UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Estudo de caso para análise de rendimento na produção de software em TI para *start-ups* explorando o modelo clássico em cascata e o modelo ágil *Scrum*

Autor: Verivaldo Teles Lobo Filho

Orientador: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos

2017

[Verivaldo Teles Lobo Filho]

Estudo de caso de rendimento na produção de software para TI explorando o modelo clássico em cascata e o modelo ágil *Scrum*

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica

ORIENTADOR: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos
2017

Teles Lobo Filho, Verivaldo
T516e Estudo de caso para análise de rendimento na produção de software em TI para start-ups explorando o modelo clássico em Cascata e o modelo ágil Scrum / Verivaldo Teles Lobo Filho; orientador Evandro Luis Linhari Rodrigues. São Carlos, 2017.

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2017.

1. Scrum. 2. desenvolvimento ágil. 3. software. 4. gestão. 5. tecnologia de informação. 6. start-up. I. Título.

FOLHA DE APROVAÇÃO

Nome: Verivaldo Teles Lobo Filho

Título: "Estudo de caso para análise de rendimento na produção de software em TI para start-ups explorando o modelo clássico em cascata e o modelo ágil Scrum"

Trabalho de Conclusão de Curso defendido e aprovado em 27/06/2017,

com NOTA 10,0 (DEZ, ZERO.), pela Comissão Julgadora:

Prof. Associado Evandro Luis Linhari Rodrigues - Orientador - SEL/EESC/USP

Prof. Associado Ivan Nunes da Silva - SEL/EESC/USP

Prof. Associado Antonio Freitas Rentes - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado José Carlos de Melo Vieira Júnior

Dedicatória

Dedico este trabalho a meus pais Verivaldo e Eliedil Lobo, minha irmã Thaiane Lobo, minha avó Cota, a Sophie Winter, a Gabriella Fonseca e a todos os meus amigos da Cubos Tecnologia.

[Verivaldo Teles Lobo Filho].

Agradecimentos

Agradeço a todas as pessoas que acreditaram e as que não acreditaram em mim. Sou grato a todos os meus amigos e conhecidos, aos que me apoiaram e aos que foram obstáculos, aos que me desafiaram e aos que me motivaram. Muito obrigado a todos, vocês ajudaram a construir quem sou hoje.

[Verivaldo Teles Lobo Filho].

"At the end of the day, we can endure much more than we think we can."

[Frida Kahlo]

Resumo

FILHO, V. T. L. Estudo de caso de rendimento na produção de *software* para TI explorando o modelo clássico em cascata e o modelo ágil *Scrum*. O avanço da tecnologia da informação está criando um campo rico para o aparecimento exponencial de *start-ups* em todo planeta. Essas jovens empresas, que têm um incrível potencial de escalabilidade, têm o objetivo de resolver problemas do dia a dia e de industrias complexas. Contudo, os empreendedores devem estar sempre alerta quanto a produtividade de suas equipes. Como essas empresas têm baixo recuso financeiro e alto risco de negócio envolvidos, todo desperdício pode significar no fim de todo o projeto. Assim, este estudo analisou quatro *start-ups* desenvolvidas pela Cubos Tecnologia. Nesses quatro projetos foram utilizados dois métodos de desenvolvimento, o tradicional modelo em Cascata e o método ágil *Scrum*. Todos os projetos possuem o mesmo nível de complexidade técnica, então o foco é entender qual dos dois métodos se apresenta mais eficiênte no processo de desenvolvimento de [software] nesse mercado com alto nível de inovação. Para tornar esta análise possível, duas principais ferramentas foram utilizadas. A primeira é o CubosTime, um programa desenvolvido pela Cubos Tecnologia que mensura o tempo gasto na fase de codificação do projeto. O segundo se trata da plataforma Asana, um gerenciador de tarefas. Deste modo, esses casos possibilitam demonstrar como a metodologia ágil pode trazer aumento de produtividade, flexibilidade e motivação na equipe. Todos esses três aspectos são vitais para o construção de um negócio de sucesso e um bom *software* que realmente preenche as necessidades do usuário.

Palavras-Chave: *Scrum*, desenvolvimento ágil, *software*, gestão, tecnologia da informação,*start-up*.

Abstract

FILHO, V. T. L. Case study of income in IT software production exploring the classic Waterfall model and the agile model *Scrum*. The advence of the informational technology is developing a rich field for the exponencial appereance of start-ups all around the globe. These young companies, witch have a huge scalable potencial, aim to solve everyday and industrial problems. Although, the enterprenuers must stay awared of their team's productivity. As these small companies have lowbudget and high risk involved in the business, every loss may bring the end of all the work. Thus, these study analised four start-ups developed by Cubos Tecnologia. In these four projects were used two different development methods, the tradicional Waterfall method and agile model Scrum. Every project has the same level of technology complexity, so the focus is to understand witch of the two methods show more efficiency in the process of developing software in a market with high level of inovation. To make this analisys possible, two main tools were used. The first one used was CubosTime, a software developed by Cubos Tecnologia that counts how much time were spent during the codification phase of the development. And the second tool was Asana, a task manager platform. Hence, these cases make it possible to show how the agile development method can bring more productivity, flexibility and motivation to a team. All these three aspects are vital to build a long term sucessful business and a good software product that actually fulfils the user's needs.

Keywords: Scrum, agile development, software, information technology, management, start-up.

Lista de Figuras

Lista de Tabelas

Siglas

UX	*User Experience* - Experiência do Usuário
UI	*User Interface* - Interface do Usuário
API	*Application Programming Interface* - Interface da Aplicação do Programa
IDE	*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado
TI	Tecnologia da Informação

Sumário

Capítulo 1

Introdução

O desenvolvimento de software é tópico constante no estudo de processo nas universidades e no mercado. Hoje, a capacidade de se organizar, padronizar e estruturar a produção em uma empresa pode determinar o seu sucesso ou fracasso. Em empresas embrionárias como *start-up* a necessidde de evitar dispedícios é ainda mais crítica do que em grandes corporações. Portanto, entender dos processos de desenvolvimento e buscar o alto rendimento é de extrema importância para o sucesso das soluções criadas por estas empresas.

No estudo aqui apresentado, tem-se uma análise de duas metodologias de desenvolvimento de software. De um lado a estrutura tradicional do método Cascata e do outro a récem-criada metodologia de desenvolvimento ágil denominada *Scrum*. Este estudo foi realizado a partir de dados obtidos de uma empresa de desenvolvimento de *software*, a Cubos Tecnologia, aplicando-se as duas metodologias e medindo-se o tempo gasto para desenvolver o código de projetos de mesmo nível de complexidade. Para o desenvolvimento de quatro projetos, dois em cascata e dois construídos com o método ágil, foram formadas três equipes. Estas foram criadas para que se pudesse cruzar informações de equipes que trabalharam com um método exclusívo e uma equipe que utilizou os dois métodos em momentos distintos.

O método Cascata, amplamente utilizado não apenas na área de software mas em diversos ramos da engenharia, vem frustrando os profissionais dos mercados em que é utilizado, assim como os seus clientes. Este método ocasiona em sua aplicação os constantes atrasos de entrega, gastos não previstos e desconforto para as equipes envolvidas.

O *Scrum* surge com a ideia de se adaptar aos obstáculos decorrentes do processo produtivo, como por exemplo a baixa capacidade do ser humano de prever problemas e de mensurá-los com precisão, como relata o criador do método. Esta metodologia tem influência direta da cultura japonesa e das práticas do Sistema Toyota de Produção e é utilizada em empresas de

grande porte, como o Google, na resolução dos mais diversos problemas, conforme descrito por Jake Knapp em seu livro *Sprint* [1]).

Nos anos 2000, após os atentados de 11 de setembro nos Estados Unidos, o FBI (*Federal Bureau of Investigation*, a polícia federal americana) decidiu investir em um sistema que unificaria todas as informações policiais do país. Para isso, foi investido inicialmentem US$ 451 milhões para construir o sistema *Sentinel* em quatro anos, como conta Jeff Sutherland em seu livro "*SCRUM*: A arte de fazer o dobro do trabalho na metade do tempo"[2]. O desenvolvimento do programa seria feito de acordo com o método em cascata. Porém, após 5 anos de desenvolvimento, já haviam sido gastos US$405 milhões e o sistema não estava pronto para uso. Sutherland conta que uma análise foi feita e a expectativa era de um acrécimo nos gastos de US$ 350 milhões, quantia proveniente dos contribuintes americanos. Jeff Johnson então foi contratado para resolver o problema e chegou-se à conclusão de que o *Scrum* seria utilizado em substituição ao método Cascata. O resultado foi que a parte mais desafiadora do projeto foi feita em um quinto do tempo estimado, com um gasto de um décimo do previsto e utilizando menos desenvolvedores.

A Cubos Tecnologia desenvolveu um *software* com a finalidade de medir quanto tempo é dedicado à fase de codificação, medindo-se assim um indicador de nível de produtividade das equipes. E com outras ferramentas utilizadas neste estudo, foi possível analisar como se comporta o desenvolvimento de programas na área da tecnologia da informação, com foco em negócios em fase embrionária, as chamadas *Startups*. Assim, este trabalho mostra, em quatro casos com foco em TI, como a metodologia utilizada em empresas como Google e Amazon pode impactar na produtividade de uma equipe.

1.1 Motivação

Desde o advento dos computadores, dos semicondutores e dos processos que permitiram que os componentes eletrônicos se tornassem mais baratos, novas áreas do conhecimento tomaram forma e força. A área de desenvolvimento de *software* se tornou muito importante, pois é responsável por fazer com que os computadores, microcontroladores e sistemas em geral possam funcionar de forma mais eficiente ou até mesmo autônoma. Hoje, programas estão empregados em diversos objetos de uso corrente como máquinas, eletrodomésticos, celulares, televisores e computadores.

Na presente era da informação, novas empresas estão surgindo para criar soluções base-

adas nas novas tecnologias. Essas empresas são conhecidas como *startups* e geralmente são criadas sem uma estrutura bem definida, plano de negócios ou estrutura de monetização. Isso ocorre devido ao valor inovador que elas trazem para o mercado. Uma estratégia muito utilizada pelas *startups* é o enfoque em MVP (*Minimum Valuable Product*, em português Mínimo Produto Viável), definido na obra "Startup Enxuta", do autor Eric Reis[3]. Nessa abordagem estratégica, o tempo de reação para o mercado precisa ser muito rápido, ou seja, evoluir o produto e responder às necessidades do mercado de maneira ágil definem o sucesso do *software* e do negócio. Portanto, no cenário atual da tecnologia da informação, o tempo é fator primordial para se alavancar o negócio e possibilitar que um programa ganhe escala na sua utilização.

Em resumo, este estudo é motivado pela busca de melhorar o processo de desenvolvimento de *software* diminuindo o tempo de produção. Isso deve ocorrer sem haver perdas na qualidade do programa.

1.2 Objetivo(s)

O objetivo principal é comparar dois métodos de desenvolvimento de software. O primeiro se trata do método Cascata, por ser tradicional não apenas na área de programação, mas também em diversas outras frentes da engenharia. O *Srum*, como segundo método estudado, é uma metodologia recente de desenvolvimento ágil que vem ganhando força nos últimos dez anos. Desse modo, este trabalho busca avaliar e contrapor, em um estudo de caso, os resultados obtidos com a metodologia ágil e com o método tradicional aplicado a projetos embrionários de *start-ups*.

1.3 Justificativas/relevância

Um bom processo de desenvolvimento de *software* é observado quando se obtém produtos de alta qualidade técnica no menor tempo possível. Neste trabalho foram escolhidos o método em cascata e o *Scrum*, pois o primeiro é um dos mais tradicionais da Engenharia de *Software* e o segundo é recente e surgiu quebrando paradigmas do processo produtivo. Colocando esses dois métodos à prova, pode-se obter informações sobre qual deles se mostra mais eficiente em projetos de tecnologia da informação, em que os mercados são extremamente dinâmicos e as tecnologias evoluem em uma velocidade nunca antes vista.

1.4 Organização do Trabalho

Este estudo está distribuído em 6 capítulos, incluindo esta introdução, dispostos conforme a descrição que segue:

Capítulo 2: descreve uma introdução à teoria que serve de base para o estudo da engenharia de software, a estrutura em cascata e a estrutura do *Scrum*.

Capítulo 3: discorre sobre as ferramentas utilizadas para se obter os dados de cada caso estudado, a organização das equipes e a composição dos projetos que foram objetos de estudo.

Capítulo 4: apresenta todos os resultados coletados pelas ferramentas e discute os pontos mais relevantes sobre os casos.

Capítulo 5: analisa o que pode ser concluído a partir dos dados coletados e propõe novos estudos de caso relevantes.

Capítulo 2

Embasamento Teórico

Este capítulo apresenta uma introdução à Engenharia de *Software*, a estrutura do método Cascata, as bases e princípios do *Scrum* e o método espiral.

2.1 A Engenharia de *Software*

Em sua obra "Engenharia de *Software*", Ian Sommerville[4] avalia que a Engenharia de *Software* é responsável por "toda documentação associada e dados de configuração necessários para fazer com que esses programas operem corretamente". Além disso, trata também do processo de gestão das informações, da gestão de pessoas e recursos. Complementando a ideia de Sommerville, Pressman[5] a define como uma tecnologia em camadas, apresentadas na figura 2.1. Estas são estruturadas para fundamentar uma organização com foco na qualidade. Filosofias como a Seis Sigma, prossegue o autor, "promovem uma cultura de aperfeiçoamento contínuo de processos, e é essa cultura que no final das contas leva ao desenvolvimento de abordagens cada vez mais eficazes".

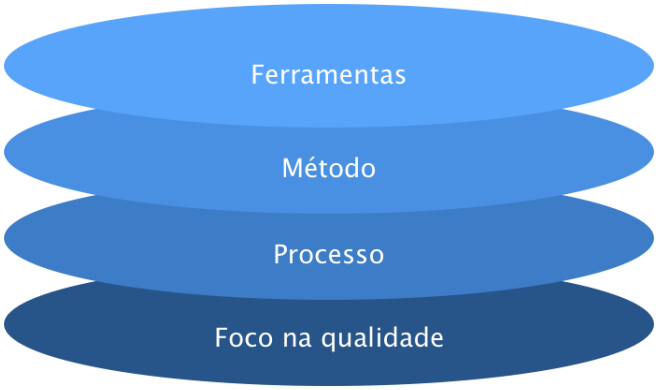

Figura 2.1: Camadas da Engenharia de Software

A camada de suporte da Engenharia de *Software* é a de processos, em que todas as demais camadas de execução se sustentam. A de foco na qualidade simboliza o objetivo final. A de método é a que permite o software ser desenvolvido, atribuindo a ela todas as atividades necessárias para que o trabalho seja concluído, enquanto a de ferramentas possibilita a execução das tarefas, facilitando, agilizando e evitando erros durante o processo de desenvolvimento. Essas camadas são definidas por Pressman[5]. No método escolhido pela equipe de desenvolvimento, deve haver impreterivelmente tarefas de comunicação, análise de requisitos, modelagem de projetos, construção de programas, testes e suporte. Sommerville[4] divide o processo de software em quatro partes: especificação do *software*; desenvolvimento do *software*; validação do *software*; evolução do *software*.

2.1.1 O Processo

Todo método em Engenharia de Software deve ter cinco fases: comunicação, planejamento, modelagem, construção e entrega. Na fase de comunicação, o desenvolvedor busca entender quais as reais necessidades do projeto em questão. Ele deve compreender de forma clara em qual contexto a solução é necessária e deve apresentar as possibilidades a serem desenvolvidas de maneira coesa e consistente .

Em todo projeto complexo é necessário criar um guia a partir do qual os membros podem se orientar. Desenvolver um software é sempre algo complicado, por mais simples que a aplicação seja, portanto é necessário planejar descrevendo o trabalho que será feito, as tarefas necessárias, os prováveis riscos do projeto, quais recursos serão alocados, qual o produto desejado e como construir um cronograma de atividades.

Para iniciar uma fase de desenvolvimento, o desenvolvedor deve projetar o que será construído, analisar o que será necessário, definir como será feito e como cada característica do software deve se comportar, o que é de vital importância para uma construção bem executada - compondo assim a fase de modelagem. Na fase de construção, a equipe técnica codifica o que foi projetado e efetua testes internos para se certificar de que aquele projeto foi bem executado. Essa fase geralmente é a mais longa do processo inteiro, e este estudo tem foco na coleta de dados nessa fase. Caso o planejamento ou o projeto tomem rumos diferentes, seja por erro ou necessidade, é neste momento do processo que o impacto de retrabalho ocorre.

Por fim o produto, o *software*, é lançado e se aguardam feedbacks por parte dos usuários, para que ele então seja melhorado.

2.1.2 Fluxo do Processo

Para um método genérico, podem ser apresentadas algumas possibilidades de fluxo de processo. Ele pode ser linear, interativo, evolucionário ou paralelo. As estruturas podem ser vistas nas figuras 2.2 a 2.5 a seguir[5].

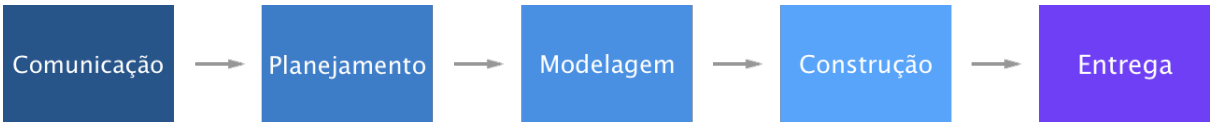

Figura 2.2: Fluxo Linear.

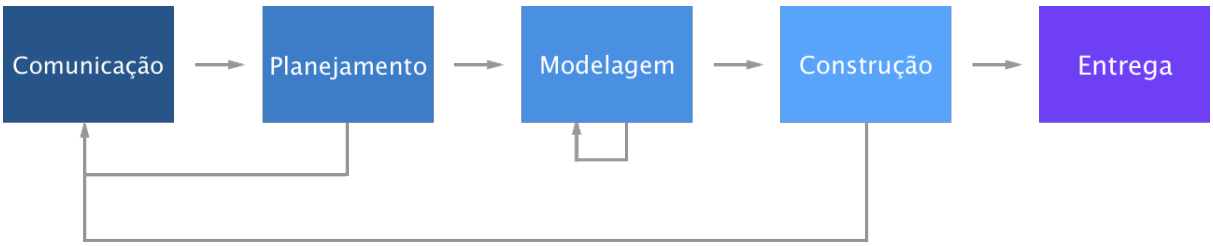

Figura 2.3: Fluxo Iterativo.

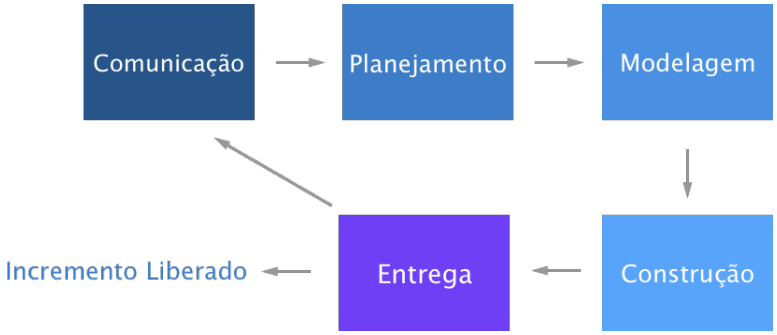

Figura 2.4: Fluxo Evolucionário.

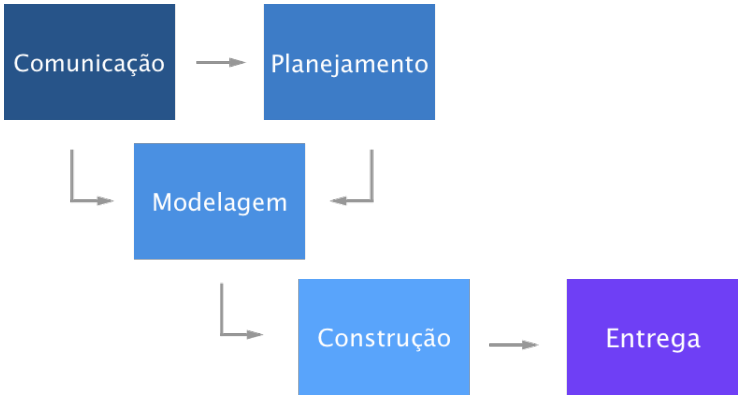

Figura 2.5: Fluxo Paralelo.

No fluxo linear, executa-se cada atividade em sequência, sendo que uma começa quando a anterior termina, culminando com a entrega. Este é de difícil ocorrência no mercado, visto que novas tecnologias e necessidades do *software* surgem de forma muito rápida. Em um ambiente de *Startup*, a velocidade dessas mudanças é ainda maior, fazendo com que a agilidade da equipe seja exigida ao máximo. Os demais fluxos são mais próximos da realidade em que os processos vão se repetindo indefinidamente ou volta-se a uma fase anterior quando necessário. Além disso, após o lançamento de funcionalidades é que realmente é avaliado se o que foi projetado obteve sucesso ou pode ser melhorado dependendo do retorno de resposta do usuário. Cada vez mais a opinião do usuário tem sido foco dos desenvolvedores de *software*. O princípio de "foco no usuário"vem sendo extremamente utilizado na criação de soluções em TI (Tecnologia da Informação) e esse é um dos grandes motivos para que os *softwares* tenham uma elevada taxa de atualização.

Para este estudo, dois fluxos de processo foram analisados. O método Cascata segue o fluxo linear, ao passo que o *Scrum* tem uma estrutura de fluxo evolucionária. O primeiro é engessado e tradicional, o segundo permite que o programa evolua para se adaptar à demanda. Assim, o fluxo evolucionário é encontrado na grande maioria das *startups* e empresas de desenvolvimento de *software*.

2.2 O Método Cascata

O método em Cascata tem sua origem nos procedimentos já utilizados em áreas mais antigas da engenharia. Assim o método tem um formato muito claro de sua estrutura e organização. Também conhecido no mercado como "escopo fechado", esta metodologia se caracteriza como um método de fluxo linear, em que uma fase só se inicia quando a anterior termina. Todavia, é possível retornar a algum estado anterior caso seja detectado algum erro. A literatura diz ainda que "a fase seguinte não deve se iniciar até que a fase precedente tenha sido concluída. Na prática, esses estágios sobrepõem e trocam informações entre si"(citar Pressman)(pág 38). Essa possibilidade prejudica diretamente na entrega final do projeto. A seguir pode-se observar como essa estrutura funciona.

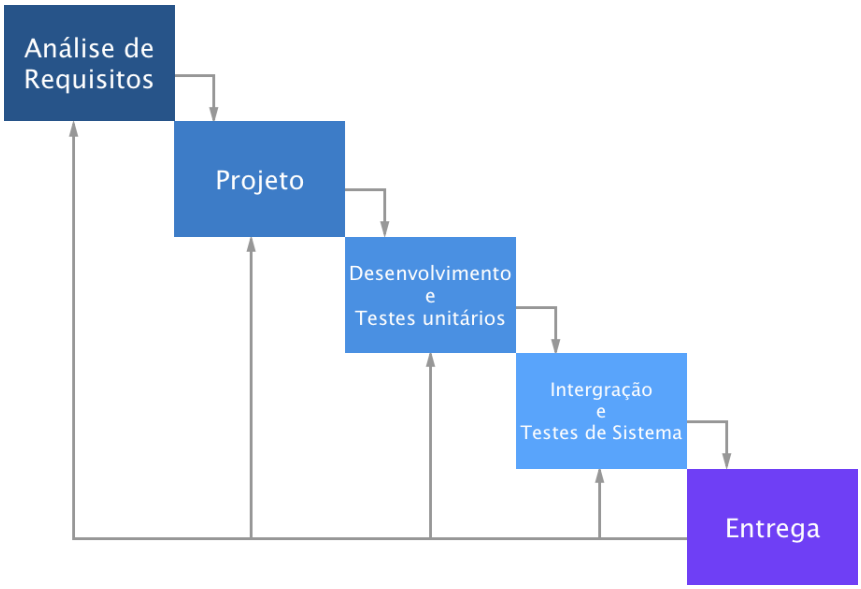

Figura 2.6: Fluxo em Cascata.

O nome "cascata"(em inglês *waterfall*) se dá devido ao formado do fluxograma da metodologia e a estrutura de que a etapa mais inferior só se inicia quando a superior é finalizada. Na primeira fase, o engenheiro de *software* deve analisar e documentar as funções necessárias para todo o projeto, verificar quais as tecnologias que poderão ser utilizadas e definir as especificações de todo o sistema que será desenvolvido. Posteriormente, a fase de projeto (ou modelagem) se inicia. Nesse momento o engenheiro, munido das informações colhidas na análise de requisitos, deve projetar todo o sistema, como ele irá se comportar, o que o usuário esperaria em termos de experiência e definir toda a estrutura do *software* em que os desenvolvedores irão se basear para poder implementar o código. Nessa fase o escopo completo do projeto deve ser definido. Essas duas fases são compostas por tarefas bem definidas, logo pode-se mensurar quais atividades estão sendo completadas nos prazos.

As fases subsequentes são as fases em que o desenvolvedor elabora o código e faz testes de unidade, ou seja, testa-se as funções que foram implementadas de forma individual. Nesse etapa este estudo mensura quanto tempo é gasto para se concluir a fase de implementação. Em seguida inicia-se a fase de integração e teste do sistema, em que todas as funções são unificadas para compor um só sistema. Como esta etapa também pode envolver trabalho de implementação por parte do desenvolvedor, o tempo de trabalho pode ser medido.

Por fim, quando o sistema é lançado para o usuário final, começa-se a fase de operação e manutenção. A última fase consiste em permitir que o programa continue em uso de maneira estável, sendo que é necessária codificação de manutenção e nenhuma nova funcionalidade

é implementada. Caso novas funções e características do sistema tenham que ser alteradas, deve-se iniciar todo o fluxo a partir da análise de requisitos.

O ideal para este método é quando o processo ocorre sem interrupções da fase de requisitos até a entrega final. Porém, isso não é o que ocorre na prática devido as mudanças de rumo no projeto por motivos como: novas tecnologias lançadas que se adequem melhor ao sistema, novas possibilidades de idéias do cliente, erro em qualquer uma das fases anteriores que só foram verificados em uma fase posterior, dentre outros. Os autores do livro "Engenharia de Software, Uma Abordagem Profissional"[5] declaram que "durante o projeto, são identificados problemas com os requisitos; durante a codificação, são verificados problemas de projeto, e assim por diante". Com isso ocorre a necessidade de retrabalhos e atraso na entrega, ou seja, elevado desperdício de recursos. Nos casos estudados, será verificado que a redefinição do projetos durante todo o fluxo é algo muito comum quando se trata de soluções em TI, especialmente em *startups*.

2.3 O *Scrum*

O método *Scrum* foi proposto na década de 80 por Hirotaka Takeuchi e Ikujiro Nonaka, visando uma estratégia de produção incremental colaborativa - como descrito em "A Guide to the Scrum body of knowledge (SBOK Guide)"[6](Um Guia para o corpo de conhecimento *Scrum*, em português). Em 1995, Ken Schwaber e Jeff Sutherland aplicaram o conceito para desenvolvimento de *software*. E gradativamente esse método vem tomando espaço no mercado, sendo incorporado por empresas em fase embrionária e empresas de grande porte que adotam uma filosofia de desenvolvimento ágil em seus processos.

O *Scrum* tem influência direta do Sistema Toyota de Produção, desenvolvido por Taiichi Ohno e, por conseguinte, tem princípios japoneses em sua filosofia. O conceito consiste em criar equipes compactas que realizam tarefas bem definidas em um intervalo de tempo curto, denominado *Sprint*. Em cada *Sprint*, um ciclo deve ser finalizado e um *software* utilizável deve ser disponibilizado para uso. Isso se assemelha muito ao método de prototipação, mas há diferenças claras, como será visto.

O *Scrum* segue seis princípios básicos[6]. Baseado no Guia SBOK, são eles: processo de controle empírico, auto-organização, colaboração, priorização baseada em valores, *time-boxing* (em português, caixa de tempo) e desenvolvimento iterativo. O princípio de controle empírico enfatiza o ponto de equilíbrio do método, baseado na transparência, inspeção e

adaptação. A auto-organização diz respeito ao apoio aos trabalhadores, que são aqueles que agregam valor ao produto. Esse princípio visa o compartilhamento de responsabilidades. O terceiro princípio coloca o foco na colaboração entre os membros da equipe e como este time deve trabalhar de forma compacta e harmônica. O princípio seguinte trata da priorização do que agrega mais valor ao produto naquele momento, sendo as tarefas mais importantes executadas primeiro. O *Scrum* percebe que tempo é um recurso escasso, assim o princípio de *time-boxing* define estratégias para utilizar o tempo da forma mais eficiente possível. Alguns elementos do *time-boxing* são os *Sprints*, *Daily Standup Meetings* (em português, reuniões diárias em pé), *Sprint Planning Meeting* (em português, reuniões de planejamento de *Sprint*) e *Sprint Review Meeting* (em português, reuniões de revisão de *Sprint*). E o último princípio se refere à estrutura iterativa de desenvolvimento. Este permite que o *software* seja desenvolvido com foco nas necessidades reais do usuário, pois neste contexto é possível se obter informações de respostas dos usuários e ter o poder de reação necessário para satisfazê-los.

O *Scrum* tem uma estrutura evolucionária muito bem definida. A figura a seguir mostra como essa dinâmica ocorre.

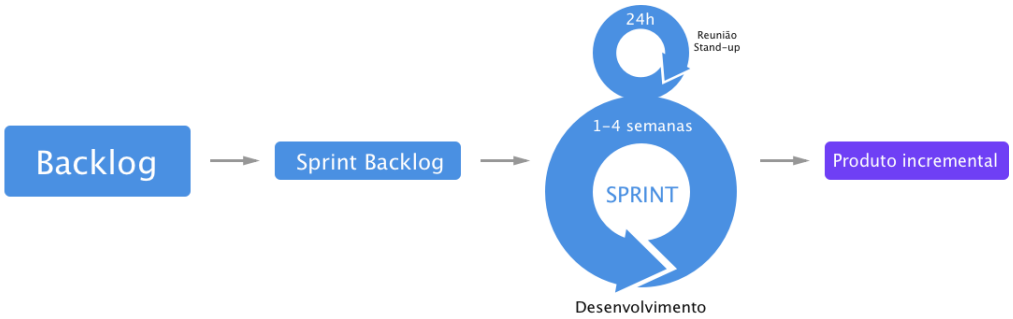

Figura 2.7: Estrutura SCRUM.

O primeiro elemento da figura é o *backlog*, que é a lista de todas as funcionalidades que o projeto visa ter a longo prazo. Essa prática é importante para a equipe ter visão das possibilidades que aquele sistema que está sendo criado pode abordar. A partir do *backlog* que é possívei se definir prioridades de desenvolvimento e construir o *sprint backlog*, que é a lista de todas as funcionalidades que serão desenvolvidas naquele*sprint* em específico. O *Sprint* é um ciclo de desenvolvimento, e a metodologia do *Scrum* indica que um *Sprint* tenha duração de duas a quatro semanas. Todavia, é possível encontrar no mercado empresas que fogem a essa regra, fazendo ciclos de desenvolvimento de apenas uma semana.

O *sprint backlog* é definido em planejamento (chamado de *planning meeting*), que é o momento em que toda a equipe se reúne para definição do *Sprint*. Essa reunião é importante para que cada membro da equipe possa opinar e que de forma colaborativa todos tenham o conhecimento do que precisa e pode ser desenvolvido. Essas definições são reavaliadas durante o desenvolvimento em encontros diários (*Daily Standup Meetings*). Estes são momentos breves de conversa, no qual se relata o que já foi feito, qual o próximo passo e se houve algum problema encontrado. Caso haja algum problema, isso fica a cargo do *Scrum Master*, que é o membro da equipe com a incumbência de minimizar despedícios e agilizar o desenvolvimento. É ele que lidera a equipe durante os *Sprints* e organiza a equipe, inclusive guiando todos na reunião de planejamento. Jeff Sutherland comenta em seu livro [] "ele ou ela conduziria todas as reuniões, se certificaria de que houvesse transparência e, o mais importante, ajudaria a equipe a descobrir o que estava atrapalhando o andamento do projeto"."

O *product owner* (em português, dono do produto) representa o cliente. Em casos de *startups*, pode ser o líder do negócio.

A equipe no *Scrum* dever ser compacta e variada. Jeff Sutherland indica que o número máximo de integrantes da equipe é de nove pessoas e ela deve englobar programadores, designers e gestores, além do próprio *Scrum Master*[2]. O objetivo dessa equipe tão enxuta é proporcionar entrosamento de forma rápida. Assim todos os membros podem interagir facilmente. Isso permite que os seis princípios da metodologia sejam possíveis. Com uma equipe pequena é possível se fazer reuniões rápidas de atualização da informação para gerir melhor as pessoas, criar empatia entre os membro para incentivar a colaboração e engajar toda a equipe no projeto.

É importante salientar dois pontos sobre o *Scrum*. Primeiramente, ele não tem um escopo fechado e bem definido. Apesar de ter um *backlog* como guia, o Scrum não indica um prazo final onde todas as funcionalidades irão estar completamente implementadas, nem mesmo se todas elas serão de fato feitas. A execução das tarefas prioriza as necessidades reais do projeto, que depende diretamente do momento de mercado em que o *software* em questão está inserido. Essa abordagem assume que é impossível prever como os usuários irão se comportar num futuro distante e se concentra em resolver problemas à medida que eles surgem. Em empresas jovens, o gasto estrategicamente equivocado de seus recursos pode levar ao fracasso, portanto é fundamental que a maior parte do tempo seja despendida em atividades que realmente agregem valor à solução.

O Google é uma das empresas que utiliza metodologia ágil em seus projetos e no "Como

O Google Funciona"[7], de autoria de Eric Schmidt e Jonathan Rosenberg, é relatado: "não temos a menor ideia de qual é seu empreendimento ou sua indústria, então não ousaremos lhe dizer como criar um plano de negócios. Contudo, podemos afirmar com 100% de certeza que, se você tem um plano, ele está errado", e continua "na verdade, é ótimo ter um plano, mas saiba que ele mudará conforme você progride e descobre novas coisas sobre os produtos e o mercado. Essa reação rápida é fundamental para o sucesso, mas as fundações do plano são tão importantes quanto". O plano é o *backlog*, e a agilidade para mudar conforme se progride é proporcionada pela dinâmica do *Scrum*. Deve-se sempre assumir que o plano pode estar errado e deve-se estar preparado para mudar de direção. As ferramentas que o método utiliza para embasar as reformulações de planejamento estão nos *feedbacks* constantes do usuário e da equipe. Este é um dos grandes benefícios do *Scrum*.

2.4 Sobre o Método Espiral ou Prototipação

Proposto por Boehm em 1988[5], a metodologia de prototipação é amplamente conhecida, assim como a tradicional em Cascata. Nesse método, representado por uma espiral, cada nova versão do *software* é obtida ao passo que se completa uma volta no plano de incremento.

Figura 2.8: Estrutura Espiral ou Prototipação.

As fases deste modelo, segundo Sommerville[4], são quatro, como ilustrado na figura 2.4. A primeira fase, de definição do objetivo, é em que se define quais os fins daquele ciclo, analisando os riscos e se é preciso tomar medidas alternativas. No segundo quadrante, seguindo o sentido horário, se faz avaliações profundas sobre os riscos que envolvem o desenvolvimento do *software* especificado e como minimizar os riscos ao máximo. Subsequentemente,

inicia-se a fase de desenvolvimento e validação, em que se escolhe qual modelo de desenvolvimento de *software* é mais adequado, dependendo de quais os riscos envolvidos e qual o foco do sistema. E por fim se planeja quais os próximos passos para novo ciclo a depender do que ocorra nas fases anteriores.

Apesar da metodologia se assemelhar ao *Scrum*, ela não define como todo o processo deve ocorrer, a estrutura da equipe nem tampouco os ciclos no método de prototipação são curtos o suficiente para reagir às necessidades dos usuários. Desse modo, é possível se cometer erros parecidos aos cometidos no modelo em Cascata. Inclusive, Sommervile[] define a fase de desenvolvimento e validação da seguinte forma: "(...) é escolhido um modelo de desenvolvimento para o sistema. Por exemplo, se forem dominantes os riscos relacionados à interface com o usuário, um modelo apropriado de desenvolvimento pode ser a prototipação evolucionária. Se os riscos de segurança forem a principal consideração, o desenvolvimento com base em transformações formais poderá ser o mais apropriado e assim por diante. O modelo em Cascata poderá ser o modelo de desenvolvimento mais apropriado se o risco principal identificado for o da integração de sistemas". Sendo assim, como o modelo de prototipação permite que qualquer método possa ser empregado na fase de desenvolvimento a depender da necessidade, ele não pode ser equivalente ao *Scrum*.

Capítulo 3

Material e Métodos

Este capítulo é exclusivamente dedicado à apresentação de todas as ferramentas que foram utilizadas nos projetos desenvolvidos, a composição das equipes e o conteúdo de cada projeto. Esta seção do trabalho tem um foco maior no conceito do Cubos Time, *software* utilizado para metrificação do uso de recurso e a interação entre os componentes das equipes. Por motivos de confidencialidade, não serão listadas as descrições das funções exatas dos *softwares* desenvolvidos, mas quais tecnologias foram empregadas. Pelo mesmo motivo, não será revelado o nome dos componentes das equipes. Além disso, é importante salientar que todos os projetos foram desenvolvidos no mesmo espaço físico seguindo as mesmas bases culturais da empresa Cubos Tecnologia, que se voluntariou a participar do estudo.

3.1 As Equipes

No estudo aqui apresentado, foram selecionadas três equipes a serem acompanhadas durante o desenvolvimento dos projetos. Os três grupos foram compostas por dois desenvolvedores e um *designer*, bem como um gerente de projeto, que para o modelo *Scrum* obteve o posto de *Scrum Master*.

O *designer* tem a função de trabalhar na UX e na UI das aplicações. Com relação à experiência do usuário, o *designer*, profissional capacitado e com domínio das tecnologias que serão empregadas, também precisa se comunicar frequentimente com o desenvolvedor para dar diretrizes do leiaute do projeto.

A equipe é composta por dois desenvolvedores, sendo um responsável por toda a construção de *BackEnd* do projeto, ou seja, estruturação e criação do banco de dados, comunicação entre o banco de dados e a aplicação, integrações com API(*Application Programing Inter-*

face, em português Interface de Aplicação do Programa), configuração das instâncias das máquinas virtuais, criação dos serviços do servidor, configura as conexão entre cliente e servidor e desenvolve as funções relacionada à lógica de negócio. O outro desenvolvedor, por sua vez, é responsável por toda codificação do leiaute da aplicação, todas as funções que são executadas em aparelhos móveis ou na interface do *browser*, requisição de dados do servidor, animações e integrações com API.

Para nomear os desenvolvedores tem-se a seguir na tabela 3.1.

Tabela 3.1: Lista e funções dos desenvolvedores

Desenvolvedor	Plataforma	Especialidade
Dev 1	Web	FrontEnd
Dev 2	Servidor	BackEnd
Dev 3	iOS e Android	FrontEnd
Dev 4	Servidor	BackEnd
Dev 5	iOS e Android	FrontEnd
Dev 6	Servidor	BackEnd

3.2 Os Projetos

Como objeto desse estudo, quadro projetos foram selecionados. Todos eles com mesmo nível de complexidade, segundo julgamento dos desenvolvedores. Para o modelo em Cascata foram selecionados os projetos A e B, e para o modelo *Scrum* os projetos C e D. A tabela 3.2 a seguir mostra essa estrutura.

Tabela 3.2: Distribuição dos projetos

Projeto(s)	Desenvolvedores	Método
Projeto A	Dev 1 e 2	Cascata
Projeto B	Dev 3 e 4	Cascata
Projeto C	Dev 1 e 2	*Scrum*
Projeto D	Dev 5 e 6	*Scrum*

O grupo que contém os desenvolvedores Dev 1 e Dev 2, participaram de um projeto no método Cascata e um projeto no método *Scrum* para assim ser feito uma análise comparativa.

Por outro lado, a equipe composta pelos demais desenvolvedores não trabalharam com mais de uma metodologia, com o objetivo de se comparar a diferença de produtividade desses dois métodos de forma isolada.

Todas as equipes já haviam trabalhado anteriormente em conjunto, sendo assim, não houve fase de adaptação dos membros entre si. A única adaptação que as equipes tiveram que passar foi cm relação ao método empregado, no caso do *Scrum*.

Cada projeto foi composto por uma serie de tecnologias agregadas e funções de usabilidade para plataformas iOS, Android e aplicações *web*. Em todos os projetos foram compostos pelas seguintes funções base: cadastro de usuários; sistema de mensagem em tempo real entre os usuários; sistema de pagamento; integração com API de pagamento; integração com Google Maps; sistema de comentários e avaliações; armazenamento dinâmico de informações; histórico de uso do programa pelo usuário; filtros de busca de informações; nível de usuário; integração com sistema de análise do comportamento de usuários. Demais funcionalidades dos projetos em específico foram consequências de um modelo de negócio de cada caso e partiram da base das funcionalidades listadas anteriormente.

O projeto A se trata de um sistema web voltado ao setor de hotelaria. Neste site o usuário pode interagir buscando hóteis, efetuar reservas e acompanhar todas as ações feitas no sistema. Todo o site foi desenvolvido em React, uma biblioteca de Typescript desenvolvida pelo Facebook. Esta tecnologia possibilta uma seria de vantagens para o usuário, como velocidade nas respostas durante a interação com a interface gráfica. Isso permite que as buscas pelos serviços desejados sejam feitas de uma forma muito dinâmica. Além disso o usuário pode editar o seu perfil, utilizar dados do Facebook para completar suas informações na plataforma.

O projeto B consiste em um sistema de de reserva de serviços residenciais. O usuário pode interagir com o sistema via interface gravida no celular. As plataformas utilizadas para o desenvolvimento dos aplicativos foram o Android e iOS, das empresas Google e Apple respectivamente. Neste aplicativo é possível se cadastrar, fazer busca e pedido de serviços e efetuar pagamento. Além disso o usuário acompanha o status de todos os serviços.

O projeto C é um sistema de acompanhamento médico *online* que possue todas funcionalidades listadas anteriormente. A diferença em relação os outros projetos está em qual funcionalidade o usuário interage mais. Neste caso o sistema de mensagens em tempo real entre os usuários é o foco da plataforma.

O projeto D é um aplicativo para iPhone e dispositivos Android para acompanhamento de gastos em eventos. O usuário pode pagar utilizando o aplicativo, acompanhar seus gastos,

entrar em contato com o suporte técnico por um *chat online*. Além disso, o usuário tem total controle sobre seus dados de perfil, cardápio do evento e local de todos os eventos que utilizam o sistema.

Portanto, em todos os projetos, seja *web* ou aplicativo para *smartphone*, as funcionalidades listadas anteriormente estavam presentes em algum momento do desenvolvimento de cada plataforma. O grau de prioridade de cada função varia de acordo com a necessidade do projeto, ou seja, depende do interesse principal do usuário em determinado caso. É importante lembrar que o foco destes sistemas sempre é o usuário. A abordagem de foco no usuário possibilita que as soluções desenvolvidas apresentem o máximo de valor agregado para as pessoas que realmente irão interagir com o *software* desenvolvido e a partir delas é possível se obter informações constantes que podem auxiliar no aperfeiçoamento do projeto.

É importante salientar que não houve reaproveitamento de código em nenhum dos projetos, isso ocorreu devido ao teor sigiloso de cada desenvolvimento. Além disso, a documentação dos projetos foi feita pela Cubos Tecnologia e este estudo não teve o intuito de avaliar esta etapa do processo de desenvolvimento.

3.3 Ferramentas Utilizadas

Para a comunicação, documentação e acompanhamento dos projetos, algumas ferramentas foram utilizadas. Estas têm o intuito de dar suporte à metodologia de desenvolvimento, buscando agilizar processos, evitar falhas, gerir tarefas, gerir informação e documentação, dar suporte ao desenvolvimento na construção da UX (User Experience, Experiência do Usuário em português) e gerenciamento de códigos.

Na comunicação entre membros da equipe e os clientes foi utilizada a plataforma Slack, que pode ser encontrado na *web*. Este *software* permite integração com outras ferramentas utilizadas pela no trabalho. Deste modo, é possível automatizar alertas, como por exemplo comunicar a todos os membros do projeto quando alguma nova versão do *software* é implementada ou alguma data de entrega está próxima. Para cada projeto dois canais de comunicação foram criado, um para comunicação interna da equipe de desenvolvimento e a outra para comunicação direta com o cliente.

No controle de desenvolvimento do processo foi utilizada a plataforma Asana. Assim, todas as atividades de gestão, *design* e programação foram listadas, associadas a um membro da equipe e estipuladas datas de entrega. No Asana também é possível anexar arquivos

relevantes às tarefas, inserir descrições, criar sub-tarefas e adicionar comentários feitos por componentes da equipe. Para todas as atividades é vital que seja vinculada a responsabilidade para algum membro da equipe. Assim, é possível se avaliar quantas atividades estão previstas no projeto, se as entregas estão sendo feitas no prazo estipulado e se o projeto está bem distribuido na equipe. Na metodologia *Scrum* esta plataforma se foi importante para o armazenamento do *backlog* e do *sprint backlog*.

Para a criação da interface do usuário e elaboração de protótipos para validações tanto de *User Interface* (UI) quanto de *User Experience* (UX), fora utilizados os programas Sketch e Marvel. O Sketch é uma ferramenta utilizada pelos *designers* para composição das telas, elaboração de ícones e criação do fluxo de experiência do usuário. O Marvel, por sua vez, é uma aplicação *web* que possibilita que a experiência criada no Sketch seja reproduzida em forma de protótipo para interação do usuário. Esta estratégia é adotada visando a validação do projeto antes de entrar na fase de codificação. Essa estratégia é tomada para que se evite retrabalhos.

Para gerenciamento de código foi utilizado o GitHub. Este, é um repositório que possibilita o controle dos códigos. Nessa plataforma os projetos são subdivididos e organizados no intuito de otimizar a atualização de novas versões, transferência de responsabilidade pelo código, desenvolvimento em paralelo por parte de mais de um programador e revisão de códigos . Em ambos os processos, tanto no método em Cascata quando no *Scrum*, os projetos foram seccionados no GitHub em pastas da seguinte maneira: uma pasta para API, uma pasta para desenvolvimento do aplicativo móvel ou *frontEnd web* e uma pasta para código do *dashboard* (quando existente no projeto).

3.4 Coletando Dados com O CubosTime

Num processo de desenvolvimento de *software*, como já foi visto, não se resume apenas ao momento de implementação do código, mas também de documentação, comunicação com o cliente e com os usuários, desenvolvimento da interface gráfica do sistema, e testes do software. Contudo a fase de codificação é a mais que contém maior carga de trabalho com valor agregado ao projeto. Por isso, neste estudo, a análise de quanto tempo foi gasto para se fazer implementação dos cógicos é uma das principais métricas utilizadas.

No mercado é possível encontrar diferente sistemas que podem medir o tempo de trabalho de uma pessoa. Porém estes programas precisam ser iniciados e finalizados manualmente.

Por isso, a Cubos Tecnologia desenvolveu o CubosTime, um programa que é instalado no computador do desenvolvedor e é executado em *background*. Este está sempre em execução para evitar que o desenvolvedor esqueça de executa-lo, problema que ocorre em outros sistemas de medição de tempo para gerenciamento de atividades. Deste modo, o desenvolvedor não precisa se preocupar em ter uma tarefa a mais na sua rotina.

O CubosTime segue a seguinte estrutura de análise de informação. O programa assume que um desenvolvedor não pode trabalhar em dois códigos ao mesmo tempo e a cada 20 minutos verifica se alguma das pastas do repositório vinculadas àquele desenvolvedor foi alterada.O *software* então verifica se a alteração sinalizada se trata de um trabalho real ou se alguma ferramenta foi acionada, pois, quando o programador inicia o Android Studio, por exemplo, a própria IDE(do inglês *Integrated Developement Enviroment*) faz alterações no código desenvolvido e salva, porém não se trata de um trabalho real do desenvolvedor. Assim, o CubosTime verifica se a atualização do código é apenas devido à mudança automática de uma das ferramentas ou se realmente um trabalho foi feito. Somente depois disso a diferença no tempo de atualização da pasta é somada ao tempo total de trabalho.

A seguir tem-se o fluxograma de como se estrutura o CubosTime.

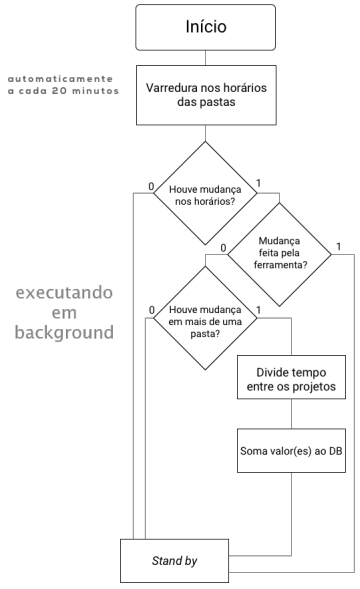

Figura 3.1: Fluxograma do CubosTimes.

Capítulo 4

Resultados e Discussões

Após o entendimento da teoria dos processos e de como este estudo se equipou de ferramentas para o acompanhamento dos projetos, pode-se emfim obter os dados necessário para análise. Para isso tem-se 3 secções onde estão expostos o progresso de cada atividade realizada, o tempo gasto em cada parte de cada projeto, e qual o impacto dos dois métodos com respeito a qualidade dos códigos e ambiente de trabalho.

4.1 Andamento das Tarefas

Com a ferramenta Asana pode-se obter dados de quantas tarefas foram definidas e quantas foram completadas ao decorrer do projeto. Essas tarefas são referentes a atividades de comunicação, projeto, codificação e testes. A seguir tem-se os gráficos dos projetos A e B, desenvolvidos pelo Método em Cascata.

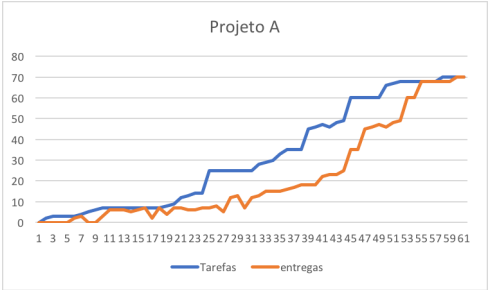

Figura 4.1: Gráficos das tarefas do projetos A - Método Cascata.

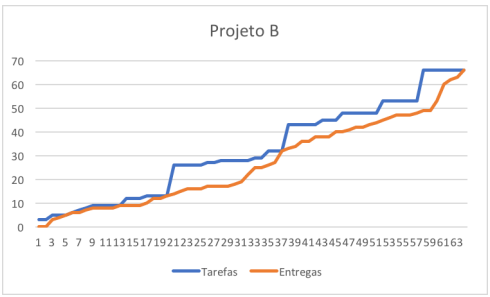

Figura 4.2: Gráficos das tarefas do projetos B - Método Cascata.

A seguir tem-se os gráficos dos projetos C e D, desenvolvidos com o *Scrum*.

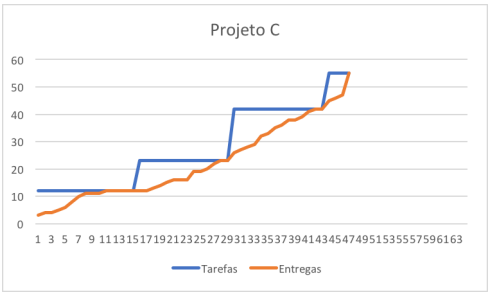

Figura 4.3: Gráficos dos projetos C - *Scrum*.

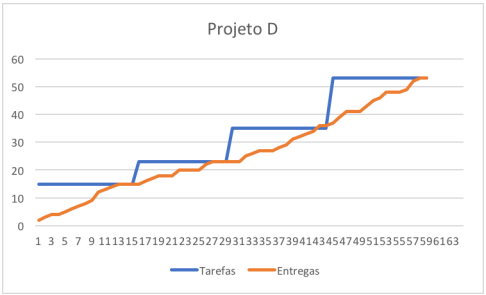

Figura 4.4: Gráficos dos projetos D - *Scrum*.

As linhas azuis representam o total de atividades criadas em cada dia de desenvolvimento, as linhas vermelhas, por sua vez, são o total de atividades completadas. Os pontos de encontro entre as duas linhas mostrar quando todas as tarefas definidas até o momento foram completadas.

Como pode ser observado nos gráficos anteriormente, tanto nos projetos feitos no Método Cascata quanto *Scrum* as linhas de tarefas e entregas são ascendentes. Isso mostra que durante o processo de desenvolvimento mais necessidades para o *software* surgem gradativamente. Mesmo que com uma análise de requisitos extremamente rigorosa, esse gráfico continua sendo real. O acrescimo no número de atividades é esperado pela metodologia ágil,

isso é evidenciado pelo formato de escada que as linhas de tarefas dos gráficos D e C traçam. O Método Cascata ignora essa realidade e faz com que prazos sejam prorrogados constantemente.

Um dos fatores que podem gerar esse gráfico crescente é uma fase de análise de requisitos mal feita, gerando assim problemas de projeto, por sua vez problemas de código e assim sucessivamente. Porém não se pode designar este fator como único motivo possível. Pois, defasamento de tecnologia, mudanças na lógica de negócio (a qual depende diretamente do comportamento do mercado que o *software* está inserido), falha na comunicação, influenciam diretamente para que esses gráficos tenham o formato apresentado.

Ainda analisando os gráficos, pode-se observar que na evolução do projeto a quantidade de entregas no método em Cascata são menores do que a do *Scrum*. Isso ocorre por uma questão de princípios das metodologias e revela a inflexibilidade do modelo em Cascata. Nos Gráficos A e B, não há intersecção das linhas em um estágio mais avançado dos projetos, essa é uma evidencia de atrasos na entrega e mudanças de plano não esperada pelos desenvolvedores. Isso gera conflitos entre desenvolvedores e clientes, causando desconforto no ambiente de trabalho.

Problemas no ambiente de trabalho proporcionam duas consequências perigosas para uma empresa. Primeiramente há um impacto direto na qualidade do produto. Conflitos desestimulam a equipe a fazer melhores entregas e manter o alto nível de comunicação interna e externa. Um outro fator que é impactado é o de formação de equipe. Em um ambiente de trabalho inóspito há perda de talentos, ou seja, funcionário infelizes tentem a buscar alternativas para uma vida melhor, assim a empresa pode perde o seu recurso mais valioso.

Outro ganho importante que esses gráficos revelam é a data final de entrega. É possível observar que, como os desenvolvedores assumiram que os projetos são equivalentes, há um ganho de cerca de 15 dias quando utilizada a metodologia *Scrum*. Tanto no caso de equipes distintas desenvolvendo *softwares* com métodos diferentes (como é o caso do gráfico B e D), quanto a mesma equipe desenvolvendo dois projetos com métodos diferentes (como é o caso dos projetos A e C), a diferença de data de entrega foi observada. Assim, é evidente que há ganho real na velocidade em que se desenvolve utilizando o método ágil. Como se gasta menos tempo para produzir o mesmo nível de trabalho, se ganha na produção.

4.2 Qualidade do *software*

É importante sempre estar atento à qualidade do *software*. Nestes estudo, o parâmetro utilizado para medir a qualidade dos códigos desenvolvidos, foram a quantidade de funções que precisaram ser refeitas. Ainda utilizando a plataforma Asana, foi possível construir a tabela a seguir que mostra quantas das funcionalidade definidas nos projetos foram produzidas mais de uma vez.

Tabela 4.1: Retrabalhos registrados

Projeto	Número de funções refeitas
Projeto A	23
Projeto B	15
Projeto C	3
Projeto D	4

Como no método em Cascata os requisitos são avaliados em uma fase prévia isolada, ele não acompanha as reais necessidades de uma *start-up* e de projetos em TI em geral. O grande número de funções refeitas nos projetos A e B ocorrem devido a mudança de realidade que o mercado vai apresentando conforme o tempo passa e o programa precisa se adaptar para isso. Nos projetos feitos com métodologia ágil, as funções vão sendo criadas conforme as necessidades do momento, logo elas se apresentam mais estáveis. Além disso, com o processo iterativo, o desenvolvedor tem tempo hábil para focar em uma funcionalidade de cada vez e assim evita erros.

4.3 Utilização dos recursos

O maior recurso em uma empresa de desenvolvimento de programas computacionais são as pessoas, os profissionais que colocam em código sua capacidade de implementação. Logo, o gasto de recurso que tem maior impacto nesse setor é o tempo. Como já explanado anteriormente, o CubosTime captura o tempo total gasto na fase de codificação e permite que todos os componentes da equipe possam se estruturar e planejar as próximas ações baseados em dados obtidos das suas experiências nos projetos.

A seguir tem-se os tempos totais de desenvolvimento em cada projeto. O fator de parada de medição, para que seja possível criar uma análise comparativa entre os métodos, foi a

estabilidade do programam no mercado.

Tabela 4.2: Medição do Cubos Time

Projeto	Tempo Total em horas
Projeto A	942,51
Projeto B	657,33
Projeto C	326,04
Projeto D	489,64

Os projetos A e C foram desenvolvidos pela mesma equipe. Esta parte do estudo tem o intuito de analisar como uma mesma equipe se comporta praticando duas metodologias distintas. Os números são evidentes quanto a diferença de horas trabalhadas e de tarefas executadas no prazo. Com o método ágil o tempo reduz consideravelmente, e isso ocorre por dois fatores importantes. O projeto evolui junto com a comunicação, assim há menos atritos entre as partes e com melhor entendimento sobre o projeto, as ações tomadas são mais precisas. Além disso, os impactos negativos que o mercado pode trazer ao projeto são mitigados pelo modo que o *Scrum* é desenhado.

Esta diferença de produtividade tem consequências diretas no orçamento dos projetos. O projeto A, por exemplo, teve um orçamento previsto de trinta mil reais. Se for proposto o custo médio de setenta reais por hora do desenvolvedor e utilizando dois desenvolvedores, tem-se cerca de 215 horas previstas para execução do projeto. Contudo, com mudanças feitas durante o desenvolvimento, o tempo total medido foi de cerca de 942 horas trabalhadas. Assim, a empresa teve um prejuízo de aproximadamente 160 mil reais. Não estão sendo contabilizados encargos e despesas internas da empresa. Para um *start-up* isso provavelmente significará o fracasso ou um gravíssimo problema com os investidores.

4.4 Depoimento das equipes

Um fator de extrema relevância na produtividade e qualidade dos produtos é o nível de satisfação das pessoas que trabalham nos projetos. Para qualquer negócio, e especialmente para empresas muito novas, é fundamental se contruir uma cultura que possa engajar os profissionais e assim elevar a criatividade ao máximo. Uma boa estrutura de processos deve está alinhada com os pensamentos da empresa e das pessoas que a sustentam.

Portanto, é importante se obter opiniões sobre o time que desenvolveu os projetos aqui

estudados. Algumas perguntas foram feitas. Para a equipe que utilizou o metodo em Cascata e o *Scrum* foi perguntado": Depois de utilizar o método os dois métodos, quais aspectos você acredita ser mais relevantes se essas duas metodologias forem comparadas? Ao adotar *Scrum*, a equipe notou alguma mudança no seu ambiente de trabalho? Ao adotar o *Scrum*, qual fator na dinâmica da comunicação com o cliente que você gostaria de resaltar? Quais os principais benefícios do *Scrum*?"

Assim, a equipe respondeu: "Acreditamos que o maior impacto que sentimos ao comparar as duas metodologias foi a dinâmica de cada uma. O *Scrum* faz com que estejamos todos juntos pensando sobre os problemas o tempo inteiro e essa interação nos faz produzir de forma mais rápida. Algo muito positivo também foi ter visão do que estavamos produzindo. Quando trabalhamos em Cascata estavamos fazendo um sistema pensando muito a frente e quando o dia chegava não era mais aquilo que precisavamos ter desenvolvido. A partir daí era redefinir o projeto e refazer o que precisava ser refeito. Como esses acontecimentos foram frustrando o cliente, a comunicação foi se tornando cada vez pior. Um outro grande benefício que sentimos do *Scrum* foi a melhora na comunicação, tanto da equipe quanto com o cliente". Declararam os desenvolvedores Guilherme Bernal e Victor Magalhães.

Para as equipe que desenvolveu o projeto B e D e não tiveram contato com apenas uma metodologia foram feitas as seguintes perguntas: "Qual a maior dificuldade do projeto? Houve alguma barreira técnica? O cliente se demontrou satisfeiro ao finalizar o projeto? Ocorreram atrasos nas entregas? Os problemas encontrados foram solucionados rapidamente?

A equipe do projeto B declarou que: "A maior dificuldade apresentada foi de comunicação, o cliente não sabia detalhar todas as funcionalidades como ele desejava e nunca tinha tempo disponível para reuniões muito longas onde nós desenvolvedores tivessemos a oportunidade de relatar cada ponto relevante. Além de que constantemente o cliente mudava de opinião com base em conversas que tinha com investidores e pessoas fora do ambiente de desenvolvimento". A equipe então prossegue falando, "Não houve nenhuma barreira técnica, as tecnologias utilizadas já eram de conhecimento da empresa, porém ocorreram problemas, novamente devido ao mecanismo de comunicação entre as partes, gerando assim atrasos nas entregas. Resolvemos os problemas o mais rápido que foi possível, mas ao final do projeto o cliente não estava 100% satisfeito. O programa ficou bom, porém os atrasos e gastos não previstos foram pontos negativos". Disseram os desenvolvedores João Pedro Gouveia e José Messias Junior.

A equipe do projeto D por sua vez declarou que: "A maior dificuldade foi entender o que os usuários queriam, tecnicamente não enfrentamos nenhum problema impossível, mas tivemos que pensar bastante como a experiência do usuário deveria ser. Mas mesmo assim, depois de muito trabalho o cliente se mostrou satisfeitos, e nós da equipe também. Não houve nenhum atraso nas entregas, porque tinhamos o controle das tarefas e das datas o tempo inteiro, se algum problema acontecia ele era rapidamente solucionado". Comentaram os desenvolvedores Clara Battesini e Rodrigo Araújo

Como relatado pelas equipes, o *Scrum* não apenas melhora a estrutura do desenvolvimento de programas dando a ela maior flexibilidade e agilidade, a metodologia também impacta na qualidade do ambiente de trabalho. Com uma equipe mais motivada e sincronizada se obtem melhores resultados a curto, médio e longo prazo. A motivação é gerada porque é criado o hábito de entregas curtas. O escritor *best seller* Chales Duhigg comenta em seu livro "O Poder do Hábito"[8]: "Uma vez que uma pequena vitória foi conquistada, forças que favorecem outra pequena vitória são postas em movimento. Pequenas vitórias alimentam mudanças transformadoras, elevando vantagens minúsculas a padrões que convencem as pessoas de que conquistas maiores estão dentro de seu alcance". Por outro lado, o método em Cascata, além de permitir que ocorram desperdícios no processo produtivo, torna os profissionais mais infelizes e os clientes insatisfeitos.

Capítulo 5

Conclusão

Após a aplicação dos dois métodos de desenvolvimento nas equipes da Cubos Tecnologia, que se voluntariou a participar deste estudo de caso, ficou evitente os benefícios que a metodologia ágil na produção de programas na área de TI. O primeira fator relevante é o menor tempo de trabalho gasto para execução de tarefas durante o decorrer do projeto. Além disso, a quantidade de atividades refeitas e acidentes de percurso utilizando o método *Scrum* caíram drasticamente quando comparados aos projetos feitos em Cascata . E por fim, o melhoramento da qualidade de vida dos profissionais envolvidos.

Quando a equipe gasta menos tempo, impacta diretamento de forma benéfica na saúde financeira da empresa contratada e do projeto em si. A alta produtividade encontrada ao se consumir menos tempo, possibilita respostas mais rápidas ao mercado em termos de disponibilizar novas versões aperfeiçoadas do *software* e assim agregar mais valor ao trabalho do desenvolvedor. Deste modo, é possível se destacadas em um cenário de concorrência com outras empresas de desenvolvimento. Como consequência, há um ganho considerávem nos custos e lucros envolvidos. Nos casos aqui apresentados, houve um ganho aproximado de 49% na receita da empresa. Este é um número que permite que a empresa atinja um outro patamar no mercado.

Ao se analisar a quantidade de retrabalhos feitos em cada projeto, percebeu-se que o método Cascata apresenta uma recorrencia maior do que o *Scrum*. Isso mostra como no método tradicional há mais desperdícios e menor qualidade nos programas desenvolvidos. Apesar de todos os projetos estarem atualmente em funcionamento de maneira estável, houve uma demora significativa na convergência dos sistemas desenvolvidos em Cascata. Portanto, pode-se dizer que houve uma influência negativa por parte da metodologia tradicional quando comparada ao desenvolvimento ágil.

Por fim, o *Scrum* se apresentou como um método que estrutura a equipe para produzir em alto nível, assumindo sempre a realidade em que as pessoas trabalham e como o mercado se comporta. Os princípios que o método se sustenta permitem que a equipe trabalhe de forma compacta, colaborativa e mais feliz. A satisfação em produzir e a dinâmica de trabalho impulsiona os profissionais a entregar produtos de maior qualidade e em menos tempo. Todos esses fatores são extremamente benéficas para emprendimentos como *startups*.

Referências

[1] Jake Knapp, John Zeratsky, and Braden Kowitz. *Sprint: how to solve big problems and test new ideas in just five days*. Simon and Schuster, 2016.

[2] Jeff Sutherland. *Scrum: a arte de fazer o dobro do trabalho na metade do tempo*. Leya, 2016.

[3] Eric Ries. *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Crown Business, 2011.

[4] Ian Sommerville, Selma Shin Shimizu Melnikoff, Reginaldo Arakaki, and Edilson de Andrade Barbosa. *Engenharia de software*, volume 6. Addison Wesley São Paulo, 2003.

[5] Roger S Pressman. Engenharia de software; tradução josé carlos barbosa dos santos. *São Paulo: Makron*, 1995.

[6] T Satpathy. A guide to the scrum body of knowledge, 2013.

[7] Eric Schmidt and Jonathan Rosenberg. *Como o Google funciona*. Editora Intrinseca, 2015.

[8] Charles Duhigg. *O poder do hábito: por que fazemos o que fazemos na vida e nos negócios*. Editora Objetiva, 2012.