CENTRO ESTADUAL DE EDUCAÇÃO TECNOLOGICA PAULA SOUZA

FACULDADE DE TECNOLOGIA DE SÃO PAULO

LUCAS EDUARDO ORMOND DE SOUSA

DESENVOLVIMENTO DE SOFTWARE PARA GESTÃO DE PROJETOS EM SCRUM

SÃO PAULO/SP

2022

LUCAS EDUARDO ORMOND DE SOUSA

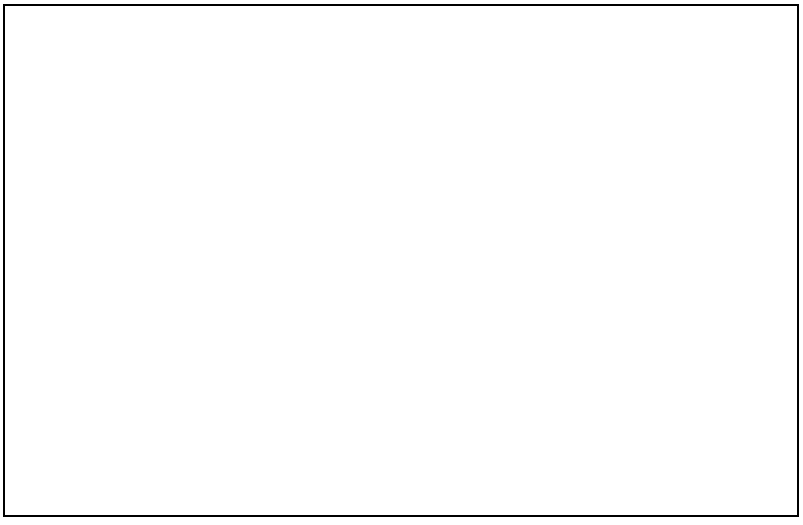
DESENVOLVIMENTO DE SOFTWARE PARA GESTÃO DE PROJETOS EM SCRUM

Monografia apresentada a FATEC – São Paulo como parte dos requisitos para obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Dr. Aristides Novelli Filho.

SÃO PAULO/SP

2022



LUCAS EDUARDO ORMOND DE SOUSA

DESENVOLVIMENTO DE SOFTWARE PARA GESTÃO DE PROJETOS EM SCRUM

Monografia apresentada a FATEC – São Paulo como parte dos requisitos para obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Prof. Dr. Aristides Novelli Filho

FATEC – São Paulo

Dedico este trabalho à minha família, que sempre me apoiou e esteve ao meu lado nos momentos mais difíceis, incentivando e renovando as minhas energias.

AGRADECIMENTOS

À Deus, por me guiar durante a minha jornada.

À minha família por todo o incentivo e apoio incondicional durante os meus estudos.

Aos meus amigos por todo o incentivo, e pela contribuição direta ou indireta à minha graduação.

Ao Prof. Dr. Aristides por ter aceitado ser meu orientador, e por todo o apoio durante o processo de elaboração do TCC.

À minha empresa atual, pela oportunidade de experiência prática no ciclo de desenvolvimento de software.

À Prof. M. Edmea pelas sugestões.

À instituição FATEC – SP juntamente com o CPS pela oportunidade de aprendizado e estudos.

A todos os outros professores da FATEC – SP pela experiência e conhecimento compartilhado durante o processo.

“Sentir é criar. Sentir é pensar sem ideias, e por isso sentir é compreender, visto que o Universo não tem ideias.”

(Fernando Pessoa, 1966)

RESUMO

Diferente do modelo em cascata, a metodologia ágil Scrum se baseia em diversos ciclos de atividades. Em cada ciclo, o projeto pode passar por alterações e ser aperfeiçoado de acordo com a interpretação obtida referente ao contexto e situação atual do projeto. O propósito deste trabalho foi analisar conceitos da engenharia de software juntamente com conceitos da metodologia ágil Scrum, para então iniciar o desenvolvimento de um sistema que permita auxiliar e gerir projetos que adotam essa metodologia em seu dia a dia. Foram abordadas as etapas necessárias que um software deve passar para ter êxito em seu processo de desenvolvimento, como especificação dos requisitos funcionais e não-funcionais do sistema, modelagem de software (diagramas de caso de uso, diagramas de classe, diagramas de evento, diagramas de atividades e diagramas de sequência), dentre outros tópicos importantes. Também foram abordados os tópicos principais da Scrum como os papéis da equipe (Product Owner e Scrum Master), Product Backlog, Sprint, dentre outras coisas.

Palavras-chave: Engenharia de software; metodologia; Scrum; desenvolvimento; projeto.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

UML - Unified Modeling Language

RUP - Rational Unified Process

SUMÁRIO

1 INTRODUÇÃO

O software está presente em diversas áreas, e por esse motivo sua existência é de extrema importância no mundo moderno. Grande parte de infraestruturas, serviços, indústrias, manufatura, dentre outras coisas, já funcionam de forma totalmente informatizada. Devido a esse fator, a engenharia de software se mostra fundamental para o funcionamento da sociedade (Sommerville, 2011).

Há diversos tipos de sistemas de softwares, e cada um possui a sua própria particularidade. Por esse motivo, apesar dessas aplicações precisarem de engenharia de software, não existem métodos ou técnicas universais a serem utilizadas. Portanto, é necessário observar a necessidade de cada sistema para abordá-lo de forma adequada (Id., 2011).

É importante ressaltar que a engenharia de software engloba diversos processos, não se limitando somente a desenvolver um simples programa de computador. Segundo Sommerville (2011, p. 3) “[...] quando falamos de engenharia de software, não se trata apenas do programa em si, mas de toda a documentação associada e dados de configurações necessários para fazer esse programa operar corretamente.”.

Tendo em vista esses conceitos e a importância da engenharia de software, ao desenvolver um sistema que consiga auxiliar na gestão de projetos em Scrum, ou qualquer outro sistema, é necessário levar em conta toda a estrutura e modelagem necessária para que ele consiga funcionar de maneira correta e consiga atender todas as expectativas.

1.1 Objetivo Geral

Aplicar de forma prática todos os conhecimentos pesquisados, para modelar e estruturar um software que permita configurar e auxiliar projetos que adotam a metodologia ágil Scrum.

1.2 Objetivos Específicos

Para que o software seja modelado e estruturado de forma correta, é necessário entender os conceitos da engenharia de software, bem como

entender como a metodologia ágil Scrum funciona. Assim, foi levantado os seguintes objetivos específicos:

- Estudar sobre a engenharia de software;
- Estudar sobre a modelagem de software;
- Estudar sobre a metodologia ágil Scrum;
- Modelar um software que permita auxiliar na gestão de projetos em Scrum.

1.3 Justificativa

A proposta na presente pesquisa vai servir para aprofundar os conhecimentos pesquisados referentes a engenharia de software juntamente com a metodologia Scrum. Há ainda a aplicação prática desses conhecimentos, no qual será modelado um software capaz de auxiliar na gestão de projetos em Scrum, permitindo assim uma visão desses conceitos que vai além do estudo da parte teórica.

2 ENGENHARIA DE SOFTWARE

Este capítulo apresenta detalhes sobre os fundamentos da engenharia de software. Serão abordados tópicos sobre modelagem de software, envolvendo diagrama de classes, diagrama de caso de uso, diagrama de atividades, diagrama de sequência e diagrama de eventos. Além disso, o capítulo também irá contemplar conceitos como especificação de requisitos funcionais e não-funcionais, ciclo de vida de software, dentre outros assuntos.

2.1 Definição

Segundo Stephen Schach (2009, p. 4) “[...] a engenharia de software é uma disciplina cujo objetivo é produzir software isento de falhas, entregue dentro do prazo e orçamento previstos, e que atenda às necessidades do cliente. Além disso, o software deve ser fácil de ser modificado quando as necessidades do usuário mudarem”.

“Engenharia de Software trata da aplicação de abordagens sistemáticas, disciplinadas e quantificáveis para desenvolver, operar, manter e evoluir software. Ou seja, Engenharia de Software é a área da Computação que se preocupa em propor e aplicar princípios de engenharia na construção de software.” (Valente, 2020, p. 2).

“Engenharia de software é uma disciplina de engenharia cujo foco está em todos os aspectos da produção de software, desde os estágios iniciais da especificação do sistema até sua manutenção, quando o sistema já está sendo usado” (Sommerville, 2011, p. 5).

Diante das citações acima, conclui-se que a engenharia de software é uma disciplina extremamente ampla e que aborda diversos temas referente ao desenvolvimento de sistemas. Existe uma preocupação desde as etapas iniciais até a sua manutenção, sendo que o software deve cumprir com o que foi especificado, e estar livre de falhas que possam o comprometer. Em suma, a engenharia de software utiliza-se de técnicas e abordagens que permitem que o produto final, no caso o software, consiga atender às necessidades do cliente, considerando funcionalidade, prazo e orçamento previsto.

2.2 Ciclo de vida do software

O modelo de ciclo de vida do software[1] é uma descrição das etapas que devem ser seguidas ao desenvolver um sistema. Por ser mais fácil realizar uma sequência de tarefas menores do que uma grande tarefa, o ciclo de vida é subdividido em uma série de etapas menores que são denominadas fases. Dependendo do modelo de ciclo de vida a ser adotado, o número de fases pode variar para mais ou para menos (Schach, 2009).

Diante de tais conceitos, será possível observar que a codificação é feita somente após uma análise detalhada do escopo do software e requisitos do cliente. Todo o projeto é dividido em fases, e cada fase do projeto possui a sua própria responsabilidade. As fases podem sofrer variações dependendo do modelo a ser adotado, alterando a quantidade de fases e o que é feito em cada uma delas.

2.2.1 Modelo cascata

Foi o primeiro modelo de ciclo de vida de software a ser publicado. No modelo em cascata, as fases são executadas em sequência. O estágio seguinte não deve ser iniciado até que a fase anterior seja concluída. Nesse modelo, as iterações podem ser dispendiosas e envolver retrabalho. Por isso, somente é recomendável seu uso quando os requisitos são bem compreendidos e que não venham a ser alterados no futuro (Sommerville, 2011). É baseado nas seguintes etapas:

- Análise e definição dos requisitos. Obtido através da consulta aos usuários;
- Projeto de sistema e software, no qual os requisitos são alocados tanto para hardware quanto para software;
- Implementação e teste unitário, no qual o software é desenvolvido em unidades de programa, e cada unidade é testada para que os requisitos sejam atendidos;
- Integração e teste de sistema, onde as unidades individuais são integradas e testadas como um sistema completo;

[1] Nota do autor: No tópico 2.4 será abordado a modelagem de software com o RUP, e no tópico 2.5 será apresentada a UML juntamente com os seus diagramas.

- Operação e manutenção. O sistema é instalado colocado em uso e sofre manutenções.

A FIGURA 1 apresenta as principais fases e fluxo do funcionamento do modelo em cascata.

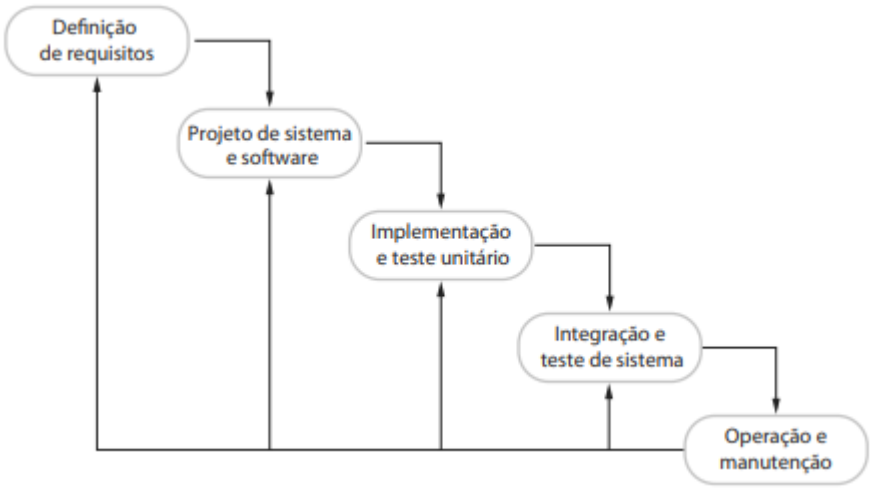


FIGURA 1 – Modelo em cascata.
Fonte: Sommerville, 2011, pág. 20.

Conclui-se que o modelo em cascata é um modelo extremamente linear, e isso acaba dificultando eventuais alterações nos requisitos que foram inicialmente estabelecidos. Portanto, caso esse modelo seja adotado pela equipe de desenvolvimento, é necessário definir da forma mais completa e correta possível os requisitos de software para evitar problemas.

2.2.2 Modelo de prototipagem rápida

É baseado na entrega de um protótipo, ou seja, é baseado na entrega de um modelo operacional que consiste em uma pequena parte funcional do software. Essa parte funcional não faz validações dos dados de entrada ou atualizações na base de dados, ela atua somente como um exemplo de funcionamento do produto final (Schach, 2009).

A FIGURA 2 apresenta as principais fases e fluxo do funcionamento do modelo de prototipagem rápida.

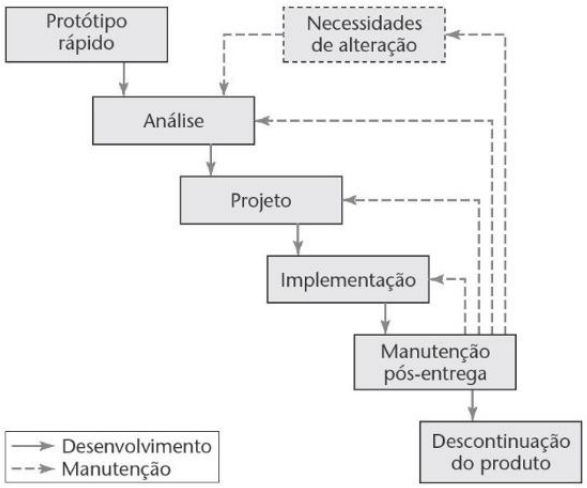


FIGURA 2 – Modelo de prototipagem rápida.
Fonte: Stephen Schach, 2009, pág. 53.

A primeira etapa é criar um protótipo de forma rápida e deixar que o cliente interaja e avalie esse protótipo. Após a avaliação do cliente, os desenvolvedores podem ter a certeza de que o sistema irá atender todas as necessidades do contratante (Schach, 2009).

Devido as interações e validações postas pelo usuário, existe uma grande chance do documento de especificações resultantes estar correto. Isso contribui para que as possíveis iterações não sejam mais necessárias. O fato de uma versão preliminar do software estar funcionando corretamente, acaba reduzindo as chances de uma correção durante ou após a implementação (Schach, 2009).

Conclui-se que o modelo de prototipagem rápida tem como pilar principal a entrega de uma versão minimalista do software para então começar todo o processo de desenvolvimento de fato. É feito de tal forma pois é mais fácil alinhar o que está sendo desenvolvido com as expectativas que o usuário possui, reduzindo a chance de eventual retrabalho.

2.2.3 Modelo espiral

O modelo espiral se baseia em diversas iterações, no qual cada iteração versões mais completas do software são construídas. Esse modelo considera os riscos e a avaliação do cliente em cada iteração. Isso ajuda a mitigar grandes

problemas que podem surgir durante a execução do projeto (Pressman, 1995). É definido pelas seguintes quatro atividades:

- Planejamento, onde ocorre a determinação de objetivos, restrições e alternativas;
- Análise dos riscos, no qual ocorre a identificação dos riscos ou resolução dos riscos;
- Engenharia, onde ocorre o desenvolvimento do produto no “nível seguinte”;
- Avaliação feita pelo cliente, onde os resultados da engenharia são avaliados.

A FIGURA 3 apresenta as principais fases e fluxo do funcionamento do modelo em espiral. Nesse modelo, cada fase é representada através de um quadrante. Por ser um modelo iterativo, é comum que uma etapa seja executada mais de uma vez.

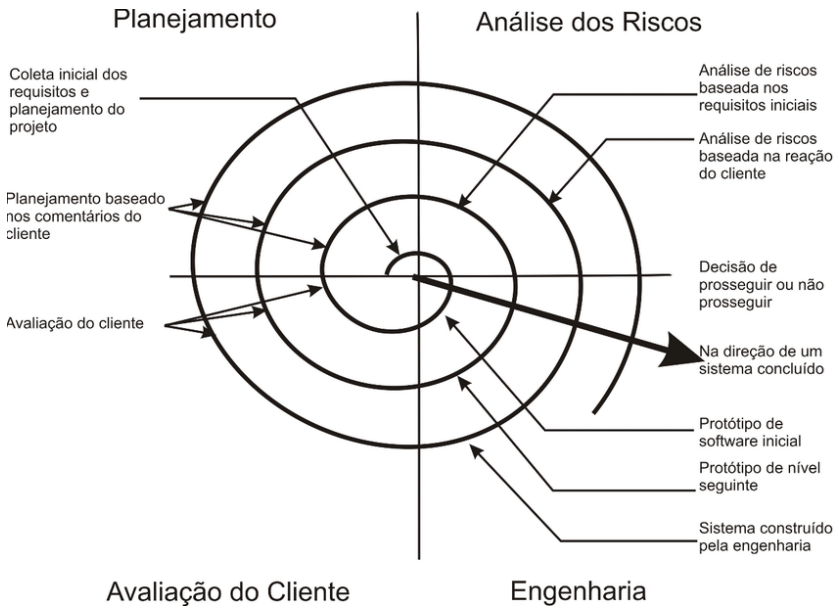


FIGURA 3 – Modelo em espiral.

Fonte: Adaptado de Pressman, 1995, pág. 39.

Pressman detalha o funcionamento do modelo em espiral da seguinte maneira:

> O cliente avalia o trabalho de engenharia (o quadrante de avaliação do cliente) e apresenta sugestões para modificações. Baseada na entrada do cliente, ocorre a fase seguinte de planejamento e análise dos riscos. Em cada arco da espiral, a conclusão da análise dos riscos resulta

> numa decisão de “prosseguir/não prosseguir”. Se os riscos forem muito grandes, o projeto pode ser encerrado (Pressman, 1995, p. 40).

Conclui-se que o modelo em espiral é um modelo em que os requisitos do software são refinados em cada iteração realizada. Em cada uma dessas iterações, é de extrema importância avaliar os riscos existentes no projeto para obter sucesso na entrega do produto final. É um modelo onde a opinião do cliente é considerada em cada ciclo, ajudando a reduzir eventuais problemas que podem surgir no processo.

2.3 Requisitos de sistema

Os requisitos de um sistema são todas as descrições das funcionalidades, dos serviços oferecidos e das restrições referente ao funcionamento do software. Os requisitos dizem a respeito sobre a necessidade e expectativas que o cliente possui relativo ao software desenvolvido. O processo de verificar essas necessidades e expectativas é chamado de engenharia de requisitos (Sommerville, 2011).

2.3.1 Requisitos funcionais

Os requisitos funcionais de um sistema servem para definir o que o sistema deve de fato fazer no que se diz a respeito as funcionalidades e serviços implementados (Valente, 2020).

Sommerville detalha a definição de requisitos funcionais da seguinte maneira:

> Os requisitos funcionais de um sistema descrevem o que ele deve fazer. Eles dependem do tipo de software a ser desenvolvido, de quem são seus possíveis usuários e da abordagem geral adotada pela organização ao escrever os requisitos. Quando expressos como requisitos de usuário, os requisitos funcionais são normalmente descritos de forma abstrata, para serem compreendidos pelos usuários do sistema. No entanto, requisitos de sistema funcionais mais específicos descrevem em detalhes as funções do sistema, suas entradas e saídas, exceções etc. (Sommerville, 2011, p. 59).

O QUADRO 1 apresenta os requisitos funcionais de um sistema utilizado para manter informações sobre os pacientes em tratamento de saúde mental.

QUADRO 1 – Requisitos funcionais de um sistema de saúde.

1	Um usuário deve ser capaz de pesquisar as listas de agendamentos para todas as clínicas.
2	O sistema deve gerar a cada dia, para cada clínica, a lista dos pacientes para as consultas daquele dia.
3	Cada membro da equipe que usa o sistema deve ser identificado apenas por seu número de oito dígitos.

Fonte: Adaptado de Sommerville, 2011, pág. 59.

Em suma, os requisitos funcionais limitam-se ao que o software deve de fato fazer, podendo ser escritos de forma abstrata ou específica, detalhando as entradas, saídas e eventuais erros que podem ocorrer. Os fatores determinantes desses requisitos levam em conta o tipo de sistema a ser desenvolvido e seus possíveis usuários.

2.3.2 Requisitos não funcionais

Os requisitos não funcionais são requisitos que não possuem ligação direta com os serviços que o software presta ao cliente. Geralmente estão relacionados a propriedades como confiabilidade, tempo de resposta, segurança, entre outros (Sommerville, 2011).

Requisitos não funcionais tendem a ser mais críticos que requisitos funcionais individuais. Sommerville explica esse fator da seguinte forma:

> Os usuários do sistema podem, geralmente, encontrar maneiras de contornar uma função do sistema que realmente não atenda a suas necessidades. No entanto, deixar de atender a um requisito não funcional pode significar a inutilização de todo o sistema. Por exemplo, se um sistema de aeronaves não cumprir seus requisitos de confiabilidade, não será certificado como um sistema seguro para operar; se um sistema de controle embutido não atender aos requisitos de desempenho, as funções de controle não funcionarão corretamente (Sommerville, 2011, pág. 60).

O surgimento desses requisitos se dá através das necessidades dos usuários, devido a restrições de orçamento, políticas organizacionais, fatores externos, entre outros (Sommerville, 2011).

A FIGURA 4 classifica os requisitos não funcionais existentes.

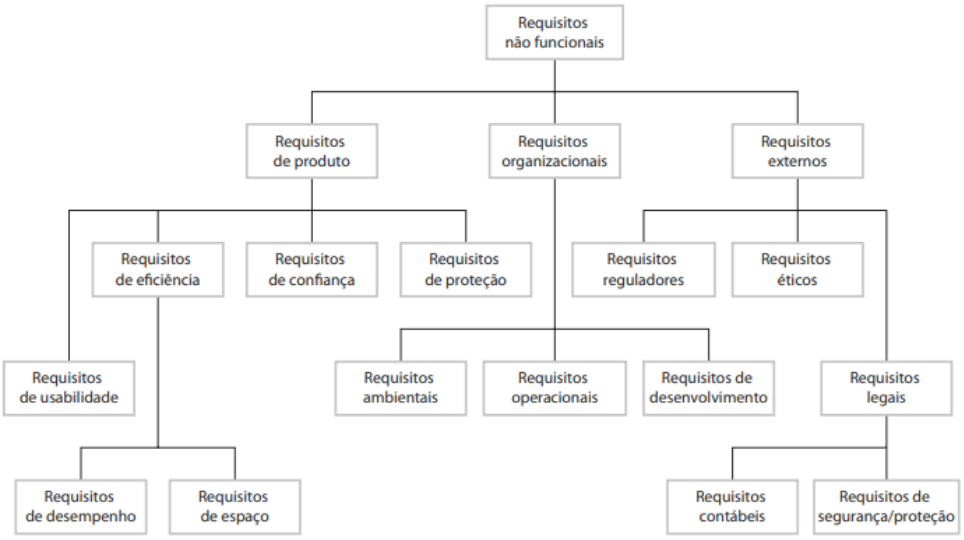


FIGURA 4 – Classificação de requisitos não funcionais.
Fonte: Sommerville, 2011, pág. 61.

Tais requisitos são definidos da seguinte maneira (Sommerville, 2011):

- Requisitos de produto são requisitos referentes ao comportamento do software, como por exemplo velocidade em requisitos de desempenho, e espaço em disco em requisitos de espaço.
- Requisitos organizacionais são requisitos decorrentes das políticas e procedimentos da organização do cliente e desenvolvedor, como por exemplo como o software deverá ser usado em requisitos operacionais, e a linguagem de programação a ser utilizada em requisitos de desenvolvimento.
- Requisitos externos são requisitos decorrentes de fatores externos ao sistema e seu processo de desenvolvimento, como por exemplo um software operante dentro da lei em requisitos legais, e o que deve ser feito para ser aceito por um regulador como o banco central em requisitos reguladores.

O QUADRO 2 apresenta os requisitos não funcionais de um sistema utilizado para manter informações sobre os pacientes em tratamento de saúde mental.

QUADRO 2 – Requisitos não funcionais de um sistema de saúde

Requisito de produto	O MHC-PMS deve estar disponível para todas as clínicas durante as horas normais de trabalho (segunda a sexta-feira, 8h30 às 17h30). Períodos de não operação dentro do horário normal de trabalho não podem exceder cinco segundos em um dia.
Requisito organizacional	usuários do sistema MHC-PMS devem se autenticar com seus cartões de identificação da autoridade da saúde.
Requisito externo	O sistema deve implementar as disposições de privacidade dos pacientes, tal como estabelecido no HStan-03-2006-priv.

Fonte: Adaptado de Sommerville, 2011, pág. 62.

Diante dos conceitos apresentados, é possível concluir que os requisitos não funcionais não estão relacionados de forma direta com o que o software deve realizar. Em suma, os requisitos não funcionais acabam se preocupando com a forma de implementação, restrições de projeto, qualidade, entre outros tópicos. Tais requisitos costumam ser mais críticos que requisitos funcionais, podendo até inviabilizar um sistema por completo.

2.4 Modelagem de software com RUP

O RUP (Rational Unified Process) é um processo de desenvolvimento de software genérico e estruturado, que pode ser adaptado e especializado para diferentes tipos de projetos, considerando fatores como área do projeto, tamanho do projeto e também as organizações envolvidas. Em suma, é um processo baseado em componentes de software interconectados com interfaces bem definidas, além de ser altamente customizável (Booch; Jacobson; Rumbaugh, 1999).

É um processo que faz uso da UML na etapa de modelagem do software, sendo responsável por ilustrar e representar os requisitos de forma gráfica. Isso faz com que a UML se torne uma parte integral e de extrema importância do RUP (Booch; Jacobson; Rumbaugh, 1999).

Tal processo de desenvolvimento de software tem como pilares fundamentais os seguintes pontos: é dirigido por casos de uso, é centralizado na arquitetura, e também é iterativo e incremental. Essas palavras-chave fazem com que o RUP seja único em seu modo de funcionamento (Booch; Jacobson; Rumbaugh, 1999). Abaixo será abordado cada um desses tópicos:

- Dirigido por casos de uso: Os casos de uso são responsáveis por capturar requisitos funcionais dos usuários do sistema em questão, e servem para analisar os resultados esperados de cada funcionalidade. Com isso, a especificação de requisitos funcionais tradicional é substituída. O termo dirigido por casos de uso significa que é através deles que o fluxo de negócio é construído.
- Centralizado na arquitetura: Antes de iniciar o desenvolvimento do sistema, é necessário considerar as diferentes visões e pontos do software a ser elaborado. A arquitetura aborda os aspectos estáticos e dinâmicos mais importantes do sistema, ela cresce com as necessidades dos usuários, e essas necessidades são refletidas em casos de uso. A arquitetura interage diretamente com os casos de uso e os dois devem ser trabalhados em paralelo.
- Iterativo e incremental: Desenvolver um software comercial é uma atividade de grande complexibilidade, e que pode demandar muito tempo para ser finalizada. Por isso, existe a prática de dividir esse grande projeto em projetos menores. Cada projeto menor se torna uma iteração, e essa iteração resulta em um pequeno incremento ao produto final. De forma complementar, cada iteração lida com grupos de casos de uso e com os riscos mais importantes. Quando uma iteração é feita com sucesso, a próxima se inicia.

O RUP é dividido em diversos ciclos que compõem toda a vida do sistema. Quando cada ciclo é finalizado, uma versão do produto pronta para a entrega é construída. Esse ciclo consiste em quatro fases, e cada uma dessas fases são divididas em iterações. A FIGURA 5 mostra como que essa divisão funciona (Booch; Jacobson; Rumbaugh, 1999):

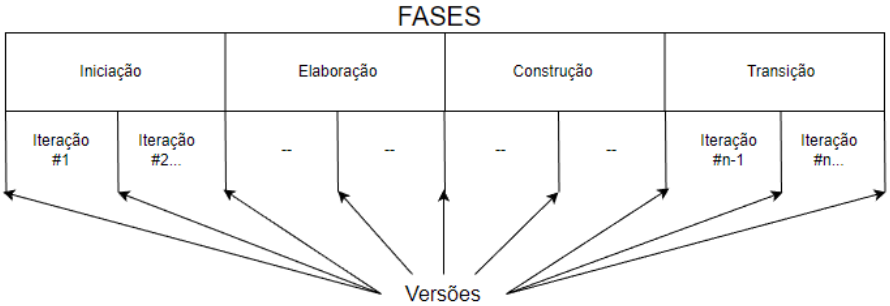


FIGURA 5 – Ciclo do RUP e suas divisões.

Fonte: Adaptado de Booch; Jacobson; Rumbaugh, 1999, pág. 9.

As quatro fases são definidas como:

- Iniciação: Essa fase tem como finalidade elaborar um caso de negócio. Deve haver um estudo dos agentes externos que utilizarão o sistema, para dessa forma definir como o sistema deve se comportar quando começar a ser utilizado por esses agentes. Esse estudo dos agentes é feito através de casos de uso críticos. Inclusive, durante essa fase os riscos são identificados e priorizados.
- Elaboração: Nessa fase, os casos de usos são especificados em detalhes e a arquitetura do sistema tem a sua projeção e base. Além disso, os casos de usos críticos descobertos na fase de iniciação são feitos. No final da fase, existe a possibilidade de planejar as atividades e estimar os recursos que são necessários para o projeto, levando também em consideração os riscos existentes.
- Construção: Nessa fase o produto começa a ser desenvolvido e a arquitetura inicialmente projetada começa a ficar mais completa. Durante essa fase a maior parte dos recursos necessários acabam sendo utilizados. No final da fase, o produto já possui as funcionalidades acordadas entre o gerente de projetos e o usuário.
- Transição: Nessa fase, o produto se torna uma versão beta, no qual os usuários irão fazer uso do sistema desenvolvido e reportar possíveis falhas existentes. Com isso, os desenvolvedores começam a trabalhar na correção desses problemas, para assim liberar uma nova versão do software. De forma complementar, essa fase também possui tarefas como o treinamento de usuários e assistência aos usuários.

Cada uma dessas fases executa um determinado tipo de trabalho durante as suas iterações, cada trabalho ocupa um certo tempo dependendo da fase e da iteração em questão, podendo sofrer grandes variações. A FIGURA 6 mostra a distribuição do tempo em cada uma dessas fases e iterações.

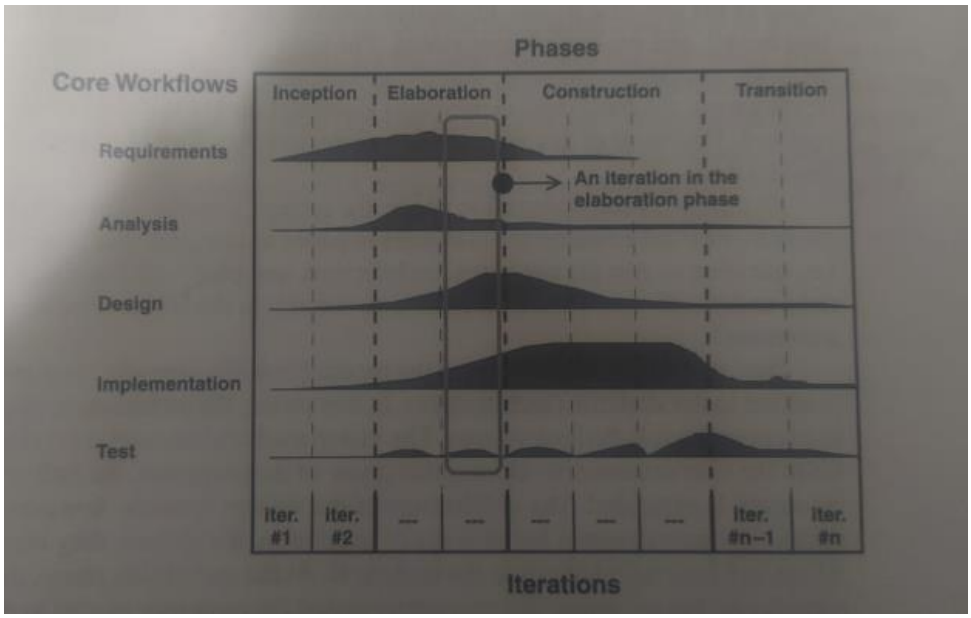


FIGURA 6 – Visão geral da distribuição do RUP.
Fonte: Booch; Jacobson; Rumbaugh, 1999, pág. 11.

No eixo horizontal da FIGURA 6, temos as fases com as suas respectivas iterações. O tempo gasto em cada iteração é distribuído em diferentes trabalhos, que estão sendo representados pelo eixo vertical. Durante a fase de construção por exemplo, o tempo gasto com implementação é extremamente superior ao tempo gasto com requerimentos. Os requerimentos ganham uma atenção maior durante a fase de iniciação e elaboração. Com isso, é possível observar que cada fase possui um foco diferente da outra.

Os fluxos de trabalho apresentados no eixo vertical da FIGURA 6 são descritos da seguinte maneira:

- Requisitos: Responsável por analisar os requisitos funcionais e não funcionais do sistema. O cliente é consultado para o alinhamento de expectativas.
- Análise: Responsável por abordar os casos de uso de forma mais detalhada para assim fazer um entendimento de como o sistema irá se comportar.
- Design: Responsável por definir a estrutura estática do software, como classes e interfaces.
- Implementação: Responsável por incluir componentes juntamente com o mapeamento de classes para componentes.

- Teste: Responsável por descrever os casos de teste que por sua vez validam os casos de uso.

Diante dos conceitos apresentados, é possível concluir que o RUP é um processo de desenvolvimento de software que engloba diferentes fases e iterações. Cada fase possui o seu foco, distribuindo mais tempo no fluxo de trabalho que vai de acordo com o objetivo da fase. O RUP trata desde a modelagem inicial até a implantação do software e treinamento aos usuários. É um modelo genérico que pode ser adaptado para a realidade da empresa que o utiliza.

2.5 UML

Criada em 1995, a UML (Unified Modeling Language) é uma linguagem que auxilia na modelagem de softwares de diferentes segmentos. Em suma, acaba sendo responsável pela visualização, especificação, construção e documentação de sistemas. Graças a UML, os desenvolvedores podem ter uma visão gráfica do que está sendo desenvolvido através de diversos diagramas e modelos (Booch; Jacobson; Rumbaugh, 1999).

A UML é uma linguagem de modelagem de objetos, e ela possui diversos recursos e formas de representações. Uma pesquisa publicada em 2007 por Erickson e Siau, mostra que a essência de um sistema está contemplada em cinco diagramas:

- Diagramas de atividades, que representam todo o fluxo de atividades desempenhadas pelo sistema;
- Diagramas de casos de uso, que representam uma determinada funcionalidade do sistema;
- Diagramas de sequência, que representam as iterações entre os componentes e usuários do sistema;
- Diagramas de classe, que representam como que as classes do sistema de relacionam entre si;
- Diagramas de estado, que mostram o comportamento e os possíveis estados que um objeto pode chegar.

2.5.1 Caso de uso

O caso de uso é uma representação de uma determinada funcionalidade que o sistema possui. Ele é responsável por retornar ao usuário algum resultado de valor, ou seja, quando o caso de uso é acionado, o software deve executar uma série de procedimentos internos afim de que o resultado esperado seja obtido (Booch; Jacobson; Rumbaugh, 1999).

Inclusive, os casos de uso capturam requisitos funcionais estabelecidos pelo cliente. Contudo, o propósito do caso de uso vai além da coleta de requisitos, pois é através dele que o design, implementação e testes do software acabam sendo conduzidos (Booch; Jacobson; Rumbaugh, 1999).

Quando há uma junção de todos os casos de uso e atores de um sistema, é obtido um modelo de caso de uso. Quando esse modelo de caso de uso é representado em partes, é obtido um diagrama de caso de uso (Booch; Jacobson; Rumbaugh, 1999). A FIGURA 7 apresenta um exemplo de um diagrama de caso de uso envolvendo um sistema de saúde.

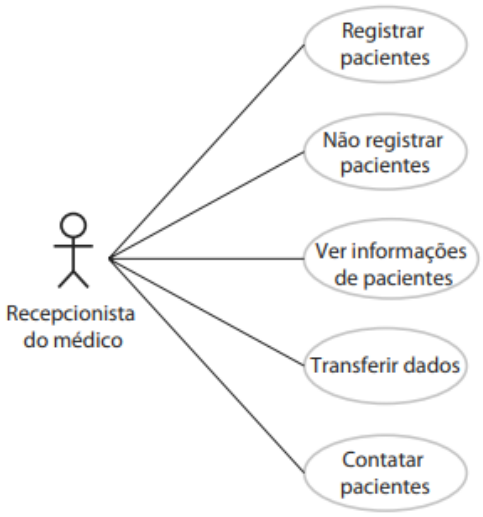


FIGURA 7 – Diagrama de caso de uso de um sistema de saúde.
Fonte: Sommerville, 2011, pág. 88.

Diante dos conceitos apresentados, é possível concluir que o caso de uso consiste em uma das diversas funcionalidades contempladas em um sistema. Além dele representar os requisitos funcionais, também conduz outras etapas importantes no desenvolvimento de software. Uma representação de caso de uso é constituída pela funcionalidade e pelo autor que a executa.

2.5.2 Diagrama de sequência

O papel do diagrama de sequência é relacionar os atores com os objetos do sistema, além das interações entre os próprios objetos. Existem diversos tipos de interações que podem ocorrer entre os itens que compõem o diagrama de sequência. A UML dispõe os recursos necessários para a elaboração desses relacionamentos (Sommerville, 2011).

Segundo Sommerville (2011, p. 87) “[...] um diagrama de sequência mostra a sequência de interações que ocorrem durante um caso de uso em particular ou em uma instância de caso de uso.”. A FIGURA 8[2] mostra um exemplo de diagrama de sequência de um sistema de saúde.

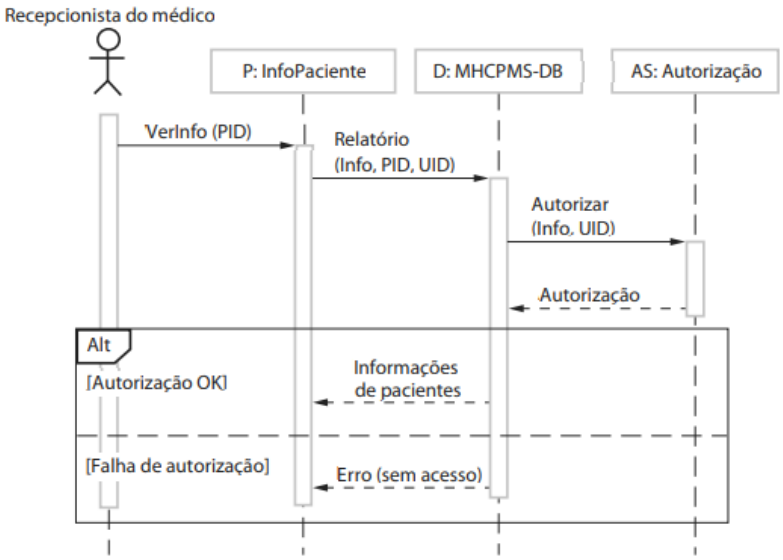


FIGURA 8 – Diagrama de sequência de um sistema de saúde.

Fonte: Sommerville, 2011, pág. 88.

Em um diagrama de sequência, os autores ficam na parte superior juntamente com uma linha tracejada na vertical. Os retângulos verticais indicam quando a instância do objeto está envolvida no fluxo. Os relacionamentos entre os autores são representados por setas horizontais. A caixa “Alt” engloba as situações condicionais que podem ocorrer durante o fluxo.

2.5.3 Diagrama de classes

Em um sistema de orientação a objetos, o diagrama de classes é responsável por mostrar as classes que constituem o sistema, e como que essas

[2] Nota do autor: O diagrama da FIGURA 8 é referente ao caso de uso “Ver informações de pacientes” apresentado na FIGURA 7.

classes se relacionam entre si. A UML permite criar diagramas de classes com diferentes níveis de detalhamento devido aos recursos disponíveis (Sommerville, 2011).

Ao desenvolver um diagrama de classe, é de suma importância identificar os objetos responsáveis por compor o sistema, para então representá-los como classes. Após a representação das classes, é possível estudar como o relacionamento entre elas funciona (Sommerville, 2011).

A FIGURA 9 apresenta um exemplo de diagrama de classes de um sistema de saúde.

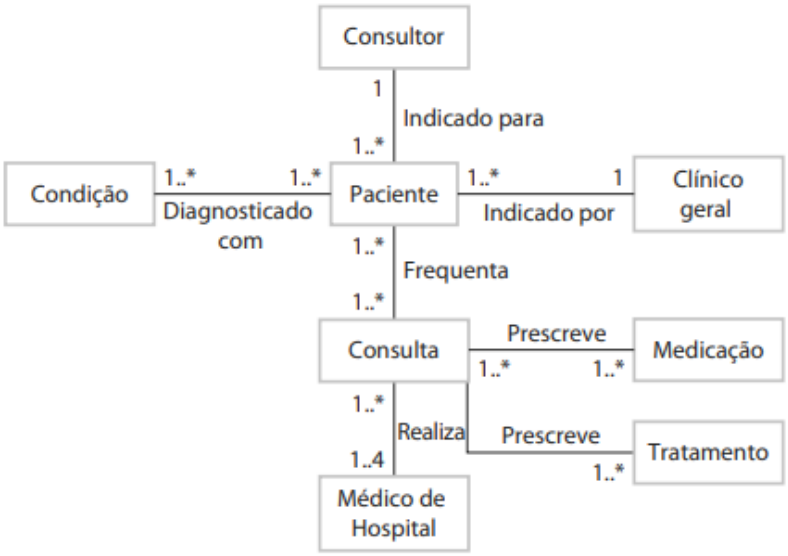


FIGURA 9 – Diagrama de classes de um sistema de saúde.
Fonte: Sommerville, 2011, pág. 91.

Na FIGURA 9, cada retângulo representa uma classe, e as linhas o relacionamento entre elas. Cada relacionamento possui uma identificação por escrito, indicando o porquê da relação existir. Nas extremidades de cada linha está indicado como a relação funciona de forma quantitativa.

Conclui-se que o diagrama de classes é uma representação dos objetos do sistema. Após a identificação dos objetos, o diagrama é construído e o relacionamento entre as classes é feito. Cada relação possui a sua particularidade. A quantidade de objetos envolvidos em cada relacionamento varia com o propósito da relação.

2.5.4 Diagrama de estados

O diagrama de estados (ou máquina de estados) tem como responsabilidade demonstrar como que o sistema se comporta referente a eventos externos e internos. É baseada na teoria na qual um sistema possui um número limitado de estados, e que os eventos existentes no software somente transferem o seu estado atual para algum outro estado (Sommerville, 2011).

Essa modelagem possui um empecilho devido ao fato de que os eventos existentes em um software podem crescer de forma extremamente rápida. Isso leva a ocultação de alguns detalhes nos modelos, que é feito através de superestados. Tais superestados englobam um determinado número de estados que podem ser expostos em outro diagrama (Sommerville, 2011).

A FIGURA 10 apresenta um exemplo de diagrama de estados de um sistema de micro-ondas.

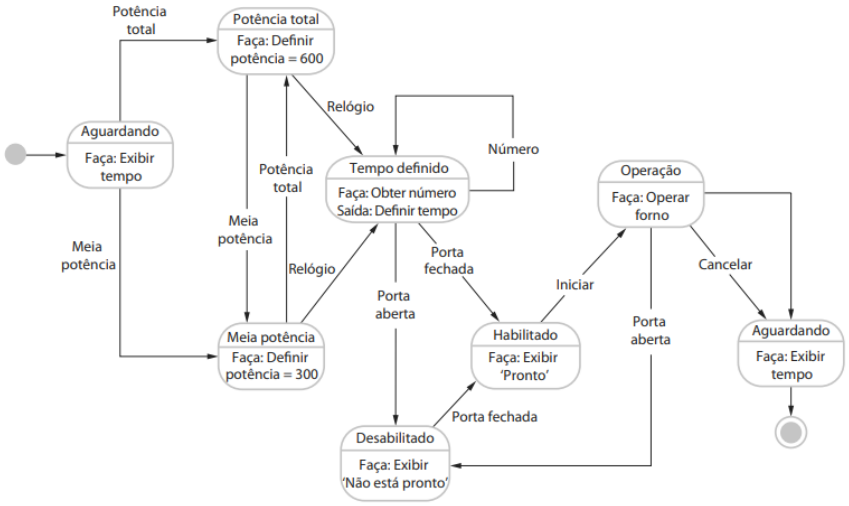


FIGURA 10 – Diagrama de estados de um sistema de micro-ondas.
Fonte: Sommerville, 2011, pág. 96.

Na FIGURA 10, os eventos estão sendo representados através de retângulos arredondados, sendo compostos por um título e uma descrição abaixo do título. As setas representam os estímulos responsáveis pela transição de um estado para outro. O início é representado por um círculo completamente preenchido, enquanto o final é mostrado por um círculo preenchido de forma parcial.

Conclui-se que o diagrama de estados é um diagrama responsável por demonstrar os eventos existentes em um sistema, juntamente com os estímulos

responsáveis pela transição entre esses eventos. Devido a quantidade de eventos que podem existir, o conceito de superestado pode acabar sendo necessário durante a construção do diagrama.

2.5.5 Diagrama de atividades

Os diagramas de atividades servem para mostrar as atividades responsáveis por constituir os processos de um sistema, juntamente com o controle que é feito para levar de uma atividade para outra atividade (Sommerville, 2011).

Esse diagrama disponibiliza a visualização do comportamento do software explicando a sequência de ações envolvidas. É similar a um fluxograma, pois tem o papel de exibir o fluxo entre as ações executadas durante uma atividade. Contudo, em um diagrama de atividades também é possível mostrar fluxos paralelos e alternativos (IBM, 2021).

A FIGURA 11 mostra um exemplo de diagrama de atividades de um sistema de saúde.

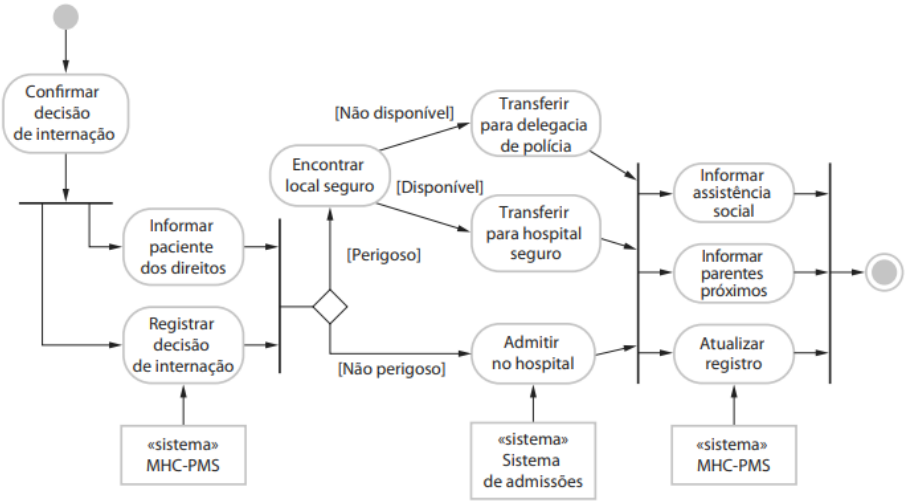


FIGURA 11 – Diagrama de atividades de um sistema de saúde.
Fonte: Sommerville, 2011, p. 85.

Na FIGURA 11, as atividades estão sendo representadas através de retângulos arredondados. As setas representam o fluxo de trabalho responsável por levar de uma atividade para outra atividade. A barra vertical sólida mostra atividades que são executadas de forma paralela entre si, sendo necessário aguardar a conclusão de todas elas antes de continuar o fluxo. Já o losango

representa uma condicional que dependendo da entrada, direciona o fluxo para uma determinada atividade.

Conclui-se que o diagrama de atividades é responsável por mostrar todo o fluxo de atividades existentes no sistema e o controle realizado durante a transição de atividades. O diagrama pode conter atividades que são paralelas, ou pode conter atividades que são situacionais dependendo da entrada recebida da atividade anterior.

3 METODOLOGIA SCRUM

Este capítulo apresenta detalhes sobre como a metodologia ágil Scrum funciona ao ser aplicada em qualquer projeto. Serão abordados tópicos sobre os fundamentos da metodologia, como por exemplo a divisão de tarefas, criação de tarefas, papéis que cada integrante do grupo possui, dentre outros tópicos.

3.1 Definição

O Scrum é uma metodologia que considera os fatores incerteza e criatividade existentes no projeto. O processo de aprendizagem da metodologia possui uma estrutura em sua volta que permite a avaliação pela equipe do que foi criado, e da forma que foi criado. É uma metodologia que enxerga a maneira que as equipes trabalham, e dispõe dos recursos necessários para aprimorar a organização, velocidade e qualidade do trabalho realizado pelo grupo (Sutherland, 2014).

Tem como base a inspeção e adaptação, que consiste na ideia de dar intervalos periódicos para avaliar o que está sendo feito, para assim verificar se o projeto e o resultado que está sendo obtido estão de acordo com as expectativas (Sutherland, 2014).

O Scrum é definido como objetivos sequenciais que devem ser concluídos em um determinado tempo. Os objetivos vão sendo finalizados de forma cíclica, e no término de cada ciclo os feedbacks são considerados e analisados para as próximas tomadas de decisões. Tais ciclos são nomeados como Sprint[3] (Sutherland, 2014).

A FIGURA 12 apresenta um exemplo de como o ciclo Scrum tem o seu funcionamento.

[3] Nota do autor: O termo Sprint será abordado de forma mais detalhada no tópico 2.3.1

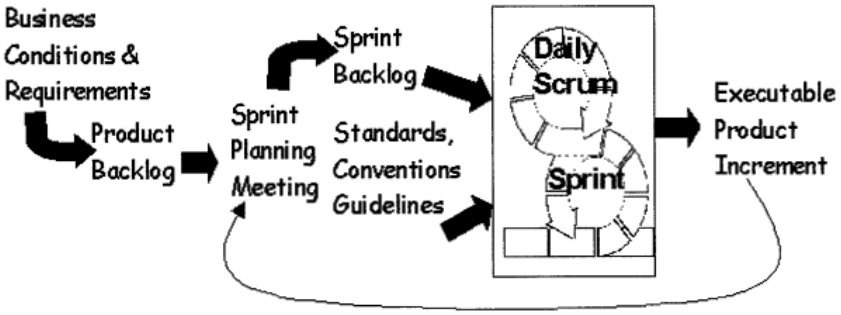


FIGURA 12 – O ciclo do Scrum.
Fonte: Schwaber; Beedle, 2001, p. 8.

Diante dos conceitos apresentados, é possível concluir que o Scrum é uma metodologia que considera os fatores imprevisibilidade e criatividade. Todo o feedback que é coletado no decorrer do projeto é analisado. Os objetivos são feitos de forma iterativa (Sprints) dentro de um tempo previamente estabelecido. Existem intervalos nos quais tudo o que está sendo feito é avaliado pela equipe, e as conclusões obtidas dessa avaliação são consideradas para as próximas tomadas de decisões.

3.2 Papéis dos integrantes

Desenvolver um produto através da metodologia ágil Scrum requer uma gama de responsabilidades e funções nas quais cada uma delas tem uma participação fundamental no sucesso do projeto. Cada pessoa integrante do projeto é alocada para uma determinada função. Tais papéis são divididos em Scrum Master, Product Owner e time de trabalho (Schwaber; Beedle, 2001).

3.2.1 Scrum Master

O Scrum Master é um papel de gerenciamento no qual tem a responsabilidade de garantir se os conceitos do Scrum estão sendo devidamente aplicados no projeto. Em cada reunião feita pela equipe, o Scrum Master analisa tudo o que é reportado e dito por cada integrante. Além disso, o Scrum Master compara o progresso que é feito com o progresso que era esperado tendo como base reuniões anteriores e os objetivos das Sprints (Schwaber; Beedle, 2001).

Ademais, o Scrum Master também tem a responsabilidade de trabalhar com os clientes e gerentes para poder definir quem será o Product Owner, que tem a função de criar as atividades que serão feitas pelo time de trabalho. O Scrum Master e o Product Owner trabalham de forma conjunta para definir quais atividades serão feitas em cada Sprint. Durante a Sprint em questão, o Scrum Master conduz as reuniões diárias (Daily) para garantir que os obstáculos foram removidos e as decisões corretas foram feitas (Schwaber; Beedle, 2001).

Tal papel costuma ser assumido pelo líder da equipe, líder do projeto ou gerente de projetos. A metodologia entrega a essa pessoa toda a estrutura necessária para cuidar de forma efetiva do andamento do projeto. O Scrum Master é responsável por tomar as decisões mais importantes, e por isso é um cargo que requer iniciativa e determinação (Schwaber; Beedle, 2001).

Diante dos conceitos apresentados, é possível concluir que o Scrum Master é um cargo responsável por dirigir e orientar o rumo do projeto. Tem a função de conversar com toda a equipe para entender a situação em que o projeto se encontra, e dessa forma atuar nas tomadas de decisões e remoções dos eventuais obstáculos que podem surgir. É um papel que requer iniciativa e determinação de quem o ocupar. Todo o progresso obtido pelo projeto deve ser avaliado pelo Scrum Master.

3.2.2 Product Owner

O Product Owner tem a responsabilidade de gerenciar e controlar o grupo de atividades que devem ser feitas durante o projeto. Tal grupo de atividades é denominado como Product Backlog na Scrum. A pessoa que recebe essa função deve garantir que o Product Backlog seja acessível por todos os integrantes da equipe, para que assim todos saibam os itens que possuem maior relevância e que devem ser feitos em cada momento (Schwaber; Beedle, 2001).

Inclusive, o Product Owner tem a responsabilidade de entender e mapear as demandas que o cliente possui. É responsável por conversar com o cliente e transmitir para a equipe o feedback obtido em cada uma das Sprints. De forma geral, o Product Owner faz a ponte entre os clientes conversando coletando opiniões acerca do produto, e com a equipe criando as pendências através do que foi considerado e valorizado pelos clientes (Sutherland, 2014).

É de suma importância que o Product Owner tenha conhecimento em relação ao setor que o projeto está atuando para conseguir levantar as necessidades dos clientes de forma efetiva para a equipe. O Product Owner deve ter um entendimento suficiente para saber o que deve ser feito de forma que agregue um valor significativo e verdadeiro (Sutherland, 2014).

Para que o Product Owner obtenha sucesso em seu trabalho, é estritamente necessário que toda a equipe respeite as decisões que são tomadas por ele. Toda a lista de tarefas e prioridades estabelecidas pelo Product Owner deve ser cumprida exatamente como foi especificada. As decisões tomadas pelo Product Owner devem ser transparentes e visíveis por toda a equipe (Schwaber; Beedle, 2001).

Conclui-se que o Product Owner tem a responsabilidade de criar a lista de atividades que devem ser realizadas para que o projeto seja concluído com sucesso. Tais atividades possuem prioridades nas quais devem ser respeitadas por todos os membros da equipe. As decisões que são tomadas pelo Product Owner têm a obrigatoriedade de serem transparentes e visíveis pelos integrantes. É através do Product Owner que a ponte entre o cliente e a equipe do projeto é feita, o Product Owner compreende as demandas do cliente e começa a criar o Product Backlog através das conclusões obtidas.

3.2.3 Time de trabalho

O time de trabalho é responsável por atender as demandas estabelecidas pelo Product Owner através do Product Backlog, para assim alcançar os objetivos de cada iteração ou Sprint do projeto. Tem a função de desenvolver o produto na prática. O time deve ter toda a autoridade necessária para tomar as ações que façam com que o objetivo do projeto seja concluído com sucesso (Schwaber; Beedle, 2001).

De forma complementar, o time de trabalho juntamente com o Scrum Master revisa o Product Backlog. Após a revisão, um trecho do Product Backlog é escolhido para que certa parte do produto seja de fato desenvolvido. Tal trecho do Product Backlog que é escolhido se torna um Sprint Backlog. O Sprint Backlog consiste em um conjunto de tarefas que devem ser atendidas pela equipe e que no final se tornam uma parte funcional do produto (Schwaber; Beedle, 2001).

É de suma importância que o time possua todas as habilidades e conhecimentos necessários para a conclusão do projeto. A equipe pode ser subdividida em outras equipes menores, cada uma com a sua especialidade e importância. Cada equipe deve concluir a sua parte para que o projeto consiga avançar para a próxima etapa. Em suma, deve haver um trabalho em conjunto, nenhuma equipe individualmente pode finalizar o produto como um todo (Sutherland, 2014).

Diante dos conceitos apresentados, é possível concluir que o time de trabalho é quem de fato tira o produto do papel e o desenvolve. O trabalho da equipe é feito em ciclos de Sprints nos quais existe uma seleção de um pedaço do Product Backlog. A equipe revisa o Product Backlog juntamente com o Scrum Master. O grupo deve ser formado por pessoas de forma que seja possível preencher as lacunas de conhecimentos necessários para o sucesso do projeto. A equipe pode ser subdividida em equipes menores, e o trabalho de cada uma deve ser considerado para que o projeto consiga avançar para a próxima etapa.

3.3 Eventos do Scrum

Os eventos na metodologia Scrum servem para entregar toda a transparência necessária ao projeto. Eles são responsáveis por criar uma rotina capaz de evitar reuniões que não foram previamente definidas, dessa forma contribuindo para que o tempo estipulado ao projeto não seja ultrapassado. A falha em qualquer dos eventos podem trazer perdas de oportunidades de revisões e adaptações (Schwaber; Sutherland, 2020).

3.3.1 Sprint

As Sprints na metodologia Scrum são onde as ideias são transformadas em algo palpável e funcional. Uma Sprint somente deve ter o seu início após a conclusão da Sprint anterior. É originado através da seleção de algumas tarefas do Product Backlog (Schwaber; Sutherland, 2020).

Elas permitem previsibilidade e garantem a inspeção e adaptação do projeto para que o objetivo seja alcançado com sucesso. A recomendação é que as Sprints não sejam muito longas para evitar possível aumento da complexibilidade e do risco. Uma Sprint longa faz com que os objetivos não

sejam definidos de forma clara. Em suma, cada Sprint pode ser vista como um pequeno projeto (Schwaber; Sutherland, 2020).

Dependendo da situação em que o projeto se encontra, a Sprint pode acabar sendo cancelada. Objetivos que se tornam obsoletos é o motivo principal do cancelamento de uma Sprint. Somente o Product Owner tem o poder necessário para cancelar a Sprint (Schwaber; Sutherland, 2020).

Conclui-se que a Sprint é responsável por agregar valor ao projeto e desenvolver algo funcional durante o seu andamento. Somente após a conclusão da Sprint anterior que uma nova pode iniciar. É recomentado que as Sprints não sejam muito longas para que os objetivos não sejam mal definidos, aumentando dessa forma o risco e a complexibilidade do projeto. Uma Sprint pode ser cancelada pelo Product Owner caso os objetivos não façam mais sentido.

3.3.2 Sprint Planning

A Sprint Planning é um planejamento responsável por iniciar a Sprint e estabelecer quais atividades devem ser feitas nela. É criado através do trabalho colaborativo de todo o time que está envolvido no projeto (Schwaber; Sutherland, 2020).

Tal planejamento deve levar em conta o porquê da Sprint em questão é importante para o projeto, o que pode ser feito nessa Sprint e como as tarefas que foram escolhidas serão feitas. A resposta para essas perguntas se dá através de uma análise de todo o projeto, envolvendo a equipe e um estudo dos clientes e objetivos (Schwaber; Sutherland, 2020).

Diante dos conceitos apresentados, é possível concluir que a Sprint Planning serve para observar e analisar como que a Sprint em questão irá funcionar no quesito de objetivos e tarefas. A análise é feita através de um trabalho conjunto de todos os integrantes da equipe.

3.3.3 Reuniões diárias

As reuniões diárias da metodologia Scrum tem como objetivo avaliar o progresso que a Sprint está tendo em relação ao seu objetivo, para assim fazer

as adaptações necessárias no Product Backlog e alinhar o próximo trabalho que deverá ser feito (Schwaber; Sutherland, 2020).

A reunião costuma ter uma duração pequena que varia em torno de 15 minutos, na qual participa todo o time de trabalho. Essa reunião é feita nos dias úteis da Sprint, e geralmente é agendada sempre no mesmo horário. As reuniões devem se concentrar no progresso e objetivo da Sprint em questão, para que dessa forma a autogestão e o foco sejam estimulados (Schwaber; Sutherland, 2020).

Tais reuniões acabam melhorando a comunicação da equipe e identificam eventuais obstáculos que podem surgir no decorrer do projeto. Além disso, as reuniões também promovem a tomada rápida de decisões. Todos esses fatores removem a necessidade de reuniões adicionais que não foram previstas (Schwaber; Sutherland, 2020).

O time de desenvolvimento pode alterar o planejamento fora das reuniões diárias através de encontros durante o dia. A equipe pode ter conversas mais detalhadas visando planejar ou adaptar o trabalho pendente da Sprint atual (Schwaber; Sutherland, 2020).

Diante dos conceitos apresentados, é possível concluir que a reunião diária serve para a equipe avaliar e planejar a situação da Sprint além de ver quais são os próximos passos. Melhora toda a comunicação da equipe e isso acaba descartando eventuais reuniões que não foram previstas. É uma reunião curta que geralmente é agendada nos mesmos horários em dias úteis.

3.3.4 Sprint Review

A Sprint Review serve para avaliar o resultado obtido pela Sprint e planejar como serão as próximas etapas. Tais resultados são apresentados para as partes interessadas do projeto e os objetivos em relação ao produto final também são colocados em pauta (Schwaber; Sutherland, 2020).

Durante o evento, todas as partes interessadas revisam o que foi feito e agregado ao produto final, e através dessas informações os participantes avaliam o que poderá ser feito no futuro. Alterações no Product Backlog podem ocorrer caso seja necessário. A Sprint Review deve ser feita para planejar e

alinhar os objetivos da equipe, portanto não pode se limitar a ser uma simples apresentação (Schwaber; Sutherland, 2020).

Diante dos conceitos apresentados, é possível concluir que a Sprint Review é um evento no qual a equipe e as partes interessadas do projetam avaliam o que foi construído na Sprint e planejam de forma conjunta como serão os próximos passos. Os objetivos são avaliados e alterações no Product Backlog podem ser feitas. O evento serve para alinhar os objetivos do projeto, e por conta disso não pode ser somente uma apresentação.

3.4 Artefatos do Scrum

Os artefatos do Scrum representam trabalho ou valor agregado dentro do projeto. Graças a eles, informações que possuem extrema importância ficam mais transparentes e acessíveis. Desse modo, qualquer integrante da equipe fica por dentro das principais atualizações da situação em que o projeto se encontra (Schwaber; Sutherland, 2020).

3.4.1 Product Backlog

O Product Backlog é uma lista ordenada de itens de tudo que é necessário ser feito para que o produto seja desenvolvido. Cada um desses itens possui uma certa prioridade, sendo que os mais prioritários ficam no topo da lista. O gerenciamento dessa lista fica a cargo do Product Owner (Schwaber; Sutherland, 2020).

Tais tarefas são selecionadas na Sprint Planning e começam a ser trabalhados dentro da Sprint atual. Esses itens que constituem o Product Backlog vão ficando cada vez mais transparentes através do refino de atividades, que consiste na ideia de quebrar as atividades existentes em atividades menores e mais precisas (Schwaber; Sutherland, 2020).

Em suma, é toda a lista de tarefas que o time de desenvolvimento irá se basear para iniciar as suas atividades. Durante esse processo, o Product Owner pode auxiliar o time ajudando-os a sanar eventuais dúvidas que podem surgir (Schwaber; Sutherland, 2020).

Diante dos conceitos apresentados, é possível concluir que o Product Backlog é uma lista de todas as atividades que devem ser concluídas para que

o produto seja de fato desenvolvido. Essa lista é manipulada pelo Product Owner, podendo sofrer refinamentos e melhorias com o passar do tempo. Em cada Sprint, um aglomerado de itens do Product Backlog é selecionado dentro da Sprint Planning para serem trabalhadas e finalizadas.

3.4.2 Sprint Backlog

O Sprint Backlog é composto pela seleção de um grupo de tarefas do Product Backlog. É um artefato que também leva em consideração o objetivo que a Sprint em questão possui, e o plano que será executado para a entrega do incremento (Schwaber; Sutherland, 2020).

Também serve como uma visão geral do trabalho que deve ser feito para que o objetivo da Sprint seja alcançado com sucesso. O Sprint Backlog é atualizado ao longo do tempo através de aprendizados e experiências previamente obtidos pela equipe. Além disso, são colocados em pauta nas reuniões diárias para que o progresso seja avaliado (Schwaber; Sutherland, 2020).

Conclui-se que o Sprint Backlog é um conjunto de tarefas previamente selecionadas do Product Backlog, no qual também leva em conta o objetivo que a Sprint atual possui. O Sprint Backlog sofre atualizações de acordo com os entendimentos que o time vai adquirindo no decorrer do tempo. São abordados nas reuniões diárias e o progresso é inspecionado.

3.4.3 Iteração

A iteração é um artefato que contempla todos os itens que foram desenvolvidos na Sprint. Cada incremento é oriundo dos incrementos anteriores e todos eles devem funcionar corretamente de forma conjunta (Schwaber; Sutherland, 2020).

Em uma única Sprint vários incrementos podem ser criados, sendo que a soma desses incrementos é apresentada e discutida na Sprint Review. O trabalho realizado somente é considerado parte do incremento caso ele esteja de acordo com os objetivos da Sprint (Schwaber; Sutherland, 2020).

Conclui-se que uma iteração é definida como o conjunto de itens trabalhados em uma determinada Sprint. Cada iteração nova é dependente das

anteriores já verificadas detalhadamente, e elas devem funcionar em conjunto. Em uma Sprint mais de um incremento pode ser criado. Todos esses incrementos são abordados na Sprint Review e devem estar de acordo com os objetivos que a Sprint possui.

4 SOFTWARE PARA GESTÃO DE PROJETOS EM SCRUM

Este capítulo apresenta a aplicação prática dos conhecimentos pesquisados com o intuito de modelar um software que seja capaz de auxiliar empresas que optaram por utilizar a metodologia Scrum em seus projetos. Para que o software possa atender a finalidade proposta, será levantado os requisitos funcionais e não-funcionais, juntamente com a construção dos respectivos diagramas UML. O sistema deve ser capaz de centralizar e facilitar o acesso aos dados, para assim evitar a adoção de eventuais processos manuais, como por exemplo o uso de papéis e planilhas.

O software não será abordado e desenvolvido em sua totalidade, ficando a implementação dos demais elementos restantes entendidos como uma continuação desta obra.

4.1 Requisitos do sistema

Serão apresentados nos tópicos seguintes os requisitos funcionais e não funcionais do software.

4.1.1 Requisitos funcionais

O QUADRO 3 apresenta os requisitos funcionais do software para gestão de projetos em Scrum.

QUADRO 3 – Requisitos funcionais do software para gestão de projetos em Scrum.

1	O Scrum Master deve ser capaz de manter projetos.
2	O Product Owner deve ser capaz manter tarefas no Product Backlog.
3	O time de desenvolvimento deve ser capaz de pesquisar e visualizar tarefas.
4	O Scrum Master deve ser capaz de manter reuniões.
5	O Scrum Master e o Product Owner devem ser capazes de manter Sprints.
6	O time de trabalho deve ser capaz de pesquisar e visualizar Sprints.
7	O Scrum Master e o Product Owner devem ser capazes de manter uma Sprint Review.
8	O Scrum Master deve ser capaz de manter integrantes do projeto.

4.1.2 Requisitos não-funcionais

O QUADRO 4 apresenta os requisitos não funcionais do software para gestão de projetos em Scrum.

QUADRO 4 – Requisitos não funcionais do software para gestão de projetos em Scrum.

1	Ser implementado na linguagem C# utilizando o ASP.NET Core 6.0.
2	Utilizar o banco de dados SQL Server.
3	Utilizar a cloud do Azure DevOps.
4	Deve ser compatível com os navegadores Internet Explorer, Google Chrome e Mozilla Firefox.
5	Ser orientado a objetos.
6	O sistema deve respeitar a LGPD.

4.2 Diagrama de casos de uso

A FIGURA 13 apresenta o diagrama de casos de uso que traduz os requisitos funcionais existentes no QUADRO 3.

REFERÊNCIAS

SOMMERVILLE, I. **Engenharia de Software 9ª edição.** São Paulo: Pearson Education do Brasil, 2011.

SCHACH, S. R. **Engenharia de Software: Os Paradigmas Clássico e Orientado a Objetos.** São Paulo: McGraw-Hill Interamericana do Brasil Ltda, 2009.

VALENTE, M. T. **Engenharia de Software Moderna.** [s.l.]: [s.n.], 2020.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. **The Unified Software Development Process.** Minneapolis: Pearson Education Corporate Sales Division, 1999.

IBM. Diagramas de Atividades. **IBM**, 2021. Disponível em: <https://www.ibm.com/docs/pt-br/rational-soft-arch/9.7.0?topic=diagrams-activity> Acesso em: 16 outubro 2022.

ERICKSON, J.; SIAU, K. Theoretical and practical complexity of modeling methods. **Research Gate**, 2007. Disponível em: <https://www.researchgate.net/publication/220422139_Theoretical_and_practical_complexity_of_modeling_methods> Acesso em: 16 outubro 2022.

PRESSMAN, R. S. **Engenharia de Software.** São Paulo: MAKRON Books do Brasil Editora Ltda, 1995.

SUTHERLAND, J. **Scrum - A Arte de Fazer o Dobro do Trabalho em Metade do Tempo.** São Paulo: Texto Editores Ltda, 2014.

BEEDLE, M.; SCHWABER, K. **Agile Software Development with Scrum.** Upper Saddle River: Prentice Hall, 2001.

SCHWABER, K.; SUTHERLAND, J. The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. **Scrum Guides**, 2020. Disponivel em:

<https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf> Acesso em: 16 outubro 2022.

DEVMEDIA. Ciclos de Vida do Software. **DEVMEDIA**, 2011. Disponível em: <https://www.devmedia.com.br/ciclos-de-vida-do-software/21099> Acesso em: 16 outubro 2022.

MÁRCIO. Apresentação do Scrum. **DEVMEDIA**, 2013. Disponível em: <https://www.devmedia.com.br/apresentacao-do-scrum/27887#:~:text=O%20Scrum%20trabalha%20com%20ciclos,a%20perceber%20os%20resultados%20rapidamente.> Acesso em: 16 outubro 2022.