UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ALESSANDRO STANKIEWICZ

MODELO DE INTERAÇÃO ÁGIL: UMA ADAPTAÇÃO DO MODELO CASCATA À ORGANIZAÇÃO DE PEQUENAS E MÉDIAS EMPRESAS

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2017

ALESSANDRO STANKIEWICZ

MODELO DE INTERAÇÃO ÁGIL: UMA ADAPTAÇÃO DO MODELO CASCATA À ORGANIZAÇÃO DE PEQUENAS E MÉDIAS EMPRESAS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. MSc. Vinícius Camargo Andrade

PONTA GROSSA

2017

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

Diretoria de Pesquisa e Pós-Graduação
Departamento Acadêmico de Informática
Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

MODELO DE INTERAÇÃO ÁGIL: UMA ADAPTAÇÃO DO MODELO CASCATA À ORGANIZAÇÃO DE PEQUENAS E MÉDIAS EMPRESAS

por

ALESSANDRO STANKIEWICZ

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 01 de novembro de 2017 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. MSc. Vinícius Camargo Andrade
Prof. Orientador

_____
Prof. Dr. Diego Roberto Antunes
Membro titular

_____
Profª Drª Simone Nasser Matos
Membro titular

_____
Profª Drª Helyane Bronoski Borges
Responsável pelos Trabalhos
de Conclusão de Curso

_____
Profª Drª Mauren Louise Sguario
Coordenadora do Curso
UTFPR - Campus Ponta Grossa

AGRADECIMENTOS

Agradeço ao meu orientador pela paciência e pela ajuda no desenvolvimento do trabalho e por acreditar na ideia.

À minha família que me ajudou e me apoiou nas escolhas e nos caminhos que trilhei ao longo do tempo. Também a minha namorada, por apoiar e ter paciência ao logo desse processo.

Ao meu atual chefe, Matheus Simões Martins pela oportunidade de estagiar em um local onde se aplica a metodologia Scrum e também por oferecer materiais para o estudo. Também ao supervisor Fernando Barreto pelos ensinamentos do funcionamento do Scrum na pratica de desenvolvimento de sistemas.

Ao meu grande amigo Sergio da Silva Santana, que me ajudou nos estudos e nos trabalhos acadêmicos.

RESUMO

STANKIEWICZ, Alessandro. **MODELO DE INTERAÇÃO ÁGIL:** Uma Adaptação do Modelo Cascata à Organização de Pequenas e Médias Empresas. 2017. 45 f. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2017.

O modelo cascata é uma metodologia de desenvolvimento sequencial conhecido como modelo tradicional de desenvolvimento, e por muito tempo atendeu as necessidades de controlar projetos de software. Ao longo dos anos, houve um aumento da complexidade dos sistemas, isto fez com que o modelo tradicional de desenvolvimento não suprisse as novas demandas. Este trabalho propõe a adaptação do modelo Cascata para que contemple características de metodologias de desenvolvimento e gerenciamento ágeis. A partir desta adaptação espera-se que o desenvolvimento possa ser incremental, bem como facilitar a identificação e correção de erros e/ou falhas durante o processo de desenvolvimento. Como avaliação, o modelo proposto foi aplicado a um estudo de caso a fim verificar sua eficácia. Uma comparação foi realizada uma comparação do modelo cascata com o modelo proposto, apontando as principais vantagens e desvantagens de ambos os modelos.

Palavras-chave: Modelo de Processo. Modelo de Interação Ágil. Modelo Cascata.

ABSTRACT

STANKIEWICZ, Alessandro. **AGILE INTERACTION MODEL:** An Adaptation of the Cascade Model to the Organization of Small and Medium Enterprises. 2017. 45 p. Work of Conclusion Course (Graduation in Technology in Systems Analysis and Development) - Federal University of Technology - Paraná. Ponta Grossa, 2017.

The cascade model is a sequential development methodology known as the traditional development model, and has long served the needs of controlling software projects. Over the years, there has been an increase in the complexity of systems, which has meant that the traditional model of development does not meet the new demands. This work proposes the adaptation of the Cascade model to include characteristics of agile development and management methodologies. From this adaptation it is expected that the development can be incremental, as well as facilitate the identification and correction of errors and / or failures during the development process. As an evaluation, the proposed model was applied to a case study in order to verify its effectiveness. A comparison was made a comparison of the cascade model with the proposed model, pointing out the main advantages and disadvantages of both models.

Keywords: Process Model. Agile Interaction Model. Cascade Model.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A demanda pelo desenvolvimento de sistemas tem aumentado consideravelmente nos últimos anos. Segundo Lee (2016), houve um crescimento de até 56% na demanda de produção de software para algumas empresas.

Com a necessidade de criar uma forma de organizar o processo, como por exemplo, padronizar a metodologia de levantamento de requisitos, criação, desenvolvimento, testes e manutenção do software, surgisse alguns modelos de ciclo de vida para softwares, tais como o Modelo Cascata (ROYCE, 1970), Modelo em Espiral (BOEHM, 1988), Modelo da Análise Estruturada (DIJKSTRA, 1962), entre outros.

A necessidade de organização dentro de uma empresa, independentemente do seu porte, é imprescindível, pois em meio as crises que todos os setores sofrem, qualquer desperdício pode acarretar em problemas sérios, como por exemplo, gastos desnecessários com tarefas sem planejamento, desperdício de tempo e, até mesmo, desenvolver um produto que não atenda aos requisitos funcionais do sistema.

Um dos problemas enfrentados pelas empresas de desenvolvimento de software, é a falta de padronização e documentação na criação do sistema. Tanto programadores iniciantes quanto experientes ainda cometem o erro de partir diretamente para a implementação, ignorando completamente as etapas inicias de desenvolvimento, como análise e modelagem de sistema (BORGES, 2017).

Segundo Borges (2017), uma fase mal planejada pode causar um erro de projeto irreversível e sem a documentação necessária, não tem como provar para o cliente a origem do erro e seus verdadeiros motivos. Outro problema relacionado a falta de documentação do sistema, ocorre sistematicamente quando um programador, por algum motivo, é demitido da empresa. Os projetos que eram de sua responsabilidade são realocados a outro pessoa. Esta mudança demanda tempo, pois o novo responsável pelo projeto terá que se inteirar de todo o funcionamento do sistema a fim de conseguir dar o suporte necessário ao mesmo.

Para que problemas desta natureza não ocorram, o modelo Cascata (ROYCE, 1970) pode ser utilizado. Comparado aos outros modelos de desenvolvimento de software, o modelo Cascata, também chamado de modelo *top-down*, possui um menor nível administrativo e uma maior rigidez. Além disso, seu desenvolvimento deu-se com o intuito de organizar desenvolvimentos de software complexos.

Segundo Royce (1970), todo o processo de desenvolvimento ocorre de forma linear, ou seja, as atividades são agrupadas em tarefas e executadas sequencialmente de maneira que uma só poderá ter início quando a sua anterior for finalizada. Ao todo, o modelo é formado por seis fases (SOMMERVILLE, 2011): análise de requisitos, projeto, implementação, testes (validação), integração, e manutenção de software.

Apesar do modelo citado ser considerado o modelo tradicional, há outros modelos derivados dele, como por exemplo: Cascata com subprojetos, Modelo V (LENZ; MOELLER, 2004) e Modelo W (SPILLNER, 2002). Porém, em nenhum é utilizado uma metodologia ágil de gerenciamento de projetos. Além disso, nos modelos V e W, por exemplo, as validações são realizadas com testes – de unidade, integração, sistema, aceitação. No modelo de subprojetos, o projeto é dividido em subsistemas que são implementados paralelamente. Ao fim do desenvolvimento destes, ocorre a integração dos mesmos e a entrega é realizada de maneira única.

Este trabalho realizou a adaptação do modelo Cascata, para que este contemple, além das vantagens do modelo tradicional de desenvolvimento, as características das metodologias de desenvolvimento/ gerenciamento ágeis. Para realizar a adaptação do modelo, uma revisão de literatura sobre modelos de processos tradicionais fez-se necessário, bem como um estudo sobre metodologias de gerenciamento ágil de projeto.

Para avaliar o modelo proposto, o mesmo foi aplicado a um estudo de caso a fim de compará-lo com o modelo cascata. Resultados indicam que o modelo proposto pode possibilitar o desenvolvimento de um projeto em um tempo reduzido, como ocorre com metodologias ágeis. Além disso, deve-se prezar por uma documentação formal de requisitos, como é o foco nos modelos tradicionais.

1.1 PROBLEMA

Características rigorosas do modelo Cascata, aliadas ao foco na documentação do projeto, fazem com que o desenvolvimento de projetos demande um tempo maior para sua conclusão e além disso a forma sequencial do projeto contribui para a perda de foco do desenvolvedor (PRESSMAN, 2011).

Como este modelo necessita de uma fase de coleta de requisitos bem defina, isso pode acarretar problemas no andamento do projeto. Segundo Sommerville (2011), em algumas ocasiões, o cliente não sabe de imediato o que precisa. E cada alteração ou incremento de funcionalidades, faz com que o escopo do projeto seja alterado, o que demanda tempo e esforço, além de elevar o custo final do projeto.

1.2 OBJETIVOS

Os objetivos gerais e específicos são descritos a seguir.

1.2.1 Objetivo Geral

Adaptar o Modelo Cascata (ROYCE, 1970) usando metodologias ágeis.

1.2.2 Objetivos Específicos

Os objetivos específicos são:

- Comparar aos modelos existentes na literatura juntamente com suas variações já existentes, na documentação do software desenvolvido.
- Analisar e propor adaptações no Modelo Cascata (ROYCE, 1970);
- Aplicar o modelo proposto em um estudo de caso a fim de levantar hipóteses acerca de seu uso;
- Realizar uma análise comparativa entre o modelo Cascata tradicional e o modelo proposto em relação ao estudo de caso.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está dividido em cinco capítulos. O capítulo 2 apresenta a revisão bibliográfica do trabalho, modelos de processo tradicionais e metodologia ágil de gerenciamento de projetos. O capítulo 3 descreve o Modelo de Interação Ágil proposto. O capítulo 4 aborda a avaliação do modelo proposto. Além disso, neste mesmo capítulo é realizada a comparação do modelo Cascata e do modelo de

Interação Ágil. Por fim, o capítulo 5 apresenta as conclusões do trabalho e traz propostas para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo aborda os tópicos referentes à revisão bibliográfica. A seção 2.1 discorre sobre os processos de software e a seção 2.2 descreve sobre o gerenciamento ágil de projetos.

2.1 PROCESSO DE SOFTWARE

Segundo Sommerville (2011), “um processo de software é um conjunto de atividades relacionadas que levam à produção de um produto de software. Essas atividades podem envolver o desenvolvimento de sistemas a partir do zero em uma linguagem padrão de programação”. Ou seja, são passos, práticas e etapas previamente estabelecidas para desenvolver softwares e os produtos associados, como, por exemplo, documentos de projeto, código-fonte, casos de uso e manual de uso do sistema, visando a qualidade e a produtividade do desenvolvedor do projeto.

Segundo Rezende (2005) os métodos utilizados atendem de forma adequada as necessidades do cliente ou usuário final. Por meio do planejamento do projeto se pode definir as etapas e estipular datas (cronograma). Para que o projeto seja concluído com êxito e no tempo determinado inicialmente, a metodologia deve ser adotada pela empresa e por todos os envolvidos, isto inclui uma parcela dos usuários finais do sistema.

O planejamento do desenvolvimento de sistemas deve, segundo Rezende (2005), visar os seguintes fatores: efetividade, continuidade, perenidade, segurança e transparência.

Para o desenvolvimento propriamente dito do projeto, Rezende (2005) define cinco fases. São elas:

- Estudos preliminares: uma visão geral do sistema e suas funções, coleta de requisitos, estipular os objetivos e verificação do impacto na empresa.
- Análise do Sistema atual: levantamento e coleta de requisitos funcionais, que definem a função de um sistema de software ou seus componentes. Analisar o ambiente de implementação e ver como o projeto se enquadra no lugar.

- Projeto Lógico: elaboração da proposta do projeto por completo, definir quais são os requisitos e fazer o detalhamento lógico do projeto; definir suas funcionalidades e ter uma visão na integra do sistema.
- Projeto físico: o projeto físico está voltado à execução e confecção do sistema, assim como os testes e verificação dos resultados, pode ter uma avaliação do sistema para a verificação do andamento do projeto.
- Projeto de implementação: após o término do projeto é necessário fazer um acompanhamento e, assim, prestar assistência, como treinamento e capacitação dos usuários.

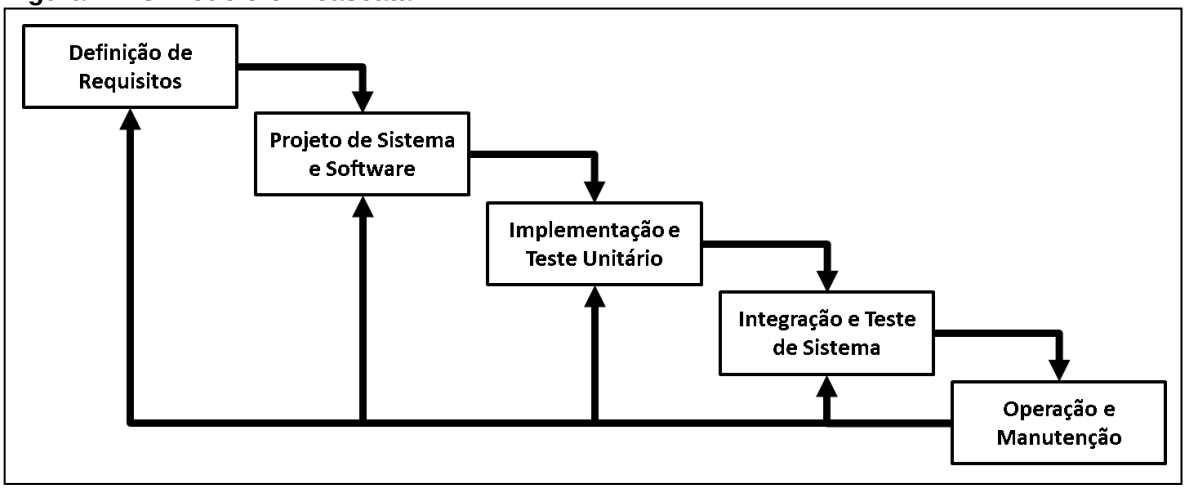
Para a aplicação destas fases é necessário uma equipe composta pelo: patrocinador, que é o cliente propriamente dito; gestor de projetos, responsável pelo comprimento do cronograma e do planejamento; equipe de usuários ou de clientes, que executam as ações que o sistema deve atender; equipe técnica, grupos de TI que devem executar toda a parte operacional do projeto; gestor técnico, que toma as decisões para a equipe de TI; coordenador de projetos, responsável pelo cronograma e coordenar as reuniões e aprovações das etapas e; especialista na área que o projeto será desenvolvido.

2.1.1 Modelo em Cascata

Conhecido como Ciclo de Vida Clássico, o Modelo Cascata foi o primeiro modelo de desenvolvimento de software a ser publicado baseado nos processos da engenharia de sistemas e seu nome foi baseado na forma sequencial de suas fases (ROYCE, 1970). Este modelo tem como ponto de partida o planejamento prévio de cada etapa, no qual é redigido um escopo do andamento de cada tarefa antes do seu início.

Segundo Sommerville (2011), o modelo cascata possui cinco fases, chamadas também de estágios: definição de requisitos; projeto de sistema e software; implementação e teste unitário; integração e teste de sistema; e operação e manutenção. O modelo é ilustrado na Figura 1.

Figura 1 – O modelo em cascata



Fonte: Sommerville (2011)

As fases do modelo são executadas de maneira sequencial e sistemática, ou seja, uma após a outra, sendo que uma nova fase só é iniciada após a conclusão de sua antecessora (SOMMERVILLE, 2011). Por este motivo, é essencial que exista um comprometimento inicial do processo e que o cliente seja específico em seus requisitos, pois só ao final do projeto será analisada uma mudança no processo.

Segundo Lawrence (2004), o modelo cascata é considerado como de alto nível, pois depende da compreensão dos envolvidos e a capacidade de prever situações na fase de desenvolvimento que possam colocar riscos ao projeto com prazos, tecnologias, entre outros. Além disso, o ciclo de vida clássico gera documentações detalhadas em cada fase do sistema a ser construído, tornando assim, o sistema mais transparente e possibilita um melhor controle por parte dos gestores.

Todavia, por não prover uma maneira de realizar mudanças necessárias que, por algum motivo, não foram identificadas nas fases iniciais do projeto, as mesmas ficam “estacionadas”. Esta modificação é destacada e passada com o *feedback* para as próximas fases e só é tratada quando todas as etapas forem concluídas (SOMMERVILLE, 2011).

As subseções a seguir apresentam de maneira detalhada cada fase do modelo cascata. A subseção 2.1.1.1 discorre sobre a fase de definição de requisitos. A subseção 2.1.1.2 descreve o estágio de projeto de sistema e software. A fase implementação e teste unitário é exibida da subseção 2.1.1.3. A subseção 2.1.1.4 apresenta a fase integração e teste de sistema. E, por fim, a subseção 2.1.1.5 discorre sobre a fase operação e manutenção, definidas por Sommerville (2011).

2.1.1.1 Definição de requisitos

Na fase de definição de requisitos são elicitadas todas as funcionalidades, serviços e restrições do sistema.

Em geral, é nesta etapa que todos os responsáveis pelo projeto se reúnem com os clientes e fazem a base do sistema, ou seja, é realizada o levantamento de requisitos, como por exemplo, definição do foco do sistema, como o software será usado, quais pessoas utilizarão este sistema, que tipo de interface ele terá e qual modelo de negócio ele representará. Em outras palavras, o cliente demonstra o que ele necessita e, em contrapartida, os desenvolvedores avaliam o que é viável no projeto.

2.1.1.2 Projeto de sistema e software

Na etapa de projeto de sistema e software ocorre a abstração do sistema. Todos os requisitos são agrupados e estudados para organizar o possível modelo de software que será gerado. Além disso, nesta fase são definidos os passos que o sistema irá seguir, como a definição de escopo, distribuição de tarefas, definição de cronograma e verificação das etapas para o desenvolvimento.

Uma vez definidos os requisitos, faz-se necessário realizar a previsão e o estudo das tecnologias que a nova plataforma irá necessitar. Caso contrário, pode ocorrer um erro de projeto, o que impactará na qualidade e nos custos finais do mesmo.

2.1.1.3 Implementação e teste unitário

A etapa de implementação e teste unitário é a fase em que os programadores e desenvolvedores se unem para a criação do *front-end* e *back-end* do sistema, ou seja, é a fase em que ocorre a codificação propriamente dita.

Um bom planejamento realizado na etapa anterior minimiza drasticamente os problemas encontrados na etapa de implementação e teste unitário, porém para se obter êxito é essencial ter uma equipe comprometida. As eventuais mudanças na

equipe de desenvolvimento do projeto podem causar atrasos significativos para a sequência do projeto.

No Modelo Cascata, em virtude do seu desenvolvimento sequencial, quando um erro é encontrado nesta fase, é documentado, pois não é permitido retornar às fases anteriores do projetos, sendo assim, o problema não é corrigido, e sim apenas contornado, o que faz com que nas fases seguintes o problema fique ainda mais evidente.

2.1.1.4 Integração e teste de sistema

Etapa responsável pela verificação do cumprimento dos requisitos determinados pelo cliente na fase de definição de requisitos. Após toda a conferência o sistema é entregue ao cliente.

Além disso, é nesta fase que ocorre a escrita do manual de utilização do sistema para o cliente.

2.1.1.5 Operação e manutenção

Fase posterior à implantação do sistema no ambiente real. Esta etapa possui foco nas correções de erros que são descobertos com o uso do sistema pelos usuários finais. Nesta fase também ocorre o treinamento destes usuários.

Após realizar as correções e adequações, pacotes de atualizações são disponibilizados ao cliente, estas novas versões podem conter apenas correções de *bugs* ou novas funcionalidades. Sendo a primeira chamada de manutenção corretiva, e a segunda, manutenção evolutiva.

2.1.2 Problemas do Modelo Cascata

A execução das fases do modelo cascata ocorre de maneira sequencial e sistemática. Estas características rigorosas, aliadas ao foco na documentação do projeto, fazem com que o desenvolvimento de projetos demande um tempo maior para sua conclusão e a forma sequencial do projeto contribui para a perda de foco (PRESSMAN, 2011).

Em um projeto, a interação entre as fases diminui o número de erros gerados pelas mesmas, visto que o cliente nem sempre consegue expressar de maneira satisfatória suas necessidades antes de visualizar a aplicação ou interagir com as primeiras versões do sistema (SOMMERVILLE, 2011; PRESSMAN, 2011).

É necessário realizar alterações no projeto em meio ao seu desenvolvimento. Infelizmente as alterações não são permitidas no modelo cascata, uma vez que os requisitos devem estar bem definidos já na fase inicial, chamada de definição de requisitos.

O ciclo de vida clássico define que os requisitos que eventualmente precisem sofrer alterações, só poderão ser atualizados ao término de todas as fases do modelo. Sendo assim, o sistema só receberá alterações para solucionar requisitos funcionais mal compreendidos pela equipe de analistas no início do projeto, após estar finalizado (SOMMERVILLE, 2011).

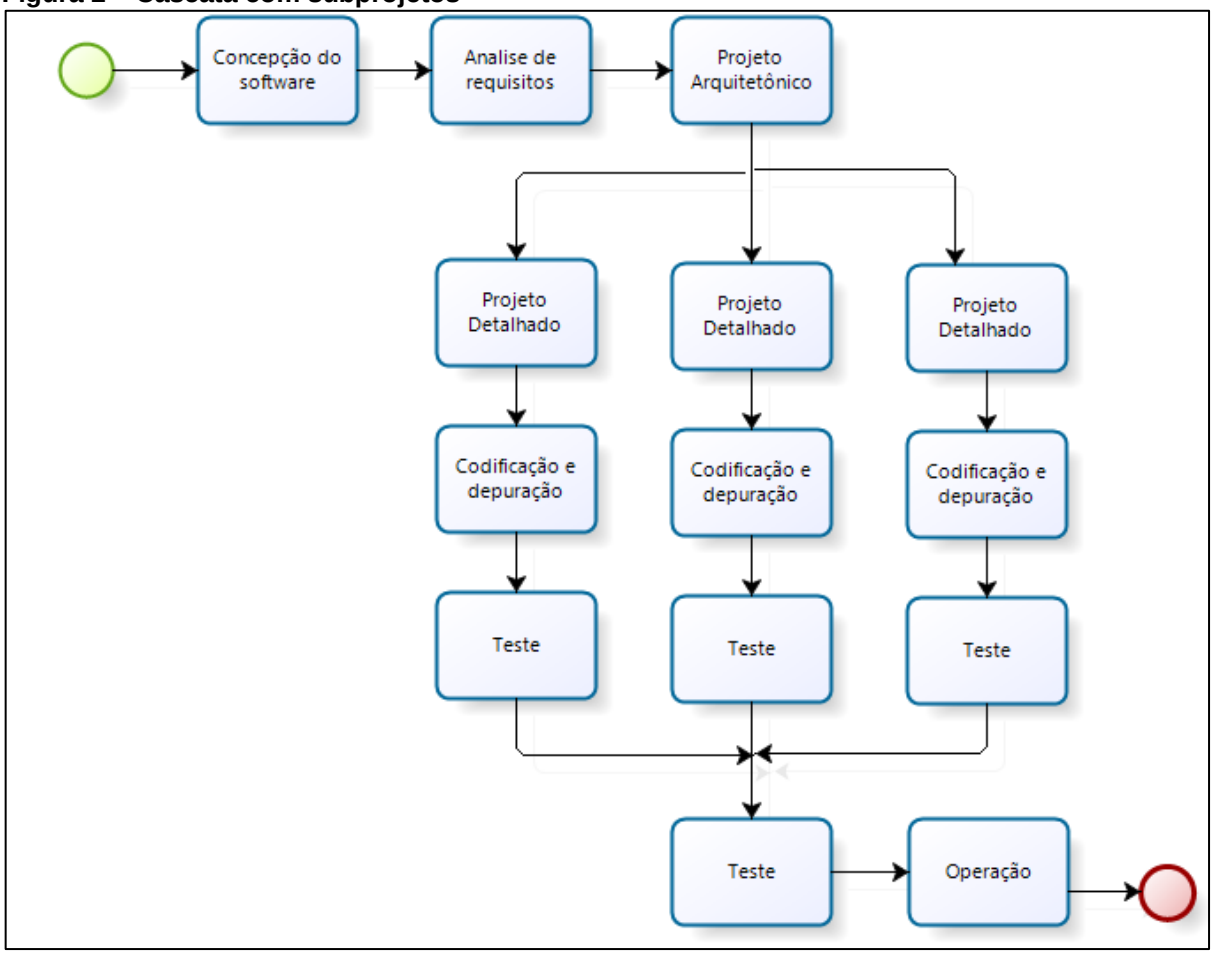
2.1.3 Cascata com Subprojetos

Neste modelo é permitido que na fase de projetos exista a divisão de tarefas para que as mesmas sejam executadas paralelamente. Deste modo, ocorre a modularização do sistema para que sejam entregues à medida que sejam desenvolvidas.

Visto que o projeto é dividido em subprojetos, o mesmo propicia um gerenciamento mais fácil, desenvolvimentos rápidos, além de permitir ao usuário visualizar o progresso mais facilmente, pois partes funcionais do sistema são entregues de acordo com que são desenvolvidas (WAZLAWICK, 2013).

Os problemas do modelo cascata com subprojetos ocorrem quando há interdependências imprevistas entre os subsistemas, quando há inconsistências que prejudiquem o projeto em uma versão final ou na integração final de todos os subsistemas. Segundo Wazlawick (2013), os principais problemas deste modelo derivam de uma arquitetura falha seguida de um gerenciamento inadequado. A Figura 2 apresenta o modelo cascata com subprojetos

Figura 2 – Cascata com subprojetos



Fonte: Wazlawick (2013)

Baseado também em desenvolvimento *top-down* a única diferença deste modelo para o modelo cascata padrão é a segmentação da fase de projetos em uma quantidade de etapas significativas.

A fase de projetos pode ser segmentada em várias partes, as quais tendem a ser executadas em paralelo. Cada segmentação se transforma em um micro projeto.

Após a finalização de cada miniprojeto, existe a fase para unir todos os módulos que foram implementados de forma separada. Em outras palavras, é a adaptação de todos os seguimentos.

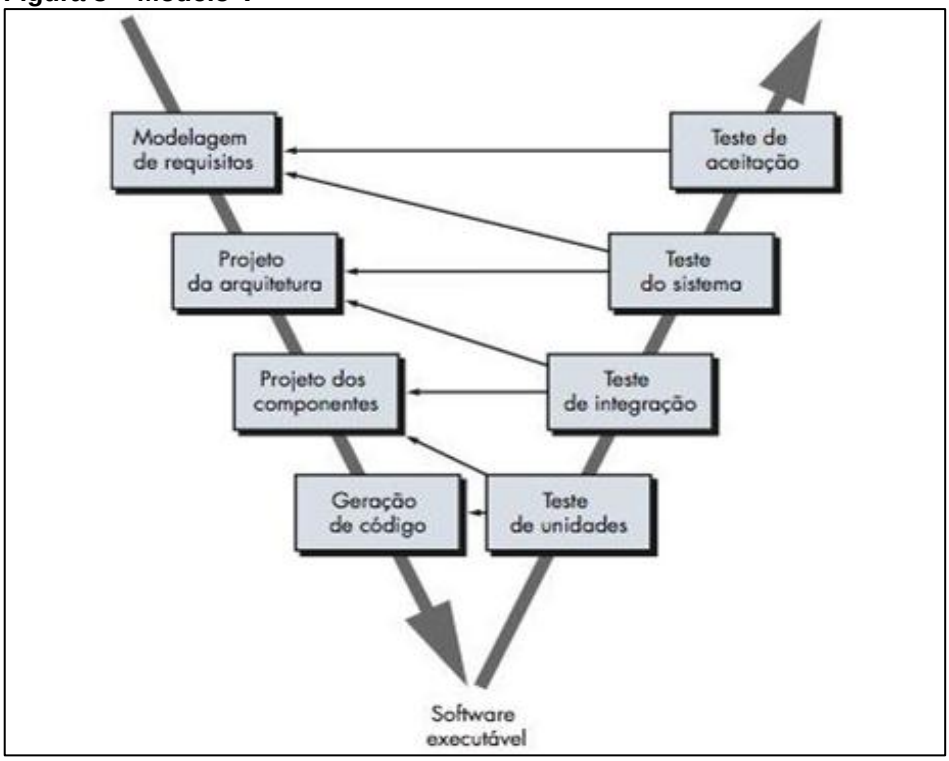
2.1.4 Modelo V

O Modelo V é uma adaptação do modelo cascata, porém ele determina uma fase de verificação e validação para cada etapa de construção. Pelo fato deste modelo

ser sequencial, como no modelo cascata, há a necessidade dos requisitos serem estáveis e o domínio reconhecido (LENZ; MOELLER, 2004).

A Figura 3 ilustra o Modelo V, este é formado pelas fases: modelagem de requisitos, projeto da arquitetura, projeto dos componentes, geração de código, teste de unidade, teste de integração, teste o sistema e teste de aceitação (PRESSMAN, 2011).

Figura 3 – Modelo V



Fonte: Pressman (2011)

Segundo Lenz e Moeller (2004), o desenvolvimento utilizando o Modelo V, inicia na fase de modelagem de requisitos. Nesta são determinados os requisitos funcionais da aplicação. A próxima fase é o projeto da arquitetura em que é escolhida a arquitetura do software que será implementada. Esta arquitetura considera os requisitos coletados na primeira etapa. A terceira e quarta fases projetam os componentes do sistema e se inicia a implementação das funcionalidades.

Ao fim da quarta etapa, iniciam-se as fases de testes. O primeiro teste a ser realizado é o de unidade, que visa encontrar problemas por módulo do sistema. Caso isso ocorra, pode-se voltar às fases de projeto dos componentes ou a geração de código.

Seguindo ao próximo teste, teste de integração, são realizadas avaliações das interfaces entre os componentes. Uma vez encontrado algum problema, pode-se retornar às fases de projeto dos componentes e projeto da arquitetura.

O teste do sistema verifica o comportamento do software em funcionamento. É focado nos requisitos funcionais e não funcionais da aplicação. Neste teste, caso identificado algum problema, pode-se retornar às fases de projeto da arquitetura e modelagem de requisitos.

O último teste, teste de aceitação, é realizado pelo cliente. Nesta etapa foca-se na satisfação das necessidades do usuário e, caso não satisfaça, deve-se retornar à fase de modelagem de requisitos.

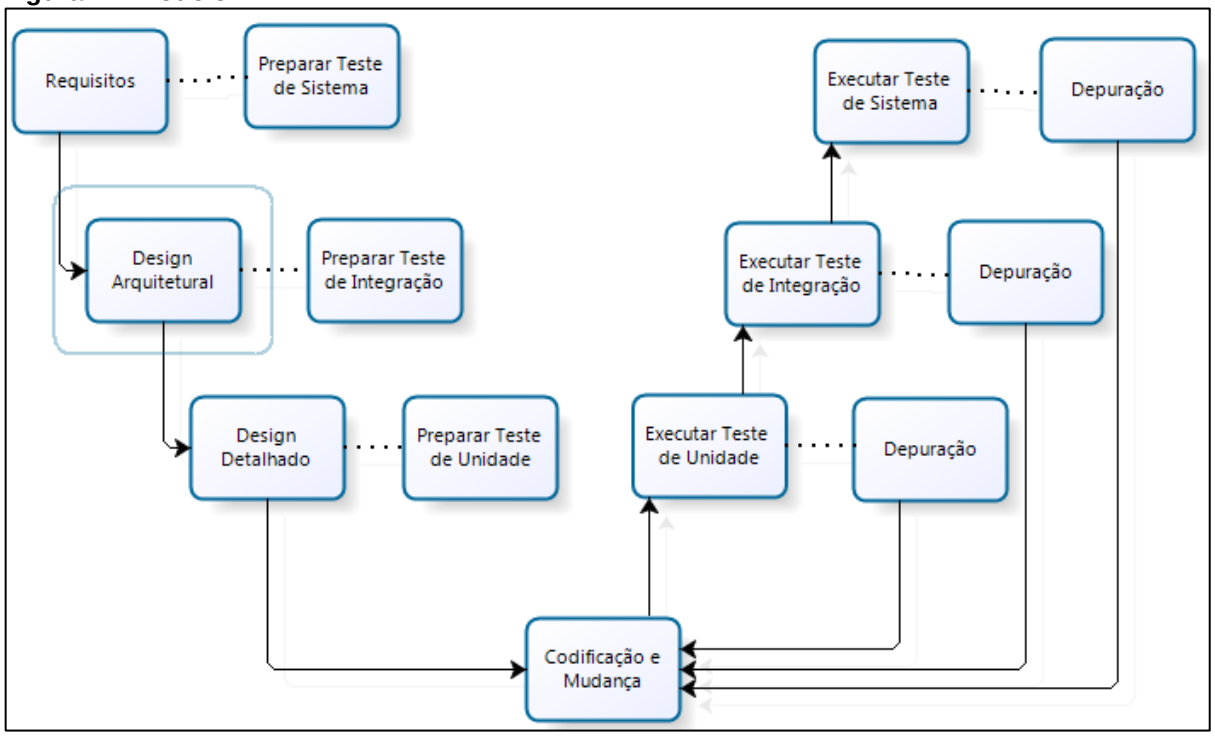
2.1.5 Modelo W

Spillner (2002) notou que no modelo V há uma divisão rigorosa entre as atividades de construção do software e as atividades de testes. A primeira se localiza ao lado esquerdo do modelo como mostrado na figura - 3, enquanto a segunda ao lado direito. Com esta análise, verificou-se a necessidade de: (i) realizar o planejamento dos testes ainda nas fases construtivas e (ii) considerar a questão de reconstrução do sistema nas fases de testes.

Neste contexto, como uma melhoria do modelo V, surgiu o modelo W. Nele, as atividades construtivas encontram-se externamente, enquanto as atividades de teste, internamente. Segundo Spillner (2002), o modelo W tem como intuito associar um teste a cada fase para que facilite o descobrimento de erros e a simplificação de arquiteturas complexas ainda no início do projeto só são realizadas com o envolvimento de responsáveis pelos testes ainda nas primeiras fases do projeto. A Figura 4 ilustra o modelo W.

Realizar o teste nos requisitos, é a primeira questão a ser considerada na fase de Requisitos. Funcionalidades testáveis são aceitas ao final desta etapa. Na fase *Design* Arquitetural, os testes servem para verificar se a estrutura desenhada (*homepage*, *index*, formulários, entre outros) é simples, caso contrário é necessário refatorá-la. Os testes relacionados a fase *Design* Detalhado também possui o intuito de verificar a simplicidade das unidades, pois devem ser coesas o que garante um nível maior de testabilidade.

Figura 4 – Modelo W



Fonte: Spillner (2002)

2.2 GERENCIAMENTO ÁGIL DE PROJETOS

Segundo Sommerville (2011), gerenciamento ágil de projeto é um conjunto de princípios de desenvolvimento, os quais tendem a facilitar a criação de sistemas.

Sua filosofia defende a entrega incremental e satisfação do cliente. Neste ponto o contato com o cliente tem uma maior frequência. E como existe fragmentação do software para ser um entrega incremental, a parte de análise não é tão forte neste processo.

2.2.1 Scrum

Scrum é um método ágil de desenvolvimento de software e foi inicialmente projetado e criado na década de 90 por Jeff Sutherland e sua equipe (PRESSMAN, 2011).

Esta metodologia é utilizada para orientar as etapas de desenvolvimento dentro de um processo que, segundo Pressman (2011), incorpora as seguintes atividades estruturais: requisitos, análise, projeto, evolução e entrega. Além disso, possui seu foco no planejamento e a revisão dos artefatos produzidos.

A base do Scrum está no conceito de transparência, inspeção e adaptação, o que o torna um modelo incremental, que gera previsão de suas fases e exibe um controle de risco (GUERREIRO, 2017). Das etapas citadas acima:

- Transparência: possui o foco na comunicação para esclarecer e definir os objetivos. Isso faz com que diminuam os retrabalhos ou falhas dos produtos. Cada etapa é relatada e nada é omitido, isso gera uma base de planejamento com dados sólidos além de facilitar a criação e segmentação das tarefas. Neste ponto se faz necessária uma linguagem comum a todos os membros, pois é por ela que se baseia-se o entendimento do projeto.
- Inspeção: define que o usuário deve fazer inspeções no processo para ajudar a identificar situações que possam ser requeridas no futuro. Porém, deve ser delimitado, pois variáveis designadas pelo usuário podem acarretar em um atraso no projeto devido às inúmeras alterações no escopo da aplicação.
- Adaptação: verifica os desvios do que foi planejamento inicialmente. Neste momento, o supervisor deve se posicionar sobre as variações. Reuniões com os membros envolvidos auxiliam na verificação do desvio das tarefas dentro do planejamento.

Na descrição de Guerreiro (2017), a metodologia Scrum é formada por perfis, eventos e artefatos, sendo como exemplos de perfis: *Product Owner*, *Development Team* e *Scrum Master*. Eventos: *Sprint Planning*, *Sprint*, *Daily Meeting*, *Sprint Review Meeting* e *Sprint Retrospective*. E artefatos: *Product Backlog*, *Sprint Backlog* e Incremento/Entrega. Os perfis, eventos e artefatos serão descritos a seguir:

2.2.1.1 Product owner

É o dono do sistema propriamente dito, responsável por fornecer o conhecimento necessário para a criação do seu software. Seu papel se define em deixar as atividades transparentes e visíveis aos integrantes do projeto, possibilitando a visualização de forma clara as metas designadas.

2.2.1.2 Development team

Equipe responsável pela implementação e cumprimento das tarefas. É a equipe de desenvolvimento que define como cada tarefa se transforma em produto. No time de desenvolvimento não existe distinção de papéis, pois todos os membros serão desenvolvedores.

É necessário que os integrantes do *development team* sejam multifuncionais, pois necessitam ter a habilidade de criação e auto supervisão para que tudo seja entregue conforme o escopo e prazos determinados.

Não existe um número ideal para o tamanho de uma equipe de desenvolvimento, porém uma equipe formada com menos de três pessoas pode gerar restrições por falta de conhecimento ou afins, e um time com cinco ou mais integrantes, pode deixar o projeto mais complexo.

2.2.1.3 Scrum master

É responsável pelo gerenciamento da metodologia Scrum. É ele quem deve fazer a definição e verificar o cumprimento das etapas. O Scrum Master faz a intermediação entre todos os envolvidos. Sendo responsável por analisar o que o cliente quer e saber o que sua equipe pode fazer. Por exemplo:

- *Scrum Master* em relação ao *Product Owner*: faz o gerenciamento de técnicas do backlog do produto. Procura manter a transparência de tudo que é informado e saber manipular tudo o que é requerido para que seja manuseado e estruturado de forma correta.
- *Scrum Master* em relação ao *Development Team*: responsável por treinar todo o time e liderar a equipe. Analisa e verifica atividades que possam impedir ou atrapalhar o desenvolvimento das tarefas, além disso, assegura que estão sendo realizadas as regras da metodologia Scrum.
- *Scrum Master* em relação ao trabalho: responsável pelo treinamento, implantação da organização de todo o sistema e analise do processo, além de possuir habilidade de replanejar, se necessário.

2.2.1.4 Product backlog

São tarefas delegadas ao *Product Owner.* O *Product Backlog* existirá enquanto o produto existir. Sua função é listar tudo que está relacionado com as funções, requisitos, melhorias e correções do sistema em relação ao que está em produção.

2.2.1.5 Backlog sprint

O backlog da *Sprint* define as tarefas de cada desenvolvedor para realizar a entrega dos itens ou produtos solicitados. Este item é importante, pois determina a separação de cada tarefa e tempo estimado para o término dos projetos.

2.2.1.6 Sprint planning

A equipe prioriza os elementos do *Product Backlog*, ou seja, seleciona por prioridade as histórias que serão implementadas em uma *Sprint.*

2.2.1.7 Sprint

São as histórias que foram selecionadas da *Sprint Planning* para serem desenvolvidas na *Sprint.* O ciclo de desenvolvimento da *Sprint* é de 2 a 4 semanas de duração sobre o qual se estrutura o Scrum.

2.2.1.8 Daily meeting

Reunião diária realizada com o intuito de sincronizar o *Scrum Team*, ou seja, deixar todos os integrantes informados do que está acontecendo e como estão os avanços de cada um. Geralmente esta reunião é realizada com todos os integrantes em pé, para que não dure mais do que 15 minutos. As perguntas que devem ser respondidas nesta reunião são:

- O que você realizou desde a última Scrum?
- Você está tendo alguma dificuldade?
- O que você irá fazer até a próxima reunião?

2.2.1.9 Sprint review meeting

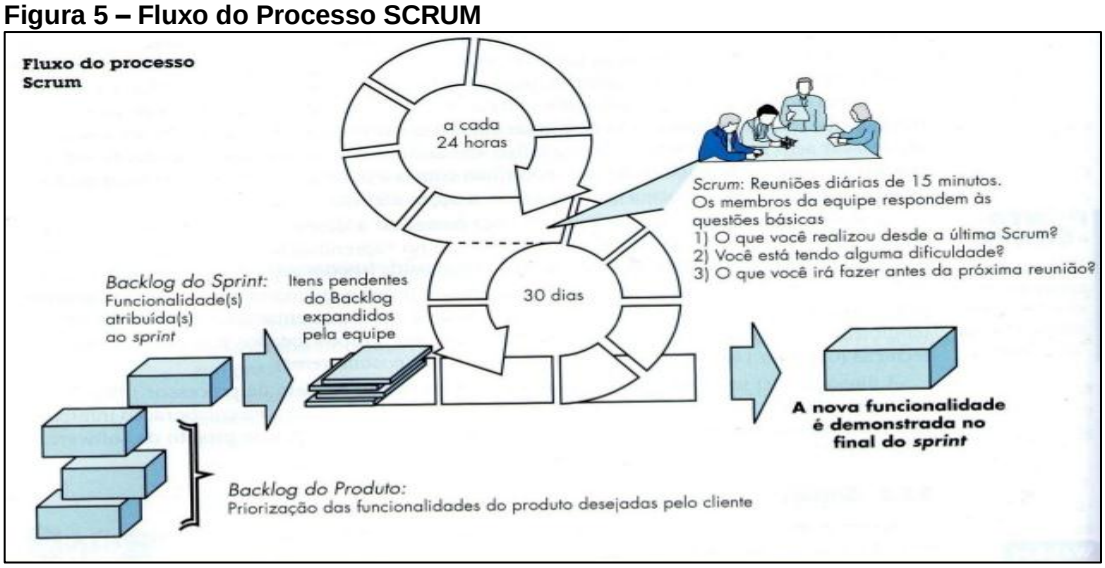
É uma reunião realizada ao final de cada *sprint* que visa avaliar o projeto em relação aos objetivos de tudo que foi planejado no *Sprint Planning Meeting*. *Product Owner*, *Development Team* e *Scrum Master*, Gerência, Cliente e Engenheiros participam desta reunião.

2.2.1.10 Sprint retrospective

Semelhante *a sprint review meeting*, ocorre no final de cada *sprint*, porém sua responsabilidade é avaliar os processos de trabalho utilizado no decorrer da *sprint.*

A Figura 5 ilustra o fluxo de processo da metodologia ágil de desenvolvimento Scrum. Nesta figura, há o *backlog* do produto que são as funcionalidades do sistema organizadas prioritariamente. A seleção dessas histórias para implementá-las é identificada pelo *backlog* da *sprint*. A *sprint,* que é o evento em que são codificadas as funcionalidades, neste exemplo, dura 30 dias, e, em cada dia, uma reunião, *Daily Meeting*, é realizada entre os integrantes do *Scrum Team.* Ao fim de cada *sprint* é realizada duas reuniões*: sprint review meeting* e *Sprint Retrospective*.

Figura 5 – Fluxo do Processo SCRUM



Fonte: PRESSMAN (2011)

3 PROPOSTA

A adequação no modelo ocorreu pela definição de Interações entre as etapas do modelo. Cada etapa pode ser redirecionada para uma fase correspondente, para que a solução do problema encontrado seja melhor trabalhada e resolvida. Pode-se citar o exemplo de um erro de *layout* encontrado na fase de projeto de sistema e software. No modelo tradicional, este erro só será resolvido no final de todo o processo. No modelo de interação ágil, após a descoberta deste erro, este poderia ser redirecionado para uma fase correspondente que o trataria de maneira adequada.

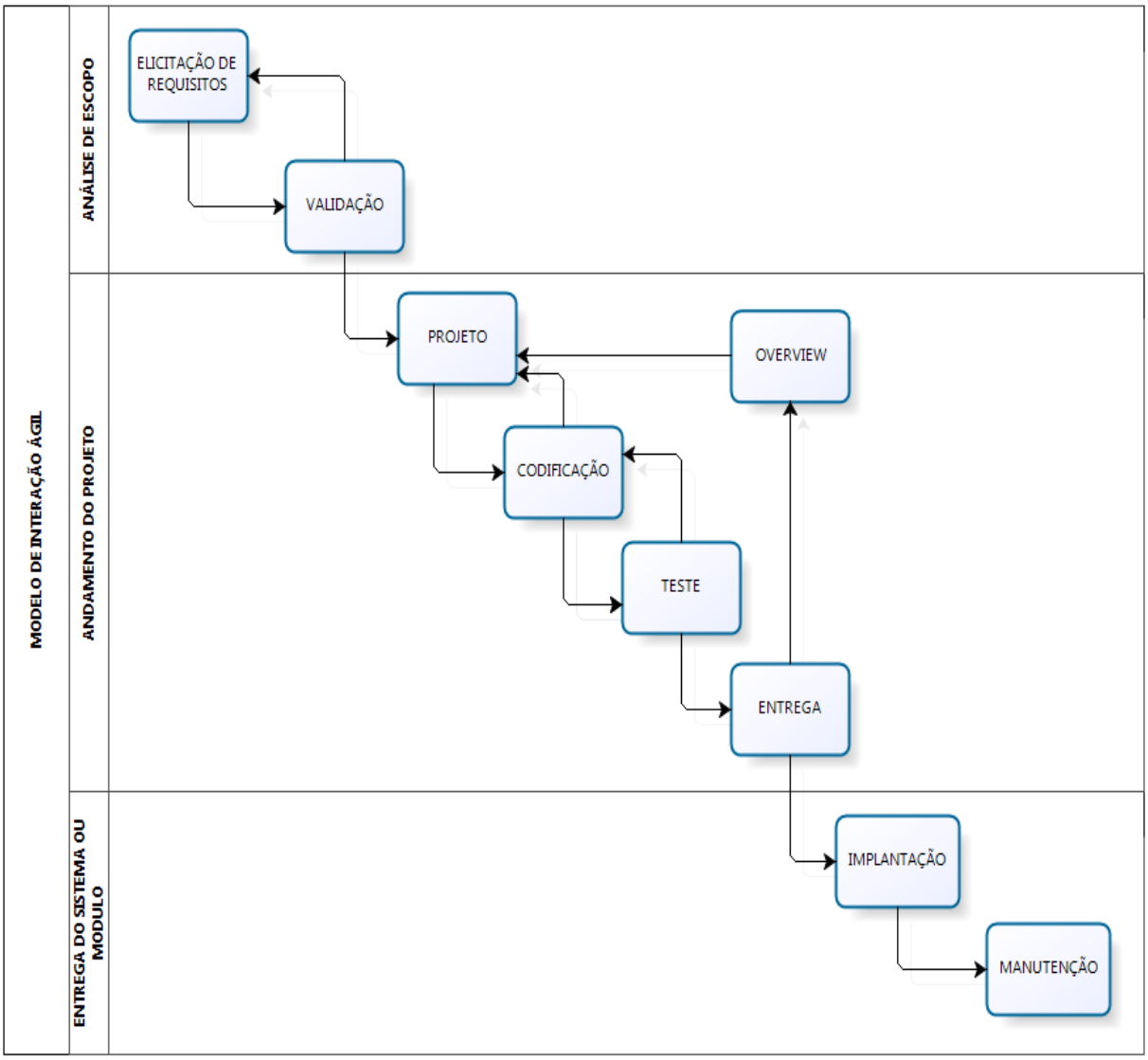
Neste contexto, o modelo Cascata, proposto por Royce (1970), recebe uma alteração no fluxo, além de ser definidas novas etapas de validações antes de permutar entre as fases do modelo. A alteração do modelo proporciona realizar o retorno a fase de projeto e identificar o erro para realizar um estudo em quais locais ele pode impactar, para que as demais falhas não venham a ocorrer.

Com o intuito de desenvolver um sistema de uma maneira incremental e ágil, inseriu-se a metodologia Scrum para o gerenciamento do desenvolvimento. O uso da mesma possibilita fragmentar a estrutura do projeto em partes menores e funcionais para facilitar a correção de erros de projeto. Diferente do modelo Cascata com subprojetos que divide todo o projeto em módulos e estes são executados em paralelo, o modelo de interação ágil define o desenvolvimento de módulo a módulo, sendo assim, o desenvolvimento e entrega ocorrem de maneira incremental. A Figura 6 ilustra o Modelo de Interação Ágil.

O modelo de interação ágil é formado por três etapas subdivididas em nove fases. A primeira etapa, nomeada de Análise de Escopo, é formada pelas fases de Elicitação de Requisitos e Validação. Seu foco é na engenharia de requisitos, ou seja, sua principal responsabilidade é coletar informações e entender a regra de negócio do projeto em questão, bem como realizar as validações dos itens coletados.

A etapa de Andamento de Projeto possui as fases: Projeto, Codificação, Teste, Entrega e *Overview.* O objetivo desta etapa concentra-se no desenvolvimento do produto propriamente dito, desta forma a definição da arquitetura do sistema, documentação, implementação e testes necessários. A execução destas etapas ocorre de maneira iterativa, ou seja, o produto é desenvolvido de forma incremental e entregue em versões ao cliente

Figure 6 – Modelo de Interação Ágil



Fonte: Autoria própria

Por fim, a terceira e última etapa, entrega do sistema ou módulo, é formada pelas fases: implantação e manutenção. Nesta etapa ocorre a entrega/implantação do sistema ou módulo – versões ainda incompletas – e são realizadas manutenções no mesmo.

As seções a seguir detalham cada fase da proposta. A seção 3.1 descreve a fase de elicitação de requisitos. A validação é apresentada na seção 3.2. A seção 3.3 discorre sobre a fase de projeto. A fase de codificação é apresentada na seção 3.4. A seção 3.5 detalha a fase de teste. A fase de entrega é descrita na seção 3.6. A seção 3.7 discorre sobre a fase *overview.* A fase de implantação do sistema é apresentada na seção 3.8. Por fim, a seção 3.9 detalha a fase de manutenção.

3.1 ELICITAÇÃO DE REQUISITOS

A Elicitação de requisitos é a primeira fase do Modelo de Interação Ágil e está localizada no setor de Análise de Escopo. Nesta fase ocorre a coleta de informações por meio de reuniões e conversas com os gestores da área, chamados *Product Owner*, pois são eles quem conhecem os detalhes, evidências, atividades e funções que o sistema deve conter..

Para que a análise de requisitos seja realizada de maneira satisfatória e não haja ambiguidades nas funcionalidades do sistema, deve haver um alinhamento entre as partes envolvidas no projeto, são elas: *Scrum Master, Product Owner* e *Development Team*.

Como objeto de saída desta fase, tem-se um conjunto de informações sobre a necessidade do cliente. Este conjunto de dados permite ao gestor do projeto criar o protótipo do sistema e documentações necessárias para a validação do projeto. Ambos serão importantes para validação dos requisitos funcionais da aplicação, que será realizado na fase seguinte.

3.2 VALIDAÇÃO

A fase de Validação tem como objetivo validar os requisitos funcionais identificados na fase anterior. Como saída desta fase, uma documentação formal de aceitação do projeto é elaborada contendo a descrição dos requisitos validados pelo *Product Owner*, bem como o planejamento do projeto.

Este documento visa facilitar e impedir que o *Product Owner* solicite alterações durante o desenvolvimento do projeto. Visto que após a validação o cliente está ciente do produto que será entregue.

3.3 PROJETO

A fase de Projeto é a primeira fase da etapa“Andamento do Projeto”. Nesta etapa a metodologia ágil de gerenciamento Scrum (SCHWABER; SUTHERLAND, 2017) atua com maior frequência, pois é neste momento que ocorre a fragmentação de todo o escopo do projeto.

Cada fragmento identificado torna-se um módulo do sistema que será entregue de forma incremental ao cliente, ou seja, em pequenas versões, chamados *sprints*. Por isso, é necessário criar cada tarefa que será implementada no *sprint*, isso inclui a descrição da tarefa, prazo máximo e definição das ferramentas que auxiliam nessa atividade.

Após elaborar todas as tarefas, as mesmas são definidas e alocadas de forma coerente a cada *Sprint*. Em cada tarefa existem vários tópicos, as quais serão desenvolvidas considerando suas prioridades.

A fase de Projeto é considerada uma das etapas mais importantes do modelo, pois é ela que delega tarefas que serão desenvolvidas.

3.4 CODIFICAÇÃO

Na fase de codificação, como no Scrum, o desenvolvedor é responsável pelo projeto e há apenas uma restrição, o *Development Team* não pode realizar alterações no escopo do projeto, pois assume a função do *Scrum Master*.

Em cada *Sprint*, o *Development Team* inicia a tarefa, a qual é uma das ações que foram planejadas na fase de projeto. Após iniciá-la, cada hora trabalhada precisa ser mencionada assim como cada ação tomada necessita ser incorporada na descrição das tarefas. Ao final do dia ou ao término desta tarefa, é necessário informar o que foi feito, o que falta fazer, quanto tempo foi utilizado e quanto tempo ainda há de trabalho para finalizar a tarefa em questão.

Dentro de um *Sprint,* a manipulação das horas pode ser realoca, pois como o planejamento é trabalhado com base em previsões, este está sugestível a erros. É normal que este procedimento acabe tendo algum erro. Sendo assim, dentro do planejamento do *Sprint* as horas alocadas em uma tarefa podem ser realocadas para outra, dentro do espaço total estipulado para a entrega da versão do sistema.

Essa fase é importante para o *Scrum Master*, pois pode-se criar indicadores de horas do projeto com base na análise dos *Sprints* anteriores. Até mesmo conseguir contabilizar a velocidade de cada desenvolvedor para determinada tarefa e, se necessário, alocar ou realocar mão de obra.

3.5 TESTE

A fase de teste é necessária para verificar se não existem erros ou falhas no sistema e se é necessário um retrabalho antes da entrega. Para isso, faz-se uso de testes estruturais e funcionais. O primeiro tem o intuito de verificar o código fonte, tentar encontrar erros ou falhas no fluxo de informações dentro do sistema que impeça o bom funcionamento. O segundo encarrega-se de testar e verificar se os requisitos correspondem ao que foi solicitado, além de controlar as versões do sistema.

Nesta fase tudo que foi desenvolvido na Codificação é testado e verificado como: *layout* do sistema, usabilidade, funcionalidades, gramática, desempenho, segurança, eficiência, confiabilidade, portabilidade, entre outros.

Como essa é uma fase do setor de projeto, define-se a tarefa de testes como um *sprint* que deve ser feito antes de realizar a entrega da nova versão ao cliente, caso seja identificado algum problema, deve ser encaminhada ao retrabalho, ou seja, retornará a fase de codificação para que o *Development Team* solucione o erro encontrado.

3.6 ENTREGA

Nesta fase ocorre uma reunião com o *Product Owner* para efetuar a entrega do módulo desenvolvido no *sprint*. Caso a versão seja validada, ocorre a implantação do mesmo. Caso contrário, o fluxo leva à fase *Overview* para solucionar o requisito identificado erroneamente e adaptar a atual e futuras *sprints*, pois o retrabalho afetará todo o planejamento e cronograma do projeto.

Ao fim desta fase é realizada uma reunião entre o *Development Team* e o *Scrum Master*. Nesta reunião são discutidas as tarefas desenvolvidas durante o *sprint*, bem como avaliações dos processos de trabalho utilizados, dificuldades encontradas e observações em geral. Como saída desta fase, é realizada a execução do próximo *Sprint*.

3.7 OVERVIEW

Fase de encerramento de um ciclo de desenvolvimento do projeto. Após a entrega de um módulo, a equipe deve se reunir e expor pontos fortes e fracos, aspectos que ocorreram com eficiência e dificuldades, além de observações e possíveis ajustes do projeto. Esta fase se assemelha a *sprint review meeting* (subseção 2.2.1.9).

3.8 IMPLANTAÇÃO

Uma vez que o *Product Owner* validou as funcionalidades implementadas no *sprint*, a fase de Implantação, primeira do setor de “Entrega do Sistema ou Módulo”, determina como esta nova versão do sistema será retirado do ambiente de desenvolvimento para ser instalado no ambiente real de utilização.

Com o sistema em utilização pelo cliente, é possível identificar erros e falhas que não foram identificadas no ambiente de desenvolvimento, bem como obter dados do próprio usuário do sistema, como por exemplo, facilidade de utilização, curva de aprendizagem, entre outros.

3.9 MANUTENÇÃO

Após ocorrer a entrega do produto em sua totalidade, inicia-se a fase de manutenção. O foco desta é prestar assistência ao produto após sua entrega. Caso seja identificada correção, adaptações ou melhorias, é nesta fase que são incorporadas. Além das manutenções corretivas, são realizadas manutenções evolutivas no software, visto que há necessidade de mantê-lo em constante evolução, prolongando ao máximo seu tempo de vida.

4 AVALIAÇÃO DO MODELO

Este capítulo aborda os resultados obtidos do estudo de caso de um sistema para coleta e processamento de dados para criação de um software de gerenciamento de torno. A seção 4.1 apresenta o estudo de caso e a seção 4.2 discorre sobre a comparação do Modelo Cascata tradicional e o Modelo de Interação Ágil.

4.1 CENÁRIO PARA APLICAÇÃO

Para realizar a avaliação do Modelo de Interação Ágil, foi utilizado o estudo de caso descrito por Fortunato et al. (2011). Nesse trabalho o Modelo Cascata é aplicado para desenvolver um sistema para coleta e processamento de dados para melhorar o processo da função de torneamento. Seu principal objetivo é realizar a comunicação em tempo real, ou seja, enquanto o torno está em funcionamento o sistema envia dados via Wi-Fi para os PDAs (*Personal Digital Assistent*), e assim o operador conseguiu gerenciar todo o processo de forma ágil.

As subseções a seguir, descrevem todo o processo de desenvolvimento de software utilizando o Modelo de Interação Ágil, proposto por este Trabalho de Conclusão de Curso. A subseção 4.1.1 descreve as fases iniciais da aplicação: Análise de Requisitos e Validação. A subseção 4.1.2 discorre sobre às fases: Projeto, Codificação, Teste, Entrega e *Overview*. Por fim, a subseção 4.1.3 apresenta às fases finais do modelo: Implantação e Manutenção.

4.1.1 Análise de Requisitos e Validação

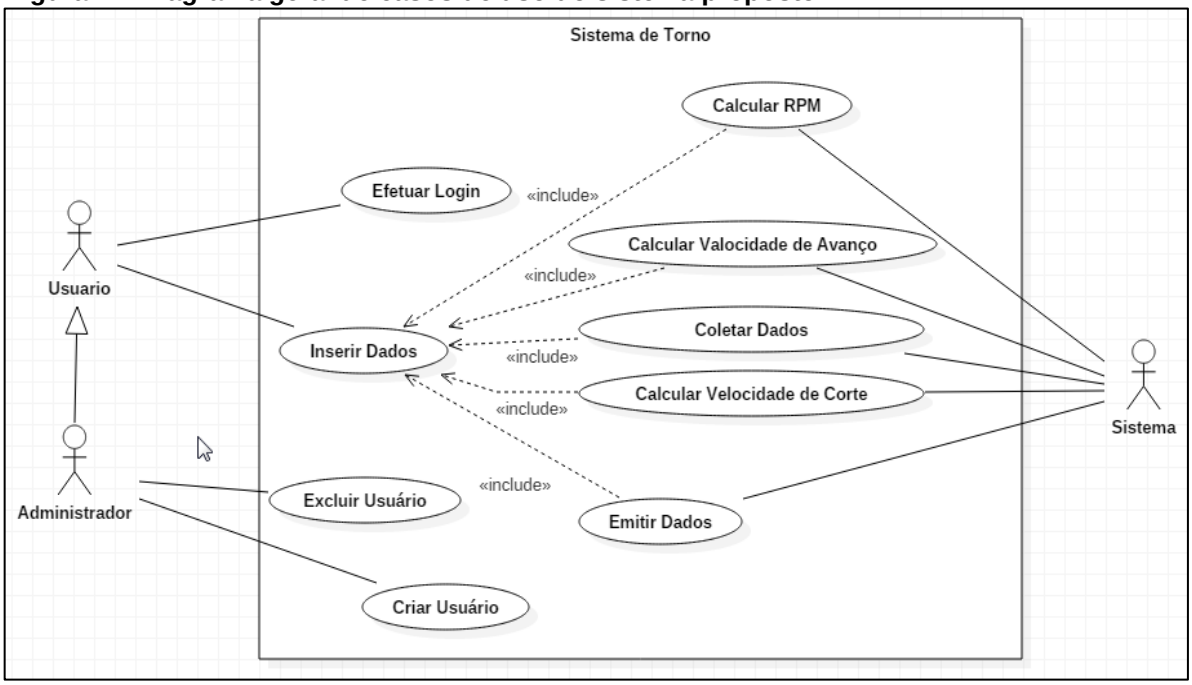
Na primeira fase do modelo, Análise de Requisitos, a coleta de dados é realizada com todos os membros envolvidos do projeto. A elicitação de requisitos visa, neste primeiro momento, compreender as funções necessárias para satisfazer os usuários do sistema. Esta fase ajuda a equipe de desenvolvimento a compreender o ponto de vista de todas as partes e a descrever e alinhar as funcionalidades pretendidas.

Considerando o trabalho de Fortunato et al. (2011), os requisitos identificados foram:

- Controle de acesso por usuário e senha, o qual o acesso só será feito por meio da autenticação dos dados da pessoa que estiver fazendo uso do software.
- Sistema precisará de um usuário administrador com todas as permissões. Estas são de inclusão e exclusão de outros usuários.
- É necessária uma tela inicial com as opções do sistema. Essas consistem em otimização, coleta de dados e emissão de relatórios.
- O menu de otimização deve permitir a regulagem da máquina com dados fornecidos pelo usuário. Além disso, deve existir as funções de regular a velocidade de corte em função das rotações por minuto.
- Também no menu de otimização deverá exibir a velocidade de avanço da máquina, pois é necessário calculá-la para etapas de acabamento e operações de desbaste.
- É necessário um menu de coleta de dados para controle das informações. Neste cenário deve constar data, velocidade de corte utilizada e quantidades produzidas.
- Um menu de relatório é necessário para obter as informações de maneira detalhada. Nesse menu os relatórios são apresentados por períodos, e devem ser definidos por uma data inicial e final. Em um primeiro momento são exibidos em tela e, posteriormente, caso seja de necessidade do usuário, poderá obter uma cópia impressa.
- Deve haver um controle na transmissão de dados. O acesso dever ser realizado por *desktops* ou sistemas com *wi-fi*.

Uma vez que os requisitos foram elicitados, a segunda fase refere-se à validação dos mesmos. Nessa etapa é convocada uma reunião com todos os responsáveis e assim, a empresa responsável pelo desenvolvimento valida o projeto com um protótipo do sistema considerando os requisitos coletados na fase anterior. Combinado com a prototipação, o diagrama de caso de uso pode ser modelado a fim de representar todos os usuários que interagirão com o sistema, funcionalidades e interações entre eles. A Figura 7 ilustra o diagrama de casos de uso proposto para o sistema.

Figura 7 – Diagrama geral de casos de uso do sistema proposto.



Fonte: Autoria própria.

Ainda na fase de validação, deve-se definir os papeis de cada equipe, segundo a metodologia de gerenciamento de projetos Scrum (GUERREIRO, 2017), o gerente de projetos é o *Scrum Master*, o cliente é o *Product Owner* e a equipe de desenvolvimento é o *Development Team*.

Após sanar os problemas de requisitos inconsistentes e/ou ambíguos, e a concordância dos clientes às funcionalidades apresentadas, a validação do modelo é finalizada, ou seja, tem-se o aceite inicial do projeto. Isso evita que o escopo do projeto sofra alterações no decorrer do mesmo, pois as partes envolvidas entraram em acordo.

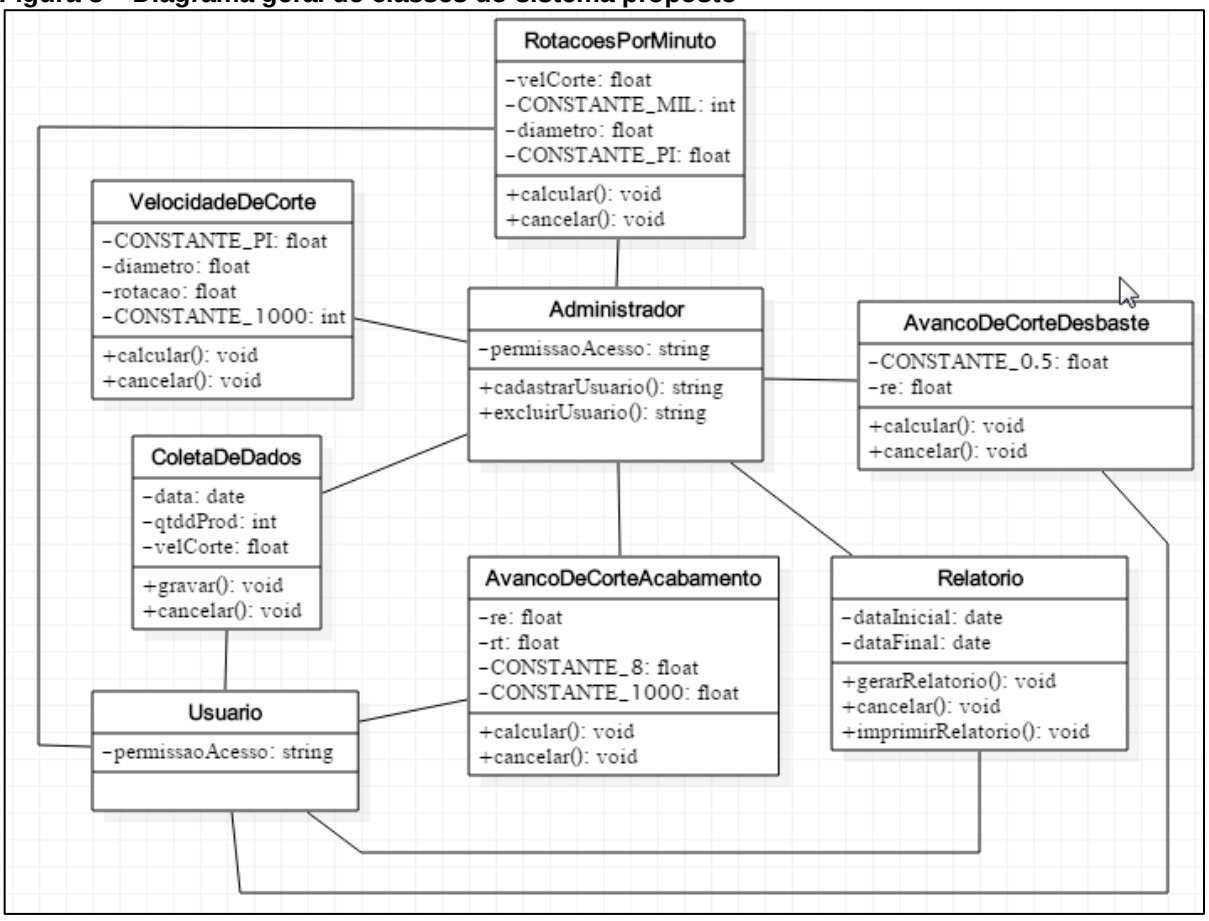
É importante salientar para o cliente que a entrega do sistema será incremental, em outras palavras, o mesmo é dividido em subsistemas na fase de projeto e estes ao serem codificados serão disponibilizados para testes ainda no ambiente de desenvolvimento. Assim que forem aprovados na fase de entrega, estes módulos serão implantados no seu ambiente real de utilização, ou seja, os usuários do sistema terão contato com a última versão do sistema.

4.1.2 Projeto, Codificação, Teste, Entrega e Overview

Na fase de Projeto é definido o escopo de projeto, o *Scrum Master* deve se reunir com o *Development Team* para definir como o sistema é modularizado e priorizado *(product backlog)*. Ainda na fase de projeto é determinada a arquitetura do sistema, linguagem de programação utilizada para a codificação, sistema de gerenciamento de banco de dados, além de discutir sobre requisitos não funcionais, como: desempenho, segurança, interoperabilidade, usabilidade, compatibilidade, flexibilidade, reusabilidade, manutenibilidade, entre outros.

Ainda na fase de projeto, pode-se gerar documentação como forma de formalizar os requisitos validados pelo cliente em fases anteriores. Como exemplo, de documentação, pode-se modelar o sistema utilizando o diagrama de classes. A Figura 8 apresenta a modelagem do sistema para coleta e processamento de dados para melhorar o processo da função de torneamento, proposto por Fortunato et al. (2011).

Figura 8 – Diagrama geral de classes do sistema proposto



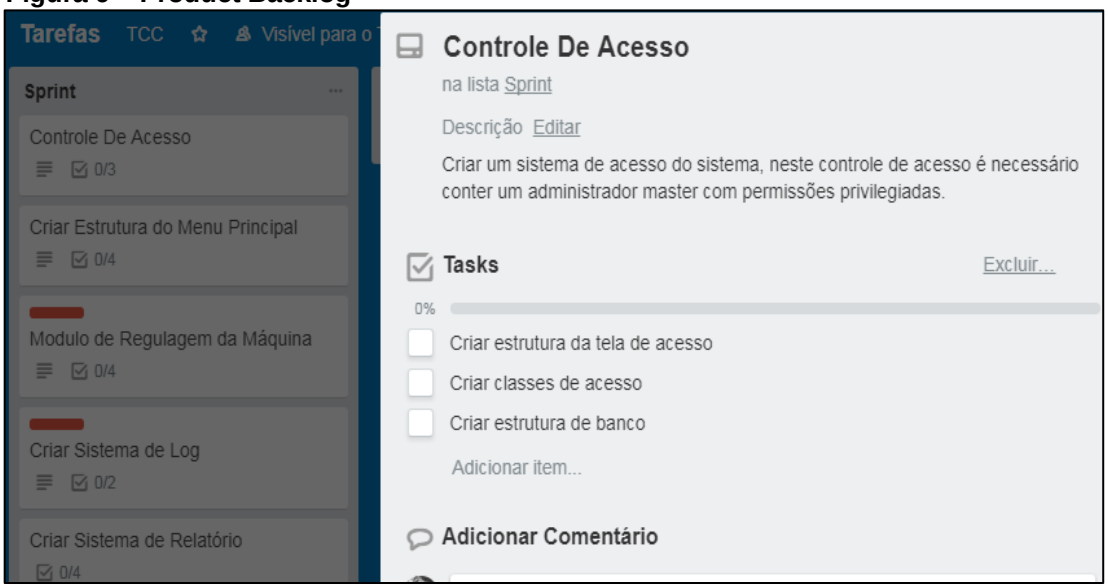
Fonte: Adaptado de Fortunato et al. (2011)

Com todas as questões definidas, um estudo de como ocorre a entrega incremental do sistema deve ser realizado. A metodologia Scrum define que deve-se implementar pequenos pacotes, para assim, agilizar ao máximo a entrega. O sistema foi dividido em subsistemas funcionais *(sprint backlog)*. E junto com esses conjuntos de soluções foi necessário criar acompanhamentos de tempos em tempos.

Como exemplo, pode ser definido que a cada duas semanas deve haver uma reunião com o cliente para efetuar a entrega de um subsistema. As atividades que ocorrem durante estas duas semanas são: verificação do planejamento, implementação do subsistema previsto, testes unitários e de integração e entrega da versão para que o cliente realize a validação da mesma.

Considerando os requisitos elicitados no trabalho de Fortunato et al. (2011), pode-se ter o seguinte *product backlog* (Figura 9).

Figura 9 – Product Backlog



Fonte: Autoria própria

Neste exemplo, as funcionalidades com maior prioridade encontram-se sinalizadas no *product backlog*, sendo assim, são codificadas as tarefas que demandam um tempo maior de desenvolvimento, pois é necessário entender a regra de negócio para execução da etapa.

Após a entrega da versão do sistema ou fechamento das tarefas, o *Scrum Master* e o *Development Team* fazem uma reunião para avaliar: (i) o projeto em relação aos objetos da *sprint (sprint planning meeting)* e (ii) os processos de trabalho utilizado na *sprint (sprint retrospective).* Ambas as reuniões fazem parte da fase

Overview do Modelo de Interação Ágil. Em resumo, é verificado se é necessário a alocação de mais recursos para o desenvolvimento ou adaptação de algum problema encontrado pelo *Product Owner* ou se precisa fazer uma análise do andamento do projeto. Com o resultado desta reunião são gerados indicadores que são úteis em *sprints* futuras.

Após finalizar a codificação de todos os subsistemas e o projeto inteiramente integrado, testado e validado, o mesmo pode ser encerrado e assim seguir o fluxo do modelo.

4.1.3 Implantação e Manutenção

Na manutenção os princípios seguem os mesmos do modelo cascata original. Na implantação, membros do *development team* são designados a realizar a instalação da nova ferramenta para que a mesma seja instalada de maneira correta no ambiente real de utilização, bem como treinar os usuários finais para a utilização do sistema.

Na manutenção, para cada solicitação de suporte deve ser coletadas as informações para fazer o controle das ocorrências. Caso uma ocorrência seja recorrente a empresa responsável pelo software, há a necessidade de avaliar em lançar uma nova versão ou pacote de atualização do sistema.

4.2 COMPARAÇÃO ENTRE MODELOS

Nesta seção são abordados o modelo cascata tradicional (ROYCE, 1970) e o modelo de interação ágil proposto pelo presente trabalho, a fim de realizar uma comparação entre a equivalência das fases de ambos. O Quadro 1 apresenta este comparativo.

Quadro 1 – Equivalência entre as fases dos modelos Cascata e de Interação Ágil

Modelo Cascata (ROYCE, 1970)	Modelo de Interação Ágil
Definição de Requisitos	Elicitação de Requisitos
	Validação
Projeto de Sistema e Software	Projeto
Implementação e Teste Unitário	Codificação
	Teste
Integração e Teste de Sistema	
Operação e Manutenção	Entrega
	Overview
	Implantação
	Manutenção

Fonte: Autoria própria

Observa-se que nas fases iniciais do projeto, a análise e validação de requisitos ocorre em uma única fase no modelo tradicional. No modelo proposto é considerado como duas fases distintas, ou seja, a análise de requisitos ocorre em uma única fase e a validação destas funcionalidades acontece em uma outra fase.

Nas etapas seguintes, tanto o modelo cascata quanto o modelo de interação ágil, possuem uma fase específica para a definição do projeto chamadas respectivamente de: “Projeto de Sistema e Software” e “Projeto”. Porém, elas possuem distinções em suas execuções. Enquanto a primeira desenvolve o projeto para uma entrega única, a segunda busca segmentar e projetar o sistema em pequenos módulos, que serão as versões de entrega.

A fase Implementação e Teste Unitário do modelo cascata é responsável por codificar o sistema e realizar testes nos módulos do programa. Por este motivo, esta fase equivale as fases de Codificação e apenas a uma fração da fase de Teste do modelo de Interação ágil, visto que esta última contempla, além dos testes unitários, testes de integração e sistemas, que são considerados apenas na fase Integração e Teste de Sistema.

Uma vez que o sistema foi implementado e testado, no modelo cascata o mesmo é colocado em operação e, posteriormente, presta-se manutenção. Esta etapa é determinada pela fase Operação e Manutenção. Diferentemente no modelo tradicional, o modelo proposto não realiza a implantação do produto apenas em sua totalidade, e sim em versões – módulos – que são entregues ao cliente na fase de

Entrega. Se o desenvolvimento da aplicação necessitar de uma nova versão, o fluxo dirige-se à fase *Overview* para a equipe discutir a nova iteração, caso contrário, a etapa seguinte será as fases Implantação e posteriormente Manutenção, que são responsáveis pela instalação do sistema no ambiente e futuras manutenções, respectivamente.

5 CONCLUSÃO

Considerando que atualmente as empresas de desenvolvimento de software necessitam entregar um sistema de qualidade em um curto espaço de tempo ao cliente, o desenvolvimento ágil consegue se enquadrar como mais uma solução. Entretanto, a documentação do software não pode ser desprezada. Neste contexto, a adaptação do modelo tradicional de desenvolvimento de software para que o mesmo contemple aspectos de um modelo de gerenciamento ágil de projeto torna-se viável.

Este trabalho apresentou uma adaptação do modelo Cascata com a metodologia de gerenciamento Scrum. Assim, o desenvolvimento de sistemas com a metodologia proposta preza por entregas ágeis e incrementais do sistema acompanhadas pelo cliente, o que torna o trabalho de desenvolvimento dinâmico.

Ao contrário do modelo tradicional, em que o projeto é desenvolvido de maneira sequencial, sistemático e a entrega ocorre apenas na versão íntegra do sistema, o modelo proposto mantém as vantagens do conceito Cascata, como por exemplo, elicitação de requisitos, projeto e documentação, entre outros, e adiciona os benefícios das metodologias ágeis, como por exemplo, entrega incremental e ágil, colaboração com os clientes, flexibilidade a mudanças, constante cooperação entre equipe de trabalho e negócio, entre outros.

Além disso, nota-se que na identificação de um erro durante o desenvolvimento do projeto, este é prontamente corrigido para que não impacte em versões futuras do sistema, o que dificultaria ainda mais sua correção aumentando os custos e tempo de entrega do projeto. Este mesmo erro descoberto em um projeto que esteja utilizando o modelo tradicional, o mesmo é documentado e corrigido apenas futuramente.

Na avaliação do modelo, utilizou-se o estudo de caso de um projeto para desenvolvimento de um sistema para coleta e processamento de dados para melhorar o processo da função de torneamento (FORTUNATO et al., 2011). Neste cenário, o modelo se fez confiável, pois gerou uma documentação formal de especificação de requisitos, quanto para a validação das funcionalidades do sistema com o cliente, quando para auxiliar nas codificações das mesmas, utilizadas pelo *Development team*.

5.1 TRABALHOS FUTUROS

A seguir são apresentadas algumas sugestões de trabalhos futuros:

- Aplicar o modelo de interação ágil em um desenvolvimento real de software. Assim será possível coletar informações como: (i) tempo de aplicação do projeto; (ii) adaptação das fases mediante a análise na aplicação em um projeto real; e (iii) quantidade de interações que a fase de projeto será submetida.
- Analisar a viabilidade e adaptar o modelo para que o fluxo, após a fase de manutenção, retorne à fase de projeto ou inicie um novo sistema.
- Realizar a integração entre o modelo proposto por este trabalho ao modelo de Computação Socialmente Consciente, proposto por Baranauskas (2009).

REFERÊNCIAS

BARANAUSKAS, M.C.C. **Socially Aware Computing. In**: Proceedings of the VI International Conference on Engineering and Computer Education (ICECE 2009). Buenos Aires, pp. 1-4, 2009.

BOEHM, B. W. **A spiral model of software development and enhancement.** Computer, v. 21, n. 5, p. 61-72, 1988.

BORGES, C. **Consequências de um processo estruturado de resposta a incidentes preocupam.** Disponível em: <http://cio.com.br/opiniao/2017/08/29/consequencias-de-um-processo-estruturado-de-resposta-a-incidentes-preocupam/>. Acesso em: 14 de set. 2017.

FORTUNATO, F. P. S. F., CUTOVOI, L.T.M., COPPINI N. L., SALLES, J. A. A., BAPTISTA, E. A**. Aplicaçao da metodologia do desenvolvimento em cascata no projeto de um sistema de coleta e otimização de dados de torneamento via rede wi-fi**. Disponível em: <http://www.abepro.org.br/biblioteca/enegep2011_tn_sto_142_899_17766.pdf> Acesso em: 02 de set. 2017.

GERRARD, P. **MODELO W** . Disponível em: <https://anielacole.wordpress.com/2011/02/10/modelo-w/>. Acesso em: 02 de ago. 2017.

GUERREIRO, C. **18 Ferramentas gratuitas para gerenciamento ágil de projetos.** Disponível em< http://blog.tecnologiaqueinteressa.com/2013/10/gerenciamento-agil-de-projetos.html>. Acesso em: 26 de mai. 2017.

LAWRENCE, P. S**. Engenharia de Software:** Teoria e Pratica**.** 2ª ed, São Paulo: Pearson Education, 2004

LEE, M. **A Pivotal anuncia financiamento Series C para abastecer a expansão continuada e atender à crescente demanda dos clientes**. Exame. Disponível em: <http://exame.abril.com.br/negocios/dino/noticias/a-pivotal-anuncia-financiamento-series-c-para-abastecer-a-expansao-continuada-e-atender-a-crescente-demanda-dos-clientes.shtml>. Acesso em: 16 mai. 2016.

LENZ, G., MOELLER, T. **Net**: A Complete Development Cycle**.** Pearson Education Inc., 2004.

PRESSMAN, R. S. **Engenharia de software:** uma abordagem profissional. 7ª Edição. Ed: McGraw Hill, 2011.

REZENDE, D. A. **Engenharia de software e sistemas de informação**. Brasport, 2005.

ROYCE, W. **Managing the development of large software systems:** Concepts and techniques. In: Proc. IEEE WESCOM. IEEE Computer Society Press, Los Alamitos. 1970.

SCHWABER, K. ; SUTHERLAND, J. **Guia do Scrum**. Disponível em <https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acesso em: 26 mai. 2017.

SOMMERVILLE, I. **Engenharia de software**. 9. ed. São Paulo, SP: Pearson Prentice Hall, 2011.

SPILLNER, A.; BREMENN, H. **The W-Model Strengthening the Bond Between Development and Test.** Int. Conf. on Software Testing, Analysis and Review. 2002.

WAZLAWICK, R**. Engenharia de software**: conceitos e práticas. Elsevier Brasil, 2013.