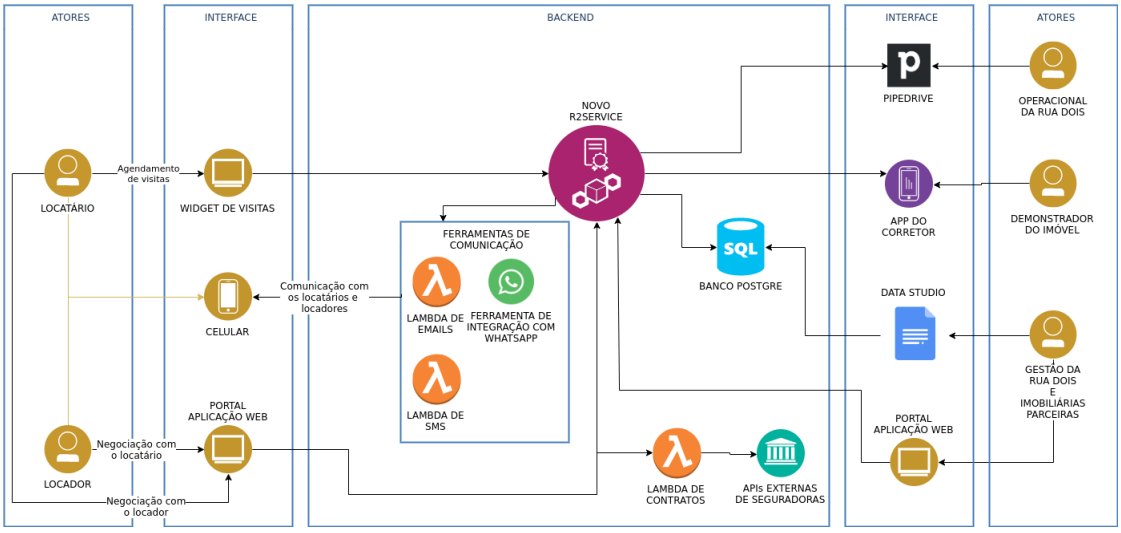

Figura 5 – Arquitetura do sistema da Rua Dois na Fase 2

5 Considerações Finais

Neste capítulo serão apresentadas considerações referentes ao que foi realizado neste trabalho até o presente momento e as expectativas e metas para a próxima fase.

As seções estão dispostas em:

1. **Primeira entrega parcial** considerações acerca dos resultados entregues na primeira etapa do respectivo trabalho, referente ao Trabalho de Conclusão de Curso 1 (TCC 1);
2. **Entrega final** considerações acerca das expectativas almejadas para a realização da segunda etapa do respectivo trabalho, referente ao Trabalho de Conclusão de Curso 2 (TCC 2);
3. **Cronograma** Distribuição planejada das fases de execução deste trabalho durante o período subsequente.

5.1 Primeira entrega parcial

Nesta primeira fase foi realizado o levantamento bibliográfico a respeito dos assuntos relacionados ao tema central com o intuito de explorar as visões dispostas na literatura sobre o mesmo. Foram encontrados diversos relatos a respeito de *Startups*, desenvolvimento de software escalável desde o início, métodos de desenvolvimento de software, etc. No entanto, a respeito de desenvolvimento de software em *startups* e desenvolvimento de software escalável em *startups* o assunto ainda é muito escasso. Nesse sentido, foi proposto uma metodologia de pesquisa afim de explorar essas duas questões dentro do contexto da Rua Dois. Também foi realizado uma análise inicial a respeito das arquiteturas de software adotadas na empresa com o intuito de compreender melhor as problemáticas enfrentadas pela *startup* e o estado atual.

5.2 Entrega final

Espera-se realizar a coleta dos dados propostos e com base nos resultados elaborar a análise comparativa entre as duas fases da Rua Dois. Assim, a partir da análise comparativa realizada propõem-se levantar diretrizes que orientem o desenvolvimento de software escalável na empresa mediante as experiências já vivenciadas pela mesma. A validação dessas diretrizes será constituída com base no que é encontrado na literatura a respeito do tema.

5.3 Cronograma

Esta seção visa apresentar o cronograma para desenvolvimento do presente trabalho. Este cronograma está dividido em duas etapas: Primeira Entrega Parcial (TCC 1) e Entrega Final (TCC 2).

O cronograma referente a Primeira Entrega Parcial é apresentado no Quadro 2. Em relação a atividade de *Pesquisa e definição da metodologia*, foi definido o **??** apresentado na **??** que será aplicado para elencar quais métricas e informações serão coletadas durante a Etapa 2: Coleta de dados a respeito dos objetos de análise. Contudo, a execução desse processo irá definir aspectos como métricas e tecnologias que ainda não estão descritos na Metodologia. Por este motivo, esta atividade foi considerada como parcialmente concluída.

Quadro 2 – Cronograma: primeira entrega parcial

Atividade	**Agosto**	**Setembro**	**Outubro**	**Novembro**	**Status**
Levantamento e definição do tema	X	X			Concluído
Definição do objetivo geral e dos específicos		X			Concluído
Pesquisa e levantamento bibliográfico		X	X		Em andamento
Pesquisa e definição da metodologia			X		Parcialmente concluído
Análise inicial da arquitetura				X	Concluído

O Quadro 3 apresenta o cronograma previsto para a execução da segunda etapa do trabalho, conforme à metodologia e ao planejamento definido no Capítulo 3.

Quadro 3 – Cronograma: entrega final

Atividade	**Fevereiro**	**Março**	**Abril**	**Maio**
Estudo acerca da arquitetura adotada	X			
Coleta e levantamento de dados referentes a qualidade do código	X	X		
Coleta e levantamento de dados referentes as práticas de desenvolvimento de software	X	X		
Coleta e levantamento de dados referentes ao conhecimento da equipe		X	X	
Coleta e levantamento de dados referentes as tecnologias utilizadas		X	X	
Coleta e levantamento de dados referentes a escalabilidade			X	X
Coleta e levantamento de dados referentes ao ciclo de vida da *startup*			X	X
Análise comparativa				X
Definição e validação das diretrizes				X
Escrita do relatório final	X	X	X	X

Referências

BECK, K. et al. *Agile Manifesto*. Agile Manifesto, 2001. Disponível em: <http://agilemanifesto.org/>. Acesso em: 24.11.2019. Citado na página 29.

BLANK, S. *What's a Startup? First Principles.* Steve Blank, 2010. Disponível em: <https://steveblank.com/2010/01/25/whats-a-startup-first-principles>. Acesso em: 29.06.2019. Citado 3 vezes nas páginas 21, 25 e 26.

BLANK, S. G. *The four steps to the Epiphany*. [S.l.: s.n.], 2013. Citado 3 vezes nas páginas 13, 26 e 27.

BOSCH, J. et al. The early stage software startup development model: A framework for operationalizing lean principles in software startups. *LESS 2013. LNBIP*, v. 167, p. 1–15, 01 2013. Citado na página 37.

BRIKMAN, Y. *Hello, Startup: A Programmer's Guide to Building Products, Technologies, and Teams*. USA: O'Reilly Media, Inc, 2015. Citado 2 vezes nas páginas 42 e 43.

CONCEPTANEXT. *The Importance of Scalability In Software Design*. Concepta, 2017. Disponível em: <https://conceptainc.com/blog/importance-of-scalability-in-software-design/>. Acesso em: 14.07.2019. Citado na página 38.

DESPA, M. L. Comparative study on software development methodologies. *Database Systems Journal*, Bucharest University of Economic Studies, V, n. 3, 2014. Citado 3 vezes nas páginas 27, 28 e 32.

DULLIUS, A. C.; SCHAEFFER, P. R. As capacidades de inovação em startups: contribuições para uma trajetória de crescimento. *Revista Alcance*, Santa Catarina, v. 23, n. 1, 2016. Citado na página 21.

GAJENDRAN, S. K. A survey on NoSQL Databases. 2012. Citado na página 40.

GARDELIN, J. P.; ROSSETTO, C. R.; VERDINELLI, M. A. O relacionamento entre a incerteza ambiental e o comportamento estratégico na percepção dos gestores de pequenas empresas. *Rausp - Revista de Administração*, São Paulo, v. 48, n. 4, 2013. Citado na página 26.

GIARDINO, C. et al. Software development in startup companies: The greenfield startup model. *IEEE Transactions on Software Engineering*, X, n. X, 2016. Citado na página 37.

HAN, J. et al. Survey on nosql database. In: *2011 6th International Conference on Pervasive Computing and Applications*. Port Elizabeth, South Africa: IEEE, 2011. Citado na página 40.

HIGHSMITH, J.; COCKBURN, A. Agile software development: The business of innovation. IEEE, v. 34, n. 9, 2001. Citado na página 22.

HUNGARY, G. Which Factors Determine the Success Or Failure of Startup Companies? A Startup Ecosystem Analysis of; US the. *Scalability Guide*. Hamburg, US: Anchor Academic Publishing, 2017. Citado na página 42.

INTERSYSTEMS, I. D. P. *Scalability Guide*. Cambridge, MA: InterSystems Corporation, 2019. Citado 2 vezes nas páginas 39 e 40.

KAVYA. *DigitalOcean vs AWS EC2: 8 Factors to Decide Who's the Winner?* ServerGuy, 2019. Disponível em: <https://serverguy.com/comparison/digitalocean-vs-aws-ec2/>. Acesso em: 07.07.2019. Citado na página 53.

KHARE, A. et al. *A Fresh Graduate's Guide to Software Development Tools and Technologies*. Singapore, Asia: School of Computing: National University of Singapore, 2012. Citado 2 vezes nas páginas 39 e 40.

MAHNIC, V.; DRNOVSCEK, S. Agile software project management with scrum. University of Ljubljana, Faculty of Computer and Information Science, Slovenia, 2005. Citado na página 30.

MARMER, M. et al. Startup genome report extra on premature scaling. *British Journal of Management*, Nova Iorque, 2011. Citado na página 42.

MONIRUZZAMAN, A. B. M.; HOSSAIN, S. A. Nosql database: New era of databases for big data analytics - classification, characteristics and comparison. *International Journal of Database Theory and Application*, Bangladesh, v. 6, n. 4, 2013. Citado na página 40.

NAGAMATSU, F. A.; BARBOSA, J.; REBECCHI, A. Business model generation e as contribuições na abertura de startups. In: . São Paulo: II Simpósio Internacional de Gestão de Projetos, Inovação e Sustentabilidade, 2013. Citado na página 21.

NEPOMUCENO, C. *Afinal, quais são os segredos de sucesso do Waze?* Administradores.com, 2013. Disponível em: <https://administradores.com.br/noticias/afinal-quais-sao-os-segredos-de-sucesso-do-waze>. Acesso em: 29.06.2019. Citado na página 25.

NEWMAN, S. *Building Microservices: Designing Fine-Grained Systems*. USA: O'Reilly Media, Inc., 2015. Citado 2 vezes nas páginas 41 e 42.

PATERNOSTER, N. et al. Software development in startup companies: A systematic mapping study. *Information and Software Technology*, Bangladesh, 2014. Citado na página 37.

PERIN, B. *Startup Genome Report Extra on Premature Scaling*. Rio de Janeiro: Alta Books, 2016. Citado 2 vezes nas páginas 25 e 26.

RIES, E. *The Lean Startup*. Nova Iorque: Currency, 2011. Citado 2 vezes nas páginas 22 e 26.

RUEDA-MANAZARES, A.; ARAGóN-CORREA, J. A.; SHARMA, S. The influence of stakeholders on the environmental strategy os service firms: the moderating effects of complexity, uncertainty and munificence. *British Journal of Management*, Nova Iorque, v. 19, n. 2, 2008. Citado na página 26.

SAKOVICH, N. *Monolithic vs. Microservices: Real Business Examples*. 2017. Disponível em: <https://www.sam-solutions.com/blog/microservices-vs-monolithic-real-business-examples/>. Acesso em: 07.11.2019. Citado na página 41.

SCHWABER, K.; SUTHERLAND, J. *The Scrum Guide*. [S.l.], 2017. Citado na página 30.

SCRUM.ORG. *What is Scrum?* Scrum.org, 2019. Data da publicação desconhecida. Disponível em: <https://www.scrum.org/resources/what-is-scrum>. Acesso em: 26.11.2019. Citado 2 vezes nas páginas 13 e 31.

SPINA, C. *O que é uma ideia escalável?* Revista Exame, 2012. Disponível em: <https://exame.abril.com.br/pme/o-que-e-uma-ideia-escalavel/>. Acesso em: 29.06.2019. Citado na página 26.

STAMPFL, G.; PRüGL, R.; OSTERLOH, V. An explorative model of business model scalability. *International Journal of Product Development*, v. 18, p. 226–248, 06 2013. Citado na página 42.

TIWANA, A. *Platform Ecosystems: Aligning Architecture, Governance, and Strategy*. Massachusetts, USA: Elsevier, 2014. Citado na página 38.

WECK, O. L. D.; ROOS, D.; MAGEE, C. L. *Engineering Systems: Meeting Human Needs in a Complex Technological World*. Massachusetts, USA: MIT Press, 2011. Citado na página 38.

WELLS, D. *Extreme Programming: A gentle introduction*. Extreme Programming Organization, 2000. Disponível em: <http://www.extremeprogramming.org/>. Acesso em: 26.11.2019. Citado 4 vezes nas páginas 13, 32, 33 e 37.

XU, L. *Modernizing Real Estate: The Property Tech Opportunity*. Forbes, 2019. Disponível em: <https://www.forbes.com/sites/valleyvoices/2019/02/22/the-proptech-opportunity>. Acesso em: 06.07.2019. Citado 2 vezes nas páginas 21 e 49.