UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC
CURSO DE CIÊNCIA DA COMPUTAÇÃO

FLARIS BARRETO MARTINHAGO

UTILIZAÇÃO DE MODELOS HÍBRIDOS PARA A DEFINIÇÃO E EXECUÇÃO DE PROCESSOS DE SOFTWARE APLICADOS NA OTIMIZAÇÃO DA PRODUTIVIDADE

CRICIÚMA
2018

FLARIS BARRETO MARTINHAGO

UTILIZAÇÃO DE MODELOS HÍBRIDOS PARA A DEFINIÇÃO E EXECUÇÃO DE PROCESSOS DE SOFTWARE APLICADOS NA OTIMIZAÇÃO DA PRODUTIVIDADE

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. MSc. Gustavo Bisognin

CRICIÚMA

2018

FLARIS BARRETO MARTINHAGO

UTILIZAÇÃO DE MODELOS HÍBRIDOS PARA A DEFINIÇÃO E EXECUÇÃO DE PROCESSOS DE SOFTWARE APLICADOS NA OTIMIZAÇÃO DA PRODUTIVIDADE

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Engenharia de Software.

Criciúma, 25 de junho de 2018.

BANCA EXAMINADORA

Prof. Gustavo Bisognin – Me. – (UNESC) – Orientador

Profa. Ana Claudia Garcia Barbosa – Ma. – (UNESC)

Prof. Matheus Leandro Ferreira – Esp. – (UNESC)

RESUMO

A busca por mais performance faz com que organizações utilizem modelos ágeis para otimizar os resultados, contudo quando esses modelos são aplicados em times inexperientes tornam-se um problema, implicando na Lei de Parkinson e na síndrome do estudante. A partir desta premissa foi criado um modelo híbrido de processo de desenvolvimento de software e aplicado em um time de desenvolvimento de uma empresa da região de Criciúma – SC, otimizando a performance do time, eliminando gargalos no processo de definição, desenvolvimento e teste, identificados por meio do *throughput*, do *lead time* e do *cumulative flow diagram.* O modelo híbrido criado eliminou os efeitos da Lei de Parkinson e da síndrome do estudante do processo de desenvolvimento, atingindo maior eficiência no tempo trabalhado durante os sprints, minimizando dias sem desenvolvimento e teste.

Palavras-chave: Engenharia de Software; Modelo Híbrido; Lei de Parkinson; Síndrome do Estudante.

ABSTRACT

The search for more performance makes organizations utilize agile models to optimize the results, however when these models are applied in inexperienced teams, they become a problem, implying in the Parkinson Law and student syndrome. From this premise, it was created a hybrid model of software development process and it was applied in a develop team in a company from region of Criciúma – SC, optimizing the team's performance eliminating bottlenecks in the definition process, development process and test process, identified through throughput, lead time and cumulative flow diagram. The hybrid model that were created, eliminated the efects from Parkinson Law and student syndrome from the development process, reaching more efficiency in the worked time during the sprints, minimizing days without development and test.

Key-words: Software Engineer; Hybrid Model; Parkinson Law; Student Syndrome.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CFD	Cumulative Flow Diagram
CMMI	Capability Maturity Model Integration
MA-MPS	Modelo de Avaliação de Melhoria de Processo de Software
MPS.BR	Melhoria de Processo de Software Brasileiro
PEP	Product Evolution Process
RUP	Rational Unified Process
SCAMPI	Standard CMMI Appraisal Method for Process Improvement
TDD	Test Driven Development
TFS	Team Foundation Server
XP	Extreme Programming

SUMÁRIO

1 INTRODUÇÃO

Conforme dados levantados pelo Standish Group somente 6% dos projetos de grande porte obtém sucesso (VENTURA, 2016). Neste contexto, os principais problemas apresentados, referem-se a atrasos na entrega, falhas de implementação, problemas relacionados a estimativas e orçamentos. Uma das formas encontradas por diversas equipes de desenvolvimento na área de software é a utilização de modelos que abordam um contexto ágil, porém estes modelos, quando aplicados em equipes com pouca maturidade acabam agravando os problemas, principalmente no que tange a produtividade bem como a qualidade das entregas. Este comprometimento pode ser observado em dados levantados nas medições realizadas durante a execução das *sprints* de desenvolvimento em diversas equipes, sendo que a principal causa está relacionada a Lei de Parkinson e a síndrome do estudante, mesmo utilizando os conceitos da metodologia ágil.

A Lei de Parkinson descreve o comportamento onde o indivíduo define um tempo para realização de uma atividade com margens de segurança e levará o tempo estimado para realização dessa atividade, ou seja, o tempo se expande para preencher o tempo disponível (PARKINSON, 1955, tradução nossa). Esse comportamento está presente nas estimativas de projetos e mesmo o utilizando projetos continuam atrasando. A síndrome do estudante se caracteriza pelo fenômeno onde as pessoas começam a finalizar uma tarefa somente no último momento possível, afetando desta forma a qualidade da entrega (GOLDRATT, 2002, tradução nossa). O problema não está na estimativa da tarefa, mas sim em dimensionar quais fatores devem ser considerados para se trabalhar na tarefa sem interrupção. A Lei de Parkinson somada com a Síndrome do Estudante contribuem para atrasos no término de projetos, conforme citado por Elder (2006) no artigo *The Five Diseases of Project Management*, sendo essas razões do comportamento humano responsáveis por atrasos nos projetos de software e que devem ser tratadas para evitar que se repitam.

Métricas e visualização de processos de software se fazem necessárias para monitorar a eficiência do processo de desenvolvimento de software, buscando propor aos times informações relevantes para avaliar o processo de desenvolvimento e a implementação dos requisitos, de forma concreta, quantitativa e

analítica (ALBINO, 2017). Desta forma, é possível projetar uma data mais precisa de entrega do software e com mais qualidade.

O presente trabalho propõe a aplicação de um modelo híbrido de processo de desenvolvimento de software como forma de otimizar e avaliar a performance de uma equipe de desenvolvimento de software em uma empresa da região de Criciúma, SC. Considerando o modelo proposto será analisado o *leadtime* da equipe, o *throughput* e o *cumulative flow diagram* determinando se existem gargalos no processo de estimativa, desenvolvimento e teste. A partir da identificação dos gargalos serão propostas soluções para maximizar o desempenho do time de desenvolvimento e assegurar um ritmo constante de entrega de valores.

1.1 OBJETIVO GERAL

Criar um modelo hibrido de processo de desenvolvimento de software para melhorar a produtividade de uma equipe de desenvolvimento em uma empresa da região de Criciúma, SC.

1.2 OBJETIVO ESPECÍFICO

Os objetivos específicos desta pesquisa são:

a) conhecer os conceitos da engenharia de software por meio do estudo dos modelos de processo de desenvolvimento;
b) criar um modelo hibrido de modelo de processo de desenvolvimento utilizando modelos ágeis e tradicionais;
c) analisar o desempenho de uma equipe de desenvolvimento de software da região de Criciúma, SC.

1.3 JUSTIFICATIVA

Se você não pode medir algo, você não pode entender. Se você não pode entender, você não pode controlar. Se você não pode controlar, você não pode aprimorar (HARRINGTON; MCNELLIS, 2006, tradução nossa). Considerando esse aspecto, a pesquisa pretende utilizar métricas ágeis a fim de proporcionar uma análise de desempenho de uma equipe de desenvolvimento de software. Realizando

a medição da performance do time é possível sugerir mudanças de modo a tornar a entrega do produto mais rápida. Conforme Sutherland (2016), um time que conhece a sua velocidade saberá quanto tempo levará concluir as atividades, ou seja, é possível prever a data de entrega dos requisitos. Nesse contexto, saber a capacidade de produção de um time é essencial para prever um prazo de entrega.

Conhecendo a performance da equipe de desenvolvimento é possível gerenciar os riscos para o projeto, por exemplo, se um novo requisito precisar ser inserido no projeto é possível determinar quanto tempo a equipe levará para implementar a nova funcionalidade e desta forma, garantir se o software ficará pronto dentro da data desejada (ALBINO, 2017).

Conforme Pressman (2011) o processo de software proporciona estabilidade, controle e organização a uma atividade que sem controle pode se tornar desordenada. Desta maneira, se torna necessário a aplicação de modelos para obter resultados de alta qualidade dentro de um prazo estabelecido. Na área da engenharia de software, a pesquisa pretende colaborar com um modelo hibrido de processo de desenvolvimento para garantir a entrega de valores no menor tempo possível de forma mais ágil e com menos falhas.

Outro aspecto que a pesquisa contribuirá é com a qualidade do software. Pois com uma equipe mais performática é possível trabalhar nos erros rapidamente, desta maneira contribuindo com a entrega de um software mais estável para os *stakeholders*[1]. Os erros que não são tratados imediatamente levam mais tempo para serem desenvolvidos, chegando a levar 24 vezes mais tempo para correção (SUTHERLAND, 2016).

Diversos são os trabalhos desenvolvidos utilizando modelos de processos, implementados com as mais diversas metodologias. Dentre esses trabalhos destaca-se o *Scaling Agile Scrum Software Development: Providing Agility and Quality to Platform Development by Reducing Time to Market* (JHA; NARAYAN; VILARDELL, 2016, tradução nossa), onde os autores descrevem o processo de implantação de um modelo hibrido de processo de desenvolvimento de software na Siemens. Sendo que o modelo hibrido foi criado a partir do modelo em cascata e adaptado para o modelo Scrum.

[1] São as pessoas diretamente envolvidas no projeto.

Portanto, mediante o exposto é de relevância computacional desenvolver um trabalho na área de engenharia de software, que ofereça um estudo dos modelos de desenvolvimento de software utilizando metodologias ágeis e tradicionais, criando um modelo hibrido e desta forma, contribuir com a otimização de uma equipe de desenvolvimento.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está divido em cinco capítulos. O capítulo 2 apresenta os modelos de processo de desenvolvimento, modelos tradicionais, modelos ágeis, modelos híbridos, técnicas de gerenciamento de processos e métricas de desenvolvimento de software. O capitulo 3 aborda os trabalhos correlatos. O capítulo 4 descreve o trabalho desenvolvido. Por fim, o capítulo 5 apresenta as conclusões e trabalhos futuros.

2 MODELOS DE PROCESSOS DE DESENVOLVIMENTO

Os modelos de processo de desenvolvimento estabelecem a base completa para o processo de desenvolvimento de um software, por meio de um pequeno número de atividade metodológicas que podem ser aplicadas a projetos de software de qualquer tamanho e complexidade (MAXIM; PRESSMAN, 2016).

Existem os mais variados modelos processos de software, onde pode-se afirmar que possuem em comum quatro atividades fundamentais para a engenharia de software (SOMMERVILLE, 2011), sendo elas:

a) especificação do software: é a fase do processo que visa documentar os requisitos que especifica um sistema, sendo esses requisitos satisfatórios para os *stakeholders*;
b) projeto e implementação do software: estágio responsável pela implementação da especificação do sistema em um software;
c) validação do software: tem como objetivo verificar se o software implementado está de acordo com os requisitos especificados;
d) evolução do software: etapa responsável pela alteração dos requisitos para atender as necessidades do cliente, pode ocorrer durante a fase de desenvolvimento ou de manutenção do software.

O processo de desenvolvimento de software possibilita ao indivíduo escolher o modelo que mais se adequa a sua realidade, sendo esse um processo adaptável onde a intenção é entregar um software estável, dentro do prazo estabelecido e que satisfaça os *stakeholders* (MAXIM; PRESSMAN, 2016).

Conforme Wazlawick (2013) dentre as vantagens de possuir um modelo de processo de desenvolvimento pode-se citar o tempo de treinamento reduzido para novos colaboradores, produtos padronizados, uma vez que existe um processo que garante a uniformidade e qualidade do software, e a melhoria continua do processo de desenvolvimento.

Os modelos de processos foram propostos para trazer ordem na área de desenvolvimento de software (SOMMERVILLE, 2011). Nesse contexto destacam-se os modelos tradicionais e os modelos ágeis, cada um adequando-se melhor a cultura de cada organização.

2.1 MODELOS TRADICIONAIS

Um modelo de processo tradicional caracteriza-se pela execução de forma sequencial das atividades definidas em um modelo padrão como o *Rational Unified Process* (RUP) entre outros. O fluxo de trabalho divido em fases e cobrem atividades como análise, modelagem, desenvolvimento e testes (MAXIM; PRESSMAN, 2016).

2.1.1 Cascata

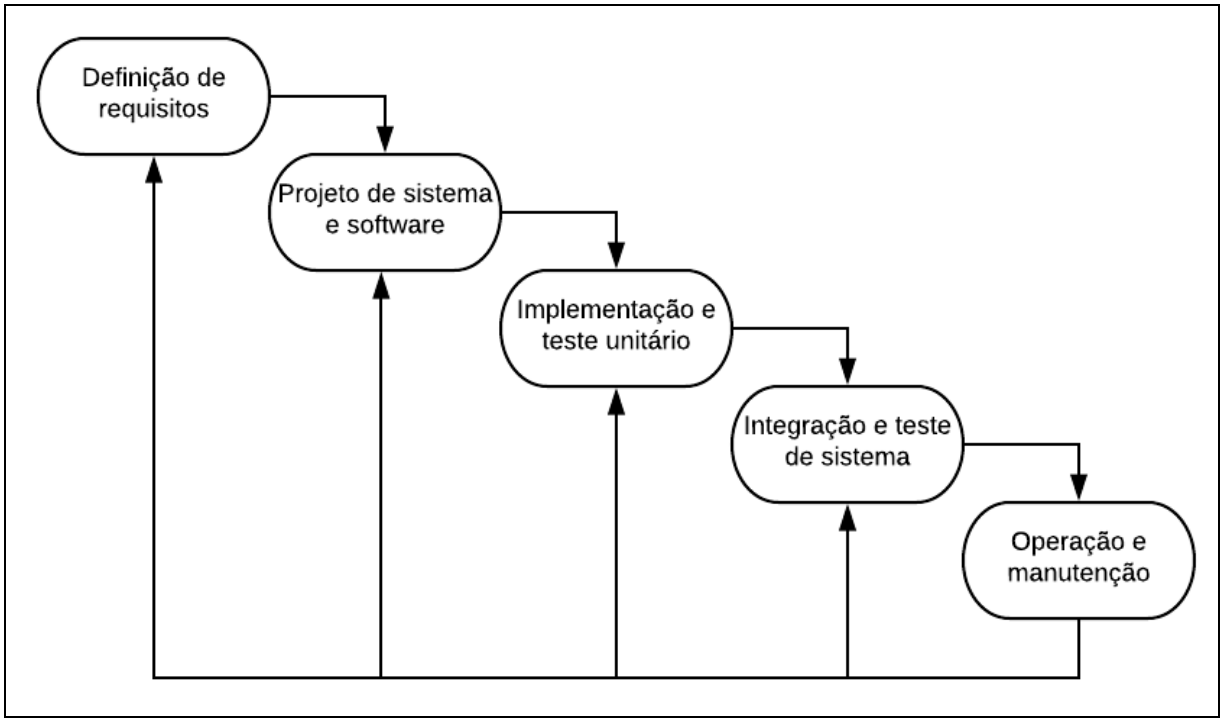
O modelo cascata possui uma abordagem sequencial e sistemática das fases do processo de desenvolvimento de software, onde o software somente avança para a próxima fase após a conclusão da fase anterior. As fases iniciam com a especificação dos requisitos, avança para o planejamento, modelagem, construção e finalmente a disponibilização comercial, após essas fases serem concluídas segue o suporte e manutenção do software (MAXIM; PRESSMAN, 2016).

Segundo Sommerville (2011) as principais fases do modelo cascata são:

a) análise e definição dos requisitos: definido junto dos *stakeholders* os requisitos do sistema e como o software deve funcionar;
b) projeto do sistema de software: nessa etapa é definida a arquitetura do sistema, é especificado no projeto de software como as partes se encaixarão;
c) implementação do teste unitário: fase responsável por gerar os testes unitários do sistema, afim de garantir que a especificação será cumprida;
d) integração e teste de sistema: parte do processo responsável pela implementação do software e teste completo do programa, nessa fase verifica-se se os requisitos estão sendo atendidos;
e) operação e manutenção: essa etapa acontece durante toda a vida após a disponibilização do software, é onde o sistema é utilizado pelo usuário final. Ocorre a manutenção de erros que não foram encontrados durante a fase de integração e teste do sistema. Além disso, nessa fase ocorre o aprimoramento do software.

Como o modelo cascata (figura 1) é linear, com uma fase somente iniciando após a conclusão da anterior, naturalmente conduz-se a estados de bloqueio, pois membros do time podem ficar bloqueados esperando a conclusão de alguma etapa anterior para prosseguir. Desta maneira, é gerado períodos de improdutividade, já que os membros do time não podem prosseguir com o seu trabalho (MAXIM; PRESSMAN, 2016).

Figura 1 – Modelo Cascata



Fonte: Sommerville (2011).

Pode-se destacar como ponto forte do modelo cascata a grande produção de documentação entre as fases do processo de desenvolvimento, contudo como ponto negativo destaca-se a imutabilidade dos requisitos uma vez que entraram em desenvolvimento (WAZLAWICK, 2013).

2.1.2 Espiral

O modelo espiral foi proposto por Boehm em 1988 e tem por característica a redução dos riscos. As fases do processo ocorrem em uma espiral, pode-se dividir a espiral em quatro setores (Sommerville, 2011):

a) definição dos objetivos: nessa fase os objetivos do projeto são definidos e os riscos são identificados;
b) avaliação e redução dos riscos: processo responsável por realizar a análise detalha dos riscos levantados e pelas medidas de contenção dos riscos;
c) desenvolvimento e avaliação: etapa responsável pela implementação do projeto;
d) planejamento: nesta etapa o projeto é revisado e avaliado a sua continuidade, caso opte por continuar o projeto é dada mais uma volta na espiral.

As primeiras iterações do modelo espiral geram modelos ou protótipos que são incrementados na medida que o projeto avança, sendo que nesta fase os maiores riscos para o projeto são identificados e tratados. Nas fases seguintes do modelo os riscos diminuem e são produzidas versões mais completas do software proposto (MAXIM; PRESSMAN, 2016).

O modelo espiral é recomendado para projetos complexos onde os requisitos não são totalmente conhecidos, pois as fases iniciais do projeto identificam e contornam problemas que o projeto pode apresentar. O modelo não é recomendado para projetos onde os requisitos são conhecidos (WAZLAWICK, 2013).

2.2 MODELOS ÁGEIS

Modelos ágeis nasceram da insatisfação com o tempo gasto em especificação de requisitos, projeto, desenvolvimento e teste do modelo tradicional, onde no caso da necessidade de alterar algum requisito é obrigatório passar novamente por todas as etapas, tornando o processo demorado e custoso, podendo ocorrer atrasos nos prazos de entrega acordados. Nesse contexto os modelos ágeis vieram para suprir a necessidade de mudança requisitos e velocidade na entrega, implementado um desenvolvimento incremental do software com entrega valores a cada duas ou três semanas (SOMMERVILLE, 2011).

Abaixo segue os doze princípios fundamentais estabelecidos no manifesto ágil para alcançar a agilidade em projetos de software (BECK et al., 2001):

a) satisfazer o cliente com a entrega de valores dentro dos prazos estipulados;
b) pedidos de mudança nos requisitos devem ser bem aceitos;
c) entregas incrementais do software funcionando em intervalos curtos;
d) equipe comercial e de desenvolvimento devem trabalhar juntas;
e) motive as pessoas e acredite que farão o trabalho corretamente;
f) a transmissão de informações para a equipe é por meio de conversas abertas e presenciais;
g) a principal medida de progresso é o software funcionando;
h) a equipe de desenvolvimento deve ser capaz de manter um ritmo sustentável de trabalho;
i) foco na excelência técnica e em processos que aumentam a agilidade;
j) simplicidade;
k) as melhores arquiteturas surgem de time auto gerenciáveis;
l) autoavaliação do time em período regulares, onde é ajustado o seu desempenho para se tornar mais eficiente.

Entregas periódicas do sistema tornam as alterações nos requisitos do software menos impactantes nos custos ou no tempo do projeto. Pois as alterações na arquitetura são realizadas de forma gradativa durante o desenvolvimento do software. Desta maneira, com *feedbacks* constantes dos *stakeholders* sobre os incrementos de software as adaptações ocorrem de modo gradativo (MAXIM; PRESSMAN, 2016).

2.2.1 Scrum

O Scrum é uma metodologia de gestão de projetos volta para times auto gerenciáveis e adaptável as mudanças inerentes dos projetos de software, onde procura-se reduzir o tempo gasto em especificação e estimativa, priorizando o desenvolvimento da aplicação e a entrega continua de versões do software a cada duas ou quatro semanas (SUTHERLAND, 2016).

Existe três papéis que o Scrum define, os quais são (WAZLAWICK, 2013):

a) *scrum master*: é o facilitador do time, responsável por mediar conflitos e pela aplicação do *framework* Scrum;

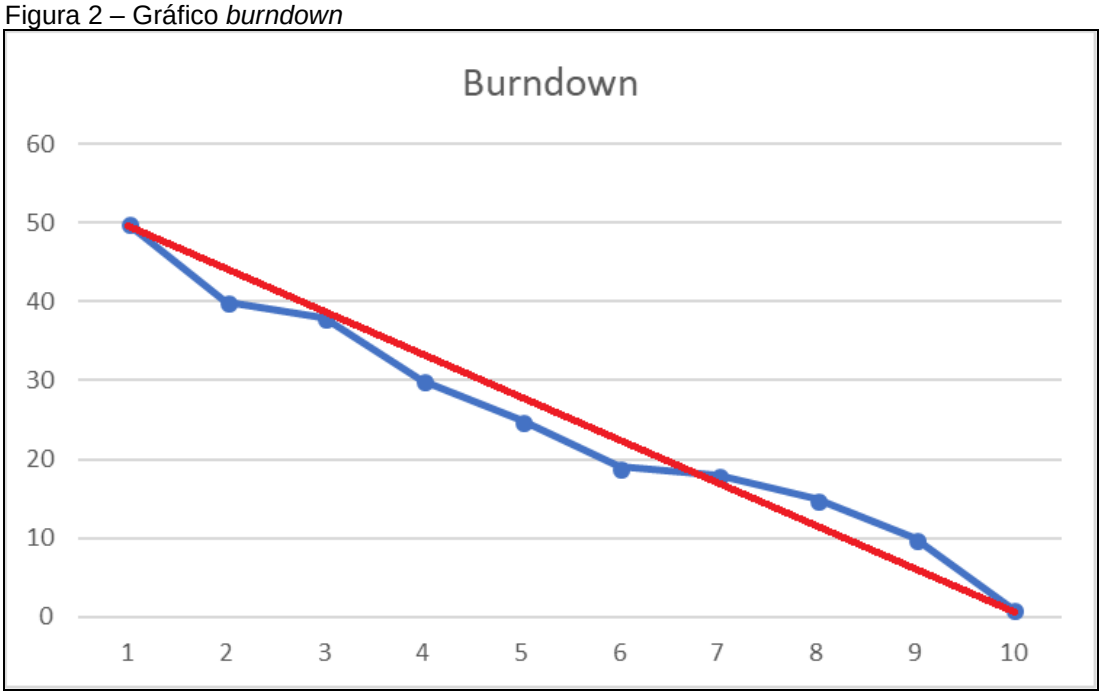
b) *product owner*: responsável pelos requisitos do sistema, conhece e avalia as necessidades do cliente. Defini a prioridade dos requisitos que devem entrar no *sprint*;
c) *scrum team*: é a equipe de desenvolvimento, não possui papéis, todos colaboram para implementar o sistema em conjunto. A recomendação é de times de sete pessoas, variando dois pra cima ou dois pra baixo.

As iterações de planejamento, desenvolvimento e teste acontecem dentro do *sprint* que pode possuir de duas a quatro semanas (SOMMERVILLE, 2011). Dentro do *sprint* existem quatro reuniões que visam garantir o processo, sendo essas (SUTHERLAND, 2016):

a) *sprint planning*: reunião que acontece no final de cada *sprint* e o time se reuni para definir os itens que serão trabalhados no próximo *sprint*, dando origem ao *sprint backlog*;
b) *daily meeting*: reunião que acontece diariamente, deve ter no máximo 15 minutos e os participantes devem ficar de pé. O objetivo é que cada participante compartilhe no que trabalhou no dia anterior, no que irá trabalhar no dia e se existe algum bloquei para execução do seu trabalho;
c) *sprint review*: é uma das mais importantes reuniões do Scrum, acontece no final de cada *sprint* e tem como objetivo mostrar o que foi produzido durante o *sprint* para os *stakeholder* e pessoas interessadas;
d) *sprint retrospective*: assim como o *sprint review* é uma reunião extremamente importante no Scrum, essa reunião é realizada com o intuito do time debater o que está dando errado no processo desenvolvimento e o que pode fazer para melhorar. Não se fala de produto nessa reunião, somente do processo. A reunião deve acontecer no final de cada *sprint* ou a cada dois *sprints*.

Para acompanhamento das atividades no Scrum é utilizado o gráfico *burndown* (figura 2). O gráfico *burndown* é atualizado diariamente e é composto por duas linhas, a primeira indica a quantidade de trabalho pendente, a segunda linha indica a quantidade de trabalho feito ao longo do período (WAZLAWICK, 2013).

Figura 2 – Gráfico *burndown*



Fonte: Wazlawick (2013).

2.2.2 Extreme Programming

O *Extreme Programming* (XP) nasceu nos Estados Unidos no final da década de 1990 (WAZLAWICK, 2013). Alguns dos princípios da metodologia ágil são implementados pelo *Extreme Programming*, conforme segue: (SOMMERVILLE, 2011).

a) desenvolvimento incremental: o projeto possui *releases* frequentes, onde os requisitos são criados como cenários ou simples estórias;
b) envolvimento do cliente: o cliente participa do time de desenvolvimento, ele quem defini os testes de aceitação da aplicação.
c) pessoas: os desenvolvedores trabalham em pares e participam da implementação de todas as partes do sistema, não criando ilhas de conhecimento;
d) mudanças: o processo é estruturado para comportar as mudanças constantes do projeto por meio da refatoração do código, *releases* frequentes e da automatização dos testes;
e) simplicidade: o código é constantemente refatorado para se tornar mais simples e de maior qualidade.

O cliente participa ativamente do processo de desenvolvimento, uma vez que faz parte da equipe de desenvolvimento, sendo ele responsável definição e priorização dos requisitos. Além disso, o cliente também é responsável por criar os testes de aceitação do sistema (MAXIM; PRESSMAN, 2016). Segundo Sommerville (2011) o fato do cliente participar ativamente do processo de desenvolvimento pode se tornar um problema, pois nem sempre ele possui tempo disponível para participar ativamente do projeto.

No processo de desenvolvimento do *Extreme Programming* a implementação dos requisitos acontece em pares, ou seja, duas pessoas trabalham no mesmo computador. Antes de iniciar a codificação são criados os testes unitários e somente após essa etapa é que codificação do requisito é iniciada. Desta maneira, em caso de refatoração o código continuará correto (WAZLAWICK, 2013).

Os testes automatizados e de integração são criados para serem executados diariamente, alertando a equipe imediatamente quando um problema ocorre. Os testes de aceitação são elaborados para assegurar que está sendo entregue o que o cliente necessita (MAXIM; PRESSMAN, 2016).

2.3 MODELOS HÍBRIDOS

Modelos ágeis ou tradicionais de processos de desenvolvimento de software nem sempre se adaptam as necessidades de uma organização, nesse contexto os modelos híbridos surgem para prover o melhor modelo de processo de desenvolvimento para cada empresa.

Modelos híbridos de processo de desenvolvimento caracterizam-se por agregar dois ou mais modelos de processos, sejam eles ágeis ou tradicionais. Modelos híbridos são concebidos para combinar o melhor de dois ou mais modelos de desenvolvimento, gerando assim um modelo único para organização onde é implementado (JHA; NARAYAN; VILARDELL, 2016, tradução nossa).

Conforme estudo realizado por Vijayasarathy e Butler (2016), 45% das empresas utilizam metodologias hibridas de processo. A pesquisa foi realizada entrevistando cerca de 2000 pessoas entre arquitetos, desenvolvedores, testadores, analistas e gerentes de projetos, abrangendo empresas com até 10.000 funcionários.

No modelo proposto nesse trabalho será utilizado um modelo hibrido de processo de desenvolvimento de software. Desta maneira, busca-se combinar as boas práticas dos modelos tradicionais e ágeis, adaptando a fase de levantamento de requisitos do modelo tradicional para trabalhar em conjunto com o processo de desenvolvimento e teste do modelo ágil, afim de criar e implementar um modelo híbrido em uma empresa de desenvolvimento de software da região de Criciúma, SC.

2.4 TÉCNICAS DE GERENCIAMENTO DE PROCESSOS

O gerenciamento de processos é uma parte vital em projetos de software pois por meio dele é atingido os objetivos planejados dentro dos prazos e do orçamento previsto (WAZLAWICK, 2013). Um bom gerenciamento do projeto não garante o seu sucesso, contudo a falha no gerenciamento é um ingrediente para o fracasso (SOMMERVILLE, 2011).

Destacam-se como principais existências para um bom gerenciamento de projeto a entrega do software dentro do prazo, manter os custos dentro do orçamento, implementar um software que atenda os requisitos do cliente e preservar um bom ambiente de trabalho com uma equipe motivada (SOMMERVILLE, 2011).

Líderes de projetos bem-sucedidos adotam uma postura de gerenciar soluções de problemas, procurando entender a adversidade a ser superada, administrando as ideias para a sua solução e comunicando-se com o time para deixá-los cientes por meio de palavras e atitudes que a qualidade não deve ser comprometida para atingir os objetivos (MAXIM; PRESSMAN, 2016).

A motivação de um time é um dos principais fatores que contribuem para um projeto de software bem-sucedido, pois pessoas são envolvidas diretamente no processo de construção do sistema. Trabalhar a motivação de uma equipe é um dos principais desafios dos líderes de projetos (WAZLAWICK, 2013).

Segundo Sommerville (2011) para assegurar a motivação de um time é necessário garantir:

a) necessidades sociais: estabelecer um canal de comunicação entre os membros da equipe para que se conheçam, tornando-se um grupo;
b) autoestima: demonstrar que as opiniões dos colaboradores são ouvidas e valorizadas;

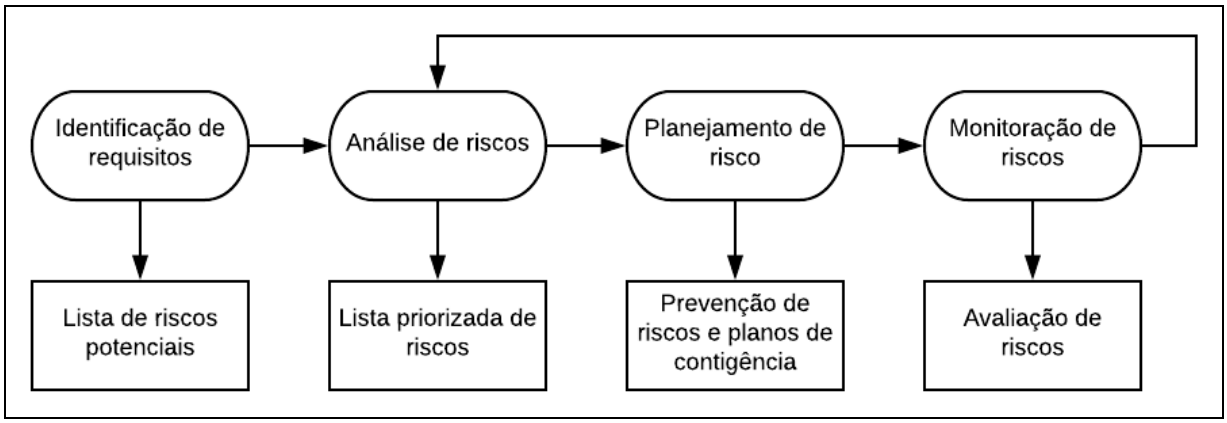
c) autorrealização: fornecer treinamentos as pessoas para que aprimorem suas habilidades, explorando o potencial de cada um durante a execução do projeto.

O gerenciamento de riscos possibilita analisar os impactos das mudanças no projeto do sistema, antecipando problemas e atuando de forma proativa nas alterações inerentes do projeto de software (HIRAMA, 2012). Pode-se dividir os riscos de software em três categorias, conforme listadas a seguir (MAXIM; PRESSMAN, 2016):

a) riscos de projeto: são os riscos que podem impactar no orçamento e cronograma do projeto;
b) riscos técnicos: influenciam na qualidade do software entregue, podem identificar problemas na implementação, interface e manutenção do software;
c) riscos de negócio: o produto entregue não atende as expectativas do mercado ou cliente.

Os riscos para o projeto devem ser documentados em um plano de gerenciamento de riscos (figura 3), onde deve existir uma análise de possíveis problemas e como contorna-los caso necessário. O plano de gerenciamento de riscos deve ser monitorado ao longo de todo projeto, caso seja identificado algum risco em potencial para o projeto deve-se atuar para evitar ou minimizar os seus efeitos (WAZLAWICK, 2013).

Figura 3 – Fluxo do gerenciamento de riscos



Fonte: Sommerville (2011)

Os riscos devem ser acompanhados regularmente pelo gerente do projeto, verificando se no decorrer da implementação do sistema eles sofreram alterações e se é necessário entrar com alguma medida para contorna-los (SOMMERVILLE, 2011).

No modelo hibrido proposto o gerenciamento do projeto ocorre por meio do acompanhamento semanal da implementação dos requisitos definidos no início do projeto. O gerente do projeto tem a responsabilidade de manter o cronograma definido e semanalmente é realizado um levantamento dos possíveis riscos, de modo que caso seja identificado algum bloqueio o problema possa ser contornado ou escalado para os superiores darem vazão ao problema.

2.5 MÉTRICAS DE DESENVOLVIMENTO DE SOFTWARE

No processo de desenvolvimento de software podemos definir uma métrica como sendo algo que pode ser objetivamente medido, sendo essa uma característica do sistema, documentação ou um processo de desenvolvimento. Pode-se citar como exemplo de métrica a quantidade de linhas de código ou a quantidade de pessoas necessárias para desenvolver um requisito (SOMMERVILLE, 2011). Métricas são uma forma de avaliar a qualidade da aplicação com um conjunto de regras claramente definidas, possibilitando a detecção de problemas antes de ocorrerem (MAXIM; PRESSMAN, 2016).

De acordo com Sommerville (2011) os principais estágios para a criação de métricas de software são:

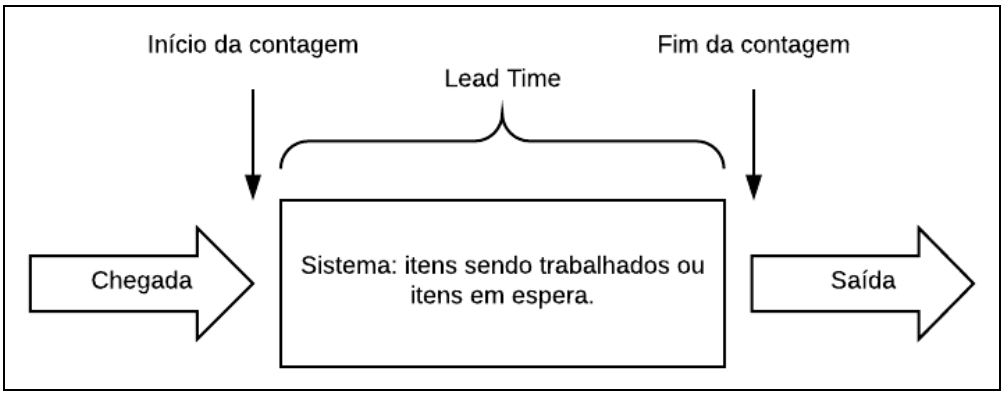
a) definir medições: etapa responsável por apontar o que será medido, quais são as questões que deseja-se obter respostas;
b) selecionar componentes: processo responsável por avaliar quais serão os componentes que serão avaliados, deve-se focar nos componentes que são mais utilizados, pois estes possuem uma representatividade maior para a qualidade do software;
c) medir características de componentes: é coletado os dados dos componentes selecionados, onde os dados selecionados são medidos e os seus valores e métricas são ligados;
d) identificar medições anômalas: rotina responsável por comparar os dados com medições coletadas anteriormente;

e) analisar componentes anômalos: avaliar os componentes que tiveram discrepância nos seus valores afim de identificar se houve perda de qualidade.

Existem diversas formas de medir o desempenho de uma equipe de software, dentre essas destacam-se o *lead time*, o *throughput* e o *cumulative flow diagram* (CFD).

O *lead time* (figura 4) é a quantidade de tempo gasto entre o início e o término de uma atividade definidos no processo de desenvolvimento. Medir o *lead time* é útil para avaliar quanto tempo o time está gastando para desenvolver um requisito, analisar a saúde do processo de desenvolvimento e identificar se o time tem entregue os requisitos dentro de um padrão de tempo (ALBINO, 2017).

Figura 4 – Representação do *lead time*

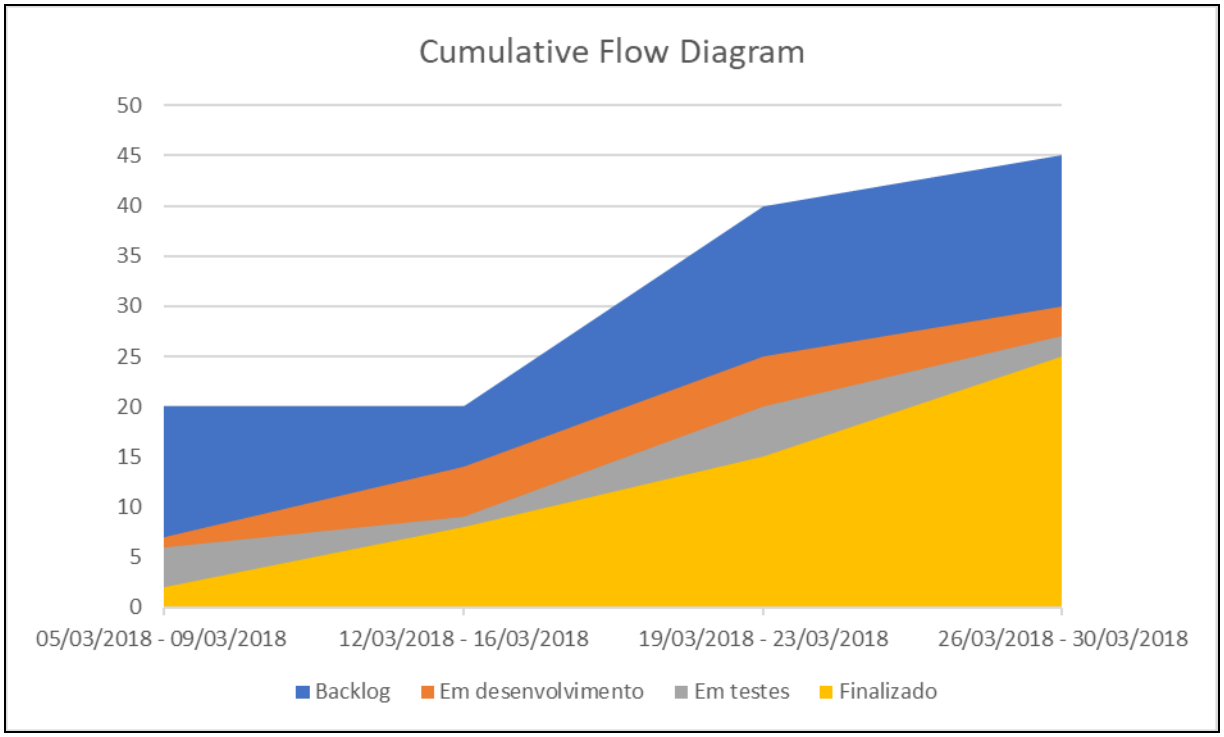


Fonte: Albino (2017)

O *throughput* é a medida de quanto está sendo possível produzir em um determinado tempo, ou seja, é a capacidade de entrega de um time. Não existe uma unidade de medida para *throughput*, isso deve ser definido por cada time, além disso a periodicidade de tempo pode variar (ALBINO, 2017).

O CFD (figura 5) é a representação gráfica do *work items* agrupados por estado, por meio dele é possível entender o estado do trabalho em desenvolvimento e o que pode ser feito para aumentar a velocidade (MICROSOFT, 2018, tradução nossa).

Figura 5 – Representação gráfica do CFD



Fonte: Albino (2017).

O gráfico é composto por um eixo X que indica o período de tempo e o eixo Y indica a quantidade de *work items* agrupados por estado.

2.5.1 MPS.BR

A Melhoria de Processo de Software Brasileiro (MPS.BR) é um modelo de avalição para empresas de software criado pela Softex com apoio do governo federal. O modelo foi concebido devido aos altos custos para implementação de certificações internacionais, oferecendo para as empresas um modelo com um custo inferior e com sete níveis de maturidade (WAZLAWICK, 2013).

Os níveis de maturidade do MPS.BR são (MPS, 2017):

a) nível A: em otimização;

b) nível B: gerenciado quantitativamente;

c) nível C: definido;

d) nível D: largamente definido;

e) nível E: parcialmente definido;

f) nível F: gerenciado;

g) nível G: parcialmente gerenciado.

Cada nível de maturidade possuí como pré-requisito o nível anterior, ou seja, para uma empresa se aplicar ao nível A é necessário que ela tenha conquistado o nível B de maturidade e assim sucessivamente (HIRAMA, 2012).

Para avaliar os níveis de maturidade de uma organização existe o Modelo de Avaliação de Melhoria de Processo de Software (MA-MPS) que é aplicado para determinar o nível de excelência na execução de seus processos de software, serviços e gestão de pessoas. O MA-MPS pode ser aplicado em toda a organização ou somente em um departamento. Como resultado do processo de avaliação é definido o nível de maturidade da instituição. A avaliação seguindo modelo MA-MPS tem validade de três anos a partir da data que a avaliação final foi concluída (MPS, 2017).

2.5.2 CMMI

O *Capability Maturity Model Integration* (CMMI) propõe um modelo para ser implementado em organizações que desejam estabelecer um processo de melhoria. O CMMI é divido em modelo continuo e modelo por estágio (MAXIM; PRESSMAN, 2016). O CMMI continuo é focado em aprimorar processos específicos de acordo com os objetivos da organização. Já o CMMI por estágio é implementado em toda a organização, permitindo que o nível de maturidade seja comparado com de outras empresas (WAZLAWICK, 2013).

O CMMI por estágio é definido por cindo níveis de maturidade, já o CMMI continuo é definido por cinco níveis de capacidade (MAXIM; PRESSMAN, 2016). A tabela 1 apresenta os níveis de capacidade e sua relação com os níveis de maturidade.

Tabela 1 – Níveis de capacidade e maturidade no modelo CMMI

Nível	Capacidade	Maturidade
0	Incompleto	
1	Realizado	Inicial
2	Gerenciado	Gerenciado
3	Definido	Definido
4		Quantitativamente gerenciado
5		Em otimização

Fonte: Wazlawick (2013).

Para avaliação do grau de maturidade ou capacidade de um modelo CMMI foi criado o *Standard CMMI Appraisal Method for Process Improvement* (SCAMPI), sendo este um método de avaliação eficiente e integrado. O SCAMPI possibilita o diagnóstico da qualidade de um processo de uma organização de qualquer porte que adote o modelo CMMI (AGUENA et al., 2006).

3 TRABALHOS CORRELATOS

Diversos são os trabalhos publicados na área de engenharia de software que propõe modelos híbridos de processo de desenvolvimento de software, os tópicos seguintes irão abordar alguns exemplos de trabalhos publicados na área.

3.1 HYBRID MODEL FOR SOFTWARE DEVELOPMENT PROCESSES

A pesquisa propõe um modelo hibrido de processo de desenvolvimento baseado nos mais comuns modelos existentes, sendo eles o modelo Cascata, o modelo Iterativo e Incremental, o modelo Espiral, o modelo V e o *Extreme Programming*. O modelo hibrido apresentado pode ser aplicado em projetos pequenos, médios e grandes (GOVARDHAN; MUNASSAR, 2010, tradução nossa).

O modelo possui nove fases, conforme segue (GOVARDHAN; MUNASSAR, 2010, tradução nossa):

a) *planning*: fase responsável por definir os recursos e as datas do cronograma;
b) *requirements*: nessa etapa os requisitos são levantados. O engenheiro de software deve entender o comportamento, funções, a performance deve ser considerada. No final dessa fase deve-se possuir os requisitos do sistema e software para o cliente revisar;
c) *design*: processo responsável por projetar o software;
d) *implementation*: etapa de implementação do software;
e) *integration development*: fase onde diferentes partes do sistema que foram implementadas separadamente juntam-se;
f) *deployment*: enviar o software desenvolvido para o cliente utilizar, sendo que o cliente indicará os problemas baseados no seu uso;
g) *testing*: o processo de teste está presente desde a primeira etapa até a etapa de *integration* d*evelopment*;
h) *maintance*: etapa que acontece após o cliente receber o software desenvolvido, onde é feito melhorias na aplicação e correção de bugs;
i) *risk* a*nalysis*: abrange a fase de p*lanning* até a fase de m*aintance* onde é calculado os riscos para o projeto e sugerido atividades para redução dos riscos.

O modelo híbrido proposto caracteriza-se por uma análise constante dos riscos para o projeto, sendo estes avaliados em cada etapa do projeto. Existe um forte encadeamento entre as fases, assim como o modelo em cascata. Além disso, a fase de testes está presente em quase todas as etapas e em cada uma delas é executada uma atividade, como planejamento dos testes, testes de regressão e integração (GOVARDHAN; MUNASSAR, 2010, tradução nossa).

3.2 SCALING AGILE SCRUM SOFTWARE DEVELOPMENT: PROVIDING AGILITY AND QUALITY TO PLATFORM DEVELOPMENT BY REDUCING TIME TO MARKET

A distância entre os times, padrões organizacionais, culturas e políticas diferentes entre diversos times contribuem para a diminuição da coesão entre as equipes. O modelo *Product Evolution Process* (PEP) foi criado para dar suporte global aos produtos desenvolvidos para a Siemens. O modelo foi concebido para trabalhar com times distribuídos em diferentes fusos horários. O PEP é uma junção do modelo cascata com o Scrum, onde as fases *Requirement Engineering*, *System Testing*, *Field Testing* e *Release to Market* segue o modelo cascata. As fases de *Design*, *Development* e *Product Testing* são baseadas nas práticas do Scrum (JHA; NARAYAN; VILARDELL, 2016, tradução nossa).

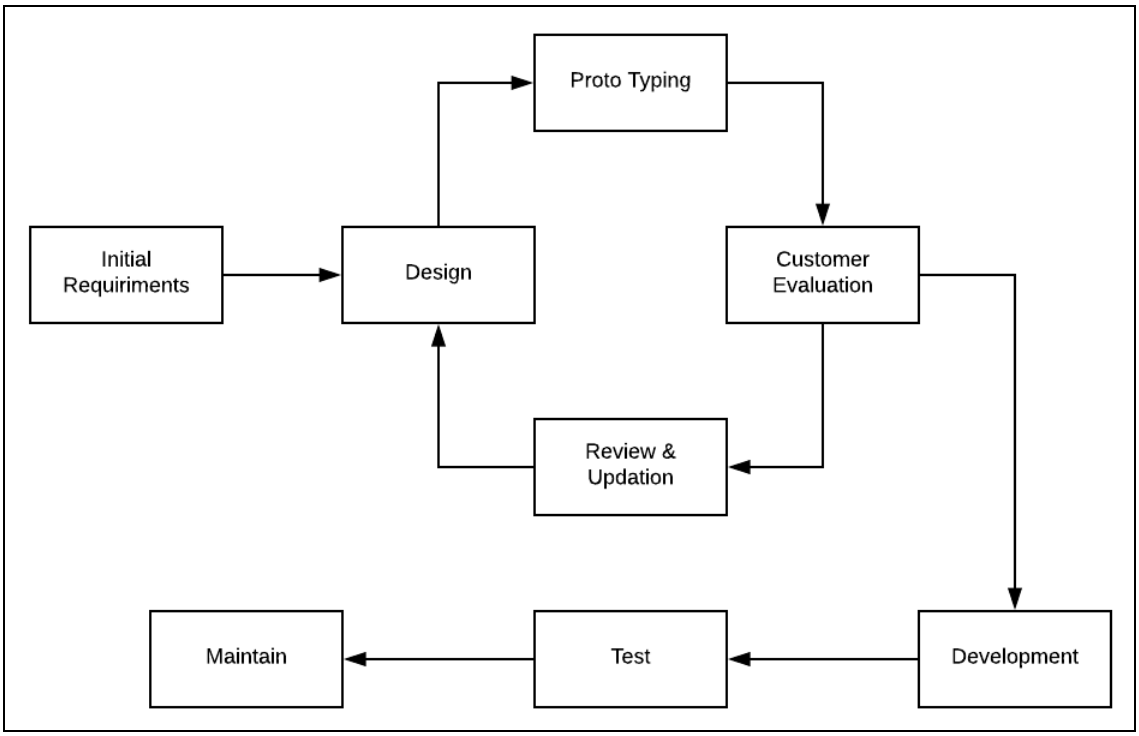
O modelo prega enxugar o *backlog* para direcionar melhor as metas do time. Todos os times devem trabalhar com metas comuns, isso engloba metas para orçamento, performance e qualidade. O PEP incentiva o conceito Lean, onde sempre que um problema ocorre é priorizado a sua resolução, mesmo que isso signifique entregar menos requisitos no final do *sprint* (JHA; NARAYAN; VILARDELL, 2016, tradução nossa).

3.3 HYBRID MODEL FOR SOFTWARE DEVELOPMENT

O modelo hibrido proposto (figura 6) utiliza o modelo Scrum alterando-o para incluir uma etapa de prototipação antes da etapa de desenvolvimento. O modelo é recomendado quando não se conhece todos os requisitos do sistema. O processo inicia-se pela identificação dos requisitos, onde após essa etapa é criado o *backlog*. Com os requisitos no *backlog* é criado um protótipo do sistema e enviado

para o cliente utilizar. A medida que o cliente utiliza e solicita alterações no protótipo o backlog é alterado. Ao término dessa etapa é gerado o *sprint backlog* (ABRAHAM; MATHAI; VENUGOPAL, 2016, tradução nossa).

Figura 6 – Modelo híbrido



Fonte: Abraham, Mathai e Venugopal (2016).

Durante a iteração o software é desenvolvido e testado, sendo que no final do *sprint* uma versão do sistema é lançada para o cliente. Ao término do *sprint* o *backlog* é atualizado e um novo protótipo é criado, dando origem a um novo *sprint* (ABRAHAM; MATHAI; VENUGOPAL, 2016, tradução nossa).

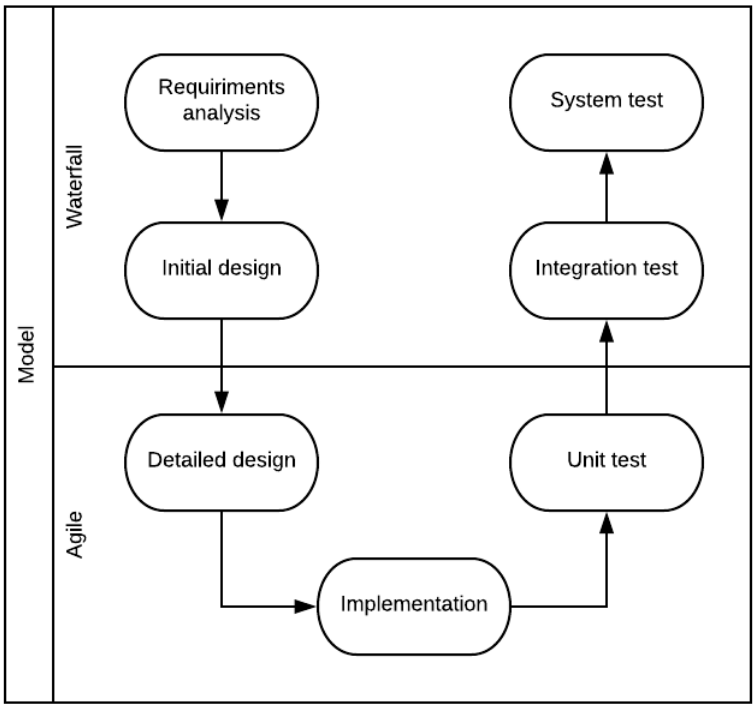
3.4 A HYBRID MODEL FOR IT PROJECT WITH SCRUM

Atualmente no mundo existem mais softwares necessitando de manutenção do que softwares novos sendo desenvolvidos. Da necessidade da manutenção de softwares foi proposto um modelo hibrido de desenvolvimento baseado no modelo cascata com Scrum, originando um modelo em V (HAN; HAYATA, 2011, tradução nossa).

O modelo hibrido (figura 7) caracteriza-se por utilizar o modelo cascata nas fases inicias de desenvolvimento e nas fases finais. Conforme imagem abaixo o

modelo cascata é aplicado nas fases de análise de requisitos, inicial designer, testes de sistema e testes de integração (HAN; HAYATA, 2011, tradução nossa).

Figura 7 – Modelo híbrido V



Fonte: Han e Hayata (2011)

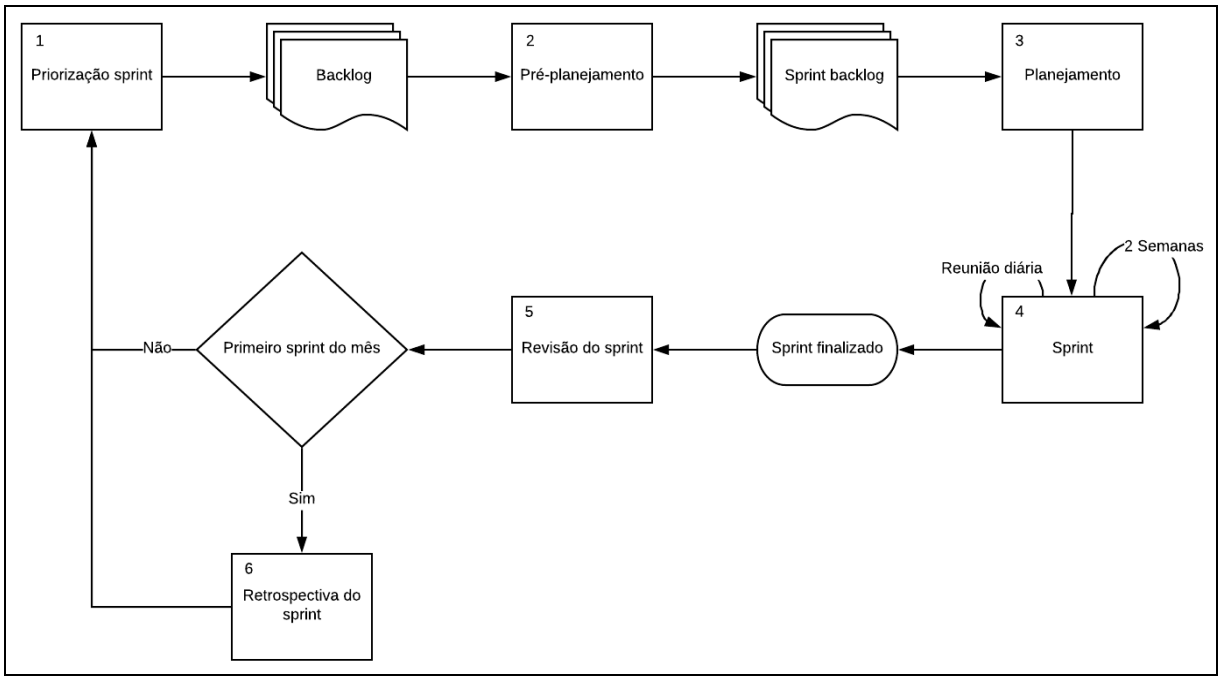
Nas fases de detalhamento do designer, implementação e testes unitário é utilizado o modelo Scrum. O Scrum foi escolhido para ser aplicado nessa fase do modelo para aplicar a metodologia *Test Driven Development* (TDD), ou seja, o teste unitário é desenvolvido antes da implementação da aplicação (HAN; HAYATA, 2011, tradução nossa).

Foi aplicado o modelo hibrido e o modelo cascata para estimar o tempo para implementação de um software. Com o modelo cascata o projeto seria concluído no prazo de 90 dias. Contudo aplicando o modelo hibrido o projeto seria concluído em 64 dias, apresentando uma redução de 23% no tempo para conclusão o projeto (HAN; HAYATA, 2011, tradução nossa).

4 TRABALHO DESENVOLVIDO

Devido a problemas de produtividade existentes no modelo em utilização por uma empresa da região de Criciúma - SC, onde após análise foi observado oportunidades de melhoria, foi proposto a criação de um modelo híbrido de desenvolvimento de software (figura 8) utilizando modelos ágeis e modelos tradicionais para obter a maior eficiência no processo de desenvolvimento.

Figura 8 – Modelo híbrido proposto



Fonte: Do autor.

O modelo híbrido é definido pelas seguintes etapas:

a) 1 – priorização do *sprint*: essa reunião tem como objetivo definir o que será trabalhado nos próximos 45 dias dando previsibilidade ao processo de definição, desenvolvimento e teste. Participa dessa reunião os responsáveis pelos setores de desenvolvimento, gerência de produto e teste. Ao término dessa reunião a gerência de produto deve colocar no *sprint* os itens que devem ser trabalhados para que a equipe de desenvolvimento e teste possam realizar a estimativa;

b) 2 – pré-planejamento: após a definição do que deverá ser trabalhado na reunião de priorização do *sprint* acontece a reunião de planejamento, onde se tem como objetivo avaliar o que entrará no

sprint. A equipe de desenvolvimento e de teste devem chegar nessa reunião com as estimativas finalizadas. Participa dessa reunião o analista de sistemas, o analista de negócio e o testador do time;

c) 3 – planejamento: ao término da reunião de planejamento a equipe de desenvolvimento se reuni e distribui o trabalho entre os desenvolvedores do time. Nessa etapa os desenvolvedores tiram dúvidas sobre as demandas atribuídas a eles. Além disso, caso seja necessário realizar alguma alteração na estimativa realizada é feito nessa reunião;
d) 4 – *sprint*: o *sprint* tem duração de duas semanas iniciando em uma quarta feira. O tempo dos desenvolvedores é alocado 70% para desenvolvimento dos itens previstos no *sprint*, os outros 30% são destinados para a correção dos erros críticos e graves que são corrigidos imediatamente que detectados;
e) reunião diária: é realizada diariamente no início do dia e tem a duração de 15 minutos no máximo, todos os integrantes do time participam. O principal objetivo da reunião é detectar bloqueios que afetam a implementação dos requisitos. Caso seja identificado algum bloqueio durante a reunião imediatamente após o término é realizada uma outra reunião com as pessoas necessárias para desbloquear o item indicado na reunião diária. Desta maneira, logo no início do dia os bloqueios são resolvidos e os integrantes do time podem continuar a implementação dos requisitos;
f) 5 – revisão do *sprint*: ao término do *sprint* é realizada uma reunião para apresentar aos *stakeholder* e pessoas interessadas o resultado do que foi implementado durante o *sprint*. Caso algum requisito não esteja de acordo com o esperado é identificado e corrigido em um *sprint* futuro;
g) 6 – retrospectiva do *sprint*: essa reunião ocorre mensalmente e é realizada no último dia do primeiro *sprint* do mês, todos os membros do time participam e o objetivo é discutir o processo de desenvolvimento, não se fala do produto nessa reunião.

Na tabela 2 é definida a composição do time de desenvolvimento:

Tabela 2 – Composição do time de desenvolvimento no modelo híbrido

Quantidade	Papel
1	Analista de Negócio
1	Testador
1	Analista de Sistemas
3	Programador

Fonte: Do autor.

As responsabilidades dos membros que compõe o time de desenvolvimento são:

a) analista de negócio: responsável por criar as definições dos requisitos que serão implementados pelo time;
b) analista de sistemas: 50% do seu tempo é destinado a realização da estimativa dos requisitos para os programadores e os outros 50% é destinado ao desenvolvimento e auxílio aos desenvolvedores quando solicitado;
c) testador: elabora os casos de teste e testa as demandas do *sprint*;
d) programador: implementa as demandas do *sprint*.

Considerando dois sprints (figura 9) com duas semanas de duração cada, a ordem cronológica das etapas de desenvolvimento do S*print* 2 no modelo híbrido ocorrem da seguinte maneira:

a) pré-planejamento: ocorre no dia 12, nesse dia todas as demandas do *Sprint* 2 devem ser colocadas para serem desenvolvidas formando *sprint backlog* do *Sprint* 2. É alocado 70% do tempo dos desenvolvedores no *sprint*, os outros 30% é deixado livre para correção de erros;
b) planejamento: ocorre no dia 12, após a reunião de pré-planejamento, o analista de sistemas se reuni com os desenvolvedores para tirar dúvidas e avaliar o que será desenvolvido no *sprint*, o analista de sistemas pode utilizar o dia todo para essa atividade;
c) revisão do *sprint*: ocorre no dia 13, é apresentado para os *stakeholders* o que foi desenvolvido nas últimas duas semanas;
d) priorização do *sprint*: ocorre no dia 16, primeiro dia do *Sprint* 2 e tem como objetivo planejar os itens que serão trabalhados no próximo *sprint*;

e) *sprint*: o *Sprint* 2 inicia no dia 16 e vai até o dia 27, contudo como o planejamento do *sprint* foi realizado no dia 12 pode-se antecipar o início do desenvolvimento do *Sprint* 2. Isso pode ocorrer para os desenvolvedores que não estiverem trabalhando na correção de erros, pois os últimos dois dias do *sprint* é reservado para correção de *bugs*. Do dia 16 ao dia 18 o analista de sistemas trabalhará na implementação de demandas. Do dia 19 ao dia 25 o analista de sistemas se dedicará a estimativa das demandas do próximo *sprint*, os itens a serem trabalhados serão colocados no *sprint* pelo analista de negócio que seguirá o que foi definido na reunião de priorização do *sprint*.

Figura 9 – Ordem cronológica das etapas de desenvolvimento

Domingo	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Sprint 1 Sprint 2

Fonte: Do autor.

O modelo híbrido foi concebido para desburocratizar o processo de desenvolvimento, procurando atingir maior eficiência no tempo trabalhado durante os *sprints* e minimizando dias sem desenvolvimento e teste. Além disso, o modelo busca dar visibilidade aos gestores por meio da previsibilidade do que irá ser trabalhado nos próximos 45 dias, facilitando a gestão do projeto e controle das entregas.

4.1 METODOLOGIA

A metodologia desta pesquisa é composta por:

a) realizar levantamento bibliográfico;

b) modelar conhecimento na área de processos de desenvolvimento;

c) definir métricas de desempenho;
d) criar modelo hibrido de processo de desenvolvimento;
e) aplicar modelo hibrido de processo de desenvolvimento;
f) realizar a análise de métricas de desempenho.

O fluxo de desenvolvimento do trabalho foi delimitado pela definição do modelo proposto, implementação na equipe do projeto, validação do modelo com a equipe do projeto, coleta e análise dos dados. Nos tópicos seguintes é exposto como cada fase do trabalho foi implementada.

4.1.1 Implementação do modelo proposto

A definição do modelo híbrido iniciou pela identificação dos gargalos no processo de desenvolvimento existente, para isso foram coletados dados de três *sprints* avaliando o *lead time*, *throughput* e CFD da equipe. Após uma análise inicial dos dados coletados foram constatados os problemas abaixo:

a) por meio do CFD foi possível verificar que existiam definições prontas para montar as *sprints*, contudo ao avaliar as demandas foi observado que haviam muitas demandas incompletas, faltando definições;
b) o *lead time* mostrou que o tempo que uma demanda ficava em desenvolvimento era alto;
c) a quantidade de demandas concluídas por semana foi verificada por meio do *throughput* que mostrou uma queda acentuada nas semanas que havia o planejamento do *sprint* seguinte.

Dadas as informações coletadas foram propostas mudanças no modelo visando:

a) atingir maior eficiência no tempo trabalhado durante os *sprints*, minimizando dias sem desenvolvimento e teste;
b) garantir que as definições das demandas fossem finalizadas com 45 dias de antecedência.

Além disso, o modelo proposto procurou desburocratizar o processo de desenvolvimento, definição e teste, aproximando fisicamente as equipes que se encontravam em andares separados.

A equipe de desenvolvimento utilizava o modelo Scrum para desenvolvimento e com base nesse modelo foram adaptados processos para garantir os resultados desejados. Desta forma, foi proposto a criação das etapas no modelo híbrido:

a) priorização do *sprint*;
b) pré-planejamento;
c) planejamento;
d) reunião diária;
e) revisão do *sprint*;
f) retrospectiva do *sprint*.

Para atingir um maior aproveitamento do tempo dos desenvolvedores durante a *sprint* foi instituído que as estimativas seriam feitas por um analista de sistemas. Além disso, para assegurar que a gerência de produtos estava trabalhando nas definições das demandas que seriam trabalhadas nos próximos *sprints* foi criada a reunião de priorização do *sprint*, desta maneira, as equipes de gerência de produtos, teste e desenvolvimento alinhavam-se com o que deveria ser dado foco no momento, dando previsibilidade ao processo de desenvolvimento.

4.1.2 Validação do modelo com a equipe do projeto

No dia 06 de março de 2018 foi realizada uma reunião com toda a equipe de desenvolvimento para apresentar o modelo híbrido proposto. Nessa reunião foi explicado como funcionaria os *sprints*, a divisão da equipe e as responsabilidades de cada papel.

A primeira ação foi colocar fisicamente os setores de desenvolvimento, teste e gerência de produtos próximos, pois estavam em andares separados. Isso facilitou a comunicação entre os membros do time, diminuindo a necessidade de reuniões para sanar dúvidas ou problemas durante as fases de definição, desenvolvimento e teste.

Nas semanas seguintes o modelo foi aplicado nos *sprints* seguindo o modelo híbrido proposto, sendo que houve um acompanhamento para garantir a correta aplicação das ações propostas.

4.1.3 Planejamento do projeto

Na semana seguinte a apresentação do modelo híbrido foi realizada a montagem do *sprint* 18.04.05 seguindo o novo modelo, onde inicialmente os analistas de negócios incluíram as demandas para estimativa e os analistas de sistemas e testador realizaram as estimativas.

Na tabela 3 é definida a composição do time de desenvolvimento no primeiro *sprint* do modelo híbrido:

Tabela 3 – Composição do time de desenvolvimento no *sprint* 18.04.05

Quantidade	Papel
2	Analista de Negócio
1	Testador
2	Analista de Sistemas
5	Programador

Fonte: Do autor.

A principal dificuldade no *sprint* 18.04.05 foi com a reunião de pré-planejamento, pois não havia ficado claro o que deveria entrar no *sprint*. Isso aconteceu devido a reunião de priorização do *sprint* ser realizada com os responsáveis das equipes de desenvolvimento, gerência de produto e teste, sendo que não houve o repasse de forma clara do que deveria ser trabalhado para os analistas de negócios, analista de sistemas e testador. No *sprint* 18.04.19 isso foi ajustado colocando em uma ata de reunião o que seria trabalhado nos próximos três *sprints* para que fosse feito a montagem dos *sprints* conforme previamente planejado.

O *sprint* 18.04.19 ocorreu sem problemas e a composição do time de desenvolvimento se manteve a mesma do *sprint* 18.04.05. Após esse *sprint* foi realizada a reunião de retrospectiva do *sprint*, onde todo o time de desenvolvimento participou. Nessa reunião foi identificada a necessidade de ordenar a sequência que as demandas eram desenvolvidas dentro do *sprint* para disponibilizar o mais rápido possível as demandas prontas para o setor de teste realizar os testes. Desta maneira, foi acordado que na reunião de planejamento do *sprint* 18.05.17 o desenvolvimento iria organizar as demandas para entregar itens para o teste o mais breve possível.

O *sprint* 18.05.03 ocorreu sem problemas e a composição do time de desenvolvimento se manteve a mesma do *sprint* 18.04.05. No *sprint* 18.05.17 foi adicionado um novo programador a equipe de desenvolvimento, nesse *sprint* ocorreu o sequenciamento por parte do desenvolvimento das demandas implementadas, mostrando-se eficiente pois houve uma diminuição no tempo para disponibilizar os itens para o teste.

4.1.4 Coleta dos dados

Os dados foram coletados por meio do Team Foundation Server (TFS) da Microsoft, foi pego dados de três *sprints* para avaliação do modelo Scrum, sendo os *sprints* 18.01.11, 18.01.25 e 18.02.08. Foi utilizada a base histórica desses *sprints* para verificar o CFD, *lead time* e *throughput* do time de desenvolvimento.

Os *sprints* 18.04.19, 18.05.03 e 18.05.17 foram desenvolvidos seguindo o modelo híbrido proposto. Os dados foram coletados após o término do *sprint* 18.05.17 para avaliação do CFD, *lead time* e *throughput*, pois a preocupação inicial foi com a aplicação do modelo proposto, após rodar o modelo por um mês e meio os dados foram coletados para avaliação do modelo híbrido.

4.1.5 Análise dos dados

Para o cálculo do *throughput* por *sprint* foi avaliado a quantidade de demandas concluídas pela equipe de teste ao longo do *sprint.* Já para o cálculo do *throughput* por semana foi agrupado as demandas pelo número da semana do ano e realizada a contagem de quantos itens foram concluídos pela equipe de testes.

O cálculo do *lead time* por *sprint* foi realizado levantando em consideração a média de tempo que uma demanda no *sprint* levava para ser colocada em desenvolvimento pelo desenvolvedor até o testador fechar a demanda. Sendo que foi possível mostrar o tempo médio que um item ficava com a equipe de desenvolvimento e teste.

A quantidade de erros por *sprint* foi calculada somando o total de *bugs* resolvidos por *sprint*, sendo que os *bugs* identificados eram imediatamente

corrigidos. A quantidade de erro por hora trabalha foi calculada dividindo o total de horas trabalhadas no *sprint* pela quantidade de erros do *sprint*.

O tempo trabalhado e o tempo estimado no *sprint* foi obtido por meio do TFS, pegando as informações geradas pelos analistas e programadores foi realizado um somatório do tempo trabalhado e estimado no *sprint*.

O CFD foi calculado pegando dados dos *sprints* 18.01.25 e 18.05.17, foi pego a quantidade de demandas criadas, em desenvolvimento, em teste e finalizadas por dia nos *sprints*.

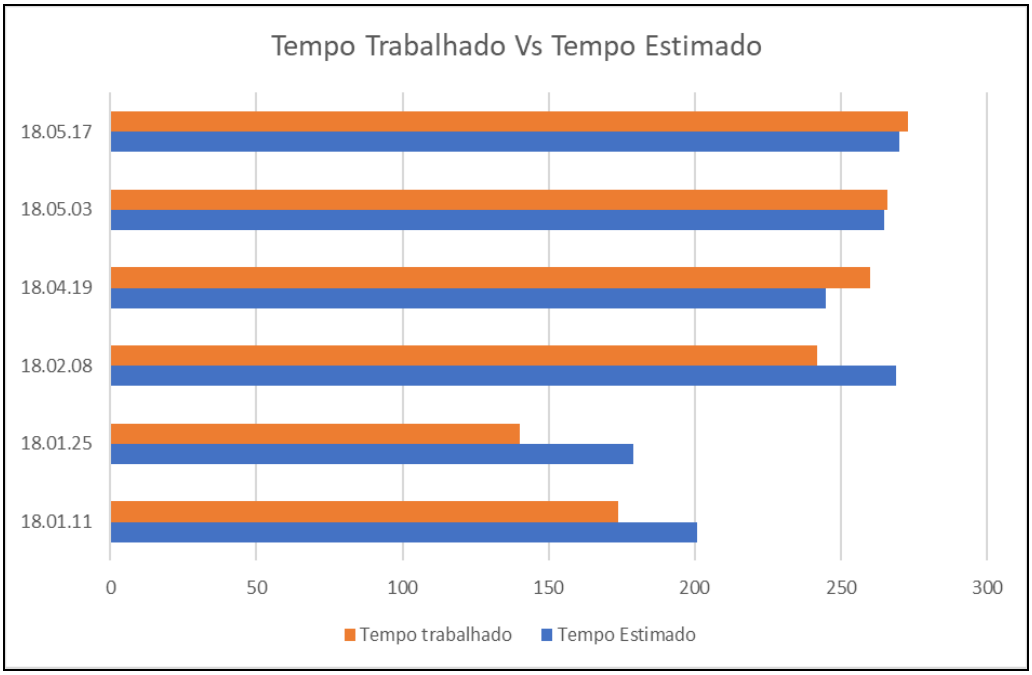
4.2 APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS

Para avaliação dos resultados foram coletados dados durante 6 *sprints*, sendo que os *sprints* 18.01.11, 18.01.25 e 18.02.08 foram desenvolvidos seguindo a metodologia do Scrum. Os *sprints* 18.04.19, 18.05.03 e 18.05.17 foram desenvolvidos seguindo o modelo híbrido proposto.

Na figura 10 é demonstrado o comparativo do tempo trabalhado versus o tempo estimado. Nos *sprints* 18.01.11, 18.01.25 e 18.02.08 as estimativas foram realizadas pelos próprios programadores. No *sprint* 18.01.11 o tempo trabalhado ficou 14% menor que o tempo estimado, no *sprint* 18.01.25 o tempo trabalhado ficou 22% menor que o tempo estimado e no *sprint* 18.02.08 o tempo trabalhado ficou 11% menor que o tempo estimado. Nos *sprints* 18.04.19, 18.05.03 e 18.05.17 as estimativas foram realizadas por um analista de sistemas, onde o tempo estimado ficou próximo ao tempo trabalhado pelos programadores, mostrando uma variação de apenas 6% para baixo.

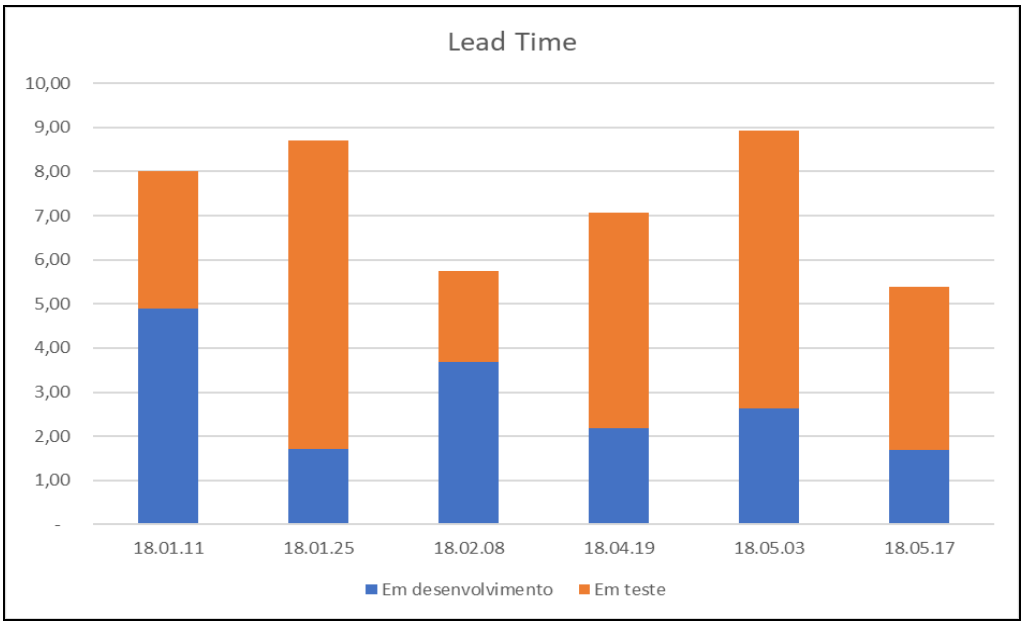
O *lead time* (figura 11) demonstrou uma variação de 5% comparando o modelo ágil com o modelo híbrido. Comparando os *sprints* 18.01.11, 18.01.25 e 18.02.08 com os *sprints* 18.04.19, 18.05.03 e 18.05.17, houve uma redução 38% do tempo que um item fica em desenvolvimento, ou seja, o desenvolvimento começou a entregar as demandas mais rapidamente para o teste. Contudo houve um aumento de 22% no tempo que o teste gasta para concluir uma demanda, isso aconteceu devido a problemas no ambiente de teste e sobrecarga do testador.

Figura 10 – Gráfico de tempo trabalhado versus tempo estimado



Fonte: Do autor.

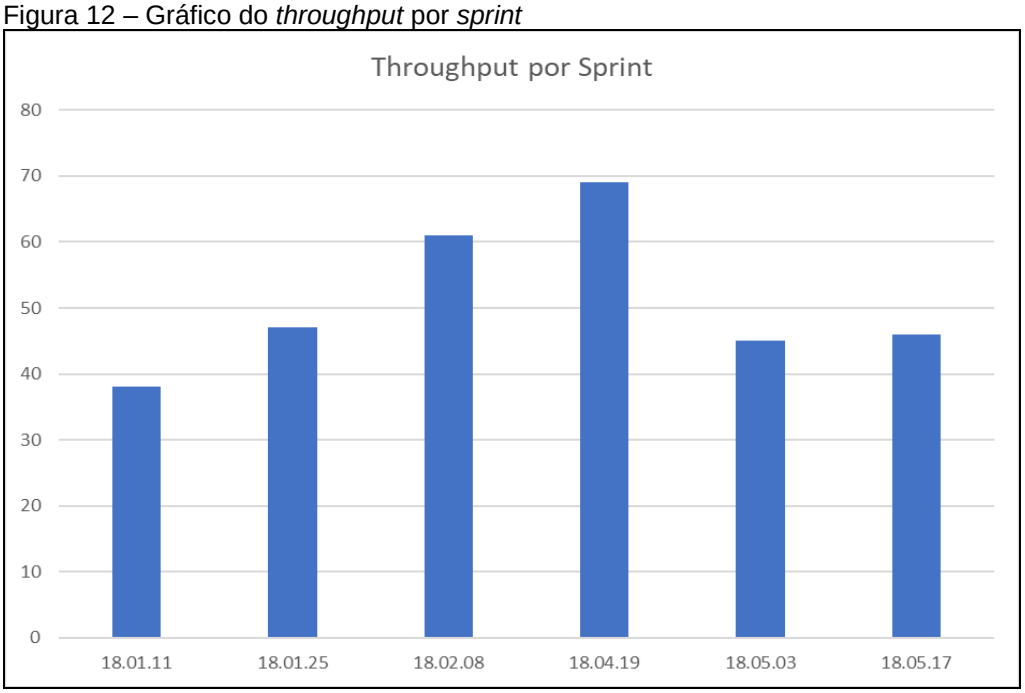
Figura 11 – Gráfico do *lead time*



Fonte: Do autor.

Com a adoção do modelo híbrido houve um aumento de 9% no *throughput* (figura 12) se comparado ao modelo ágil.

Figura 12 – Gráfico do *throughput* por *sprint*



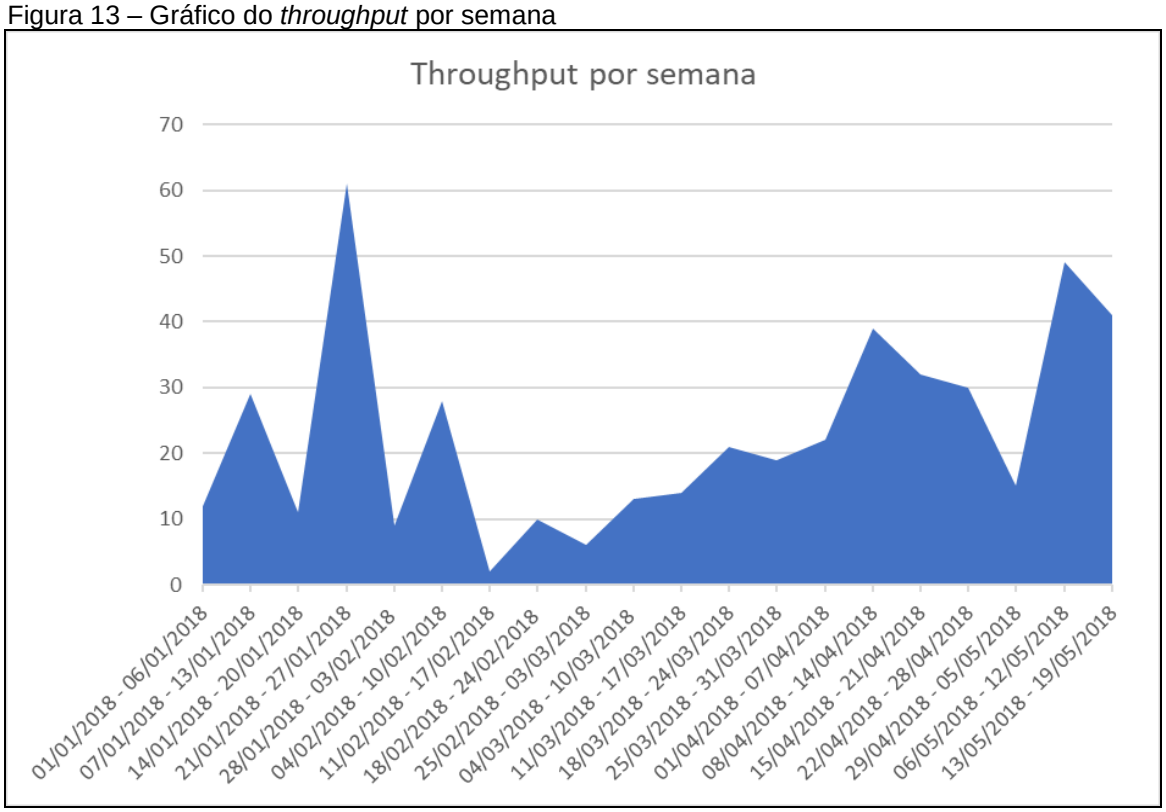
Fonte: Do autor.

O modelo híbrido foi adotado no dia 07/03/2018, comparando as semanas anteriores a adoção do modelo (figura 13) nota-se pico nas entregas das demandas, sendo que na semana seguinte existe uma queda. Essa queda existia devido aos programadores serem alocados na estimativa dos requisitos quando se finalizava um *sprint*. Durante dois dias não havia desenvolvimento, o time era alocado na preparação do próximo *sprint*, o que gerava a queda observada no gráfico.

Com a adoção do modelo híbrido as entregas ficaram constantes, não existindo mais quedas no *throughput* no final dos *sprint*, isso se deve as estimativas serem realizadas por um analista de sistemas. Desta forma, os programadores passam mais tempo implementando os requisitos, não precisando mais parar por dois dias para realizar estimativas.

Na figura 13 a semana do dia 29/04/2018 à 05/05/2018 apresenta uma queda no *throughput*, essa queda aconteceu devido a essa semana possuir um feriado, reduzindo a quantidade de dias trabalhados pela equipe de desenvolvimento.

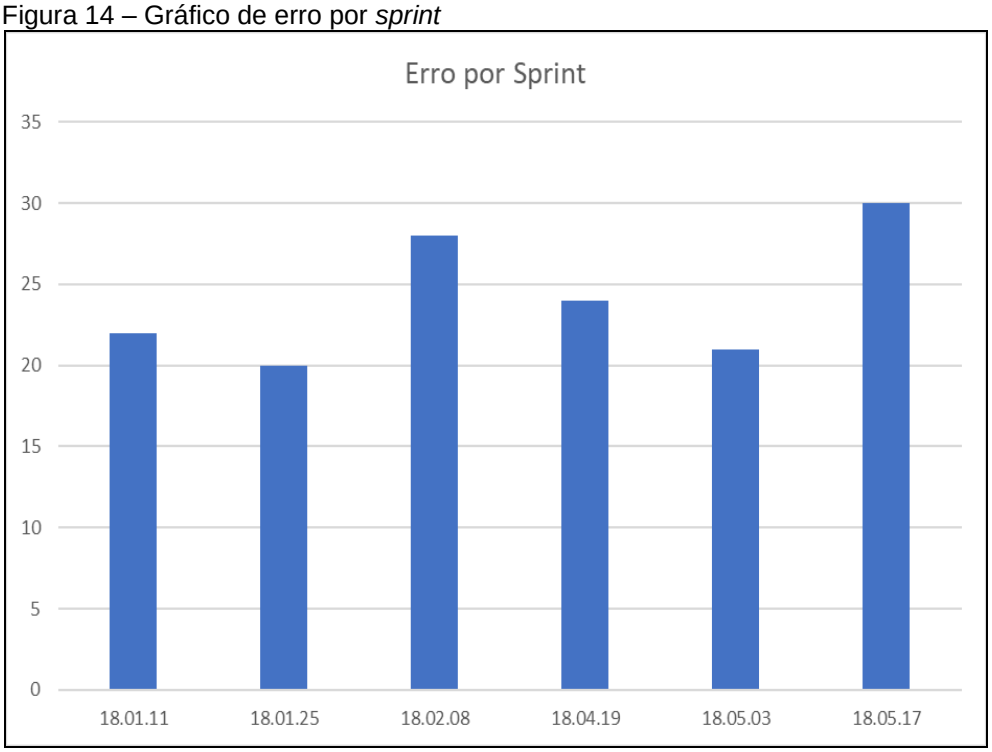
Figura 13 – Gráfico do *throughput* por semana



Fonte: Do autor.

Avaliando a quantidade de erro (figura 14) gerado por *sprint* a metodologia ágil e a metodologia híbrida mostraram-se iguais.

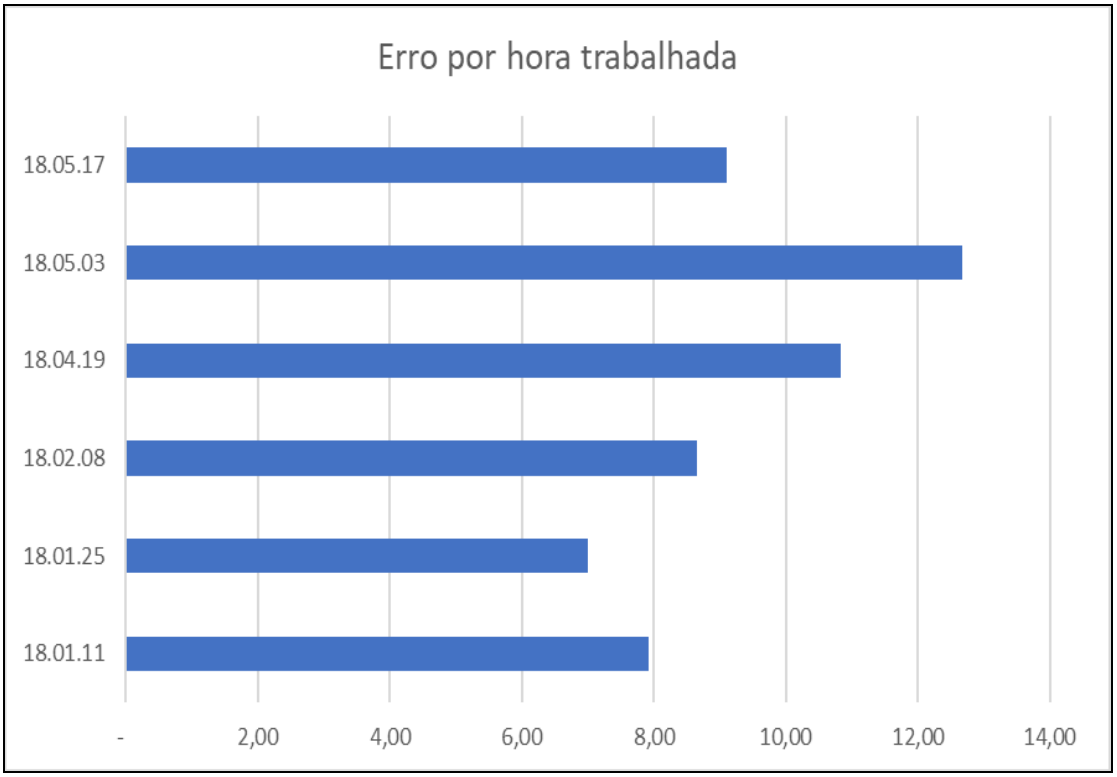
Figura 14 – Gráfico de erro por *sprint*



Fonte: Do autor.

Comparando os *sprints* (figura 14) 18.01.11, 18.01.25 e 18.02.08 com os *sprints* 18.04.19, 18.05.03 e 18.05.17 ocorreu um crescimento de 7% na quantidade de erros cadastrados. Contudo quando se avalia a quantidade de erros gerados por hora trabalhada (figura 15) constata-se uma melhora de 38% no aproveitamento do tempo. Se comparar o *sprint* 18.01.25 com o *sprint* 18.05.03 houve uma melhora de 81% na quantidade de erros gerado por hora trabalhada.

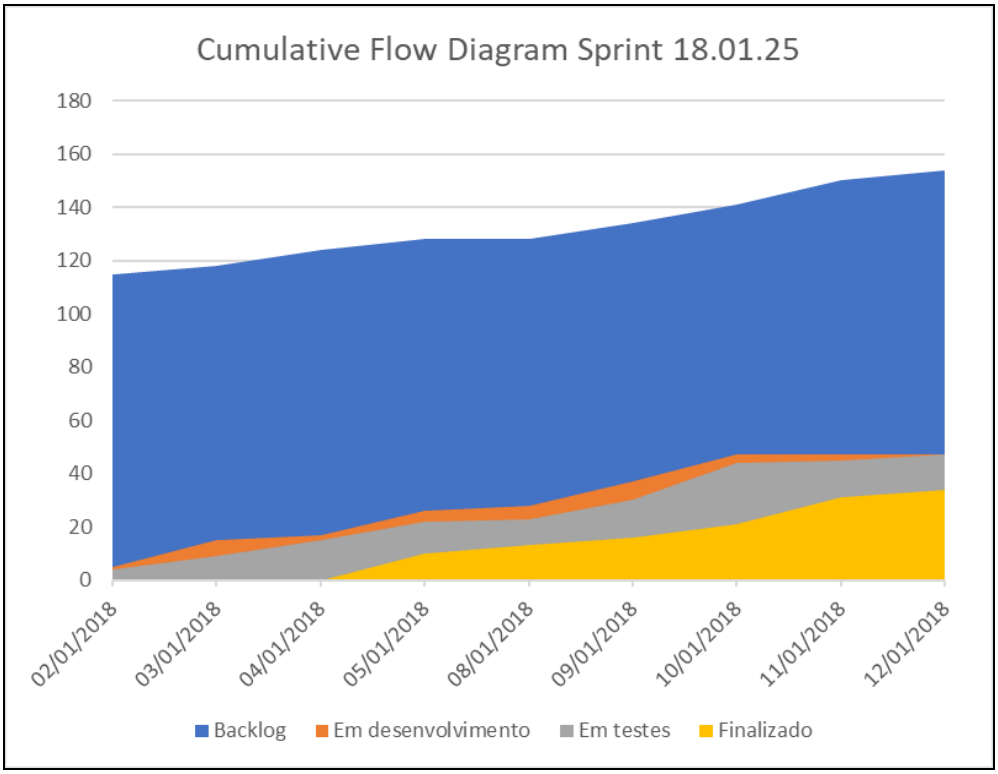
Figura 15 – Gráfico de erro por hora trabalhada



Fonte: Do autor.

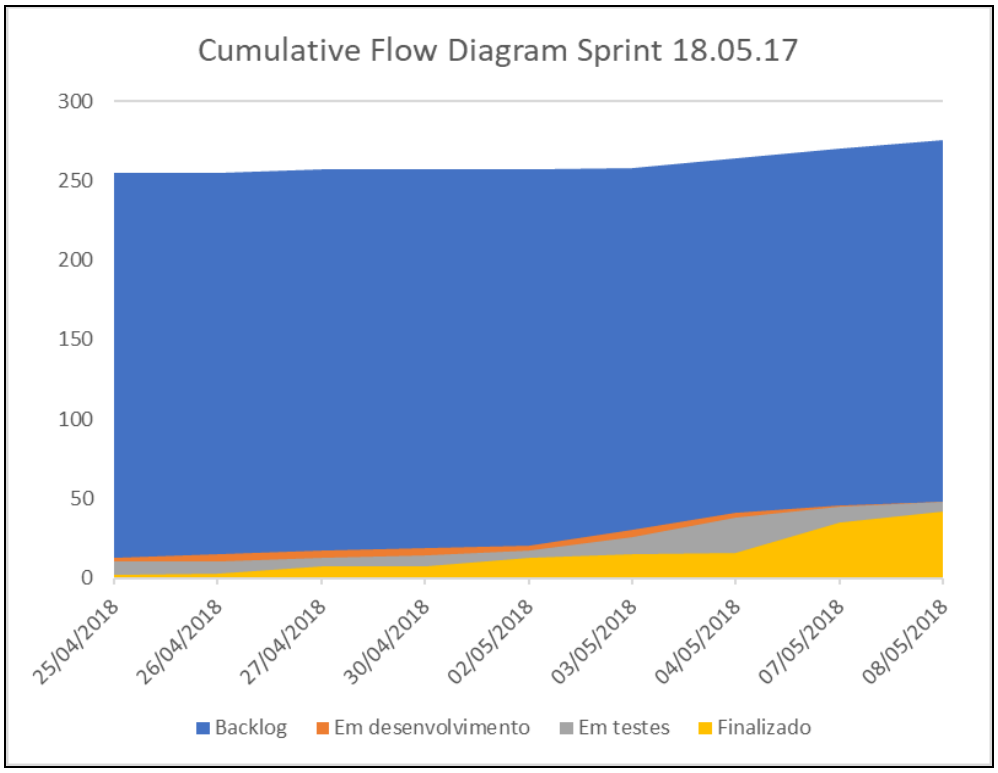
Comparando o CFD do *sprint* 18.01.25 (figura 16) com o do *sprint* 18.05.17 (figura 17) observa-se que o tempo que uma demanda fica em desenvolvimento diminuiu, houve uma melhora de 61% se compararmos os dois *sprints*, corroborando os números obtidos por meio do *lead time*. Além disso, o tempo que uma demanda fica pronta para testes mostrou-se igual entre os dois *sprints*, contudo constatou-se uma melhora no aproveitamento do tempo do teste, onde houve um ganho de 53% entre os dois *sprints*.

Figura 16 – Gráfico CFD do *sprint* 18.01.25



Fonte: Do autor.

Figura 17 – Gráfico CFD do *sprint* 18.05.17



Fonte: Do autor.

Portando após análise das métricas do CFD, *throughput* e *lead time*, comparando três *sprints* com o modelo híbrido desenvolvido e o modelo utilizado anteriormente na organização pode-se observar que houve uma melhora no aproveitamento do tempo trabalhado pela equipe. Além disso, houve uma melhora significativa no desempenho da equipe de desenvolvimento para conclusão de uma demanda, constatado por meio do gráfico do *lead time*.

5 CONCLUSÃO

Criar times de desenvolvimento produtivos é um dos grandes desafios das organizações. Para contornar essa dificuldade as empresas adotam modelos de desenvolvimento ágeis, que acabam se tornando um problema quando aplicados em times com pouca maturidade, comprometendo a qualidade e a produtividade das entregas. Um dos grandes fatores da queda de produtividade está relacionado com a Lei de Parkinson e a síndrome do estudante.

Com o objetivo de otimizar o desempenho de uma equipe de desenvolvimento de uma empresa da região de Criciúma – SC foi proposto e aplicado um modelo híbrido de processo de desenvolvimento de software. O modelo híbrido mostrou-se eficiente ao tratar a Lei de Parkinson conseguindo uma variação de 6% entre o tempo estimado e o tempo trabalhado. Além disso, houve um aumento significativo da qualidade das entregas, chegando a uma melhora de 38% na quantidade de erros gerados por hora trabalhada, contornando desta forma o problema apresentado pela síndrome do estudante.

A utilização do *throughput* e do *lead time* foram primordiais na detecção dos gargalos da equipe de desenvolvimento, contudo o CFD mostrou-se uma ferramenta de apoio.

O *throughput* auxiliou na identificação dos períodos de queda de produtividade do time, fator determinante para definir no modelo o papel do analista de sistemas, retirando dos programadores a responsabilidade da estimativa do tempo das demandas e direcionando ao analista de sistemas, o que ajudou a corrigir o problema com a Lei de Parkinson.

O *lead time* colaborou na quantificação do tempo gasto pelos programadores para implementar uma demanda e disponibilizar para testes, isso influenciou na decisão de reduzir o tempo de planejamento do *sprint* para 1 dia. Desta maneira, os programadores passam 90% do *sprint* desenvolvendo, fazendo com que exista uma entrega constante de valores para o setor de testes, reduzindo assim o tempo que o testador fica parado esperando a implementação dos requisitos.

Portanto após a implantação do modelo híbrido de processo de desenvolvimento em uma equipe de desenvolvimento da região de Criciúma – SC foi

constatado a eficiência das práticas propostas, sendo essas pautadas nos resultados obtidos por essa pesquisa.

Como proposta de trabalho futuro, sugere-se a criação de métricas para aprimorar a inclusão das demandas dentro dos *sprints*, desta forma, conseguindo um melhor preenchimento do tempo dos desenvolvedores. Além disso, a criação de métricas para trabalhar nos erros dentro do *sprint*, buscando assim um parâmetro para ser utilizado no momento do preenchimento das demandas dentro do *sprint*.

REFERÊNCIAS

ABRAHAM, John T; MATHAI, Manju K; VENUGOPAL, Rakhi. HYBRID MODEL FOR SOFTWARE DEVELOPMENT. **International Journal Of Research In Engineering And Technology.** S. L., p. 198-202. jan. 2016. Disponível em: <http://esatjournals.net/ijret/2016v05/i01/IJRET20160501040.pdf>. Acesso em: 20 mar. 2018.

AGUENA, Márcia Luciana Silva et al. CMMI e SCAMPI: uma visão geral dos modelos de qualidade e de um método formal para sua avaliação. **Revista de Ciências Exatas e Tecnologia**, S. L., v. 1, p.22-31, 2006. Anual. Disponível em: <http://www.pgsskroton.com.br/seer/index.php/rcext/article/view/2392/2296>. Acesso em: 22 abr. 2018

ALBINO, Raphael Donaire. Métricas Ágeis: Obtenha melhores resultados em sua equipe. São Paulo: Casa do Código, 2017. 261 p.

BECK, Kent et al. **Manifesto for Agile Software Development**. 2001. Disponível em: <http://www.agilemanifesto.org/> Acesso em: 10 mar. 2018.

ELDER, Allan. The Five Diseases of Project Management. 2006. Disponível em: <http://www.nolimitsleadership.com/images/The Five Diseases of Project Management.pdf>. Acesso em: 10 mar. 2018.

GOLDRATT, Eliyahu M.. Critical Chain. Great Barrington: The North River Press, 2002. 246 p.

GOVARDHAN, A.; MUNASSAR, Nabil Mohammed Ali. **HYBRID MODEL FOR SOFTWARE DEVELOPMENT PROCESSES.** 2010. Disponível em: <https://pdfs.semanticscholar.org/6c66/61ab8ae91a59c28803bcae7e6d63eadfc904.pdf?_ga=2.135071858.2140202031.1527000280-781384142.1527000280>. Acesso em: 20 mar. 2018.

HAN, Jianchao; HAYATA, Tomohiro. A hybrid model for IT project with Scrum. **Proceedings Of 2011 Ieee International Conference On Service Operations, Logistics And Informatics**, [s.l.], p.285-290, jul. 2011. IEEE. http://dx.doi.org/10.1109/soli.2011.5986572.

HARRINGTON, H. James; MCNELLIS, Thomas. Mobilizing the Right Lean Metrics for Success. 2006. Disponível em: <https://www.qualitydigest.com/may06/articles/02_article.shtml>. Acesso em: 6 mar. 2018.

HIRAMA, Kechi. **Engenharia de Software:** Qualidade e produtividade com tecnologia. Rio de Janeiro: Elsevier Editora Ltda, 2012. 280 p.

JHA, Madan Mohan; VILARDELL, Rosa Maria Ferrer; NARAYAN, Jai. Scaling Agile Scrum Software Development: Providing Agility and Quality to Platform Development by Reducing Time to Market. 2016 Ieee 11th International Conference On Global

Software Engineering (icgse), [s.l.], p.84-88, ago. 2016. IEEE. http://dx.doi.org/10.1109/icgse.2016.24.

MAXIM, Bruce R.; PRESSMAN, Roger S.. **Engenharia de Software:** Uma abordagem profissional. 8. ed. São Paulo: Bookman, 2016. 940 p.

MELHORIA DE PROCESSO DO SOFTWARE BRASILEIRO. **MA-MPS**: Guia de Avaliação Parte I – Processo e Método de Avaliação MA-MPS. 2017. 116 p. Disponível em: <http://www.softex.br/wp-content/uploads/2017/11/MPS.BR_Guia_de-Avaliacao_2017-Parte-1-abril2017.pdf>. Acesso em: 22 abr. 2018.

MICROSOFT. **Cumulative flow, lead time, and cycle time guidance.** 2018. Disponível em: <https://docs.microsoft.com/en-us/vsts/report/dashboards/cumulative-flow-cycle-lead-time-guidance?view=vsts>. Acesso em: 27 mar. 2018.

PARKINSON, Cyril Northcote. Parkinson's Law. 1955. Disponível em: <https://www.economist.com/node/14116121>. Acesso em: 6 mar. 2018.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011. 530 p.

SUTHERLAND, Jeff. Scrum: A arte de fazer o dobro do trabalho na metade do tempo. 2. ed. São Paulo: Leya, 2016. 240 p.

VENTURA, Plínio. Porque os projetos dão errado. 2016. Disponível em: <http://www.ateomomento.com.br/porque-os-projetos-projetos-dao-errado/>. Acesso em: 28 mar. 2018.

VIJAYASARATHY, Leo R.; BUTLER, Charles W.. Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter?. **Ieee Software**, [s.l.], v. 33, n. 5, p.86-94, set. 2016. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/ms.2015.26.

WAZLAWICK, Raul S.. **Engenharia de Software:** Conceitos e Práticas. Rio de Janeiro: Elsevier, 2013. 368 p.

APÊNDICE(S)

Utilização de Modelos Híbridos para a Definição e Execução de Processos de Software Aplicados na Otimização da Produtividade

Flaris B. Martinhago[1], Gustavo Bisognin[1]

[1]Departamento de Ciência da Computação
Universidade do Extremo Sul Catarinense (UNESC) – Criciúma, SC – Brasil

{flaris, gustavo}@unesc.net

Abstract. *The search for more performance to make organizations use agile models to optimize the results, however when this models are applied in inexperienced teams become a problem, it's implying in the Parkinson Law and student syndrome. From this premise was created a hybrid model of process of development of software and applied in a develop team in a company from region of Criciúma – SC, it's optimizing the team performance. The hybrid model created eliminated the efects from Parkinson Law and student syndrome from the development process, it's reaching more efficiency in the worked time during the sprints, it was mimimize days without development and test.*

Resumo. *A busca por mais performance faz com que organizações utilizem modelos ágeis para otimizar os resultados, contudo quando esses modelos são aplicados em times inexperientes tornam-se um problema, implicando na Lei de Parkinson e na síndrome do estudante. A partir desta premissa foi criado um modelo híbrido de processo de desenvolvimento de software e aplicado em um time de desenvolvimento de uma empresa da região de Criciúma – SC, otimizando a performance do time. O modelo híbrido criado eliminou os efeitos da Lei de Parkinson e da síndrome do estudante do processo de desenvolvimento, atingindo maior eficiência no tempo trabalhado durante os sprints, minimizando dias sem desenvolvimento e teste.*

1. Introdução

Conforme dados levantados pelo Standish Group somente 6% dos projetos de grande porte obtém sucesso [Ventura 2016]. Neste contexto, os principais problemas apresentados, referem-se a atrasos na entrega, falhas de implementação, problemas relacionados a estimativas e orçamentos. Uma das formas encontradas por diversas equipes de desenvolvimento na área de software é a utilização de modelos que abordam um contexto ágil, porém estes modelos, quando aplicados em equipes com pouca maturidade acabam agravando os problemas, principalmente no que tange a produtividade bem como a qualidade das entregas. Este comprometimento pode ser observado em dados levantados nas medições realizadas durante a execução das *sprints* de desenvolvimento em diversas equipes, sendo que a principal causa está relacionada a Lei de Parkinson e a síndrome do estudante, mesmo utilizando os conceitos da metodologia ágil.

A Lei de Parkinson descreve o comportamento onde o indivíduo define um tempo para realização de uma atividade com margens de segurança e levará o tempo estimado para realização dessa atividade, ou seja, o tempo se expande para preencher o tempo disponível [Parkinson 1955]. Esse comportamento está presente nas estimativas de projetos e mesmo o utilizando projetos continuam atrasando. A síndrome do estudante se caracteriza pelo fenômeno onde as pessoas começam a finalizar uma tarefa somente no último momento possível, afetando desta forma a qualidade da entrega [Goldratt 2002]. O problema não está na estimativa da tarefa, mas sim em dimensionar quais fatores devem ser considerados para se trabalhar na tarefa sem interrupção. A Lei de Parkinson somada com a Síndrome do Estudante contribuem para atrasos no término de projetos, conforme citado por Elder (2006) no artigo *The Five Diseases of Project Management*, sendo essas razões do comportamento humano responsáveis por atrasos nos projetos de software e que devem ser tratadas para evitar que se repitam.

O presente trabalho propõe a aplicação de um modelo híbrido de processo de desenvolvimento de software como forma de otimizar e avaliar a performance de uma equipe de desenvolvimento de software em uma empresa da região de Criciúma, SC. Considerando o modelo proposto será analisado o *leadtime* da equipe, o *throughput* e o *cumulative flow diagram* (CFD) determinando se existem gargalos no processo de estimativa, desenvolvimento e teste. A partir da identificação dos gargalos serão propostas soluções para maximizar o desempenho do time de desenvolvimento e assegurar um ritmo constante de entrega de valores.

2. Modelo Híbrido Desenvolvido

Devido a problemas de produtividade existentes no modelo em utilização por uma empresa da região de Criciúma - SC, onde após análise foi observado oportunidades de melhoria, foi proposto a criação de um modelo híbrido de desenvolvimento de software (figura 1) utilizando modelos ágeis e modelos tradicionais para obter a maior eficiência no processo de desenvolvimento.

O modelo híbrido é definido pelas seguintes etapas:

a) 1 – priorização do *sprint*: essa reunião tem como objetivo definir o que será trabalhado nos próximos 45 dias dando previsibilidade ao processo de definição, desenvolvimento e teste. Participa dessa reunião os responsáveis pelos setores de desenvolvimento, gerência de produto e teste. Ao término dessa reunião a gerência de produto deve colocar no *sprint* os itens que devem ser trabalhados para que a equipe de desenvolvimento e teste possam realizar a estimativa;

b) 2 – pré-planejamento: após a definição do que deverá ser trabalhado na reunião de priorização do *sprint* acontece a reunião de planejamento, onde se tem como objetivo avaliar o que entrará no *sprint*. A equipe de desenvolvimento e de teste devem chegar nessa reunião com as estimativas finalizadas. Participa dessa reunião o analista de sistemas, o analista de negócio e o testador do time;

c) 3 – planejamento: ao término da reunião de planejamento a equipe de desenvolvimento se reuni e distribui o trabalho entre os desenvolvedores do time. Nessa etapa os desenvolvedores tiram dúvidas sobre as demandas atribuídas a eles. Além disso, caso seja necessário realizar alguma alteração na estimativa realizada é feito nessa reunião;

d) 4 – *sprint*: o *sprint* tem duração de duas semanas iniciando em uma quarta feira. O tempo dos desenvolvedores é alocado 70% para desenvolvimento dos itens previstos no *sprint*, os outros 30% são destinados para a correção dos erros críticos e graves que são corrigidos imediatamente que detectados;

e) reunião diária: é realizada diariamente no início do dia e tem a duração de 15 minutos no máximo, todos os integrantes do time participam. O principal objetivo da reunião é detectar bloqueios que afetam a implementação dos requisitos. Caso seja identificado algum bloqueio durante a reunião imediatamente após o término é realizada uma outra reunião com as pessoas necessárias para desbloquear o item indicado na reunião diária. Desta maneira, logo no início do dia os bloqueios são resolvidos e os integrantes do time podem continuar a implementação dos requisitos;

f) 5 – revisão do *sprint*: ao término do *sprint* é realizada uma reunião para apresentar aos *stakeholder* e pessoas interessadas o resultado do que foi implementado durante o *sprint*. Caso algum requisito não esteja de acordo com o esperado é identificado e corrigido em um *sprint* futuro;

g) 6 – retrospectiva do *sprint*: essa reunião ocorre mensalmente e é realizada no último dia do primeiro *sprint* do mês, todos os membros do time participam e o objetivo é discutir o processo de desenvolvimento, não se fala do produto nessa reunião.

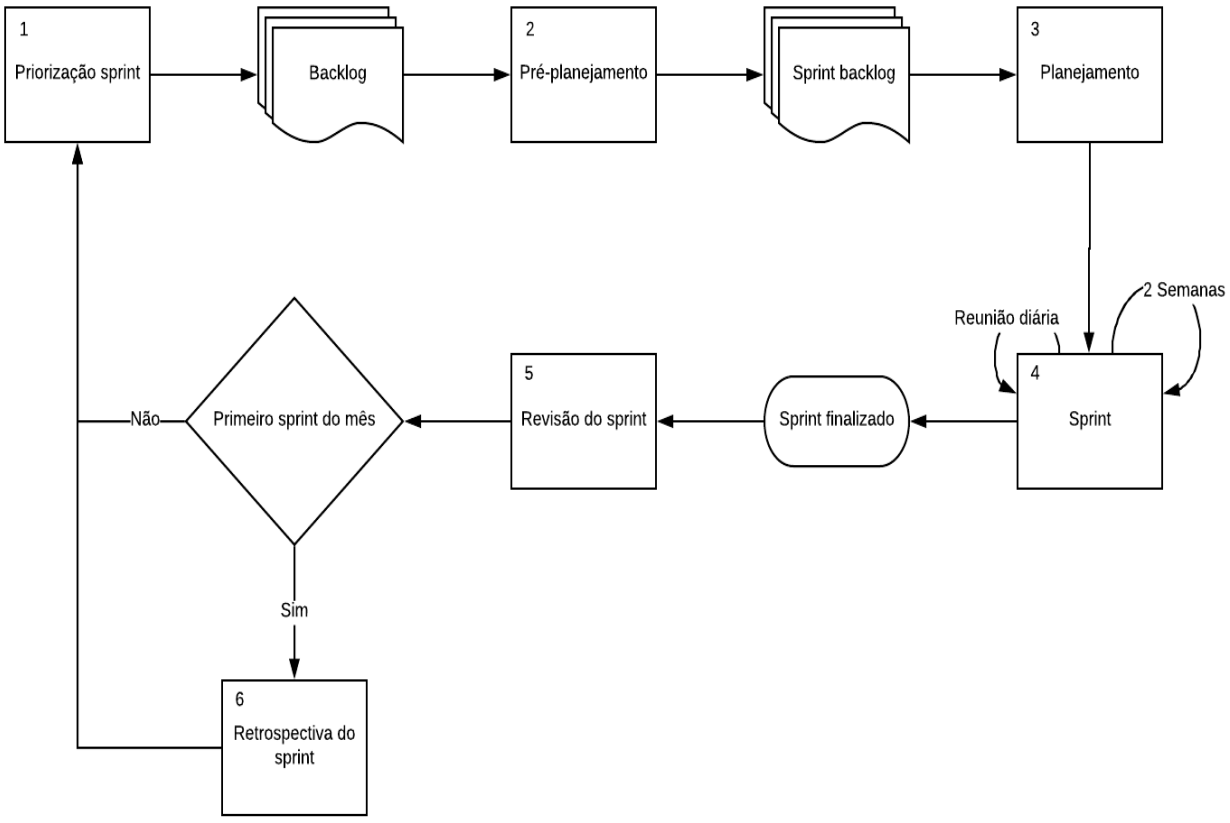


Figura 1. Modelo híbrido proposto

Na tabela 1 é definida a composição do time de desenvolvimento:

Tabela 1. Composição do time de desenvolvimento no modelo híbrido

Quantidade	Papel
1	Analista de Negócio
1	Testador
1	Analista de Sistemas
3	Programador

As responsabilidades dos membros que compõe o time de desenvolvimento são:

a) analista de negócio: responsável por criar as definições dos requisitos que serão implementados pelo time;

b) analista de sistemas: 50% do seu tempo é destinado a realização da estimativa dos requisitos para os programadores e os outros 50% é destinado ao desenvolvimento e auxílio aos desenvolvedores quando solicitado;

c) testador: elabora os casos de teste e testa as demandas do *sprint*;

d) programador: implementa as demandas do *sprint*.

O modelo híbrido foi concebido para desburocratizar o processo de desenvolvimento, procurando atingir maior eficiência no tempo trabalhado durante os *sprints* e minimizando dias sem desenvolvimento e teste. Além disso, o modelo busca dar visibilidade aos gestores por meio da previsibilidade do que irá ser trabalhado nos próximos 45 dias, facilitando a gestão do projeto e controle das entregas.

3. Implementação do modelo híbrido

A definição do modelo híbrido iniciou pela identificação dos gargalos no processo de desenvolvimento existente, para isso foram coletados dados de três *sprints* avaliando o *lead time*, *throughput* e CFD da equipe. Após uma análise inicial dos dados coletados foram constatados os problemas abaixo:

a) por meio do CFD foi possível verificar que existiam definições prontas para montar as *sprints*, contudo ao avaliar as demandas foi observado que haviam muitas demandas incompletas, faltando definições;

b) o *lead time* mostrou que o tempo que uma demanda ficava em desenvolvimento era alto;

c) a quantidade de demandas concluídas por semana foi verificada por meio do *throughput* que mostrou uma queda acentuada nas semanas que havia o planejamento do *sprint* seguinte.

Para atingir um maior aproveitamento do tempo dos desenvolvedores durante a *sprint* foi instituído que as estimativas seriam feitas por um analista de sistemas. Além disso, para assegurar que a gerência de produtos estava trabalhando nas definições das demandas que seriam trabalhadas nos próximos *sprints* foi criada a reunião de priorização do *sprint*, desta maneira, as equipes de gerência de produtos, teste e desenvolvimento alinhavam-se com o que deveria ser dado foco no momento, dando previsibilidade ao processo de desenvolvimento.

No dia 06 de março de 2018 foi realizada uma reunião com toda a equipe de desenvolvimento para apresentar o modelo híbrido proposto. Nessa reunião foi explicado como funcionaria os *sprints*, a divisão da equipe e as responsabilidades de cada papel.

A primeira ação foi colocar fisicamente os setores de desenvolvimento, teste e gerência de produtos próximos, pois estavam em andares separados. Isso facilitou a comunicação entre os membros do time, diminuindo a necessidade de reuniões para sanar dúvidas ou problemas durante as fases de definição, desenvolvimento e teste.

Nas semanas seguintes o modelo foi aplicado nos *sprints* seguindo o modelo híbrido proposto, sendo que houve um acompanhamento para garantir a correta aplicação das ações propostas.

4. Resultados

Para avaliação dos resultados foram coletados dados durante 6 *sprints*, sendo que os *sprints* 18.01.11, 18.01.25 e 18.02.08 foram desenvolvidos seguindo o modelo ágil. Os *sprints* 18.04.19, 18.05.03 e 18.05.17 foram desenvolvidos seguindo o modelo híbrido proposto.

Na figura 2 é demonstrado o comparativo do tempo trabalhado versus o tempo estimado. Nos *sprints* 18.01.11, 18.01.25 e 18.02.08 as estimativas foram realizadas pelos próprios programadores. No *sprint* 18.01.11 o tempo trabalhado ficou 14% menor que o tempo estimado, no *sprint* 18.01.25 o tempo trabalhado ficou 22% menor que o tempo estimado e no *sprint* 18.02.08 o tempo trabalhado ficou 11% menor que o tempo estimado. Nos *sprints* 18.04.19, 18.05.03 e 18.05.17 as estimativas foram realizadas por um analista de sistemas, onde o tempo estimado ficou próximo ao tempo trabalhado pelos programadores, mostrando uma variação de apenas 6% para baixo.

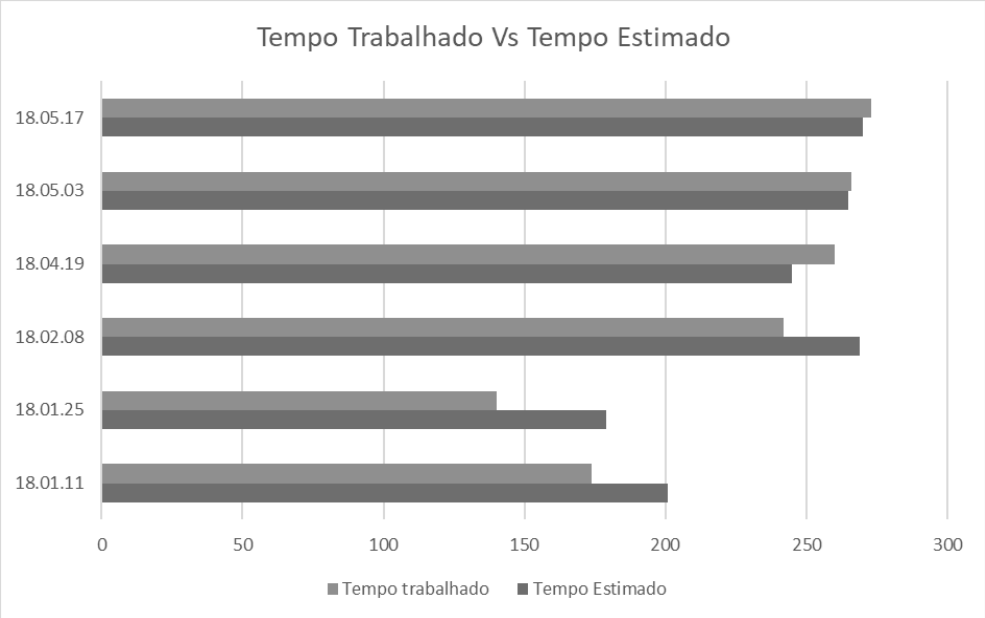


Figura 2. Gráfico de tempo trabalhado versus tempo estimado

O *lead time* (figura 3) demonstrou uma variação de 5% comparando o modelo ágil com o modelo híbrido. Comparando os *sprints* 18.01.11, 18.01.25 e 18.02.08 com os *sprints* 18.04.19, 18.05.03 e 18.05.17, houve uma redução 38% do tempo que um item fica em desenvolvimento, ou seja, o desenvolvimento começou a entregar as demandas mais rapidamente para o teste. Contudo houve um aumento de 22% no tempo que o teste gasta para concluir uma demanda, isso aconteceu devido a problemas no ambiente de teste e sobrecarga do testador.

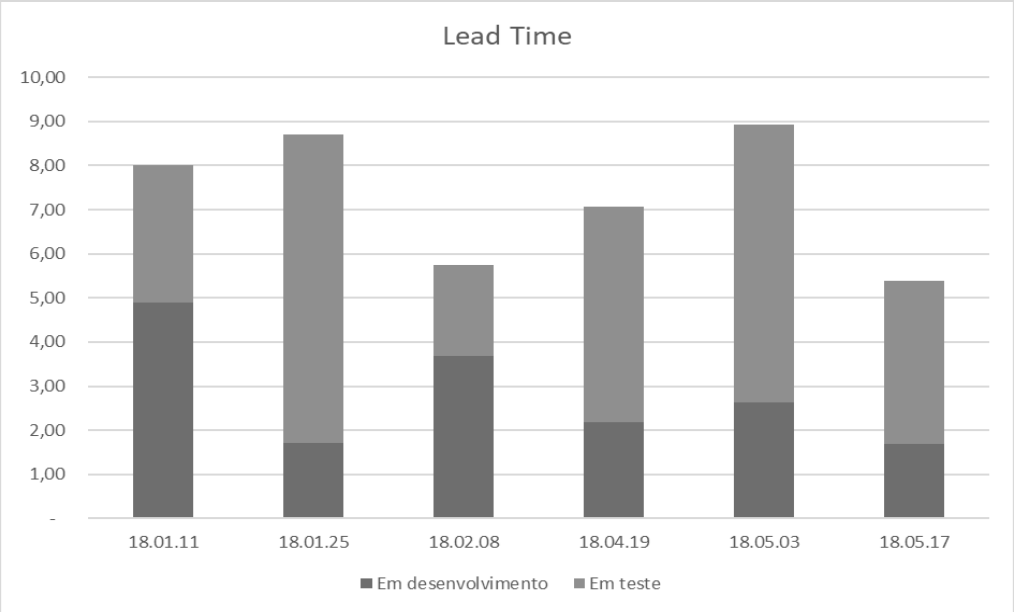


Figura 3. Gráfico do *lead time*

O modelo híbrido foi adotado no dia 07/03/2018, comparando as semanas anteriores a adoção do modelo (figura 4) nota-se pico nas entregas das demandas, sendo que na semana seguinte existe uma queda. Essa queda existia devido aos programadores serem alocados na estimativa dos requisitos quando se finalizava um *sprint*. Durante dois dias não havia desenvolvimento, o time era alocado na preparação do próximo *sprint*, o que gerava a queda observada no gráfico.

Com a adoção do modelo híbrido as entregas ficaram constantes, não existindo mais quedas no *throughput* no final dos *sprint*, isso se deve as estimativas serem realizadas por um analista de sistemas. Desta forma, os programadores passam mais tempo implementando os requisitos, não precisando mais parar por dois dias para realizar estimativas.

Na figura 4 a semana do dia 29/04/2018 à 05/05/2018 apresenta uma queda no *throughput*, essa queda aconteceu devido a essa semana possuir um feriado, reduzindo a quantidade de dias trabalhados pela equipe de desenvolvimento.

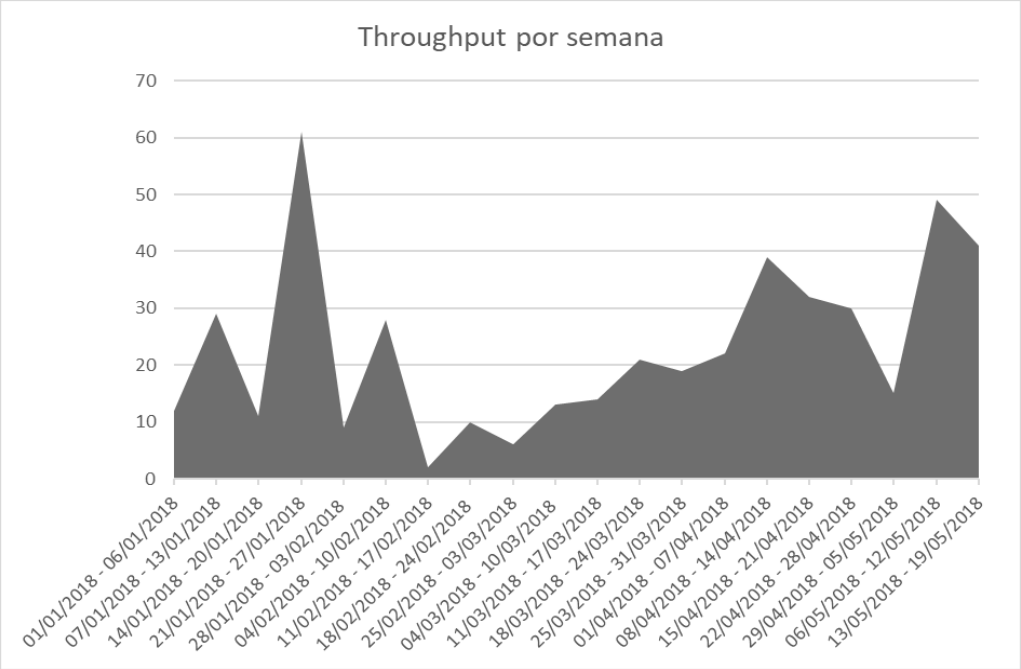


Figura 4. Gráfico do *throughput* por semana

Comparando os *sprints* 18.01.11, 18.01.25 e 18.02.08 com os *sprints* 18.04.19, 18.05.03 e 18.05.17 ocorreu um crescimento de 7% na quantidade de erros cadastrados. Contudo quando se avalia a quantidade de erros gerados por hora trabalhada (figura 5) constata-se uma melhora de 38% no aproveitamento do tempo. Se comparar o *sprint* 18.01.25 com o *sprint* 18.05.03 houve uma melhora de 81% na quantidade de erros gerado por hora trabalhada.

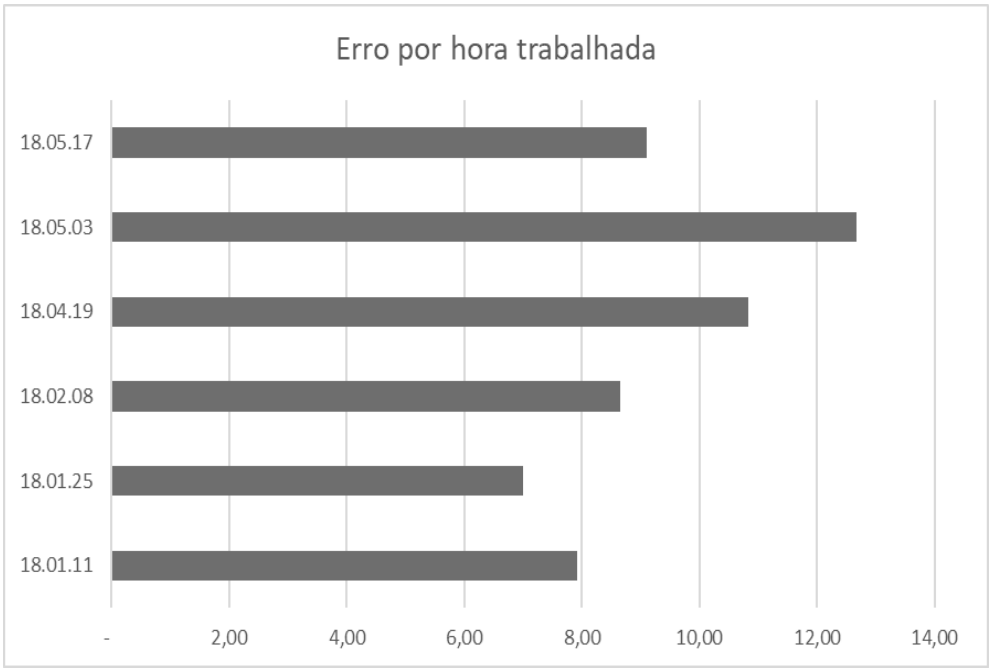


Figura 5. Gráfico de erro por hora trabalhada

Comparando o CFD do *sprint* 18.01.25 (figura 6) com o do *sprint* 18.05.17 (figura 7) observa-se que o tempo que uma demanda fica em desenvolvimento diminuiu, houve uma melhora de 61% se compararmos os dois *sprints*, corroborando os números obtidos por meio do *lead time*. Além disso, o tempo que uma demanda fica pronta para testes mostrou-se igual entre os dois *sprints*, contudo constatou-se uma melhora no aproveitamento do tempo do teste, onde houve um ganho de 53% entre os dois *sprints*.

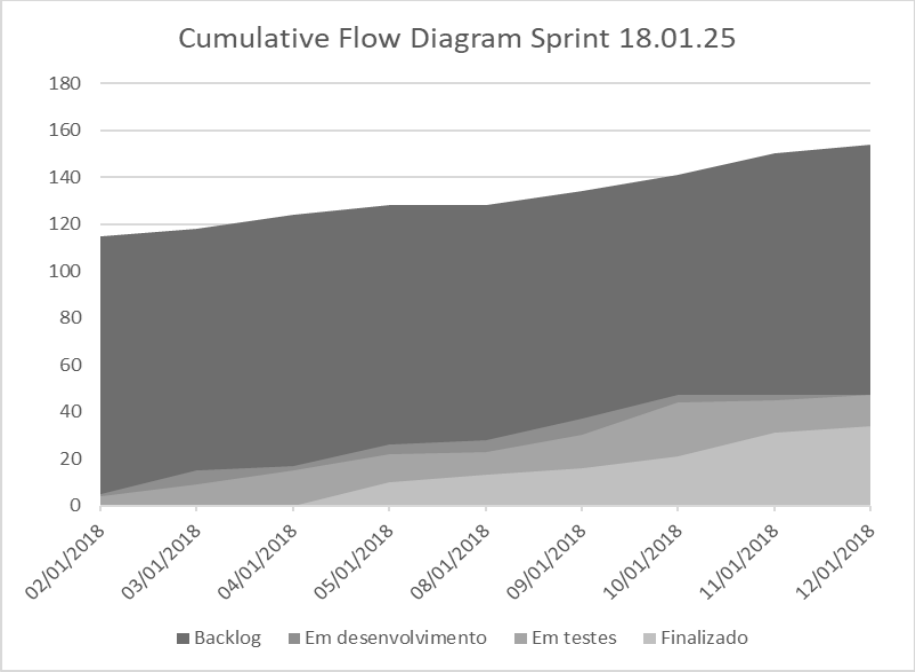


Figura 6. Gráfico CFD do *sprint* 18.01.25

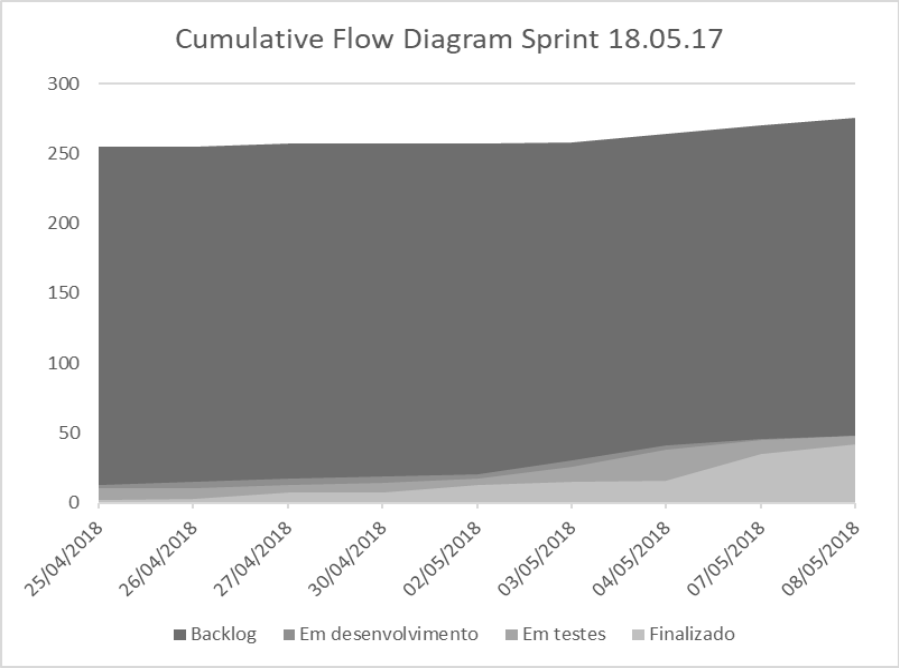


Figura 7. Gráfico CFD do *sprint* 18.05.17

Portando após análise das métricas do CFD, *throughput* e *lea time*, comparando três *sprints* com o modelo híbrido desenvolvido e o modelo utilizado anteriormente na organização pode-se observar que houve uma melhora no aproveitamento do tempo trabalhado pela equipe. Além disso, houve uma melhora significativa no desempenho da equipe de desenvolvimento para conclusão de uma demanda, constatado por meio do gráfico do *lead time*.

5. Conclusão

Criar times de desenvolvimento produtivos é um dos grandes desafios das organizações. Para contornar essa dificuldade as empresas adotam modelos de desenvolvimento ágeis, que acabam se tornando um problema quando aplicados em times com pouca maturidade, comprometendo a qualidade e a produtividade das entregas. Um dos grandes fatores da queda de produtividade está relacionado com a Lei de Parkinson e a síndrome do estudante.

Com o objetivo de otimizar o desempenho de uma equipe de desenvolvimento de uma empresa da região de Criciúma – SC foi proposto e aplicado um modelo híbrido de processo de desenvolvimento de software. O modelo híbrido mostrou-se eficiente ao tratar a Lei de Parkinson conseguindo uma variação de 6% entre o tempo estimado e o tempo trabalhado. Além disso, houve um aumento significativo da qualidade das entregas, chegando a uma melhora de 38% na quantidade de erros gerados por hora trabalhada, contornando desta forma o problema apresentado pela síndrome do estudante.

A utilização do *throughput* e do *lead time* foram primordiais na detecção dos gargalos da equipe de desenvolvimento, contudo o CFD mostrou-se uma ferramenta de apoio. O *throughput* auxiliou na identificação dos períodos de queda de produtividade do time, fator determinante para definir no modelo o papel do analista de sistemas, retirando dos programadores a responsabilidade da estimativa do tempo das demandas e direcionando ao analista de sistemas, o que ajudou a corrigir o problema com a Lei de Parkinson.

O *lead time* colaborou na quantificação do tempo gasto pelos programadores para implementar uma demanda e disponibilizar para testes, isso influenciou na decisão de reduzir o tempo de planejamento do *sprint* para 1 dia. Desta maneira, os programadores passam 90% do *sprint* desenvolvendo, fazendo com que exista uma entrega constante de valores para o setor de testes, reduzindo assim o tempo que o testador fica parado esperando a implementação dos requisitos.

Portanto após a implantação do modelo híbrido de processo de desenvolvimento em uma equipe de desenvolvimento da região de Criciúma – SC foi constatado a eficiência das práticas propostas, sendo essas pautadas nos resultados obtidos por essa pesquisa.

6. Referências

Elder, Allan. The Five Diseases of Project Management. 2006. Disponível em: <http://www.nolimitsleadership.com/images/The Five Diseases of Project Management.pdf>. Acesso em: 10 mar. 2018.

Goldratt, Eliyahu M.. Critical Chain. Great Barrington: The North River Press, 2002. 246 p.

Parkinson, Cyril Northcote. Parkinson's Law. 1955. Disponível em: <https://www.economist.com/node/14116121>. Acesso em: 6 mar. 2018.

Ventura, Plínio. Porque os projetos dão errado. 2016. Disponível em: <http://www.ateomomento.com.br/porque-os-projetos-projetos-dao-errado/>. Acesso em: 28 mar. 2018.