UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
UNIDADE ACADÊMICA ESPECIALIZADA EM CIÊNCIAS AGRÁRIAS – ESCOLA AGRÍCOLA DE JUNDIAÍ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Liryel Belo Aguiar

PROJETO DE ESTRATÉGIA DE MODELAGEM DE SISTEMAS APLICADA A PROCESSOS ÁGEIS DE DESENVOLVIMENTO DE SOFTWARE

Macaíba, RN

2024

LIRYEL BELO AGUIAR

PROJETO DE ESTRATÉGIA DE MODELAGEM DE SISTEMAS APLICADA A PROCESSOS ÁGEIS DE DESENVOLVIMENTO DE SOFTWARE

Monografia apresentada ao curso superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Federal do Rio Grande do Norte, como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador(a): Prof(a). Carla Fernandes Curvelo.

Macaíba, RN

2024

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Setorial Prof. Rodolfo Helinski - Escola Agrícola de Jundiaí - EAJ - Macaiba

Aguiar, Liryel Belo.
Projeto de estratégia de modelagem de sistemas aplicada a processos ágeis de desenvolvimento de software / Liryel Belo Aguiar. - Macaíba, 2024.
60f.: il.

Universidade Federal do Rio Grande do Norte, Unidade Acadêmica Especializada em Ciências Agrárias, Curso Superior de Tecnologia em Analise e Desenvolvimento de Sistemas, Macaíba, RN, 2024.
Orientação: Profa. Dra. Carla Fernandes Curvelo.

1. Metodologias ágeis - Monografia. 2. Modelagem de sistemas - Monografia. 3. Documentação de sistemas - Monografia. 4. Scrum - Monografia. 5. Diagramas UML - Monografia. I. Curvelo, Carla Fernandes. II. Título.

RN/UF/BSPRH CDU 004.43

Elaborado por Valéria Maria Lima da Silva - CRB-15/451

Liryel Belo Aguiar

PROJETO DE ESTRATÉGIA DE MODELAGEM DE SISTEMAS APLICADA A PROCESSOS ÁGEIS DE DESENVOLVIMENTO DE SOFTWARE

Trabalho de conclusão de curso de graduação apresentado à Unidade Acadêmica Especializada em Ciências Agrárias – Escola Agrícola de Jundiaí da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do título de Tecnólogo (a) em Análise e Desenvolvimento de Sistemas.

Aprovado em: _____ de _____ de 20____.

BANCA EXAMINADORA

_____

Nome do professor - instituição (orientador)

_____

Nome do professor - instituição

_____

Nome do professor - instituição

Dedico este trabalho à minha mãe. Sem ela, eu não conquistaria nada do que tenho. Agradeço por sua insistência em me tornar melhor.

AGRADECIMENTOS

Agradeço a Deus, pois Ele é a razão da minha existência e minha vida é para a glória dEle. Sou grata à minha família pelo apoio incondicional e sustentação, especialmente à minha mãe, que sempre me encorajou e abriu inúmeras portas para mim. Sem ela, eu não teria superado os momentos difíceis e não teria alcançado nada do que tenho hoje. Desejo ser cada vez mais parecida com ela.

Agradeço também aos professores, pela dureza, franqueza e pela insistência em nos desafiar. À minha orientadora, Carla, expresso minha gratidão por ter me dado um "norte" e por sua ajuda nos momentos de desespero. Apesar das dificuldades, tudo deu certo.

A única maneira de ir rápido é ir bem.

Autor desconhecido

RESUMO

Esta pesquisa explora a integração da modelagem de sistemas em processos ágeis de desenvolvimento de software, analisando os principais desafios, benefícios e limitações envolvidos. O estudo investiga como as metodologias ágeis, conhecidas por sua flexibilidade e foco em entregas rápidas, podem ser harmonizadas com a necessidade de uma modelagem de sistemas estruturada e documentada. Um dos focos centrais é a análise do impacto do volume de documentação nas metodologias ágeis, bem como a forma como as mudanças frequentes de requisitos, características desses processos, afetam a modelagem de sistemas.

Como parte dos resultados, desenvolvem-se estratégias e recomendações práticas para facilitar essa integração, fornecendo orientações claras para profissionais de desenvolvimento de software e equipes de projeto. Essas recomendações visam não apenas melhorar a comunicação e a coordenação em ambientes ágeis, mas também assegurar que a documentação permaneça atualizada e relevante, acompanhando as mudanças significativas ao longo do projeto.

Além de contribuir para a prática profissional, o estudo oferece uma importante contribuição para o conhecimento acadêmico sobre a integração da modelagem de sistemas em metodologias ágeis. Ele reforça a importância da documentação contínua como parte integrante das tarefas diárias, destacando seu papel crítico na melhoria da comunicação e na coordenação eficaz dentro das equipes de desenvolvimento ágil.

Palavras-chave: Metodologias ágeis. Documentação contínua. Desenvolvimento de software.

ABSTRACT

This research examines how systems modeling can be integrated into agile software development processes, addressing the main challenges, benefits, and limitations of this integration. The study investigates how agile methodologies, known for their flexibility and focus on rapid delivery, can align with the need for structured and well-documented systems modeling. A central aspect is understanding the impact of documentation on agile methodologies and how frequent changes in requirements, inherent to these processes, influence systems modeling.

The results include the development of strategies and practical recommendations to assist with this integration, providing valuable guidance for software professionals and project teams. These recommendations aim to improve communication and coordination in agile environments, as well as ensure that documentation remains up-to-date and relevant, keeping pace with changes throughout the project.

In addition to benefiting professional practice, the study also contributes to academic knowledge on how systems modeling can be effectively integrated into agile methodologies. It highlights the importance of continuous documentation, underscoring its crucial role in communication and coordination within agile development teams.

Keywords: Agile methodologies. Continuous documentation. Software development.

LISTA DE SIGLAS

UML	Unified Modeling Language

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

A modelagem de sistemas é uma técnica ou abordagem que envolve a criação de representações simplificadas e abstratas de sistemas reais ou propostos. Esses modelos são usados para entender, analisar, comunicar e tomar decisões sobre o sistema em questão. A modelagem de sistemas é amplamente aplicada em uma variedade de campos, incluindo engenharia de software, engenharia de sistemas, ciência da computação e muitos outros.

Grady Booch, um dos principais especialistas na área de engenharia de software, enfatiza que a modelagem de sistemas é fundamental para a compreensão de sistemas complexos e para a comunicação eficaz entre os membros da equipe. Booch argumenta que a modelagem ajuda a capturar a essência dos sistemas e a identificar problemas de forma antecipada, o que é crucial para a validação e a implementação bem-sucedida das soluções propostas (BOOCH, 1994).

O sucesso de uma produção de sistema está ligado diretamente à sua capacidade de atender às necessidades do usuário, que definem os requisitos mínimos que o software deve ter. O custo e os prazos definidos também são importantes na produção de um sistema e definem tanto seu sucesso quanto sua qualidade. O produto final depende então de uma série de definições pré-estabelecidas e documentadas, que é o que chamamos de modelagem de sistemas.

O uso da modelagem de sistemas é discutido nos campos de desenvolvimento de sistemas dividindo opiniões acerca de sua importância. Podemos fazer uma rápida comparação para entender o valor que ela agrega a um projeto. Usaremos então o exemplo das plantas de um edifício,

do que elas valem para seus moradores? Talvez nunca cheguem a vê-las, mas esses registros são, de forma indiscutível, vital para o projeto de um edifício, seja para sua construção ou para possíveis reformas e eventuais problemas. Da mesma maneira podemos dizer que uma modelagem de sistema é importante para o desenvolvimento de um projeto, tais documentações podem poupar horas de trabalho no futuro, seja de alguém que já está habituado ao sistema ou que está aprendendo sobre ele, e ainda podemos fazer uso de soluções que sejam compatíveis com a arquitetura definida.

Ainda utilizando o exemplo do edifício citado anteriormente, vamos imaginar como funciona o processo para construí-lo, mesmo que tenhamos as documentações e materiais necessários, se não houver organização e uma equipe devidamente treinada onde cada um exerce uma função específica, como ficará esse processo? Certamente que haverá muitos problemas, pode ocorrer um atraso na obra e claro, terá um equipe estressada e sobrecarregada. Tão importante quanto um trabalho bem documentado é um trabalho com processos bem definidos e organizados. O exemplo que usamos se torna pertinente se imaginarmos que ninguém constroem um edifício sem que antes as fundações estejam prontas, assim como ele é entregue em pequenas partes, que vão dando forma ao produto final. Da mesma forma é a produção de um sistema, nesse processo se faz necessário o que chamamos de Metodologias ágeis, que são soluções implementadas com o intuito de promover mais rapidez, flexibilidade e eficiência nas rotinas organizacionais.

As metodologias ágeis utilizam de algumas ferramentas organizacionais como quadros para acompanhar tarefas a serem feitas ou já concluídas, reuniões rápidas, uso dos meios digitais e muitas outras. Uma das plataformas de hospedagem de código mais populares entre os

desenvolvedores, o Github, tem uma ferramenta que se chama “Projetos” onde o desenvolvedor pode organizar suas tarefas em quadros Kanban, e isso nos mostra que esse processos ágeis estão ganhando cada vez mais espaço.

Jeff Sutherland, co-criador do Scrum, destaca a importância da adaptabilidade e da melhoria contínua no contexto das metodologias ágeis. Ele afirma que:

> O Scrum acolhe a incerteza e a criatividade. Coloca uma estrutura em volta do processo de aprendizagem, permitindo que as equipes avaliem o que já criaram e, o mais importante, de que forma o criaram. A estrutura do Scrum busca aproveitar a maneira como as equipes realmente trabalham, dando a elas as ferramentas para se auto-organizar e, o mais importante, aprimorar rapidamente a velocidade e a qualidade de seu trabalho.
>
> — SUTHERLAND, Jeff. *Scrum: A Arte de Fazer o Dobro do Trabalho na Metade do Tempo*. Crown Business, 2014.

A maior parte das empresas implementam processos ágeis em busca de eficiência, rapidez e melhor organização no desenvolvimento de projetos. Com o crescimento do seu uso tanto desenvolvedores como profissionais de tantas outras áreas precisam aprender sobre processos ágeis de documentações, o que já está acontecendo no mercado, onde aparecem cada vez mais vagas de emprego exigindo tais conhecimentos.

Esse aumento do uso se dá pelos benefícios das metodologias ágeis, alguns deles são a flexibilidade e as entregas rápidas, o que torna a modelagem de sistemas em processos de desenvolvimento ágil mais complicado. O desenvolvimento ágil busca sanar algumas problemáticas encontradas no desenvolvimento de sistemas.

Este trabalho é importante porque ajuda a esclarecer e defender a relevância da modelagem de sistemas no desenvolvimento de software,

especialmente dentro das metodologias ágeis. Mostramos que, assim como as plantas de um edifício são essenciais para sua construção e manutenção, a modelagem de sistemas é crucial para o sucesso de um projeto de software. Ao adotar práticas ágeis, como sprints e colaboração constante, é possível resolver problemas comuns como documentação excessiva ou insuficiente, mudanças não documentadas e dificuldades de integração entre as equipes. Em resumo, este trabalho destaca como a combinação de modelagem de sistemas e metodologias ágeis pode levar a um desenvolvimento mais eficiente, flexível e alinhado às necessidades dos usuários, trazendo benefícios significativos para o processo de desenvolvimento de software. Trazendo aqui uma proposta documentada de desenvolvimento utilizando metodologias ágeis e diagramas UML.

1.1 JUSTIFICATIVA

Há uma crescente relevância da integração da modelagem de sistemas em processos ágeis de desenvolvimento de software. Os processos ágeis têm ganhado ampla adoção na indústria de desenvolvimento de software devido à sua capacidade de entregar software funcional de alta qualidade de forma rápida e interativa. No entanto, a agilidade e a entrega contínua muitas vezes resulta em uma modelagem de sistemas mínima, o que pode dificultar a compreensão e comunicação de requisitos e design de software.

Essa pesquisa pode beneficiar profissionais da área, acadêmicos e até empresas a gerir melhor e documentar de forma clara e eficiente os seus projetos. Por isso a pesquisa e a proposta deste trabalho são importantes, para contribuir e atender a uma necessidade crescente.

1.2 OBJETIVOS GERAIS

Entendendo a problemática que a falta de harmonia entre metodologias ágeis e modelagem de sistemas trás, percebemos a necessidade de preencher essa lacuna, fornecendo orientações para profissionais de desenvolvimento de software.

1.2.1 Objetivos específicos

- Analisar os desafios da integração da modelagem de sistemas em processos ágeis de desenvolvimento de software.
- Desenvolver estratégias e recomendações para facilitar a integração eficaz da modelagem de sistemas em processos ágeis de desenvolvimento de software.
- Fornecer orientações práticas para profissionais de desenvolvimento de software e equipes de projeto interessados em aprimorar a comunicação e documentação em ambientes ágeis.
- Contribuir para o conhecimento acadêmico e prático sobre a integração da modelagem de sistemas em metodologias ágeis.

2. REFERENCIAL TEÓRICO

Como já citado neste trabalho, foi preciso uma pesquisa aprofundada sobre os assuntos que abordaremos tais como: modelagem de sistemas, metodologias ágeis e suas aplicações no mercado atual. A pesquisa foi fundamental para a conclusão deste trabalho. Segue-se neste capítulo parte da pesquisa feita, incluindo alguns conceitos importantes e documentações necessárias para compreender a proposta deste trabalho.

2.1 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

O processo de desenvolvimento de software é em resumo a divisão de etapas para o desenvolver um software com melhor qualidade, menos tempo e maior organização desde sua análise de viabilidade até sua manutenção. Para um bom produto final é preciso ter sólidas etapas de desenvolvimento, capacidade de planejamento de projetos e colaboradores bem capacitados, podemos ainda acrescentar um item óbvio que seria uma documentação bem elaborada.

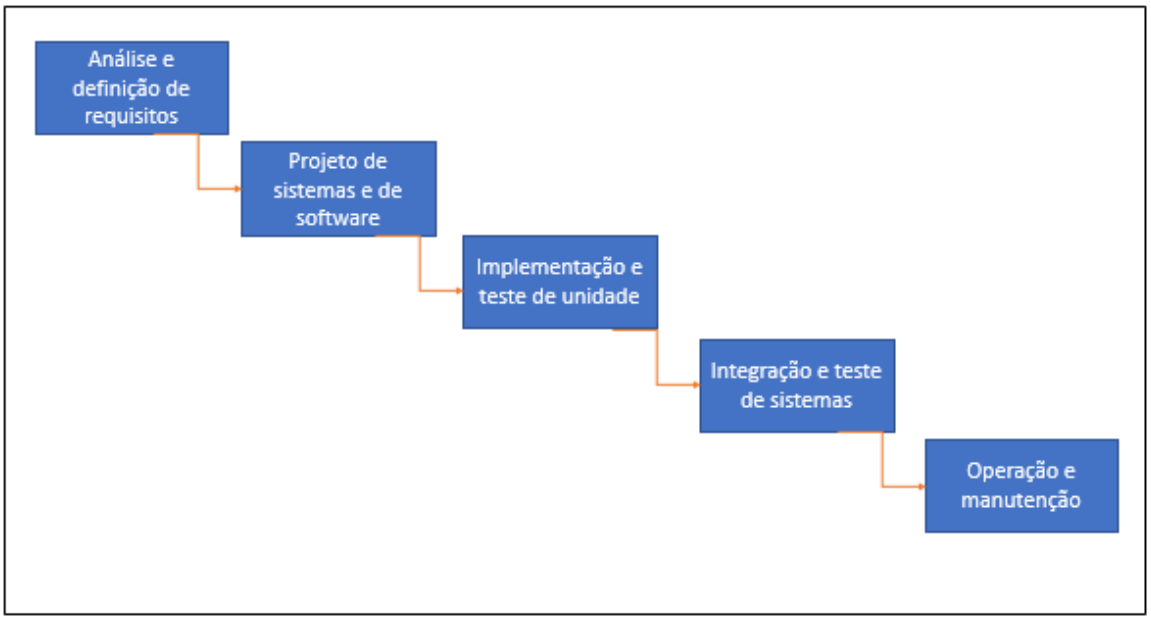
São muitos produtos que se propõem a fazer sucesso no mercado digital, mas nem todos eles têm sucesso e muitos acabam antes mesmo de sua proposta sair do planejamento para a comercialização. O que diferencia software de sucesso de uma ideia frustrada? Bom, muitas equipes de desenvolvimento não seguem processos bem definidos, não sabem exatamente o que querem e acabam entregando um produto genérico, sem documentação e com um tempo de entrega extenso, isso quando terminam o projeto. Quando seguimos um processo de desenvolvimento bem definido sabemos exatamente o que fazer e no momento que surgir uma dúvida, o sistema de organização que seguimos nos ajudará a saná-la e ainda descobrir outras necessidades que o cliente nem mesmo sabia que tinha.

Roger Pressman, em Software Engineering: A Practitioner's Approach, ressalta que ter uma abordagem estruturada e uma documentação adequada é essencial para o desenvolvimento bem-sucedido de software. Segundo ele, a ausência desses componentes críticos pode levar ao fracasso de muitos projetos, uma vez que compromete a clareza dos requisitos e dificulta a comunicação eficaz entre todos os envolvidos (PRESSMAN, 2014).

David Norris, em Why Software Fails: Insights from a Software Engineering Perspective, também destaca a importância de processos bem definidos e de uma boa documentação. Norris explica que a falta de planejamento adequado e de documentação pode ser um fator decisivo para o insucesso dos projetos de software, levando a uma gestão ineficaz e a produtos que não atendem às expectativas dos usuários (NORRIS, 2012).

Vamos ver melhor as etapas de um processo de desenvolvimento de software genérico (Figura 1).

Figura 1 : Processo de desenvolvimento de software genérico



Fonte: Ciclo de vida do software: por que é importante saber? | Blog da TreinaWeb

- **Levantamento e análise de requisitos:** O levantamento e análise de requisitos é a fase inicial de qualquer projeto de desenvolvimento de software. Envolve a coleta e compreensão das necessidades do cliente e dos usuários finais. Os requisitos funcionais e não funcionais são documentados nesta etapa para estabelecer as bases do projeto.

- **Design do projeto:** Nesta etapa, os requisitos identificados na fase anterior são transformados em um plano de alto nível para o sistema. O design do projeto inclui a arquitetura do sistema, a estrutura de dados, as interfaces do usuário e outros aspectos técnicos. O objetivo é criar um modelo abstrato do sistema antes de passar para a implementação.
- **Implementação:** Após a fase de design, a implementação é o processo de traduzir o design em código real. Os programadores escrevem o software de acordo com as especificações estabelecidas no design. Essa fase é fundamental para a construção efetiva do sistema.
- **Teste:** O teste é uma etapa crítica para garantir que o software atenda aos requisitos e funcione conforme o esperado. Diferentes tipos de testes, como testes unitários, de integração, de sistema e de aceitação, são conduzidos para identificar e corrigir falhas e garantir a qualidade do software.
- **Documentação:** A documentação é uma parte essencial do ciclo de vida do software. Ela inclui a criação de manuais de usuário, documentação técnica e qualquer outra informação relevante que auxilie na compreensão, uso e manutenção do sistema. Uma boa documentação facilita a gestão, treinamento e suporte contínuo.
- **Suporte, manutenção e atualização:** Após o lançamento do software, é fundamental fornecer suporte contínuo aos usuários, monitorar o desempenho do sistema e realizar atualizações para corrigir bugs, adicionar novos recursos e manter a segurança. O suporte, a manutenção e as atualizações garantem que o software continue atendendo às necessidades em constante evolução dos usuários.

O processo bem detalhado do desenvolvimento de software, principalmente quando em conjunto com as metodologias ágeis e boa documentação, nos ajudam a solucionar alguns problemas do desenvolvimento. Sendo alguns deles:

- Documentação em excesso
- Documentação com falta de detalhes
- Documentação muito extensiva
- Mudanças de requisitos não documentadas
- Dificuldade de integração entre os envolvidos

Mike Cohn, em *Agile Estimating and Planning*, explora como metodologias ágeis podem enfrentar desafios como documentação excessiva e mudanças de requisitos, promovendo um ambiente adaptável e eficiente (COHN, 2005). Jeff Sutherland, em *Scrum: The Art of Doing Twice the Work in Half the Time*, destaca que o Scrum melhora a colaboração e a comunicação, ajudando a resolver problemas relacionados à documentação e à integração da equipe (SUTHERLAND, 2014). Roger Pressman, em *Software Engineering: A Practitioner's Approach*, enfatiza a importância de uma abordagem estruturada e de uma documentação adequada para evitar problemas comuns e facilitar a integração entre as partes envolvidas (PRESSMAN, 2014).

Vejamos cada um desses problemas citados acima mais detalhadamente:

- **Documentação em excesso:** Processos ágeis geralmente utilizam de documentações menores, um dos motivos disso é seu foco em processos rápidos. E assim como poucas documentações podem prejudicar e atrasar o desenvolvimento de um sistema,

documentações em excesso podem dificultar a implementação de um processo ágil.

- **Documentações com falta de detalhes:** A falta de detalhes nas documentações podem prejudicar o desenvolvimento e o suporte a futuros problemas, fazendo muitos desenvolvedores gastarem tempo tentando compreender aquilo que não foi detalhado.

- **Documentações muito detalhadas:** Uma das maiores dificuldades em se implementar modelagem de sistemas em metodologias ágeis é encontrar o equilíbrio ideal na quantidade de documentação e nos detalhes contidos nelas. Imagine que a empresa quer facilitar e agilizar processos, mas os desenvolvedores gastam muito tempo apenas detalhando as documentações e mais tarde tentando analisar cada detalhe.

- **Mudança de requisitos:** Em metodologias ágeis é comum os requisitos mudarem a cada ciclo de desenvolvimento, o que pode tornar a modelagem de sistemas difícil de ser implementada e também estática.

- **Dificuldade de integração entre os envolvidos:** Nas metodologias ágeis temos reuniões diárias e uma dinâmica de trabalho fluida que ajudam na integração dos colaboradores, que é uma das suas características marcantes, já na modelagem de sistemas é promovida atividades individuais ou em grupos pequenos. Dessa

forma se torna difícil envolver cada colaborador nos processos de modelagem.

2.2 METODOLOGIA ÁGIL

A metodologia ágil é uma abordagem para o desenvolvimento de software que prioriza flexibilidade, colaboração e entrega contínua de valor. Diferente das metodologias tradicionais, como o modelo em cascata, as metodologias ágeis utilizam ciclos curtos de desenvolvimento chamados sprints. Esses sprints, que duram geralmente de duas a quatro semanas, permitem adaptações rápidas às mudanças e entregas frequentes de incrementos do software (Cohn, 2005).

Os princípios ágeis, que estão no coração do Manifesto Ágil, valorizam mais os indivíduos e as interações do que processos e ferramentas. Eles também colocam mais ênfase em software funcionando do que em documentação extensa, mais colaboração com o cliente do que negociação de contratos, e mais capacidade de responder a mudanças do que seguir um plano fixo (Agile Alliance, 2024). Esses princípios têm o objetivo de tornar o desenvolvimento mais eficiente, melhorar a comunicação e garantir que o produto final realmente atenda às necessidades dos usuários.

2.2.1 SCRUM

Entre as metodologias ágeis, o SCRUM é uma das mais populares e é amplamente utilizado para gerenciar o desenvolvimento de produtos complexos. O SCRUM se baseia em sprints curtos e fixos, que geralmente duram de duas a quatro semanas. Cada sprint inicia com uma reunião de planejamento para definir as tarefas a serem realizadas e encerra com uma

revisão e retrospectiva para avaliar o que foi feito e como a equipe pode melhorar (Sutherland, 2014; Schwaber & Beedle, 2002).

No framework SCRUM, há três papéis principais: o Product Owner, que representa os interesses dos stakeholders e prioriza o backlog do produto; o SCRUM Master, que facilita o processo e remove obstáculos; e o Time de Desenvolvimento, responsável por entregar os incrementos do produto. Além disso, o SCRUM promove reuniões diárias rápidas, conhecidas como daily stand-ups, para garantir que todos na equipe estejam alinhados e para identificar e resolver problemas rapidamente.

O objetivo do SCRUM é entregar incrementos de valor de forma contínua e adaptativa, permitindo que a equipe responda rapidamente às mudanças nos requisitos e no ambiente de trabalho (Sutherland, 2014).

2.3 MODELAGEM DE SISTEMAS

A modelagem de sistemas é uma prática essencial, ela envolve a criação de representações abstratas e simplificadas de sistemas complexos, para que possamos compreendê-los, analisá-los e comunicar suas características de maneira mais clara e eficiente.

Ivar Jacobson, que contribuiu significativamente para a área com seu enfoque na engenharia de software orientada a casos de uso. Jacobson destaca que uma abordagem estruturada para a modelagem é essencial para o desenvolvimento de sistemas eficientes e eficazes. Ele argumenta que a modelagem deve guiar tanto a implementação quanto a evolução do software, proporcionando uma base sólida para o desenvolvimento contínuo (JACOBSON, 1992). Mostrando assim a importância para o desenvolvimento bem-sucedido de sistemas de software. Vejamos alguns benefícios da modelagem de sistemas:

- **Compreensão:** A modelagem ajuda os desenvolvedores e stakeholders a compreenderem a estrutura e o comportamento do sistema.
- **Análise:** Permite a análise de diferentes aspectos, como desempenho, segurança e confiabilidade, antes da implementação.
- **Comunicação:** Facilita a comunicação entre membros da equipe e partes interessadas, fornecendo uma representação visual do sistema.
- **Identificação de Requisitos:** Ajuda na identificação e especificação dos requisitos do sistema, garantindo que todos os envolvidos tenham uma compreensão comum.

A modelagem de sistemas é realizada em várias fases do ciclo de vida do desenvolvimento de software, incluindo a análise de requisitos, design, implementação e manutenção. Durante a análise de requisitos, os modelos ajudam a capturar e representar os requisitos do sistema. Durante o design, são utilizados para criar planos detalhados da arquitetura e da estrutura do sistema. Na implementação, os modelos desenvolvidos durante a análise de requisitos e design servem como guias detalhados para os programadores. Eles ajudam a traduzir especificações abstratas em código funcional, garantindo que a arquitetura planejada seja seguida rigorosamente. A modelagem também facilita a comunicação entre equipes, reduzindo mal-entendidos e aumentando a consistência do código. Durante a manutenção, os modelos continuam a ser uma referência vital. Eles são usados para entender a estrutura e o comportamento do sistema existente, facilitando a localização e correção de defeitos. Modelos atualizados também são cruciais para a implementação de novas funcionalidades, permitindo que as alterações sejam realizadas sem comprometer a integridade do sistema. Além disso, a documentação contínua através de

modelos ajuda a treinar novos membros da equipe, garantindo que o conhecimento do sistema seja preservado e compartilhado.

Existem várias formas de modelagem de sistemas, incluindo:

- **Modelagem de Dados:** Representação dos dados que serão armazenados e processados pelo sistema.
- **Modelagem de Processos:** Descreve as atividades ou processos que ocorrem no sistema e como eles interagem.
- **Modelagem de Comportamento:** Captura o comportamento dinâmico do sistema, como fluxos de eventos e estados.
- **Modelagem de Arquitetura:** Descreve a estrutura geral e as relações entre os componentes do sistema.

Muitas empresas, profissionais e acadêmicos ainda não realizam a modelagem de sistemas mesmo com os benefícios que citados acima, e existem alguns motivos disso ocorrer:

- **Falta de Tempo e Recursos:** Em projetos com prazos apertados, pode haver uma tendência de pular a fase de modelagem para acelerar o desenvolvimento.
- **Falta de Habilidade ou Conhecimento:** Algumas equipes podem não ter a experiência necessária para realizar uma modelagem eficaz.
- **Percepção de Complexidade:** Algumas pessoas podem considerar a modelagem como uma atividade complexa e demorada, sem perceber os benefícios a longo prazo.

Apesar dos desafios, a modelagem de sistemas é uma prática valiosa que, quando realizada de maneira adequada, contribui significativamente

para o sucesso de projetos de desenvolvimento de software e outros sistemas complexos.

2.4 DIAGRAMAS UML

Os diagramas UML (Unified Modeling Language) são ferramentas visuais amplamente utilizadas na modelagem de sistemas de software. Eles permitem a representação clara e compreensível de diversos aspectos de um sistema, facilitando a comunicação entre desenvolvedores e stakeholders. A UML fornece um conjunto padronizado de notações e diagramas que ajudam na especificação, visualização, construção e documentação dos componentes de um sistema de software.

Os diagramas UML fornecem uma forma padronizada e eficiente de modelar os diferentes aspectos de um sistema de software. Eles são fundamentais para a análise, design e comunicação entre as equipes de desenvolvimento, garantindo uma compreensão clara e compartilhada dos requisitos e da arquitetura do sistema. A escolha dos diagramas apropriados depende da fase do desenvolvimento e dos aspectos específicos do sistema que precisam ser modelados.

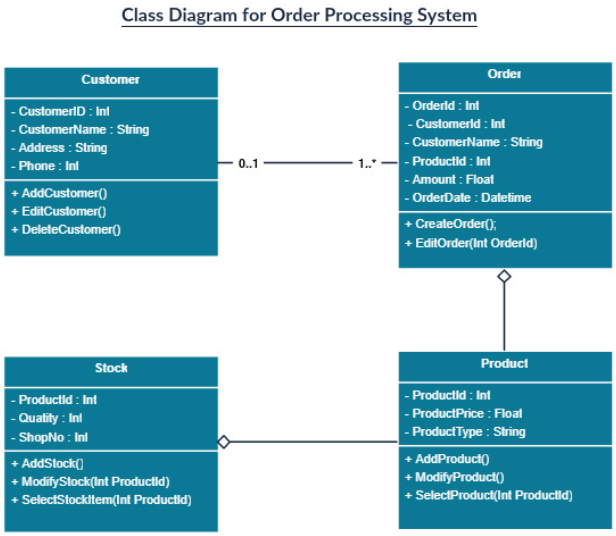
Os principais diagramas da UML são divididos em duas categorias: diagramas estruturais e diagramas comportamentais. Os diagramas estruturais incluem o Diagrama de Classes, o Diagrama de Objetos, o Diagrama de Pacotes, o Diagrama de Componentes e o Diagrama de Implantação. Já os diagramas comportamentais compreendem o Diagrama de Casos de Uso, o Diagrama de Sequência, o Diagrama de Colaboração, o Diagrama de Atividades e o Diagrama de Estados.

2.4.1 Diagramas Estruturais

A. Diagrama de Classes:

- Representa a estrutura estática do sistema mostrando classes, atributos, métodos e os relacionamentos entre elas.
- Útil para detalhar a arquitetura do sistema e definir as responsabilidades de cada classe.

Figura 2 : Exemplo de diagrama de classes

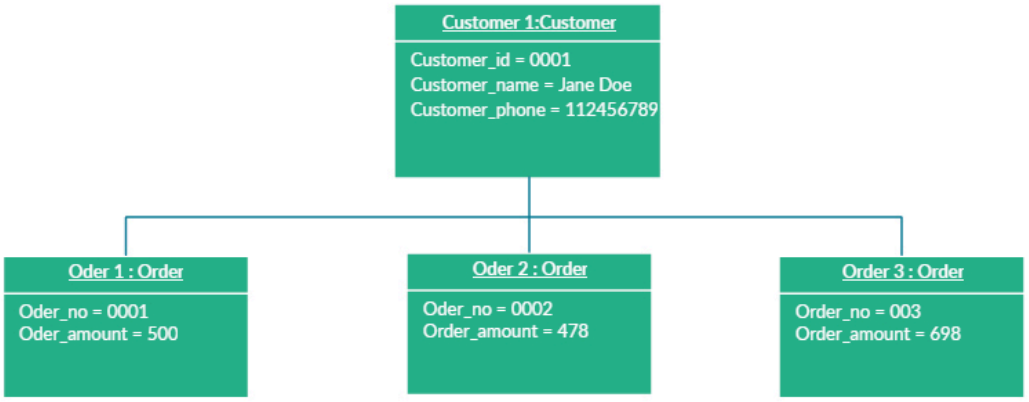


Fonte: CREATELY, 2022

B. Diagrama de Objetos:

- Exibe uma instância de um diagrama de classes em um ponto específico no tempo.
- Ajuda a ilustrar como os objetos interagem entre si e seus estados.

Figura 3 : Exemplo de diagrama de objetos

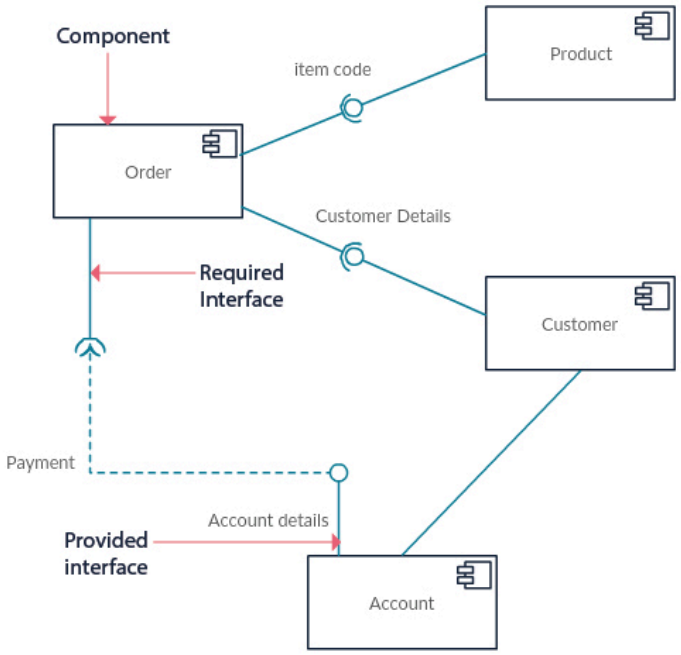


Fonte: CREATELY, 2022

C. Diagrama de Componentes:

- Mostra a organização e dependência dos componentes de software.
- Ideal para modelar a estrutura física do código, incluindo bibliotecas e módulos.

Figura 4 : Exemplos de diagrama de componentes

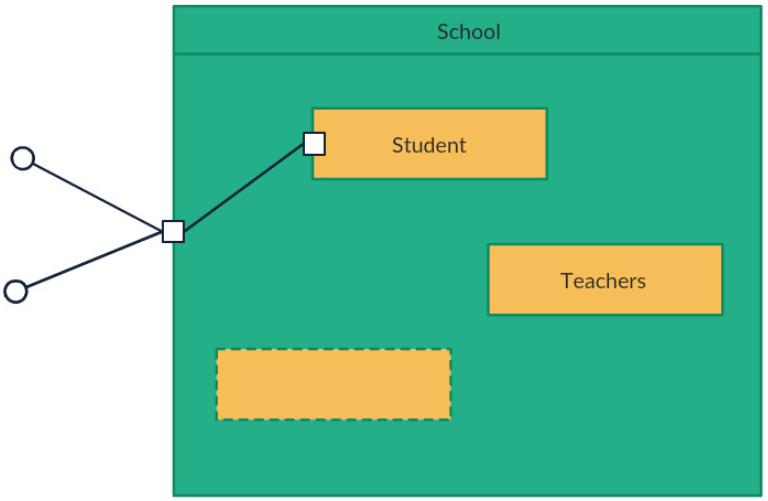


Fonte: CREATELY, 2022

D. Diagrama de Estrutura Composta:

- Descreve a estrutura interna de uma classe e como seus componentes colaboram para realizar comportamentos.
- Útil para representar detalhes internos de uma classe complexa.

Figura 5: Exemplo de diagrama de estrutura composta

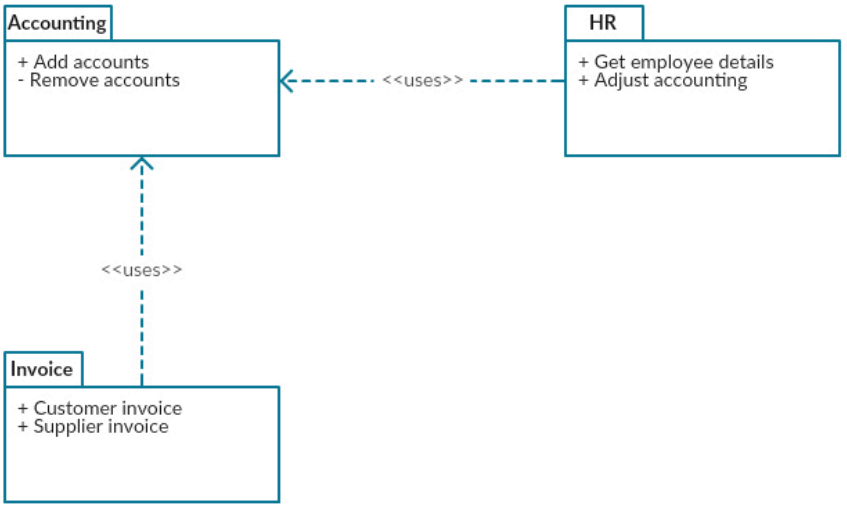


Fonte: CREATELY, 2022

E. Diagrama de Pacotes:

- Agrupa elementos relacionados em pacotes, mostrando a organização e a hierarquia dos componentes do sistema.
- Facilita a gestão de grandes sistemas, organizando-os em módulos.

Figura 6: Exemplo de diagrama de pacotes

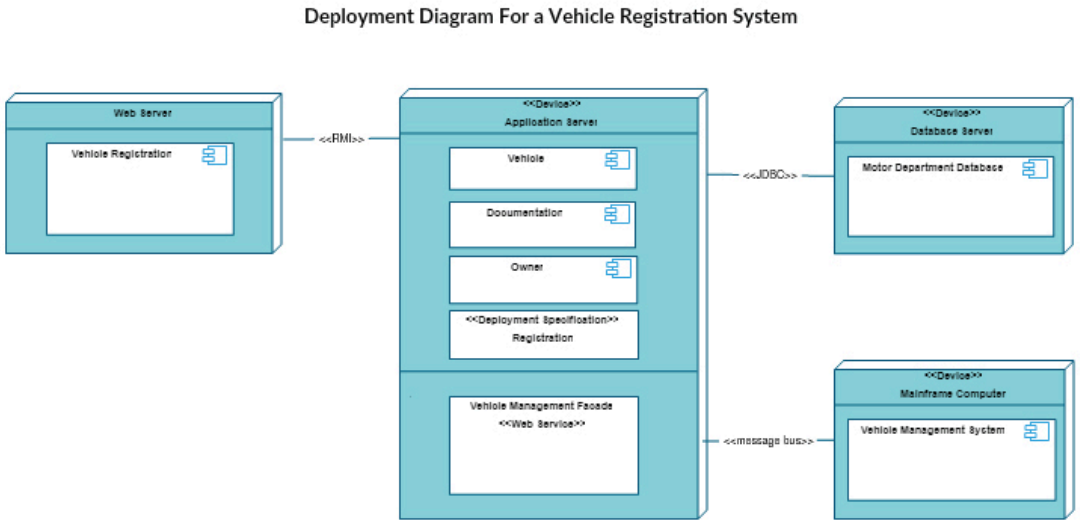


Fonte: CREATELY, 2022

F. Diagrama de Implantação:

- Representa a disposição física dos artefatos do sistema em nós de hardware.
- Essencial para entender a arquitetura física e a distribuição do sistema.

Figura 7: Exemplo de diagrama de implantação



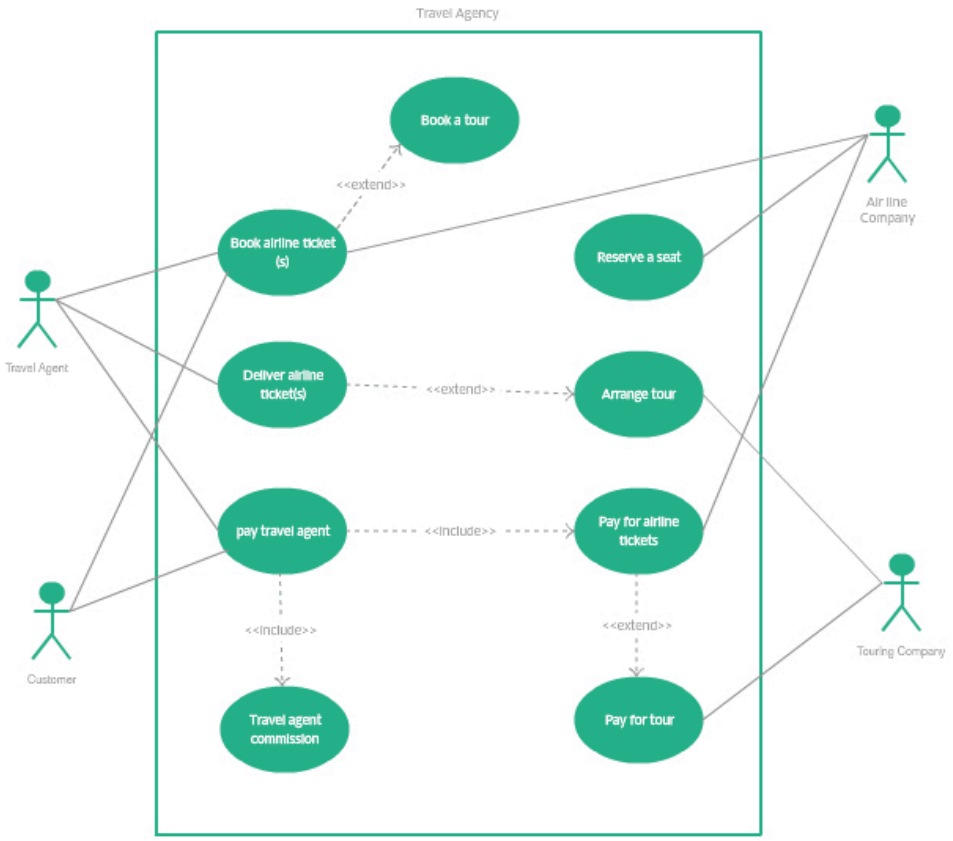
Fonte: CREATELY, 2022

2.4.2 Diagramas Comportamentais

A. Diagrama de Casos de Uso:

- Mostra a interação entre os atores (usuários ou outros sistemas) e os casos de uso (funcionalidades) do sistema.
- Importante para capturar os requisitos funcionais e ilustrar como os usuários irão interagir com o sistema.

Figura 8 : Exemplo de diagrama de caso de uso



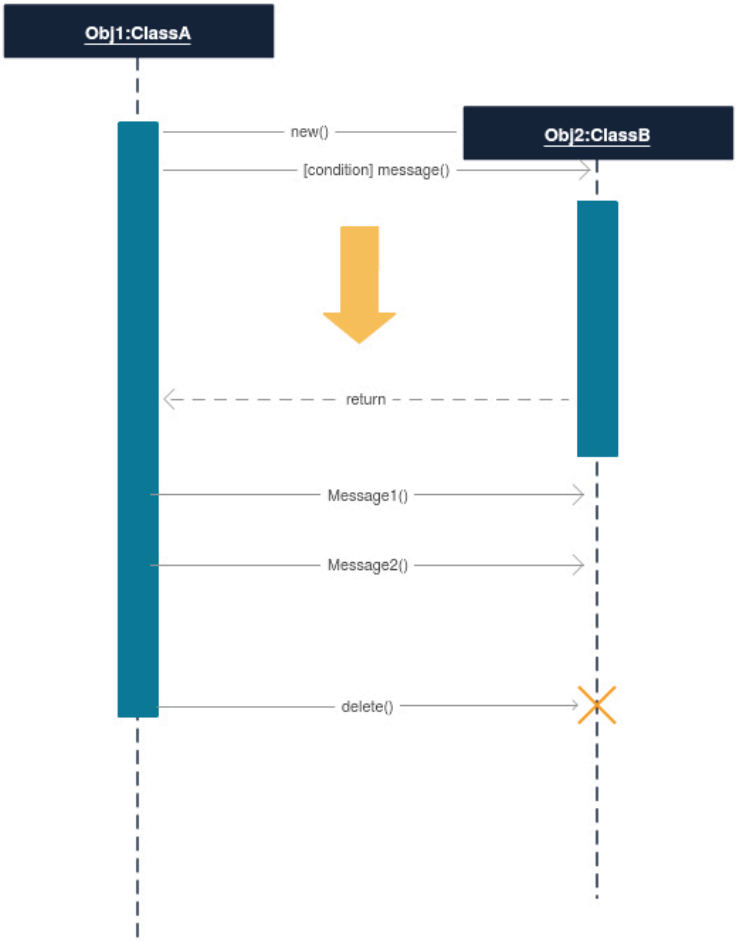
Fonte: CREATELY, 2022

B. Diagrama de Sequência:

- Descreve como os objetos interagem em um fluxo temporal, mostrando a troca de mensagens entre eles.

- Útil para detalhar o comportamento dinâmico do sistema em cenários específicos.

Figura 9: Exemplo de diagrama de sequência

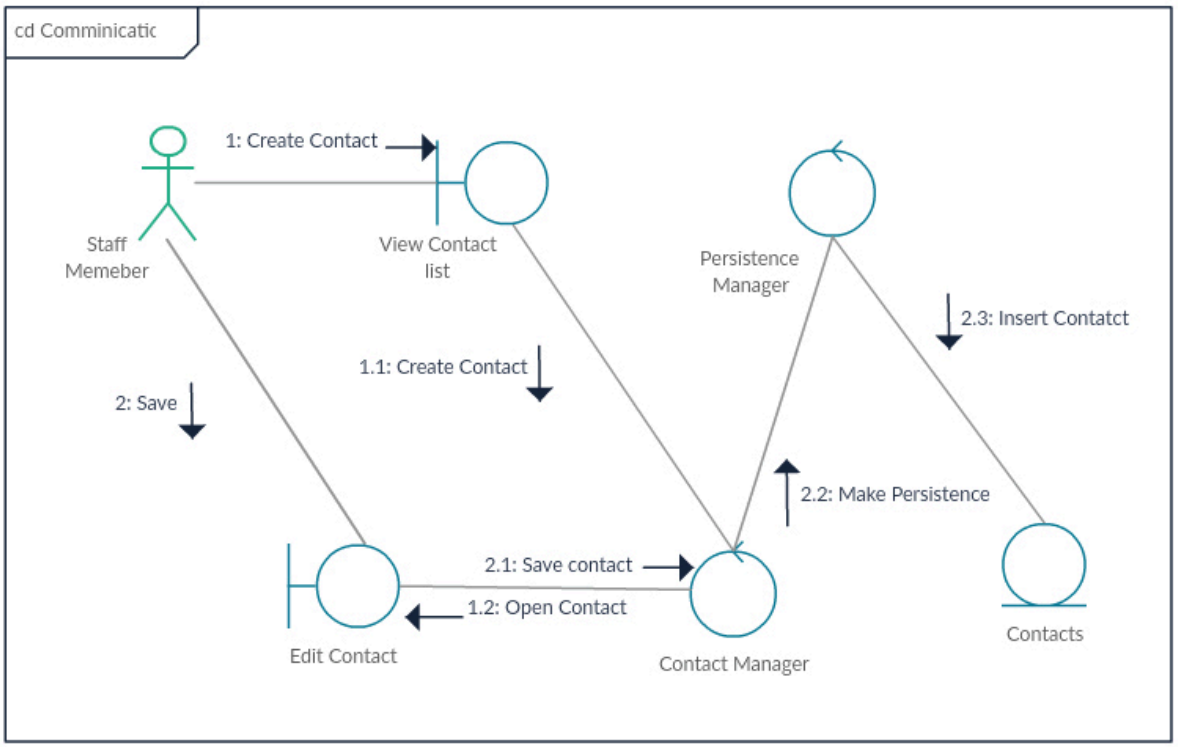


Fonte: CREATELY, 2022

C. Diagrama de Colaboração:

- Foca na interação entre objetos e suas ligações, destacando as colaborações que realizam um comportamento.
- Complementar ao diagrama de sequência, mas enfatiza a estrutura das colaborações.

Figura 10: Exemplo de diagrama de colaboração

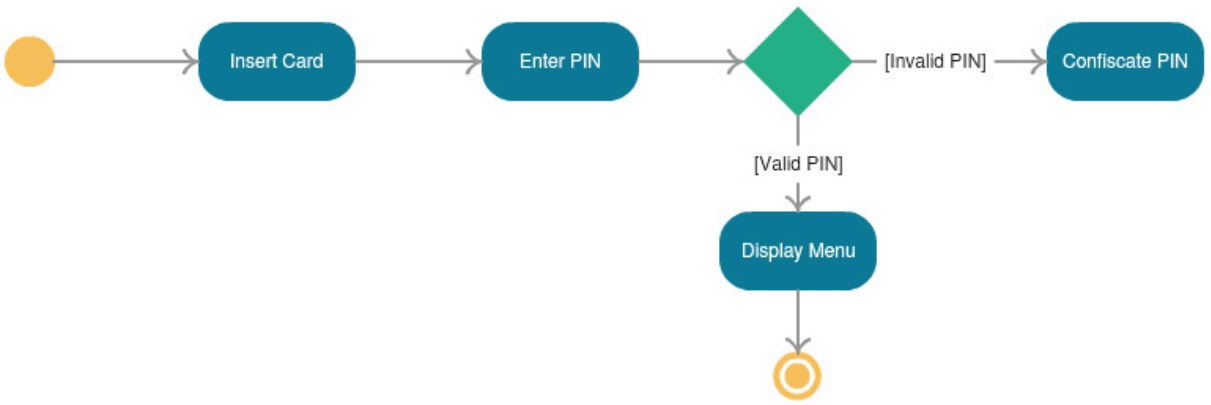


Fonte: CREATELY, 2022

D. Diagrama de Atividades:

- Representa fluxos de atividades e ações no sistema, incluindo decisões, paralelismos e fluxos de controle.
- Importante para modelar processos de negócio e algoritmos complexos.

Figura 11: Exemplo de diagrama de atividades

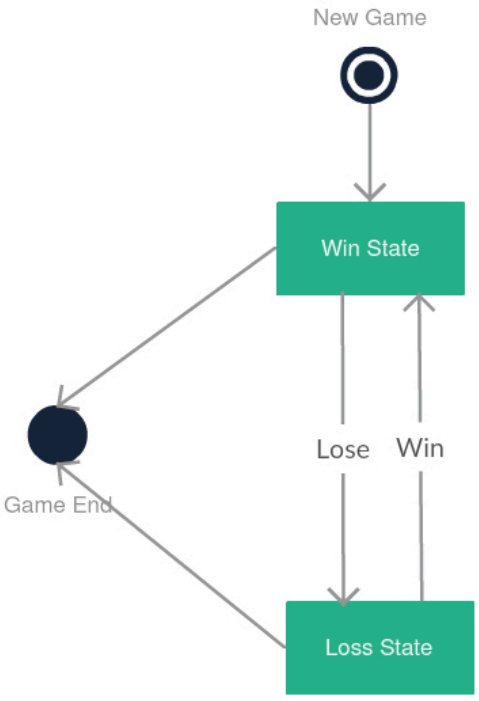


Fonte: CREATELY, 2022

E. Diagrama de Estados:

- Mostra os estados pelos quais um objeto passa durante seu ciclo de vida e os eventos que causam a transição entre esses estados.
- Útil para modelar o comportamento de objetos reativos e de controle.

Figura 12: Exemplo de diagrama de estados

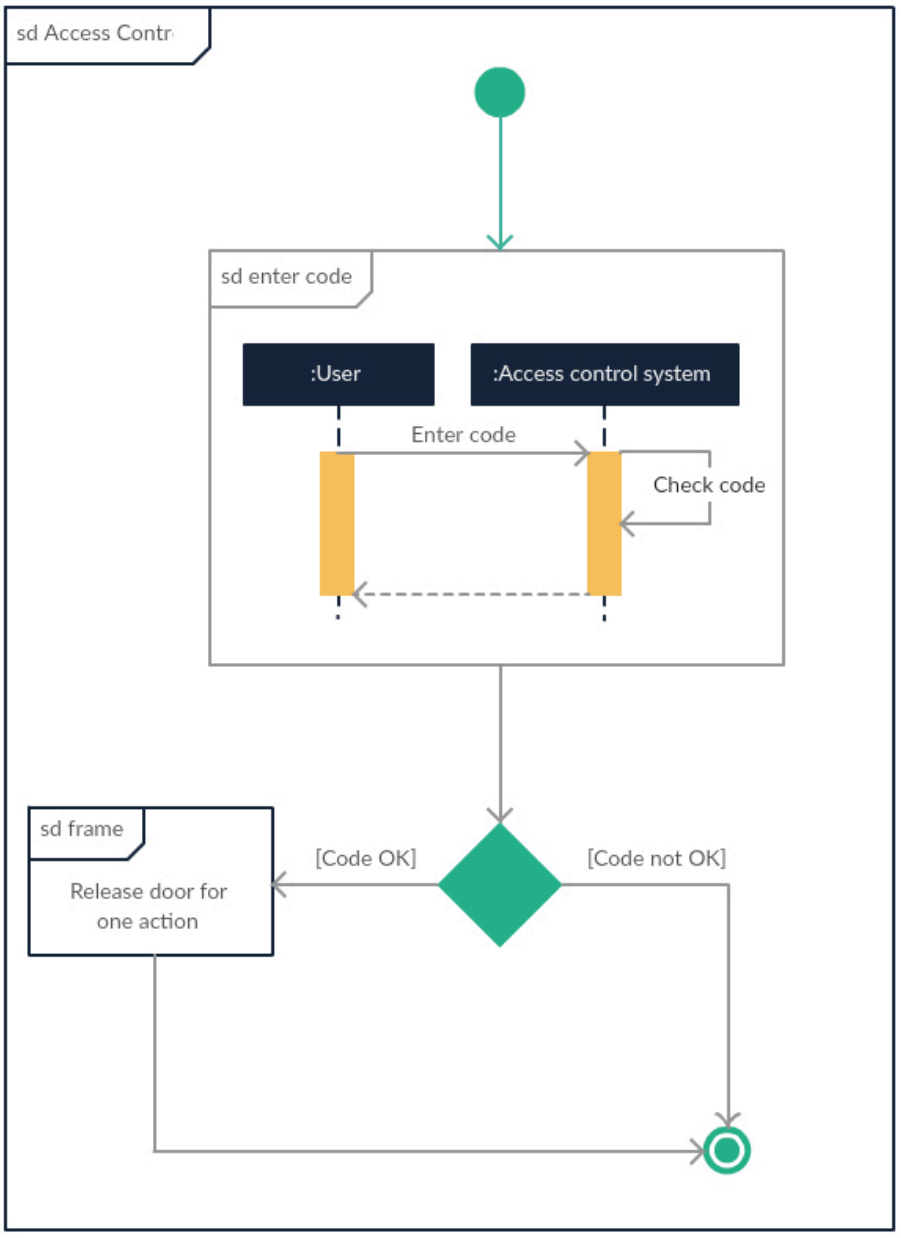


Fonte: CREATELY, 2022

F. Diagrama de Interação Geral:

- Combina aspectos de diagramas de sequência e diagramas de atividades para mostrar interações em um nível mais alto.
- Útil para visualizar a interação geral entre vários objetos ao longo de um cenário.

Figura 13: Exemplo de diagrama de Interação geral

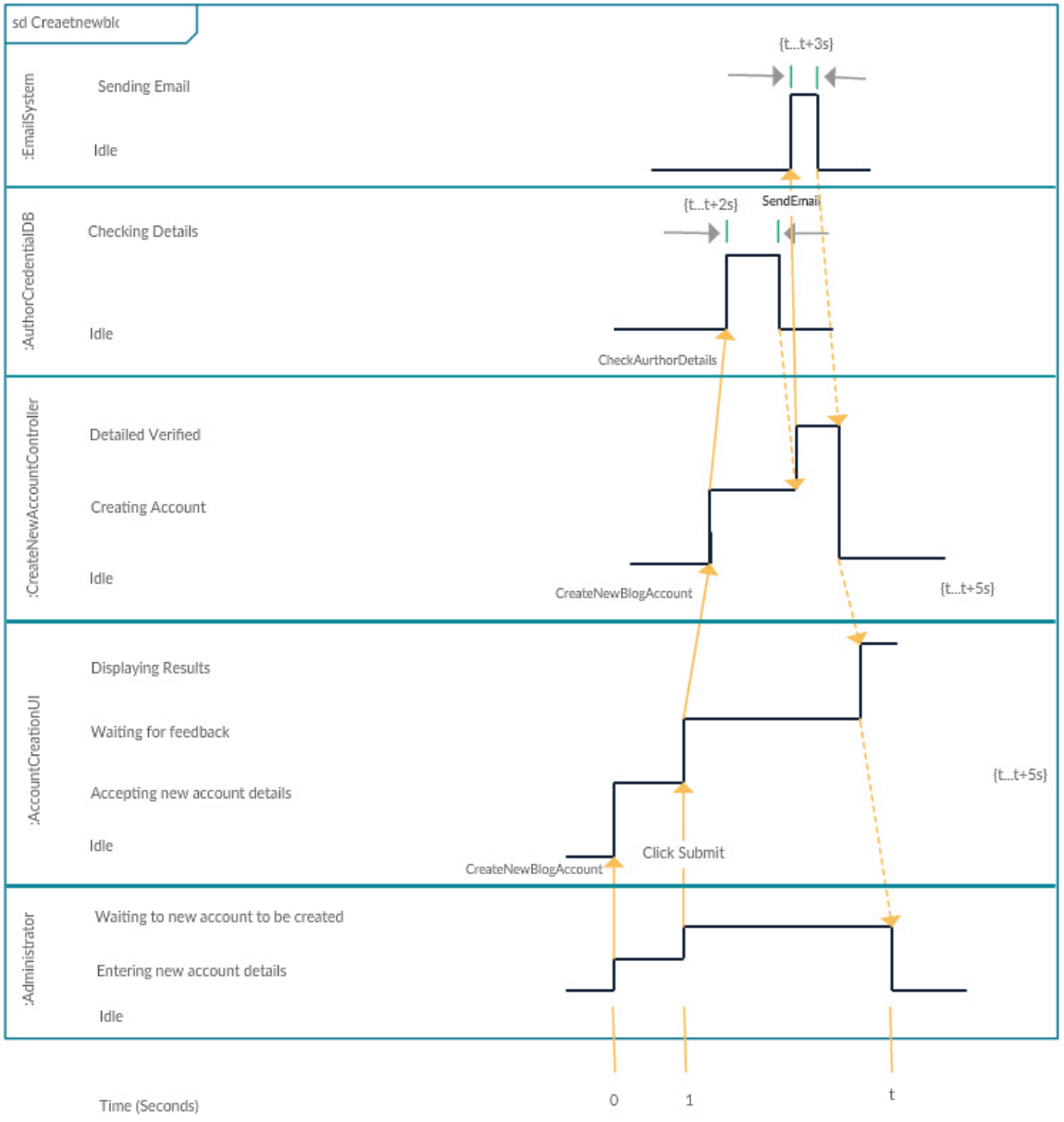


Fonte: CREATELY, 2022

G. Diagrama de Tempo:

- Representa a mudança de estados de um ou mais objetos ao longo do tempo.
- Essencial para sistemas onde o tempo é um fator crítico, como sistemas em tempo real.

Figura 14 : Exemplo de diagrama de tempo



Fonte: CREATELY, 2022

2.5 FERRAMENTAS PARA DOCUMENTAÇÃO DE SOFTWARE

Neste tópico exploraremos algumas das ferramentas mais conhecidas no mercado para criação de diagramas UML e, em seguida, discutiremos o Writerside da JetBrains como uma ferramenta de documentação excepcional.

2.5.1 Visual Paradigm

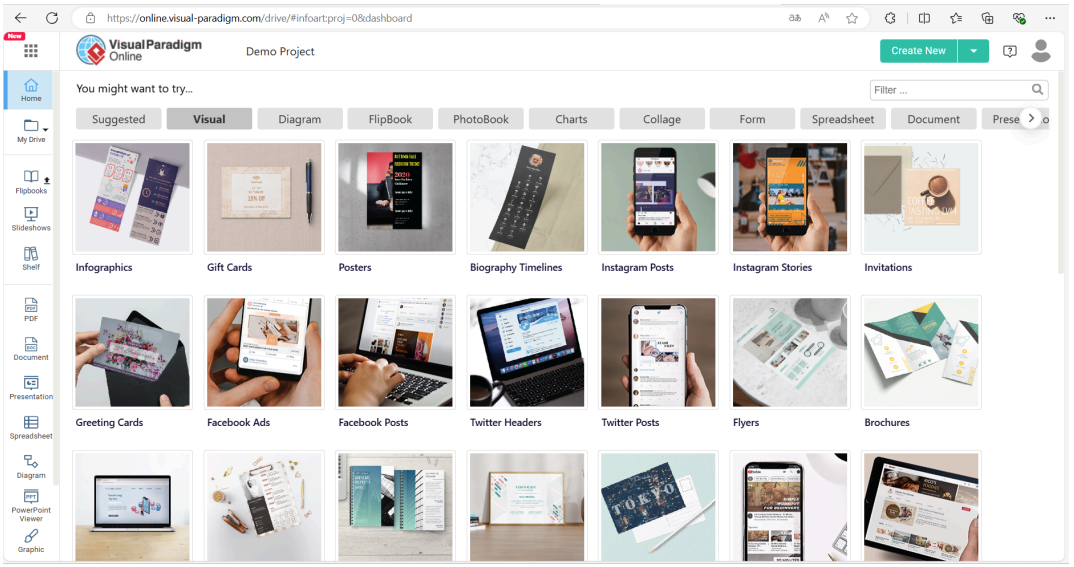
Visual Paradigm é uma ferramenta poderosa e abrangente para a criação de diagramas UML (figura 15). Ela oferece uma ampla gama de recursos, incluindo:

Modelagem UML: Suporte completo para todos os tipos de diagramas UML, como diagramas de classe, sequência, caso de uso, estado e muito mais.

Integração com IDEs: Integração com IDEs populares como Eclipse e IntelliJ IDEA, facilitando a sincronização entre o código e os modelos UML.

Colaboração em Equipe: Recursos robustos para trabalho colaborativo, permitindo que várias pessoas trabalhem simultaneamente em projetos e revisem mudanças.

Figura 15 : Visual Paradigm



Fonte: Visual Paradigm

2.5.2 Lucidchart

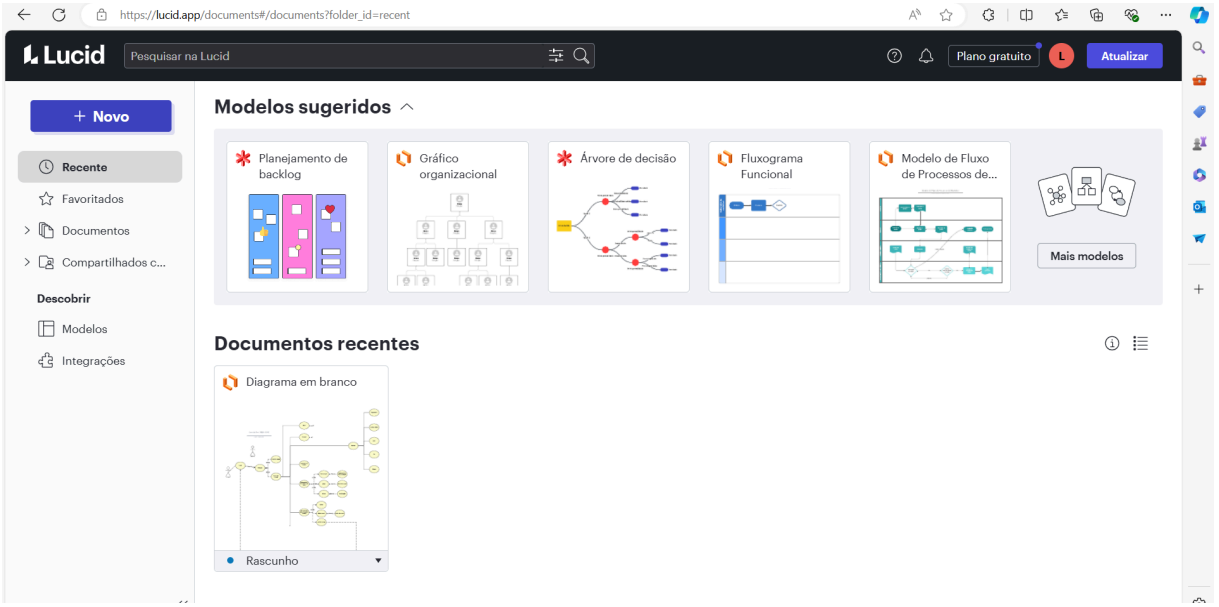
Lucidchart é uma ferramenta baseada em nuvem que se destaca pela sua facilidade de uso e flexibilidade (figura 16). Entre seus principais recursos estão:

Interface Intuitiva: Interface drag-and-drop que facilita a criação de diagramas complexos sem a necessidade de um treinamento extenso.

Colaboração em Tempo Real: Permite que equipes colaborem em tempo real em documentos compartilhados, promovendo uma comunicação eficiente.

Integrações: Integra-se com várias outras ferramentas, como Google Drive, Slack, e Atlassian, facilitando o fluxo de trabalho.

Figura 16 : Lucidchart

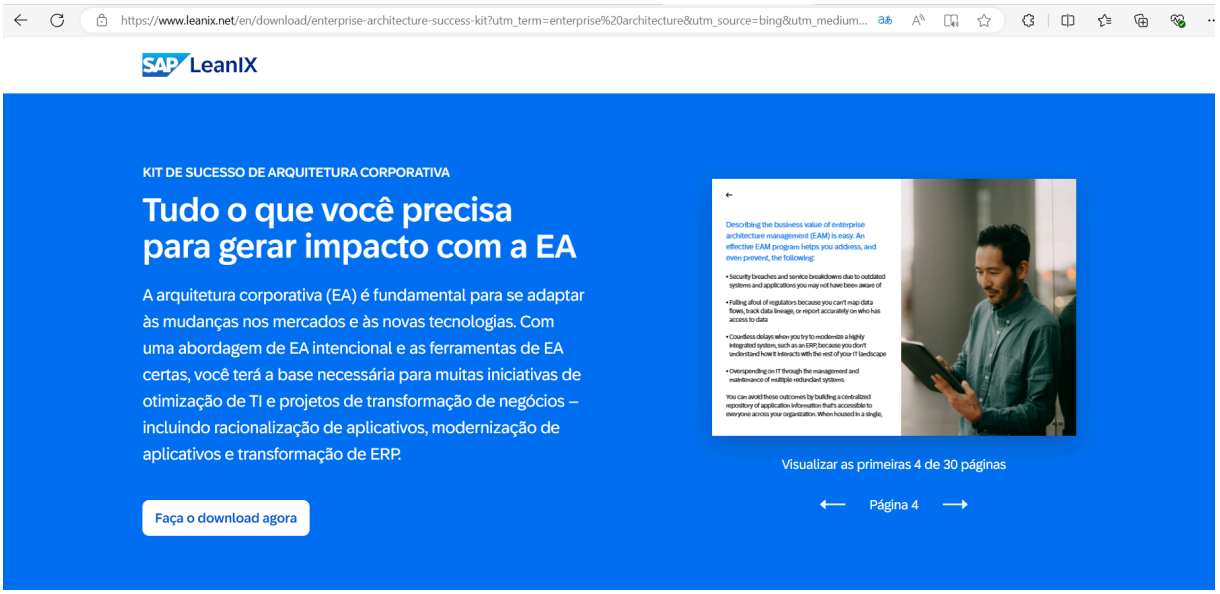


Fonte : Lucidchart

2.5.3 Enterprise Architect

Enterprise Architect é uma ferramenta de modelagem UML robusta e altamente configurável, adequada para grandes projetos e ambientes corporativos. (figura 17)

Figura 17 : Enterprise Architect



Fonte: Enterprise Architecture

Seus principais recursos incluem:

Amplitude de Modelagem: Suporte para UML, BPMN, SysML e outros padrões de modelagem.

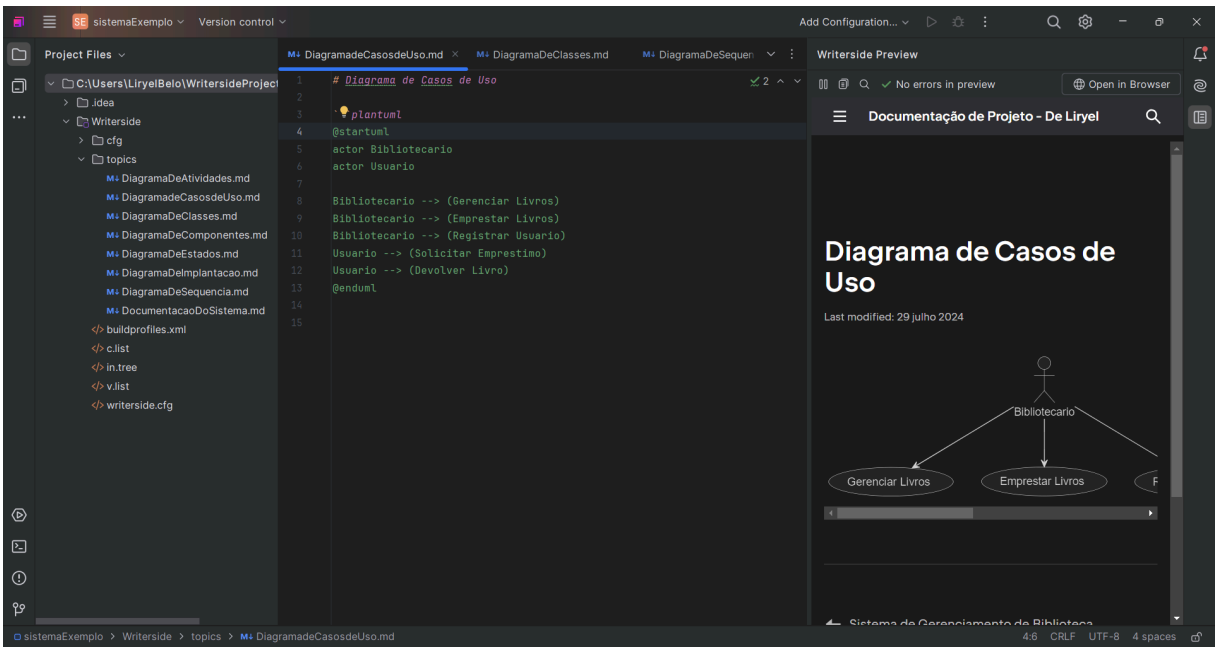
Gerenciamento de Requisitos: Ferramentas integradas para capturar, gerenciar e rastrear requisitos ao longo do ciclo de vida do projeto.

Simulação e Análise: Capacidades avançadas de simulação e análise para validar e otimizar modelos.

2.5.4 Writerside

Writerside é uma ferramenta gratuita da JetBrains que se destaca pela sua capacidade de facilitar a documentação de software de forma eficiente e integrada. Abaixo, discutimos alguns dos benefícios que fazem do Writerside uma excelente escolha para documentar software. (figura 18)

Figura 18 : Writerside



- **Integração com Repositórios de Código:** Writerside permite que a documentação seja versionada juntamente com o código fonte em repositórios Git. Isso garante que a documentação esteja sempre sincronizada com o estado atual do software, facilitando o controle de versões e o histórico de mudanças.
- **Facilidade de Escrita e Formatação:** A ferramenta oferece uma interface de escrita intuitiva e poderosa, com suporte a Markdown, permitindo que os desenvolvedores escrevam e formatem documentação de maneira rápida e eficaz.
- **Colaboração e Revisão:** Writerside suporta colaboração em equipe, com recursos para revisão de documentos, comentários e controle de versões. Isso promove um ambiente de trabalho colaborativo, onde os membros da equipe podem contribuir e revisar a documentação de forma organizada.
- **Publicação e Distribuição:** A ferramenta facilita a publicação da documentação em diferentes formatos e plataformas, permitindo que ela seja acessível a todos os stakeholders do projeto. Pode-se gerar documentação em HTML, PDF, entre outros formatos.
- **Personalização e Extensibilidade:** Writerside é altamente personalizável, permitindo a criação de templates e a extensão das funcionalidades de acordo com as necessidades específicas do projeto ou da organização.

Em resumo, o Writerside da JetBrains se destaca como uma ferramenta excepcional para documentação de software devido à sua integração com repositórios de código, facilidade de uso, recursos de colaboração e revisão, e capacidades de publicação versáteis. Esses atributos fazem do Writerside

uma escolha ideal para equipes de desenvolvimento que buscam manter uma documentação precisa, atualizada e acessível.

3. METODOLOGIA

Para a realização deste trabalho foi preciso uma pesquisa inicial sobre metodologias ágeis, modelagem de sistema, como utilizá-las e como estão aplicadas no mercado de trabalho hoje em dia. Também foi preciso analisar se as documentações são indispensáveis para as empresas, contando com a experiência de modelagem de sistemas, em trabalhos anteriores.

Com as pesquisas em mãos e o mercado analisado, foi preciso um sistema exemplo para que possamos além de argumentar textualmente mas também trazer a prática, para ter uma imagem mais clara de como seria aplicada a estratégia sugerida no mercado real. Sendo assim, o sistema que veremos neste trabalho é fictício e para fins acadêmicos, veremos como modelar o sistema apresentado e usaremos uma metodologia ágil em todo o processo.

4. RESULTADO

Neste tópico, apresentarei o resultado de uma estratégia de documentação UML dentro de uma metodologia ágil, especificamente utilizando o SCRUM. Através do exemplo de um sistema para bibliotecas, demonstrarei como documentar eficientemente utilizando o Writerside da JetBrains. Explicarei quem é responsável por essa documentação e em quais etapas do SCRUM ela deve ser realizada.

4.1 CONTEXTO E ESCOLHA DA FERRAMENTA

No desenvolvimento ágil, a documentação é frequentemente vista como um desafio devido à natureza iterativa e incremental das metodologias. No entanto, a documentação UML continua sendo crucial para garantir a clareza do projeto e facilitar a comunicação entre os membros da equipe. Para esse fim, foi escolhido o Writerside da JetBrains, uma ferramenta que integra perfeitamente a documentação com repositórios de código e oferece uma interface amigável e colaborativa.

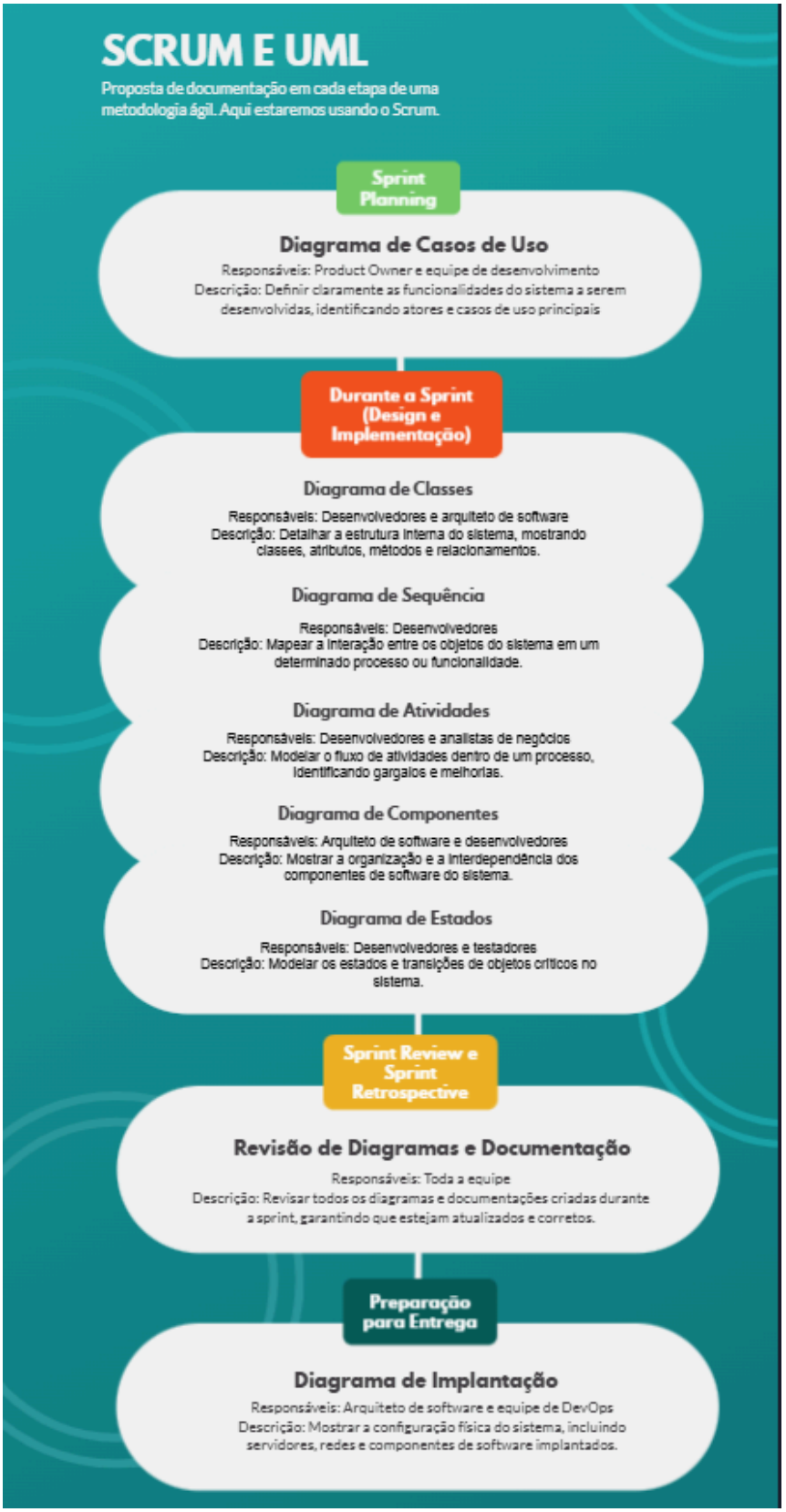
4.2 RESULTADO DA ESTRATÉGIA DE MODELAGEM DE SISTEMA EM PROJETOS SCRUM

Neste projeto, busca-se integrar a modelagem de sistemas a uma metodologia ágil, nesse caso o framework SCRUM, para tornar o processo mais claro e compreensível para todos os envolvidos. A ideia é usar a modelagem para mostrar de forma eficaz como os diferentes componentes do sistema se conectam e funcionam juntos.

Ao aplicar a metodologia ágil SCRUM, o objetivo é tornar o planejamento, a execução e a entrega mais flexíveis e adaptáveis às mudanças. Essa abordagem ajuda a responder rapidamente a novas demandas e ajustes ao longo do projeto, além de promover uma colaboração mais eficaz entre as equipes.

Para apoiar essa metodologia, será criado um guia prático (figura 19) e acessível, útil em todas as etapas do projeto. Esse guia servirá como um recurso que orienta e apoia desde a concepção inicial até a entrega final, com o intuito de aumentar a eficiência e garantir que todos na equipe estejam alinhados e bem informados ao longo do desenvolvimento.

Figura 19 : Linha do Tempo do Projeto com SCRUM



Fonte: Autor(a): Liryel Belo Aguiar, 2024.

4.2.1 Sprint Planning

Durante o planejamento da sprint, é crucial definir as funcionalidades do sistema a serem desenvolvidas. O Diagrama de Casos de Uso é utilizado para identificar os atores e os principais casos de uso, com responsabilidade do Product Owner e da equipe de desenvolvimento. Esse diagrama ajuda a estabelecer uma visão clara das interações e requisitos.

Figura 20 : Etapa de Sprint Planning

- **Diagrama de Casos de Uso**

  Responsáveis: Product Owner e equipe de desenvolvimento

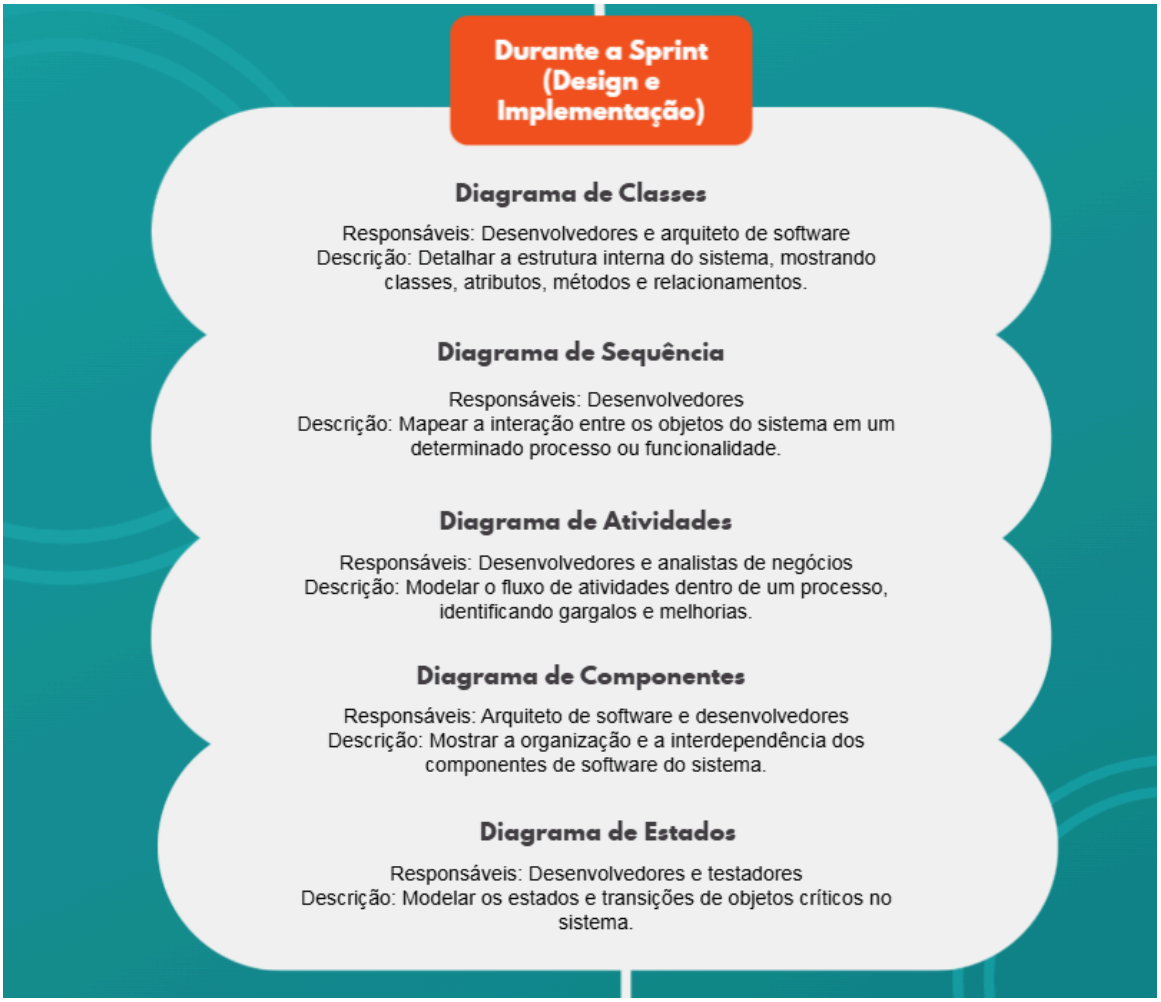
  Descrição: Definir claramente as funcionalidades do sistema a serem desenvolvidas, identificando atores e casos de uso principais.

4.2.2 Durante a Sprint (Design e Implementação)

Durante a sprint, vários diagramas são usados para detalhar o desenvolvimento. O Diagrama de Classes, feito por desenvolvedores e arquitetos, representa a estrutura interna do sistema, detalhando classes e relacionamentos. O Diagrama de Sequência mostra a interação entre

objetos e o fluxo de mensagens. O Diagrama de Atividades modela o fluxo de processos e ajuda a identificar gargalos. O Diagrama de Componentes ilustra a organização e interdependência dos componentes de software. Por fim, o Diagrama de Estados mapeia os estados e transições dos objetos críticos do sistema.

Figura 21 : Etapa da Sprint



- **Diagrama de Classes**

  Responsáveis: Desenvolvedores e arquiteto de software

  Descrição: Detalhar a estrutura interna do sistema, mostrando classes, atributos, métodos e relacionamentos.

- **Diagrama de Sequência**

  Responsáveis: Desenvolvedores

Descrição: Mapear a interação entre os objetos do sistema em um determinado processo ou funcionalidade.

- **Diagrama de Atividades**

  Responsáveis: Desenvolvedores e analistas de negócios

  Descrição: Modelar o fluxo de atividades dentro de um processo, identificando gargalos e melhorias.

- **Diagrama de Componentes**

  Responsáveis: Arquiteto de software e desenvolvedores

  Descrição: Mostrar a organização e a interdependência dos componentes de software do sistema.

- **Diagrama de Estados**

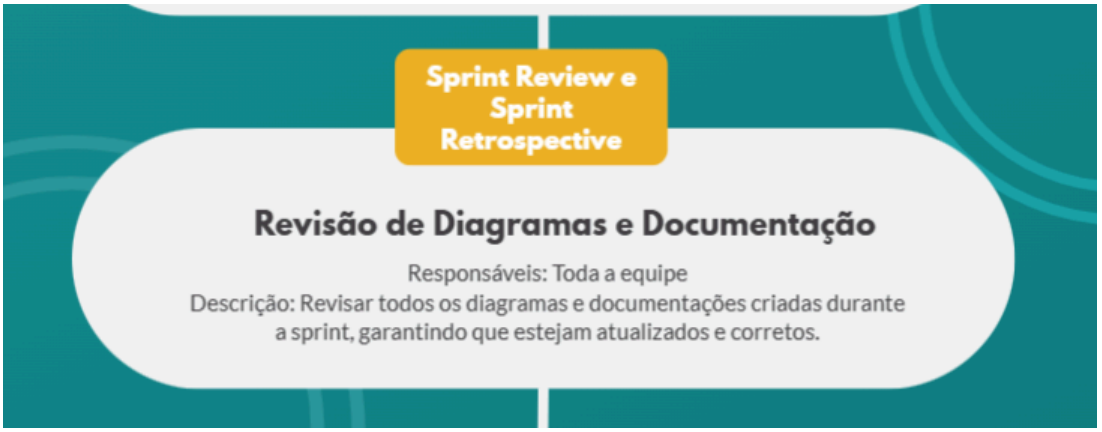
  Responsáveis: Desenvolvedores e testadores

  Descrição: Modelar os estados e transições de objetos críticos no sistema.

4.2.3 Sprint Review e Sprint Retrospective

Na revisão e retrospectiva, toda a equipe revisa os diagramas e a documentação criados durante a sprint. O objetivo é garantir que todos os documentos estejam atualizados e corretos, assegurando que a base de conhecimento esteja alinhada com o desenvolvimento realizado.

Figura 22 : Etapas da Sprint Review e Sprint Retrospective



- **Revisão de Diagramas e Documentação**

  Responsáveis: Toda a equipe

  Descrição: Revisar todos os diagramas e documentações criadas durante a sprint, garantindo que estejam atualizados e corretos.

4.2.4 Preparação para Entrega

Durante a implantação, o Diagrama de Implantação é utilizado pelo arquiteto de software e pela equipe de DevOps para mostrar a configuração física do sistema. Este diagrama inclui detalhes sobre servidores, redes e componentes de software implantados, assegurando uma visão clara da infraestrutura.

Figura 23 : Etapa da entrega

- **Diagrama de Implantação**

  Responsáveis: Arquiteto de software e equipe de DevOps

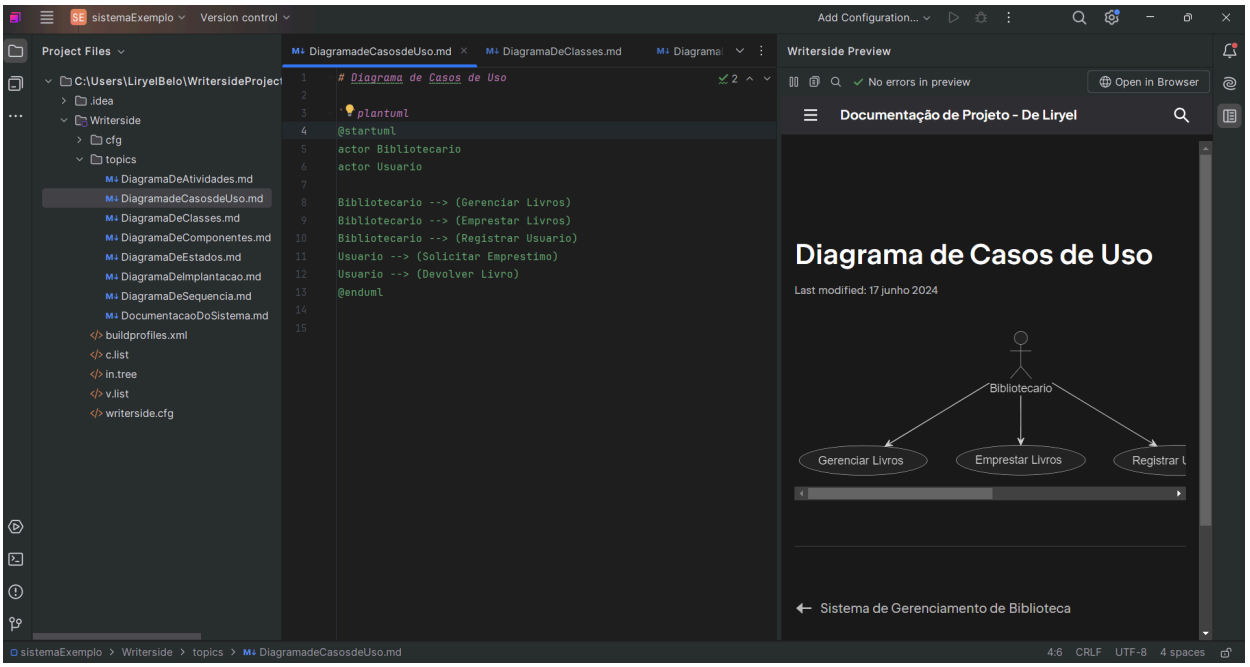
  Descrição: Mostrar a configuração física do sistema, incluindo servidores, redes e componentes de software implantados.

4. 3 DOCUMENTAÇÃO NO WRITERSIDE

3.3.1 Como funciona o Writerside

O Writerside é uma ferramenta de documentação baseada em Markdown, uma linguagem de marcação simples e fácil de aprender que permite criar documentos formatados com texto plano. A ferramenta é integrada ao ecossistema de desenvolvimento da JetBrains, permitindo que a documentação seja versionada junto com o código-fonte nos repositórios Git.(figura 24)

Figura 24: Documentação no Writerside



3.3.2 Documentação

O WriterSide facilita a inserção de diagramas diretamente em textos, permitindo uma integração fluida entre gráficos e conteúdo textual. Com o WriterSide, é possível criar e editar diagramas como fluxogramas, organogramas e diagramas de processos, oferecendo uma maneira prática e eficiente de ilustrar ideias e processos dentro de documentos e relatórios.

Figura 25 : Diagrama de casos de uso no writerside

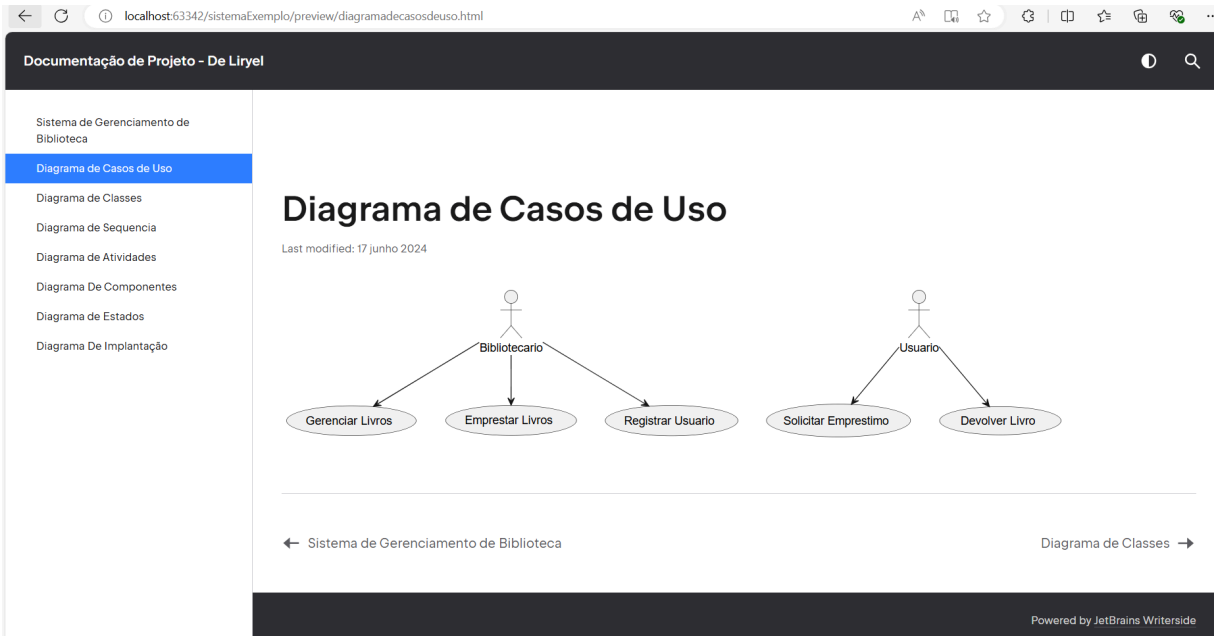


Figura 26 : Diagrama de classes no writerside

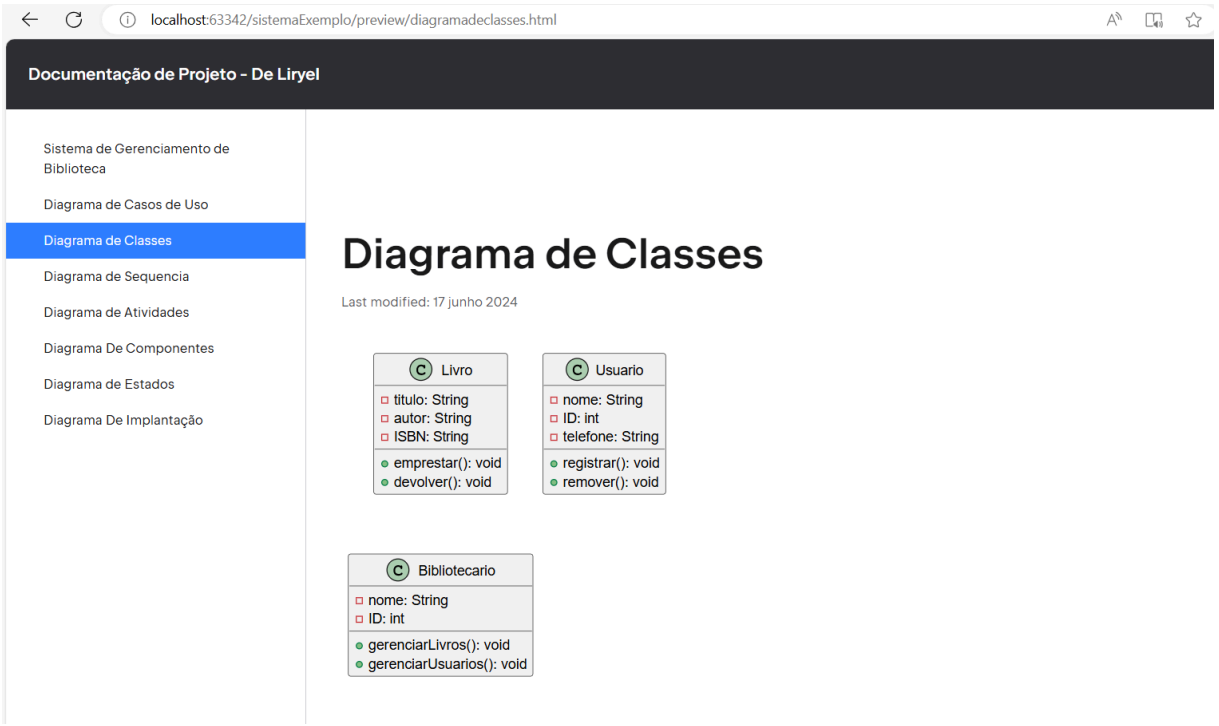


Figura 27 : Diagrama de sequência no writerside

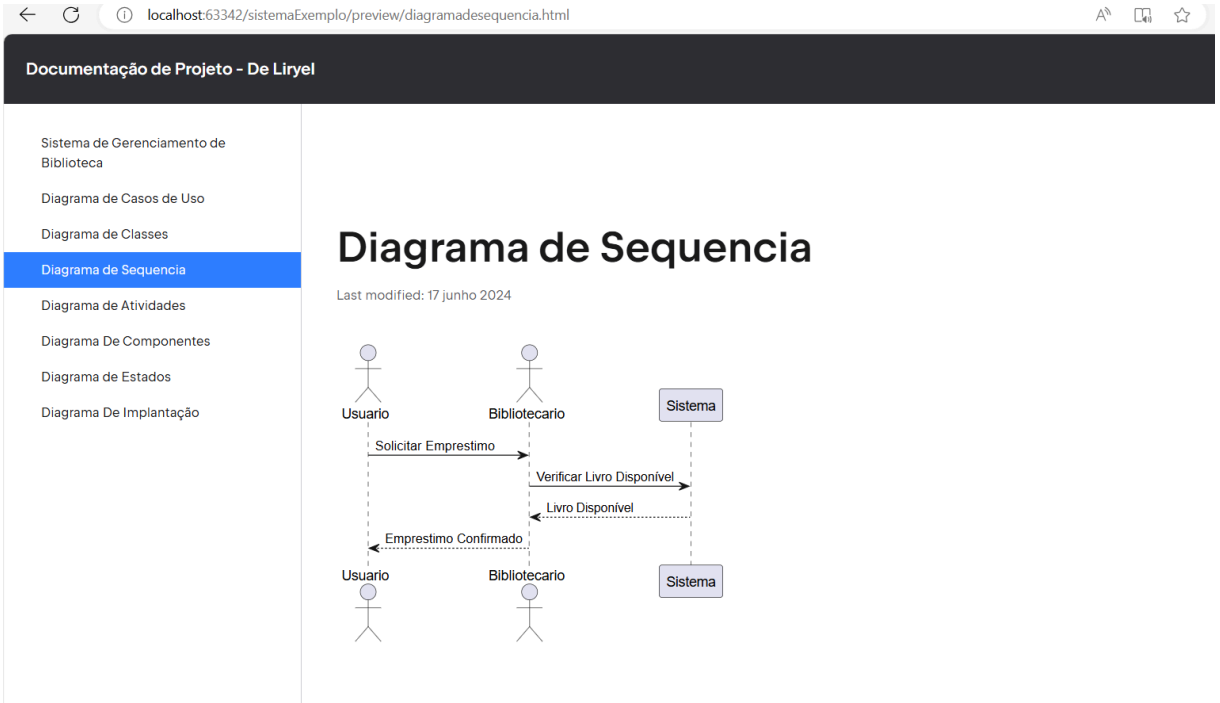


Figura 28 : Diagrama de atividades no writerside

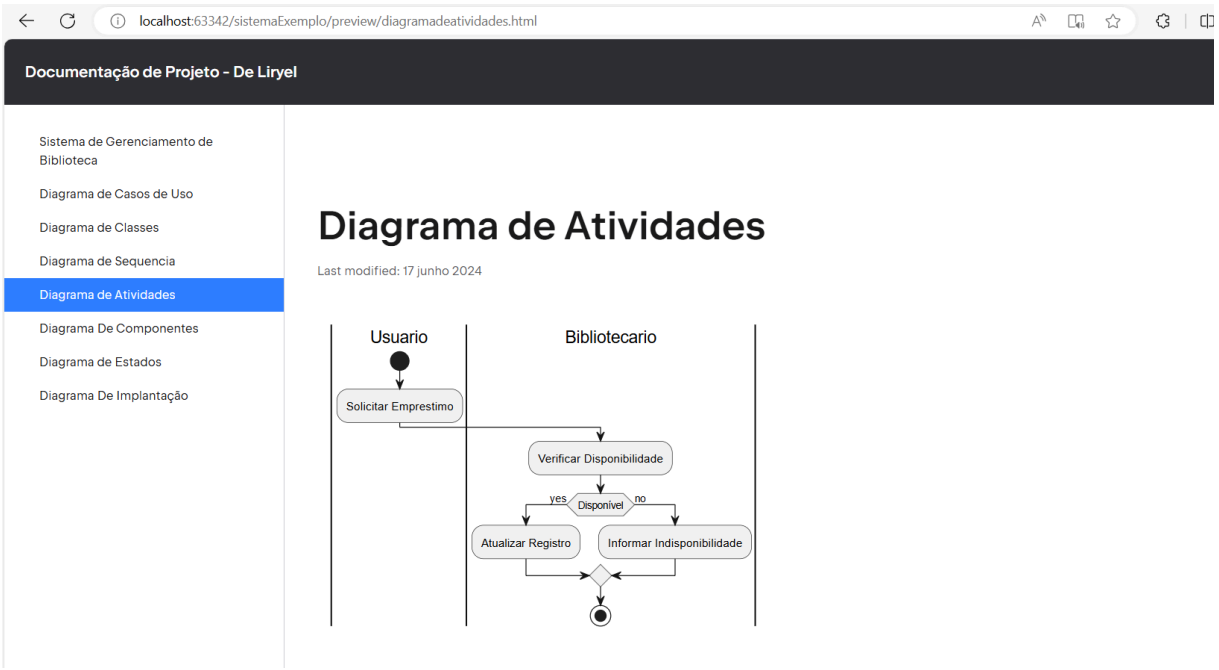


Figura 29 : Diagrama de componentes no writerside

Figura 30: Diagrama de estados no writerside

Figura 31 : Diagrama de implantação no writerside

O WriterSide é uma ferramenta útil para criar e gerenciar diagramas diretamente em documento, no entanto, possui algumas limitações. A funcionalidade pode ser restrita em comparação com ferramentas especializadas em diagramas, limitando a complexidade e personalização dos diagramas. Além disso, o suporte para integração com outras

ferramentas e formatos pode ser limitado, o que pode afetar a flexibilidade na exportação e compartilhamento dos diagramas.

Apesar dessas limitações, o WriterSide continua a ser uma opção valiosa para a criação de diagramas dentro de documentos, oferecendo uma solução conveniente para muitos casos de uso.

5. CONCLUSÃO

A integração eficaz da modelagem de sistemas em processos ágeis de desenvolvimento de software apresenta desafios, mas também inúmeras oportunidades. Um dos aspectos facilitadores é a repartição das tarefas de documentação entre os membros da equipe. Quando cada integrante sabe exatamente o que fazer em cada etapa do processo, a tarefa de documentar torna-se mais simples e eficiente. Muitas empresas reconhecem a importância da documentação contínua e, por isso, implementam sistemas de bonificação para aqueles que completam as tarefas de documentação.

Para assegurar que a documentação permaneça atualizada, é essencial que esta seja incorporada como parte das tarefas diárias dos membros da equipe. Sempre que houver mudanças significativas, a documentação deve ser imediatamente revisada e atualizada. A documentação contínua não só melhora a comunicação e a coordenação dentro da equipe, mas também garante que todos os envolvidos no projeto tenham acesso à informação mais recente e relevante.

A documentação deve acompanhar todas as etapas do desenvolvimento ágil, servindo como um registro vivo do progresso e das mudanças. Isso não só facilita a manutenção do software a longo prazo, mas também permite uma melhor compreensão do processo de desenvolvimento, contribuindo para a melhoria contínua das práticas e metodologias ágeis.

6. SUGESTÕES PARA TRABALHOS FUTUROS

Com base nas descobertas e insights deste trabalho, há várias formas de levar a pesquisa adiante. Uma possibilidade é aplicar o guia em um projeto real para testar e ajustar suas diretrizes na prática. Por exemplo, alguém poderia usar o guia para desenvolver um projeto, documentar todo o processo e avaliar como as práticas recomendadas se saem no mundo real.

Outra opção é colaborar com startups ou empresas juniores. Esse tipo de parceria pode oferecer uma perspectiva prática sobre como o guia funciona em diferentes contextos e ajudar a refinar as abordagens apresentadas. Documentar essas experiências e os resultados obtidos seria uma excelente maneira de validar e aprimorar o guia.

REFERÊNCIAS

ANÁLISE DE REQUISITOS. O que são diagramas UML. Análise de Requisitos, 2024. Disponível em: https://analisederequisitos.com.br/o-que-sao-diagramas-uml/. Acesso em: 10 mar. 2024.

ATLASSIAN. Scrum. Atlassian, 2024. Disponível em: https://www.atlassian.com/br/agile/scrum. Acesso em: 20 abr. 2024.

COHN, Mike. *Agile Estimating and Planning*. Upper Saddle River: Prentice Hall, 2005.

CREATELY. Guia de tipos de diagramas UML: aprenda sobre todos os tipos de diagramas UML com exemplos. Creately, 2024. Disponível em: https://creately.com/blog/pt/diagrama/guia-de-tipos-de-diagramas-uml-aprenda-sobre-todos-os-tipos-de-diagramas-uml-com-exemplos/#ProfileDiagram. Acesso em: 20 jun. 2024.

DEVMEDIA. Modelagem de software: por que, como e o que deve ser feito. DevMedia, 2024. Disponível em: https://www.devmedia.com.br/modelagem-de-software-porque-como-e-o-que-deve-ser-feito/33794. Acesso em: 25 jul. 2024.

IEBT INNOVATION. Metodologias ágeis: o que são e qual a sua importância. IEBT Innovation, 2024. Disponível em: https://iebtinnovation.com/blog/artigos/metodologias-ageis-o-que-sao-e-qual-a-sua-importancia/. Acesso em: 30 jul. 2024.

JETBRAINS. Writerside. JetBrains, 2024. Disponível em: https://www.jetbrains.com/pt-br/writerside/. Acesso em: 15 fev. 2024.

JONES, Capers. *Estimating Software Costs: Bringing Realism to Estimating*. 2. ed. Redmond: Microsoft Press, 2007.

NORRIS, David. *Why Software Fails: Insights from a Software Engineering Perspective*. Boston: Addison-Wesley, 2012.

PRESSMAN, Roger S. *Engenharia de Software: uma abordagem profissional*. 8. ed. São Paulo: McGraw-Hill, 2014.

SCHWABER, Ken. *Agile Project Management with Scrum*. Microsoft Press, 2004.

SPACE PROGRAMMER. Introduzindo o conceito de modelagem e diagramação. Space Programmer, 2024. Disponível em: https://spaceprogrammer.com/uml/introduzindo-o-conceito-de-modelagem-e-diagramacao/. Acesso em: 12 jun. 2024.

VISME. Visme Dashboard. Visme, 2024. Disponível em: https://dashboard.visme.co/v2/projects/own. Acesso em: 26 jul. 2024.

WORKOVER. Curso de Scrum. Workover, 2024. Disponível em: https://workover.com.br/cursos/228/scrum. Acesso em: 5 mai. 2024.