Índice

1.Introdução 2

2.UML a

2.1 Diagramas de Classes x

2.2 Diagramas de Sequencias x

2.3 Diagramas de Atividades

3. Conclusão x

4. Referências Bibliográficas

5. Referência das Figuras a

1. **Introdução**



Figura 1

Na engenharia de software os requisitos documentam “o que um sistema deve fazer” valendo-se de um nível de abstração próximo do problema e de seus usuários. Por outro lado, o código fonte é uma representação concreta, de baixo nível e executável do comportamento de um sistema. Portanto existe, uma lacuna entre esses dois mundos: requisitos e código fonte. Para preencher essa lacuna, os Engenheiros de Software criam **modelos**, o quais são criados para ajudar no entendimento e análise de um sistema [1]. Os modelos são mais detalhados que os requisitos e menos complexos que os códigos fonte de um sistema.

Importante pontuar que os modelos de software, devido a natureza abstrata deste, são menos efetivos que os modelos matemáticos e físicos utilizados em outras engenharias. Sobre esse ponto transcrevo o ensaio de Frederick Brooks *“Não existe bala de Prata”* [2].

“*A complexidade de um software é uma propriedade essencial e não acidental. Portanto, representações de uma entidade de software que abstraem sua complexidade normalmente também abstraem sua essência. Por três séculos, matemáticos e físicos obtiveram grandes avanços construindo modelos simplificando fenômeno complexo, derivando propriedades de tais modelos e verificando tais propriedades por meios de experimentos. Esse paradigma funcionou porque as complexidades ignoradas não são propriedades essenciais do fenômeno sob estudo. Porém, essa abordagem não funciona quando as complexidades são essenciais”.*

Essa introdução visa “calibrar” a expectativa do leitor em relação as ferramentas de modelagem a serem apresentadas neste trabalho. Iremos apresentar um subconjunto de diagramas propostos pela UML (*Unified Modeling Language*). Vamos descrever um breve histórico em seguida, estudar alguns dos principais diagramas UML .

1. **Unified Modeling Language (UML)**

A Unified Modeling Language (UML) é uma linguagem de modelagem visual usada para descrever sistemas de software complexos. A UML foi criada na década de 1990 por Grady Booch, James Rumbaugh e Ivar Jacobson. Desde então, a UML se tornou uma das linguagens de modelagem mais usadas na indústria de software.

A UML tem sido utilizada em vários processos de desenvolvimento de software, desde modelagem de requisitos até a modelagem de arquitetura de software. Ela oferece uma linguagem visual com uma série de diagramas, cada um com uma finalidade específica.

Neste texto, exploraremos alguns dos principais diagramas UML e sua aplicação na modelagem de sistemas de software.

**ESCREVER SOBRE OS DIAGRAMAS ESTÁTICOS (OU ESTRUTURAIS ) E DINÂMICOS OU COMPORTAMENTAIS**

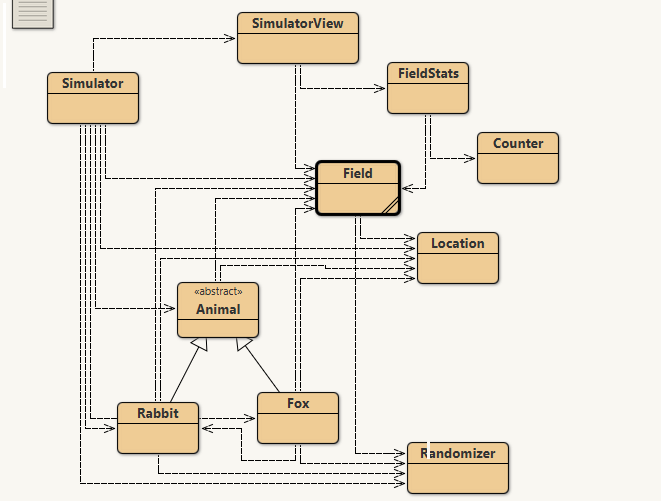
**2.1 Diagrama de Classes**

O Diagrama de Classes é usado para representar a estrutura do sistema. Ele mostra as classes do sistema, seus atributos e métodos, bem como as relações entre elas. O diagrama de classes é uma das principais ferramentas para a modelagem da arquitetura de software.

Uma classe é uma abstração de um objeto do mundo real. Ela define um conjunto de atributos e métodos que descrevem as propriedades e comportamentos do objeto. As relações entre as classes são representadas por meio de linhas que conectam as classes.

Os Diagramas de classes são os mais utilizados da UML. Eles oferecem uma representação gráfica para um conjunto de classes, provendo informações sobre atributos, métodos e relacionamentos que existem entre as classes[1].

A figura abaixo mostra um exemplo de um diagrama de classes.



Figura\_2

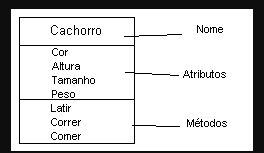
Essa figura foi extraída do exemplo descrito no livro Programação orientada a objetos com Java – Uma introdução prática utilizando o Blue J. O compilador Blue J gera automaticamente ao se compilar um Código em JAVA a imagem com a UML deste software.

Existe uma dependência de uma classe A para uma classe B, representada por uma seta com uma linha tracejada de A para B, quando a classe A usa a classe B, porém esse uso não ocorre por meio de uma associação (isto é, A não é tem um atributo tipo B) ou herança (isto é, A não é uma subclasse de B). Dependências ocorrem, por exemplo, quando um método de A declara um parâmetro ou variável local tipo B ou quando um método A lança uma exceção do tipo B. As linhas cheias representam **heranças** e as tracejadas **dependências** entre as classes.

As linhas representam relação de **herança** entre as famílias e a linha tracejada indica que a classe de origem instância objetos da classe destino ou que chama métodos da classe destino.

Pode-se também representar na UML mais informações sobre as classes como atributos e métodos.

Segue uma Figura com um exemplo



Figura\_3 Exemplo de uma Classe “Cachorro” com seus Atributos e Métodos.

**2.1 Diagrama de Sequências**

Diagramas de sequência são diagramas dinâmicos, O Diagrama de Sequência é usado para representar a interação entre os objetos do sistema ao longo do tempo. Ele mostra a ordem em que as mensagens são trocadas entre os objetos durante a execução do sistema.

Vamos considerar o caso do acesso a uma aplicação *web* por exemplo o *alunoonline* cujos requisitos foram descritos em um trabalho.

## Diagrama de Atividades

O Diagrama de Atividades é usado para modelar um processo de negócios ou uma atividade do sistema. Ele mostra o fluxo de atividades, decisões e eventos do sistema. O diagrama de atividades é muito útil para modelar a lógica de negócios de um sistema, ou seja, como o sistema funciona para atender às necessidades dos usuários.

As atividades são representadas por retângulos, as decisões por losangos e as ações por símbolos específicos. As setas indicam a direção do fluxo de atividades. O diagrama de atividades é especialmente útil para mostrar a ordem das atividades em um processo e para destacar os pontos de decisão.

A figura abaixo mostra um exemplo de um diagrama de atividades.

## Diagrama de Componentes

O Diagrama de Componentes é usado para descrever a estrutura do sistema em termos de seus componentes e suas interações. Ele mostra as partes do sistema e como elas estão conectadas umas às outras. O diagrama de componentes é muito útil para a modelagem de arquiteturas de software complexas.

Um componente é uma unidade de software que pode ser implementada e substituída de forma independente. Os componentes podem ser bibliotecas, módulos ou serviços. As relações entre os componentes são representadas por meio de linhas que conectam os componentes.

A figura abaixo mostra um exemplo de um diagrama de componentes.

O Diagrama de Implantação é usado para descrever como o sistema é implantado em um ambiente de hardware. Ele mostra a relação entre os componentes do sistema e o hardware em que eles são executados. O diagrama de implantação é muito útil para a modelagem de arquiteturas distribuídas.

Os nós são usados para representar o hardware em que os componentes são implantados. Os componentes são colocados nos nós para mostrar como eles são implantados no hardware. As relações entre os componentes e os nós são representadas por meio de linhas que conectam os componentes e os nós.

A figura abaixo mostra um exemplo de um diagrama de implantação.

A UML é uma linguagem de modelagem visual que é amplamente usada na indústria de software. Ela oferece uma linguagem visual com uma série de diagramas, cada um com uma finalidade específica. Os principais diagramas da UML incluem o Diagrama de Caso de Uso, o Diagrama de Classes, o Diagrama de Sequência, o Diagrama de Atividades, o Diagrama de Componentes e o Diagrama de Implantação.

Cada um desses diagramas tem uma finalidade específica e pode ser usado em diferentes etapas do processo de desenvolvimento de software. Por exemplo, o Diagrama de Caso de Uso pode ser usado para descrever os requisitos funcionais do sistema, enquanto o Diagrama de Sequência pode ser usado para descrever a interação entre os objetos do sistema ao longo do tempo. Com a UML, os desenvolvedores podem criar modelos visuais que ajudam a entender o sistema de forma mais clara e completa, reduzindo erros e problemas na implementação. Além disso, a UML ajuda a documentar e comunicar as decisões de design para todas as partes interessadas envolvidas no projeto de desenvolvimento de software.

**Referências Bibliográficas**

**[1]** Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, Marcos Túlio Valente, ASIN ‏ : ‎ B086K5QJ9V

**[2]** O mítico homem-mês: ensaios sobre engenharia de software Capa comum, Frederick P. Brooks Jr, ISBN-13 ‏ : ‎ 978-8550802534

**[3]** Programação Orientada a Objetos com Java: Uma Introdução Prática Usando o BlueJ Capa comum – 28 novembro 2008, David J. Barnes e Michael Kölling ISBN-13 ‏ : ‎ 978-8576051879

**Referências das Figuras**

**Figura 1 –** Disponível em: <http://4.bp.blogspot.com/-PnWH7p-nCMk/UbUOX0w8O-I/AAAAAAAAAFw/JOI45pH38Ak/s1600/topoblog.png>

**Figura 2 –** Disponível em: [**https://www.bluej.org/**](https://www.bluej.org/)

**Figura 3 –** Disponível em <https://www.macoratti.net/net_uml3.gif>