

GILVAN SOUSA DA SILVA RA: 6820508946

LEONARDO CÉSAR VALENTINO COSTA RA: 6247228721

MARCUS ALVES FERREIRA RA: 6277261905

RODRIGO DOURADO NUNES RA: 6818457769

MODELO DE DESENVOLVIMENTO DE SOFTWARE PARA UMA REDE DE DROGARIAS

BRASÍLIA

2016

GILVAN SOUSA DA SILVA

LEONARDO CÉSAR VALENTINO COSTA

MARCUS ALVES FERREIRA

RODRIGO DOURADO NUNES

MODELO DE DESENVOLVIMENTO DE UM SOFTWARE PARA UMA REDE DE DROGARIAS

Projeto apresentado ao Curso em Bacharelado de Sistemas de Informação da Instituição Faculdade De Negócios e Tecnologias da Informação – FACNET Anhanguera  
Orientador: Mariana Monteiro Nunes

BRASÍLIA

2016

**SUMÁRIO**

[**1. INTRODUÇÃO 4**](#_Toc446565869)

[1.1 PROBLEMA 5](#_Toc446565870)

[**2.** **OBJETIVOS** **6**](#_Toc446565871)

[2.1 OBJETIVO GERAL OU PRIMÁRIO 6](#_Toc446565872)

[2.2 OBJETIVOS ESPECÍFICOS OU SECUNDÁRIOS 6](#_Toc446565873)

[**3.** **JUSTIFICATIVA** **7**](#_Toc446565874)

1. INTRODUÇÃO

O projeto trata o modelo de desenvolvimento de um software, exemplificado em um sistema para uma rede de drogarias, onde são abordadas as etapas de desenvolvimento desse sistema seguindo desde concepção até a implantação.

Serão contempladas aqui também as regras de negócio encontradas em uma drogaria real, elucidando assim, as necessidades reais, as quais são muito evidentes, ainda mais se levarmos em conta o momento atual, em que mudanças são cada vez mais rápidas e influenciadoras, o que já traz a necessidade de utilizar ferramentas de software que sejam capazes de prestar auxílio.

Será levada em conta uma série de padrões ditados pela engenharia de software, os quais farão parte do tema central ser pesquisado. De acordo com a evolução do projeto, cada padrão que for utilizado será explicado por meio de pesquisa, para isso, buscaremos explicar o porquê utilizar determinado padrão, mostrando assim, as vantagens de se fazer uso desses padrões.

Este projeto pretende aplicar as melhores práticas da engenharia de software para resolver o problema da rede de drogarias, de forma a mostrar a importância da engenharia de software para que se possa ter um sistema de qualidade. Além de demonstrar como a sociedade envolvida no desenvolvimento de sistemas pode se beneficiar destas técnicas, como empresas, usuários de sistemas e estudantes. E tentar esclarecer para todos os estudantes, que venham a ler este projeto, a importância da engenharia de software.

# PROBLEMA

Atualmente os sistemas apresentam muitos problemas, derivados da má concepção e de processos ruins. Como criar sistemas que satisfação os clientes e atendam as especificidades de seus negócios?

1. OBJETIVOS

# OBJETIVO GERAL OU PRIMÁRIO

Desenvolver um sistema para uma rede de drogarias com o uso da engenharia de software.

# OBJETIVOS ESPECÍFICOS OU SECUNDÁRIOS

1. Estudar o negócio do cliente, procurar entender sua forma de trabalho e criar uma linguagem comum entre a equipe técnica e o cliente;
2. Pesquisar sobre sistemas semelhantes, para analisar pontos fortes e fracos, averiguar o que pode ser aproveitado do sistema atual, assim estabelecendo o que de melhor pode ser aplicado para o cliente;
3. Realizar os processos de engenharia de software, que serão estabelecidos para melhor atender ao cliente;
4. Implementar e testar o sistema.
5. JUSTIFICATIVA

As mudanças no atual cenário corporativo estão sempre acontecendo, surge então, a necessidade de as empresas possuírem sistemas de informações que sejam capazes de se adequar a essas mudanças. Portanto, sistemas obsoletos que não sejam flexíveis e escaláveis devem ser na medida do possível substituídos. Queremos assim, deixar mais transparente os benefícios trazidos pela modernização dos sistemas utilizados pelas empresas.

Outro alvo importante deste projeto é a importância da utilização dos padrões engenharia de software, por se tratar de padrões aceitos internacionalmente, que já foram testados inúmeras vezes. Construir um software tem suas particularidades, por ser algo mais lógico do que concreto, a necessidade de ter um controle do projeto é ainda mais fundamental, por meio de prazos e custos bem definidos, documentação clara e abrangente, entre muitos outros fatores que são aqui abordados.

É claro que, uma empresa possuir sistemas de informações atuais e desenvolvidos seguindo todos os padrões não garantirá seu bom desempenho, mas, sem dúvida, facilitará enfrentar este mercado cada vez mais difícil, no qual turbulências podem surgir a qualquer momento, ainda mais levando em conta a atual situação da economia do país.

1. FUNDAMENTAÇÃO TEÓRICA

Abordaremos neste tópico o respaldo teórico metodologia que utilizaremos para desenvolver o sistema.

# Engenharia de sofware

Nesta parte abordaremos os principais conceitos da Engenharia de Software respaldado por autores reconhecidos no cenário internacional.

Cada vez mais o mundo ganha um dinamismo que influencia muito a vida das pessoas, a forma obter informações alcançou um nível que no passado era inimaginável, e os sistemas de informações têm um papel fundamental nesse acontecimento. Dessa forma buscar formas de otimizar o seu processo de desenvolvimento torna-se cada vez mais necessárias, e é esse o principal o objetivo da Engenharia de Software.

Conforme Pressman “A Engenharia de Software é uma tecnologia dividida em camadas (ferramentas, métodos, processo e foco na qualidade) ” (PRESSMAN, 2011, p. 15).

“Engenharia de Software é a criação e a utilização de sólidos princípios de engenharia a fim de obter software de maneira econômica, que seja confiável e que trabalhe em máquinas reais. ” (FRIEDRICH LUDWIG BAUER)

“Engenharia de Software é uma disciplina cujo foco está em todos os aspectos da produção de software, desde os estágios iniciais da especificação do sistema até sua manutenção...” (SOMMERVILLE, 2013, p. 5)

Há vários exemplos de sistemas, em que foram gastos recursos consideráveis e que não atenderam às necessidades de seus usuários, e ainda por cima foram entregues fora do prazo estabelecido no cronograma. Esses acontecimentos são muitos graves, haja vista que as organizações estão cada vez mais dependentes dos sistemas de informações. Pressman descreve esta importância: “O software distribui o produto mais importante de nossa era – *a informação*. Ele transforma dados pessoais de modo que possam ser úteis num determinado contexto” (PRESSMAN, 2011, p. 31). Isso deixa muito claro o tamanho da importância da Engenharia de Software.

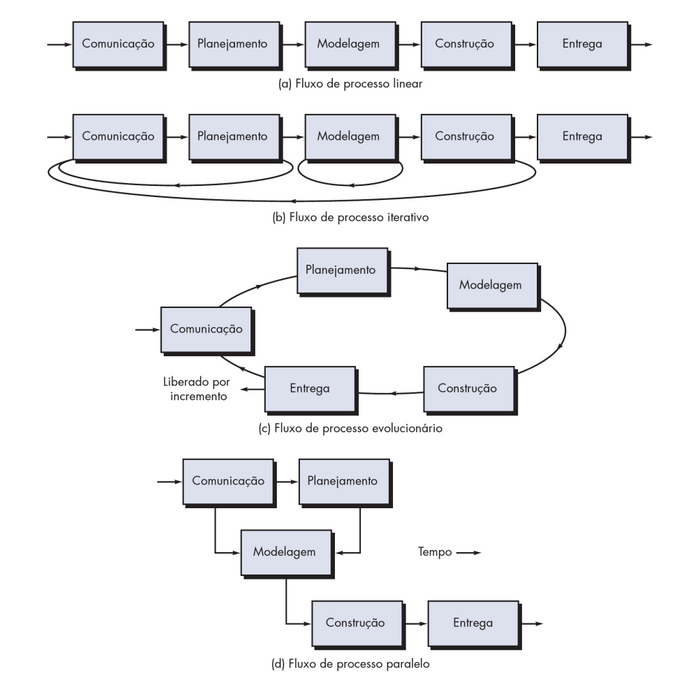
Um dos pontos mais importantes da Engenharia de Software é a divisão do desenvolvimento de software em fases definidas.

Para Pressman, a Engenharia de Software é dividida em cinco etapas:

1. **Comunicação –** [...] A intenção é compreender os objetivos das partes interessadas para com o projeto e fazer o levantamento das necessidades que ajudarão a definir as funções e características do software.
2. **Planejamento –** [...] define o trabalho de engenharia de software, descrevendo as tarefas técnicas a ser conduzidas, os riscos prováveis, os recursos que serão necessários, os produtos a ser produzidos e um cronograma de trabalho.
3. **Modelagem –** [...] Cria-se um “esboço” da coisa, de modo que se possa ter uma ideia do todo – qual será seu aspecto em termos de arquitetura, como as partes constituintes se encaixarão e várias outras características. [...]
4. **Construção –** Essa atividade combina geração de código (manual ou automatizada) e testes necessários para revelar erros de codificação.
5. **Emprego –** O software é entregue ao cliente, que avalia o produto entregue e fornece feedback, baseado na avaliação.(PRESSMAN, 2011, p. 41)

1.1.2.1 Processos Genéricos

Pressman defende que essas etapas, que ele chama de processos, podem vários fluxos, vejamos na figura 1.1 esses exemplos.

Figura 1- Fluxos de processo

Fonte: PRESSMAN (2011, p. 54)

1.1.2.2 Modelo de Processo Prescritivo

Foram propostos alguns modelos de processos para situações mais específicas, diante de uma enorme necessidade.

“A história tem demonstrado que esses modelos tradicionais proporcionaram uma considerável contribuição quanto à estrutura utilizável no trabalho de equipes de software. Entretanto, o trabalho de engenharia de software e o seu produto permanecem à beira do caos”. (PRESSMAN, 2011, p. 59)

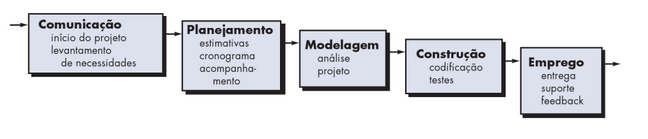
1.1.2.2.1 O Modelo Cascata

É mais indicado em situações nas quais os requisitos estão em um nível de definição elevado, e que por isso tendem a mudar pouco. Isso é mais comum em mudanças bem definidas e um sistema existente.

O Modelo Cascata “...sugere uma abordagem sequencial e sistemática para o desenvolvimento de software”. (PRESSMAN, 2011, p. 59)

Vejamos na figura 2 como evolui o Modelo Cascata, segundo Pressman.

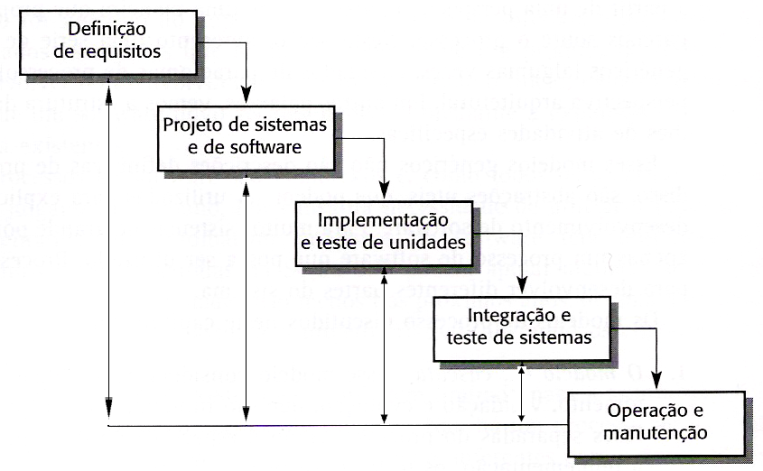
Figura 2- Modelo Cascata - Pressman



Fonte: PRESSMAN (2011, p. 60)

Vejamos na figura 3 como evolui o Modelo Cascata segundo Sommervile.

Figura 3- Modelo Cascata - Sommerville



Fonte: SOMMERVILLE (2013, p. 20)

O modelo Cascata tem suas qualidades, como organizar e dividir responsabilidades no desenvolvimento de software. Entretanto, tem sofrido inúmeras críticas por não atender a complexidade que é desenvolver um software, ainda mais nos dias atuais.

“Para maioria dos sistemas, esse processo não oferece custo-benefício significativo”*.* (SOMMERVILLE, 2013, p. 21)

Pressman cita três principais defeitos:

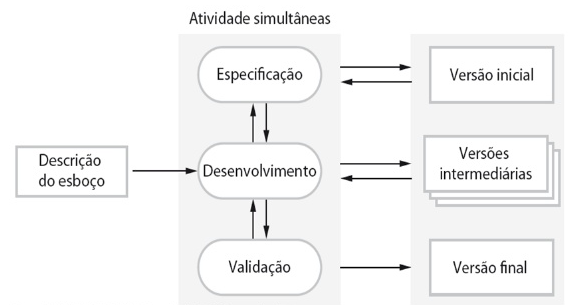
* Projetos reais raramente seguem o fluxo sequencial[...]
* Frequentemente, é difícil para o cliente estabelecer explicitamente todas as necessidades[...]
* O cliente dever ter paciência. Uma versão operacional do(s) programa(s) não estará disponível antes de estarmos próximo do final do projeto[...] (PRESSMAN, 2011, p. 61)

“Na média, 45% das características nos requisitos em cascata nunca são usadas e o cronograma e orçamentos iniciais, em cascata, variam até 400% do que ocorre na realidade”. (LARMAN, 2007, p. 61)

1.1.2.2.2 O Modelo Incremental

Este modelo foi proposto inicialmente por Mills em 1980 e veio para superar a dificuldade que o Modelo Cascata tem em lidar com mudanças frequentes durante o processo de desenvolvimento de software, principalmente por conta da sua maneira de lidar com os requisitos, mais flexível considerando mudanças de requisitos algo esperado e normal. O Modelo Incremental pode ser definido como: “O desenvolvimento incremental é baseado na ideia de desenvolver uma implementação inicial, expô-la aos comentários dos usuários e continuar por meio de criação de várias versões até que um sistema adequado seja desenvolvido” (SOMMERVILLE, 2013, p. 21).

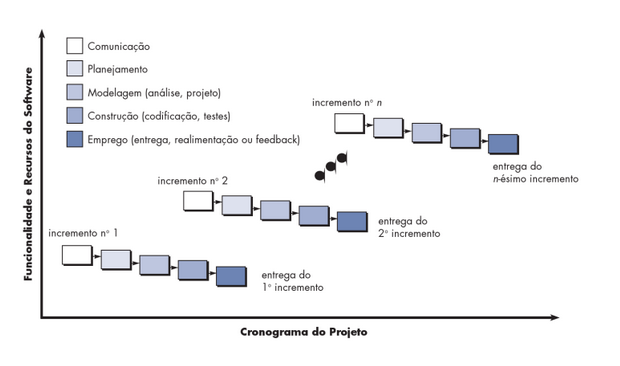
A figura 4 demonstra como o modelo incremental se comporta segundo Sommerville.

Figura 4- Modelo Incremental - Sommerville

Fonte: SOMMERVILLE (2013, p. 22)

A figura 5 demonstra como o modelo incremental se comporta segundo Pressman.

Figura 5- Modelo Incremental - Pressman



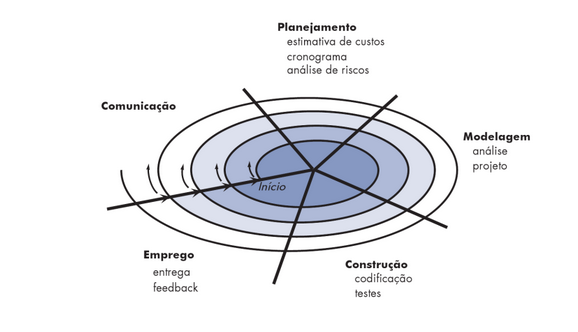
Fonte: PRESSMAN (2011, p. 61)

O modelo incremental se mostra eficaz em muitas situações, porém há algumas específicas em que não possível obter resultados satisfatórios com sua utilização, como em sistemas de bancários de grande porte, nos quais várias equipes de trabalho têm de estar envolvidas, exigindo dessa forma, uma arquitetura bem definida previamente.

1.1.2.2.3 Modelo Espiral

Segundo PRESSMAN 2011 p. 65 “o Modelo Espiral é um modelo de software evolucionário que acopla a natureza iterativa da prototipação com os aspectos sistemáticos e controlado do Modelo Cascata”. Neste modelo o cliente vai recebendo e testando o sistema em várias versões entregues, as primeiras podem ser um protótipo até chegar na versão final.

Figura 6- Modelo Espiral



Fonte: PRESSMAN (2011, p. 65)

O Modelo Espiral é eficaz na redução de riscos a que o processo de desenvolvimento de software está sujeito, por utilizar a prototipação em qualquer estágio.

Pressman explica o motivo do nome do modelo ser espiral:

Assim que esse processo evolucionário começa, a equipe de software realiza atividades indicadas por um circuito em torno da espiral no sentido horário começando pelo centro. Os riscos são considerados à medida que cada evolução é realizada[...](PRESSMAN, 2011, p. 41)

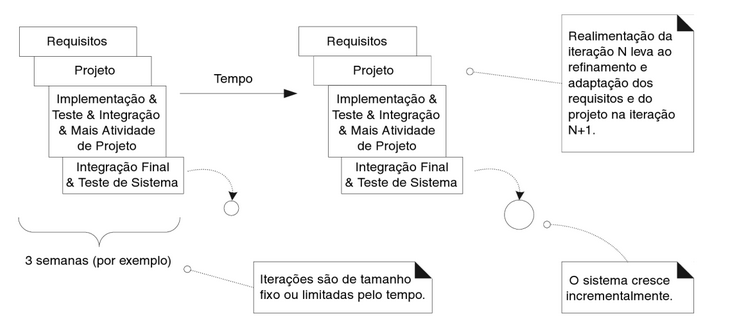
Como nenhum modelo é perfeito, o Modelo Espiral também tem suas limitações, como o fato de exigir avaliação de riscos, pois se algum importante não detectado e controlado poderá ocorrer sérios problemas.

1.1.2.2 Processo Unificado

O Processo Unificado é um processo para desenvolvimento de software iterativo que busca tirar proveito dos melhores recursos dos modelos mais tradicionais citados anteriormente. “Processo Unificado surgiu comoum processo iterativo popular para o de o desenvolvimento de software visando à construção de sistemas orientados a objetos”. (LARMAN, 2007, p. 46).

O Processo Unificado segue uma abordagem de desenvolvimento iterativa, que se baseia em incrementos e melhorias de acordo com a evolução do desenvolvimento, as mudanças são vistas como inevitáveis e essenciais para sucesso do projeto, ou seja, a adaptação faz parte do modelo. Vejamos na figura 7 como funciona o fluxo do desenvolvimento iterativo.

Figura 7- Desenvolvimento Iterativo e Evolutivo



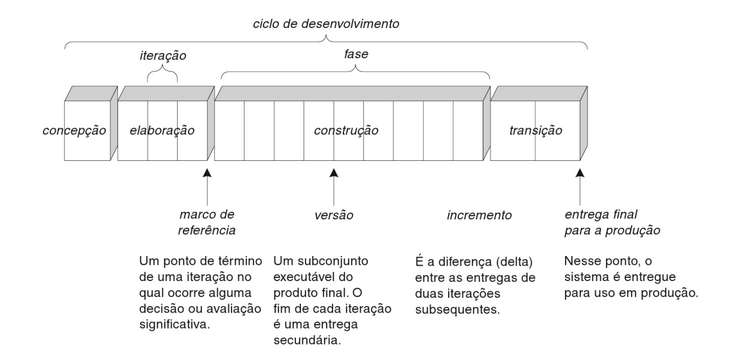
Fonte: LARMAN (2007, p. 48)

LARMAN descreve as quatro fases do Processo Unificado:

1. **Concepção –** visão aproximada, casos de negócio, escopo e estimativas vagas.
2. **Elaboração –** visão refinada, implementação iterativa da arquitetura central, resolução dos altos riscos, identificação da maioria dos requisitos e do escopo e estimativas mais realistas.
3. **Construção –** implementação iterativa dos elementos restantes de menor risco e mais fáceis e preparação para a implantação.
4. **Transição –** testes beta e implantação. (LARMAN, 2010, p. 61)

Vale ressaltar que o processo pode ser composto de várias iterações de acordo a evolução, vejamos na figura 8 mais detalhes.

Figura 8- Cronograma do Processo Unificado

Esse modelo é seguido por uma metodologia de desenvolvimento de software muito difundida pertencente à IBM, o RUP (Rational Unified Process), que será mais detalhado posteriormente.

Fonte: LARMAN (2007, p. 62)

4.2 PARADIGMAS DE PROGRAMAÇÃO

Atualmente existem diversos paradigmas de programação, cada um com suas características e com sua melhor aplicabilidade. Ter conhecimento sobre os paradigmas de programação é essencial para que se possa saber qual a melhor maneira de desenvolver um projeto. Os principais paradigmas de programação são:

**Programação Orientada a Objetos (POO):** A programação orientada a objetos, do inglês *Object Oriented Programming,* é um modelo de programação que procura representar as coisas do mundo real através de objetos, neste modelo tudo é um objeto, e esses objetos possuem características e comportamentos, conhecidos como atributos e métodos, respectivamente, estas características e comportamentos são organizados dentro de um modelo do objeto, que é chamado de classe. O principal objetivo da POO é aproximar o mundo real do virtual, através de representações e linguagens de fácil entendimento. Alguns exemplos de linguagens orientadas a objetos são: C#, C++, Objective C, Java, Objective Pascal, Python, entre outras;

**Programação Estruturada:** Tipo de programação que é voltada para estruturas de decisão como, “se...então” e “caso algo...então faça”, além de se utilizar de interadores, que determinam condições para que algo seja repetido. Estás estruturas de decisão e integradores são definidos dentro de funções que podem ser chamadas uma ou várias vezes dentro de um mesmo programa, estás funções ainda podem ser abstraídas para sub programas. A programação estruturada ainda permite a definição de estruturas que são dados organizados para representar algo mais complexo. Exemplos de linguagens de programação estruturada são Pascal e C;

**Programação Modular:** Modelo em que os programas são divididos em vários módulos, estes, são conectados através de uma interface em comum;

**Programação procedural:** Neste modelo os programas são executados através de que podem ser chamados sucessivas vezes. Exemplos de linguagem são o Fortran e o BASIC.

O problema a ser solucionado indica qual paradigma deve ser utilizado, e logo o paradigma irá influenciar na linguagem escolhida. Para resolver o problema proposto deve se utilizar o paradigma POO, para que caso seja necessário alterar ou expandir, torne-se mais simples de se realizar está tarefa. *Sistemas orientados a objetos são mais fáceis de mudar do que os sistemas desenvolvidos em abordagens funcionais*(SOMMERVILLE,2011).

1. **METODOLOGIA**

Com base na fundamentação teórica, o melhor paradigma para desenvolvimento do sistema é o Orientado a Objetos, então iremos expor a linguagem selecionada e as motivações de seu uso.

* 1. **Java**

**5.1.1 Introdução ao Java**

O Java hoje não é apenas uma linguagem de programação, e sim uma plataforma de desenvolvimento de sistemas e aplicações. O Java hoje é composto por uma série de componentes e frameworks que são baseados na linguagem de programação Java e que foram criados para apoiar e facilitar o desenvolvimento de aplicações, estes frameworks seguem uma série de especificações e padrões de desenvolvimento, os chamados padrões de projeto. Antes de se entender o que é essa plataforma e como ela funciona é necessário o entendimento da linguagem de programação JAVA.

A linguagem de programação Java foi desenvolvida pela Sun Microsystems na década de 90, a grande característica do Java é o fato de ele ser multiplataforma, pode rodar em diversas plataformas e em diferentes sistemas operacionais, essa portabilidade do Java só é possível graças a *Java Virtual Machine* – JVM e ao bytecode, quando o Java é compilado ele gera um bytecode a JVM, que é uma máquina virtual que lê os códigos criados em Java e os interpreta para o sistema operacional, lê o bytecode gerado no processo de compilação e a partir dele cria uma linguagem própria para cada sistema operacional. A linguagem Java é uma linguagem orientada a objetos, ela segue o paradigma de análise e desenvolvimento orientado a objetos, ela é baseada no C++.

Algumas características do Java:

* Sintaxe derivada do C++;
* Criação de bytecode, ao em vez de interpretação direta para o sistema operacional;
* Multiplataforma;
* Desenvolvimento Orientado a Objetos.

**5.1.2 As diversas plataformas do Java**

As plataformas do Java são um conjunto de interfaces de programação de aplicações, do inglês *Application Programing interface* (API), estas APIs seguem alguns padrões de projeto. Cada plataforma possui uma outra embutida dentro dela e funcionalidades a mais. Vamos as plataformas e suas definições:

**JSE:** *Java Standard Edition –* A plataforma mais básica do Java, com ela é possível criar aplicativos para computadores pessoais;

**JME:** *Java Micro Edition –* Uma plataforma criada para o desenvolvimento de aplicações para dispositivos móveis e embarcados;

**JEE:** *Java Enterprise Edition –* A plataforma mais completa do Java, voltada para o desenvolvimento de sistemas mais robustos, para empresas e para a internet;

**JavaFX:** Voltada para o desenvolvimento de aplicações multimídia tanto para desktop/web como para dispositivos moveis;

**Java Card:** Plataforma criada para o desenvolvimento de aplicações para dispositivos embarcados que possuem limitação de processamento e armazenamento de dados.

**5.1.3 O que será utilizado do Java?**

Para resolver os problemas da irá se utilizar as APIs e frameworks da plataforma do JEE que serão, a JPA, o JSF, o Hibernate e o Primefaces.

A Java Peristence API (*JPA*) é uma API baseada em especificações para frameworks de Object Relational Mapping (*ORM*), que são frameworks que fazem todo o mapeamento entre as especificações do banco de dados e as aplicações, estes frameworks abstraem a camada de persistência e trazem uma maior portabilidade para a aplicação, entre os diversos Bancos de Dados. A JPA foi criada a partir de um framework gratuito que é o Hibernate, apesar de ter sido criado antes as novas implementações do Hibernate seguem as definições da JPA.

O Java Server Faces (*JSF*) é um framework que segue o padrão de desenvolvimento Model View Controller (*MVC*), este padrão define um nível de abstração e baixo acoplamento entre as diversas camadas.

* Model – a camada de modelo é onde fica toda a lógica de negócio.
* View - a camada de visão é onde está definida todas as lógicas de tela, é nela que é definida as interfaces as quais os usuários podem acessar.
* Controller – o controlador é a camada que recebe e distribui para os locais certos as requisições do usuário e as respostas do servidor, local onde a aplicação está armazenada.

O JSF é um framework para desenvolvimento web, ele é bastante utilizado, para melhorar a forma como o usuário irá ver este framework será combinado com uma biblioteca de componentes, o Primefaces, que é uma forma de abstrair os códigos JavaScript e as marcações de CSS e HTML.

**5.1.4 Conclusões sobre a escolha do Java**

Como foi exposto o Java é uma plataforma de desenvolvimento de aplicações, que é composta pela linguagem Java, que é orientada a objetos, e diversas APIs e frameworks, Uma linguagem orientada a objetos permite uma melhor representação do mundo real no virtual. A plataforma do Java permite o desenvolvimento de aplicações robustas de uma forma mais ágil e com maior qualidade, ao utilizar as APIs e Frameworks que seguem padrões de desenvolvimento e melhoram a abstração.

Com isso para desenvolver o sistema necessário o uso do Java vai permitir uma maior portabilidade entre os sistemas operacionais ou bancos de dados, através da JVM e JPA, respectivamente, além de permitir o desenvolvimento de uma única solução para todas as filiais, está solução irá rodar no servidor e as filiais poderão acessar via web, por causa do uso do JSF.

**5.2 SOBRE UML**

A Unified Modeling Language ou linguagem unificada de modelagem ou ainda UML, e um tipo de linguagem padrão voltada para modelagem orientada a objetos, foi iniciada com a junção de outros três métodos, que são BOOCH (Fortemente preparado para a fase de projeto), OMT (Propicio para fazer a analise com base nas informações) e OOSE (Focos em casos de uso). Esta linguagem de modelagem não e por si um método de desenvolvimento. Ela Possui a função de auxiliar a visualização o desenho e a comunicação entre objetos criados. Suas funções são capazes de permitir que desenvolvedores consigam visualizar os produtos dos trabalhos feitos por eles em diagramas padrões, fora isso a UML e muito utilizada para criar modelos de sistemas de softwares.

Especificando melhor a UML, devemos saber então que esta linguagem e composta por quatro documentos importantes, infra-estrutura de UML, Superestrutura de UML, Linguagem para Restrições de objetos (OCL) e Intercâmbio de diagramas UML.

Infra - estrutura de UML – E formada por um conjunto de diagramas UML, que forma uma linguagem que e secundaria a outra linguagem que foi definida a partir dos elementos fundamentais e aqui onde a linguagem suporta as definições dos diagramas em UML.

Superestrutura de UML – São os documentos que servem para complementar os outros documentos da infra - estrutura e que também serve para definir a linguagem para o melhor acesso do usuário.

Linguagem para restrições de Objetos – OCL e o documento que faz a representação da linguagem que serve para descrever as expressões usadas em modelos UML, através de pré – condições, pós – condições e as suas invariantes.

Intercâmbio de diagramas UML – Nos mostra uma extensão ou meta – modelos ligada às informações gráficas. Essa extensão nos permite criar uma descrição que posteriormente fica no estilo XMI que e usada nos aspectos de gráficos, quando usada com a XMI original e possível produzir as representações portáveis de especificados UML.

Esta linguagem possui muitas ferramentas tecnológicas que são necessárias para o apoio da pratica de engenharia de software, voltados à orientação de objetos. Alem disso e uma linguagem de modelagem padronizada para diversos modelos de sistemas distribuídos ou concorrentes. Para se utilizar da UML e necessário um conjunto de técnicas que envolvem noções gráficas para a criação dos modelos visuais dos possíveis softwares de sistemas que chegam a ser intensivos, com a boa pratica e possível fazer uma combinação excelente de modelos de dados, objetos, negócios e componentes. Essa linguagem e muito comum e bastante utilizada.

Com a UML, mesmo que possível fazer a representação dos softwares, isso por meio de seus modelos a objetos, essa linguagem não mostra que tipo de trabalho deve ser feito, ela não tem um processo que padroniza como o trabalho deve realmente ser feito. Então assim temos muitos outros objetivos como, o que fazer como fazer, quando deve ser feito e o porquê deve ser feito. Torna-se necessário a criação completa de um dicionário de dados, que vai nos descrever todas as entidades envolvidas no projeto, e assim nos mostrando detalhadamente, quais são os requisitos funcionais daquele software ou projeto.

A UML, ainda possui os seus diagramas, que forma as representações através de gráficos do modelo de um sistema ou software, que tornasse necessários o uso de combinações para se obter os melhores aspectos e visões do software.

**DIAGRAMAS ESTRUTURAIS**

Os diagramas estruturais formam o conjunto de outros diagramas em UML. Esse diagrama e utilizado para mostrar, fazer especificações, construções e documentações na criação de um sistema.

**Diagramas de Classe** – E o diagrama fundamental e o que mais e utilizado na UML, e também serve de apoio para os demais diagramas, esse diagrama e responsável por mostrar o aglomerado conjunto classes e seus determinados atributos e também os seus métodos, sem esquecer os relacionamentos existentes nas classes de uso.

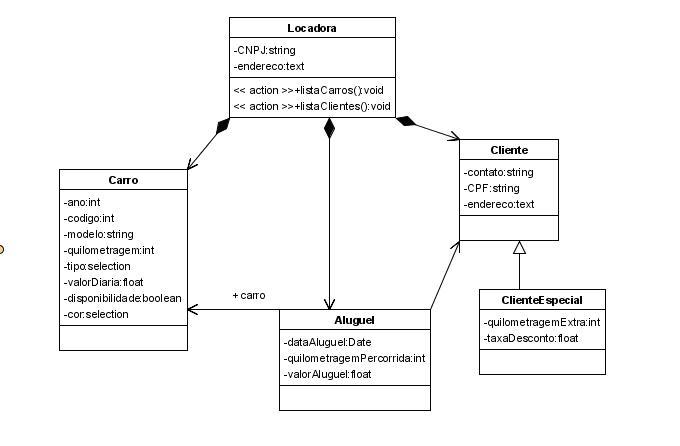


Diagrama de Classe disponível em <http://www.ebah.com.br/content/ABAAAA7sEAD/uml-2-0>

**Diagrama de Objeto** – Este diagrama esta relacionado ao diagrama de classes, praticamente o diagrama em questão e um complemento para o diagrama de classe. Responsável por fornecer uma determinada visão dos valores que são armazenados pelos objetos do diagrama de classe, ocorrendo em um determinado momento em que acontece a execução de um processo de sistema.

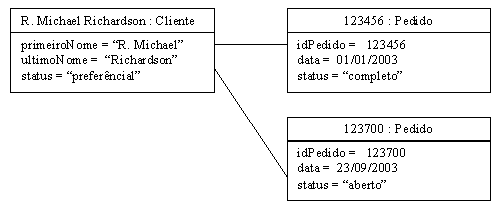


Diagrama de Objeto disponível em <http://homepages.dcc.ufmg.br/~amendes/GlossarioUML/glossario/conteudo/objetos/FigurasDigObjetos/diagramaObjetos.gif>

**Diagrama de Componentes** – Diagrama que possui uma ligação direta com a linguagem de desenvolvimento tem como objetivo fazer a indicação dos componentes do sistema e seus dados relacionamentos.

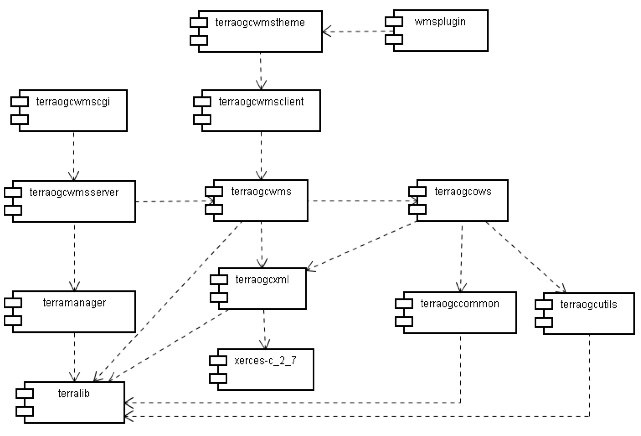


Diagrama de Componentes disponível em <https://lh3.googleusercontent.com/lJN4APf5abN6i-OIsKhTfsruo03YQFv4H5ioTEmnLRdeK20oT-3MGag3rShl4jA8mqjsFmFLkU4tuJWgfLS6zwDW2chas72YxImPJobaN6LYC8fryqomaywdxEZZI3B-Ji31BNI>

**Diagrama de Implementação** – Faz a modelagem dos relacionamentos entre os recursos de infraestrutura, de artefatos ou de redes do sistema. Ele e muito usado nos projetos onde geralmente ocorre uma interdependência entre os hardwares e os softwares. Neste diagrama a representação de cada nó mostra uma unidade física representando um determinado recurso computacional, como os roteadores, processadores ou qualquer outro equipamento que sirva para o software em sua importância.

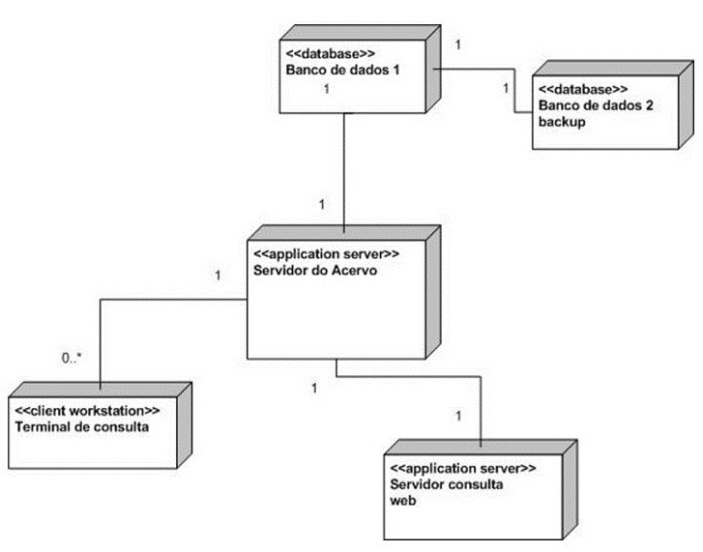


Diagrama de Componentes disponível em <http://images.slideplayer.com.br/1/327235/slides/slide\_6.jpg>

**Diagrama de Pacotes** – Seu objetivo e fazer a representação dos subsistemas dentro de um sistema maior, sua tarefa e identificar as partes de subsistemas que compõe esse sistema maior, para mostrar este diagrama ele e representado por diversos outros diagramas, como por exemplo, o diagrama de classe que representa e detalha um pacote qualquer.

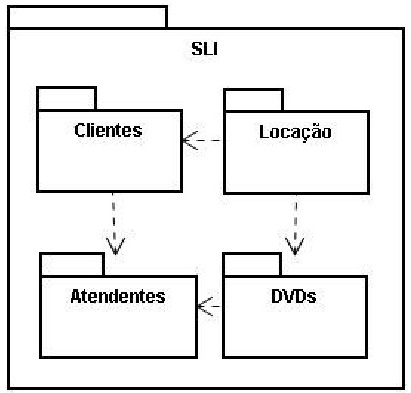


Diagrama de Componentes disponível em <http://image.slidesharecdn.com/umlpacotes-100819163942-phpapp02/95/uml-diagrama-de-pacotes-6-728.jpg?cb=1282236021>

**Diagrama de Estrutura** – Utilizado para a criação de modelagem das chamadas colaborações. Para um melhor entendimento uma colaboração mostra uma visão de um aglomerado de instancias que se ajuda para completar uma determinada tarefa a ser realizada, de tal forma que a mesma instancia mostra uma ligação entre as outras instâncias e os seus respectivos papeis a serem cumpridos naquela tarefa.

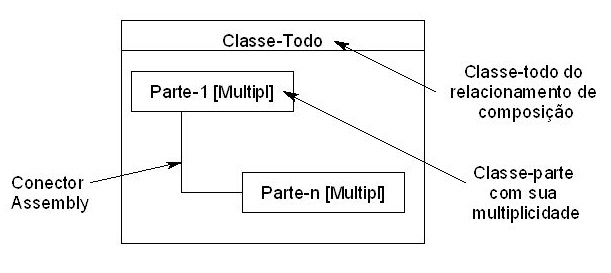
****

Diagrama de Componentes disponível em <http://images.slideplayer.com.br/10/2843796/slides/slide\_27.jpg>

**DIAGRAMAS COMPORTAMENTAIS**

Como o nome já diz os diagramas comportamentais são aqueles que mostram alguma alteração na classe. Em UML e permitido aos seus desenvolvedores fazer a visualização dos produtos de seus trabalhos através dos diagramas padronizados, mostrando uma visão graficamente, facilitando assim uma possível explicação de um projeto.

**Diagrama de Caso de uso** – Utilizado para fazer a analise e o levantamento de requisitos de um determinado sistema seja por meio geral ou ate mesmo informal em suas distintas fases de projeto.

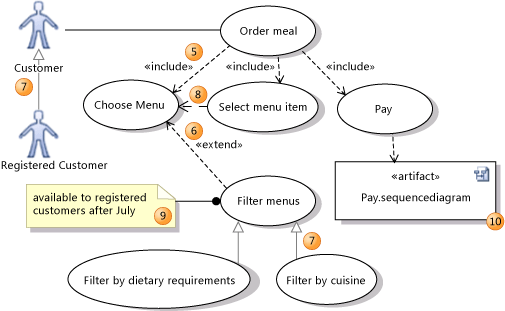


Diagrama de caso de uso disponível em <http://www.micreiros.com/wp-content/uploads/DiagramaCasoDeUso.png>

**Diagrama de Máquina de Estados** – Necessário para acompanhar todas as mudanças sofridas em um projeto em sua fase de desenvolvimento. Aconselhável o uso em sistemas com mais complexidade, o diagrama de estado possui um início, porem pode ter várias saídas e fins.

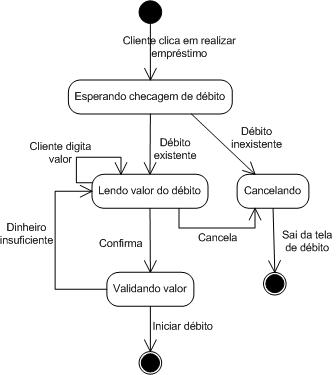


Diagrama de Estado disponível em <http://www.micreiros.com/wp-content/uploads/Diagrama-de-Estado.jpg>

**Diagrama de Atividades** – Mostra todos os passos as serem feitos ate o fim daquela determinada atividade, e igualmente ao diagrama de estado, esse diagrama também pode conter várias saídas ou fins.

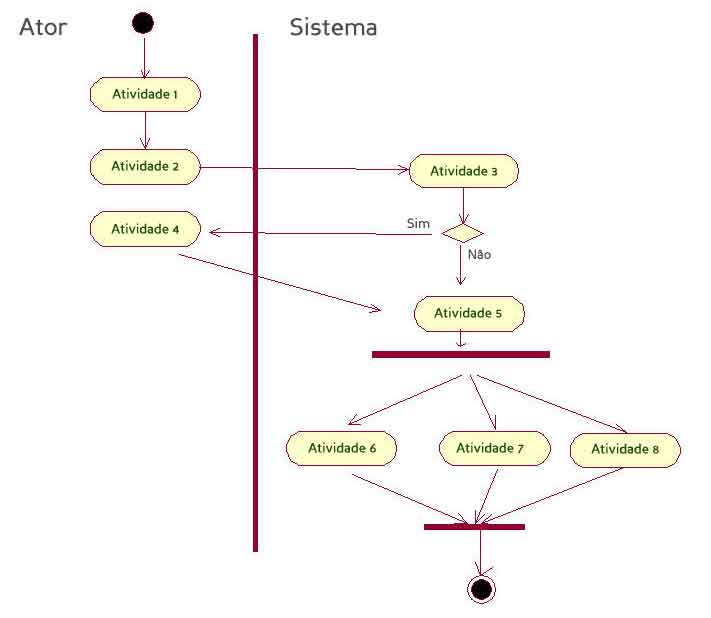
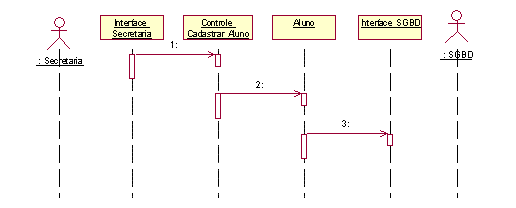


Diagrama de Atividades disponível em <http://www.micreiros.com/wp-content/uploads/DiagramaAtividades.jpg>

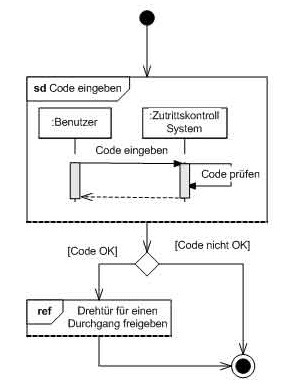
**Diagrama de Interação** – Este diagrama se divide em outros quatro diagramas sendo eles, diagrama de Seqüência (Mostra em ordem de tempo quando as mensagens são trocadas entre os objetos), Diagrama Geral de Interação (Exibe a variação dos diagramas de atividades que mostra uma visão geral dentro de um projeto ou sistema), Diagrama de Comunicação (Ligado ao diagrama de seqüência, completando-o e mantendo a concentração de como os objetos se vinculam), Diagrama de Tempo (Nos fala sobre a mudança que ocorre no estado ou na condição de determinada instancia de uma classe ou o papel que ela exerce durante um determinado tempo).

Diagrama de Seqüência.



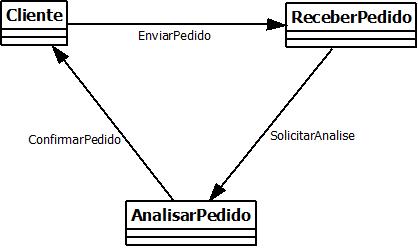
Disponível em <http://www.macoratti.net/vb\_uml2b.gif>

Diagrama Geral de Interação.



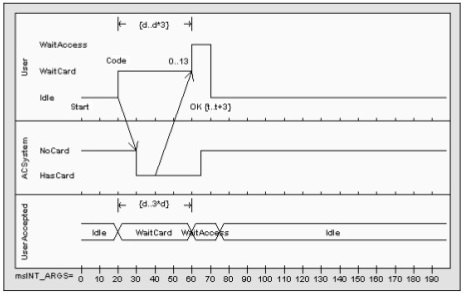
Disponível em <http://nusseagora.dominiotemporario.com/wp-content/uploads/2009/09/Interacao.jpg>

Diagrama de Comunicação.



Disponível em <https://upload.wikimedia.org/wikipedia/commons/7/72/Diagrama\_de\_Colaboracao.jpeg>

Diagrama de Tempo.



Disponível em <<http://image.slidesharecdn.com/projetodesistemascomuml-parte1-131022134221-phpapp02/95/projeto-de-sistemas-com-uml-parte-1-43-638.jpg?cb=1382449959>>

1. CRONOGRAMA DE DESENVOLVIMENTO

**Quadro 1** – Calendário de execução das atividades do Projeto e do Trabalho de Conclusão de Curso

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ATIVIDADE** | **2016** | | | | | **2016** | | | | |  |
| **JAN** | **FEV** | **MAR** | **ABRIL** | **MAI** |  | **AGOS** | **SETEM** | **OUTUB** | **NOVEM** | **DEZEM** |
| Escolha do tema | X |  |  |  |  |  |  |  |  |  |  |  |
| Revisão bibliográfica | X | X | X | X | X |  | X | X | X | X | X |  |
| Elaboração do projeto | X | X | X | X |  |  |  |  |  |  |  |  |
| Elaboração de questionários, tópico de entrevistas etc. \* |  | X | X | X |  |  |  |  |  |  |  |  |
| Entrega do projeto |  |  |  |  | X |  |  |  |  |  |  |  |
| Elaboração da monografia (TCC) |  |  |  |  |  |  | X | X | X | X | X |  |
| Realização dos capítulos |  |  |  |  |  |  | X | X | X |  |  |  |
| Coleta e análise de dados / amostragens \* |  |  |  |  |  |  | X | X | X | X |  |  |
| Realização da conclusão e introdução |  |  |  |  |  |  |  |  | X | X |  |  |
| Correção de textos | X | X | X | X |  |  | X | X | X | X | X |  |
| Elaboração de elementos pré e pós-textuais |  |  |  |  |  |  |  |  | X | X |  |  |
| Entrega da monografia |  |  |  |  |  |  |  |  |  |  | X |  |
| Defesa da monografia |  |  |  |  |  |  |  |  |  |  | X |  |

REFERÊNCIAS

SOMMERVILLE, Ian. **Engenharia de Software**. 9ª ed. São Paulo. Pearson. 2013

PRESSMAN, Roger S. **Engenharia de Software.** 7ª ed. São Paulo. Bookman. 2011

LARMAN, Craig. **Utilizando UML e Padrões.** 3ª ed. São Paulo. Bookman. 2007