

**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 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**

Trabalho de Conclusão de Curso. 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**](#_Toc452319547)

[1.1 PROBLEMA 5](#_Toc452319548)

[**2. OBJETIVOS 6**](#_Toc452319549)

[2.1 OBJETIVO GERAL OU PRIMÁRIO 6](#_Toc452319550)

[2.2 OBJETIVOS ESPECÍFICOS OU SECUNDÁRIOS 6](#_Toc452319551)

[**3. JUSTIFICATIVA 7**](#_Toc452319552)

[**4. FUNDAMENTAÇÃO TEÓRICA 8**](#_Toc452319553)

[4.1 ENGENHARIA DE SOFTWARE 8](#_Toc452319554)

[4.1.1 PROCESSOS GENÉRICOS 9](#_Toc452319555)

[4.1.2 MODELO DE PROESSO PRESCRTIVO 10](#_Toc452319556)

[4.2 PARADIGMAS DE PROGRMAÇÃO 15](#_Toc452319557)

[4.3 RUP 16](#_Toc452319558)

[4.4 BANCO DE DADOS RELACIONAL 18](#_Toc452319559)

[4.5 ARQUITETURA DE SOFTWARE 20](#_Toc452319560)

[4.5.1 ARQUITETURA EM CAMADAS 20](#_Toc452319561)

[4.5.2 ARQUITETURA CLIENTE-SERVIDOR 21](#_Toc452319562)

[4.5.3 ARQUITETURA ORIENTADA A SERVIÇOS (SOA) 22](#_Toc452319563)

[**5. METODOLOGIA 23**](#_Toc452319564)

[5.1 AQUITETURA DO SISTEMA 23](#_Toc452319565)

[5.2 JAVA 23](#_Toc452319566)

[5.2.1 INTRODUÇÃO AO JAVA 23](#_Toc452319567)

[5.2.2 AS DIVERSAS PLATAFORMAS DO JAVA 24](#_Toc452319568)

[5.2.3 O QUE SERÁ UTILIZADO DO JAVA 25](#_Toc452319569)

[5.2.4 CONCLUSÕES SOBRE A ESCOLHA DO JAVA 25](#_Toc452319570)

[5.3 UML 26](#_Toc452319571)

[5.3.1 DIAGRAMAS ESTRUTURAIS 27](#_Toc452319572)

[5.3.2 DIAGRAMAS COMPORTAMENTAIS 30](#_Toc452319573)

[5.4 HIBERNATE 34](#_Toc452319574)

[**6. CRONOGRAMA 37**](#_Toc452319575)

[**7. REFERÊNCIAS 38**](#_Toc452319576)

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.

* 1. 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
   1. OBJETIVO GERAL OU PRIMÁRIO

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

* 1. OBJETIVOS ESPECÍFICOS OU SECUNDÁRIOS
* Estudar o negócio do cliente, procurar entender sua forma de trabalho e criar uma linguagem comum entre a equipe técnica e o cliente;
* 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;
* Realizar os processos de engenharia de software, que serão estabelecidos para melhor atender ao cliente;
* Implementar e testar o sistema.

1. 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 para a metodologia que utilizaremos para desenvolver o sistema.

* 1. ENGENHARIA DE SOFTWARE

Nesta etapa 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 necessário, 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:

**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.

**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.

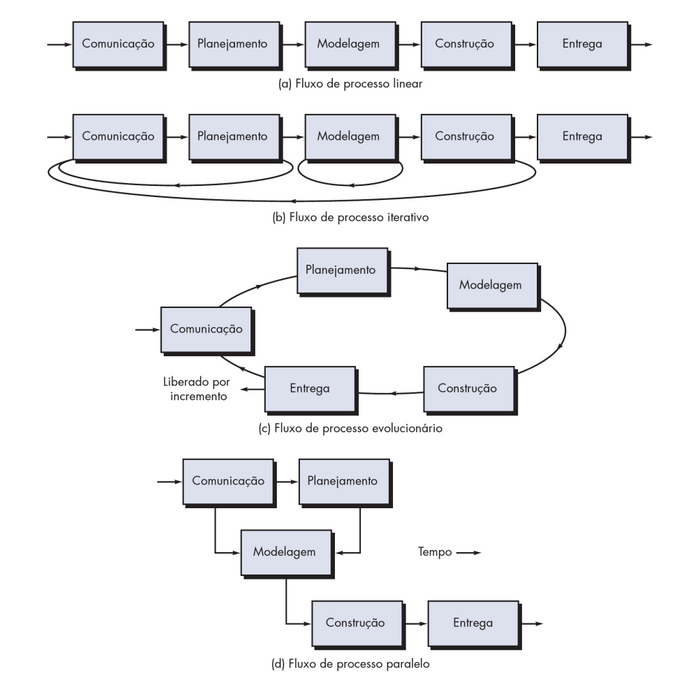
**Modelagem** – [...] Cria-se um “esboço” da coisa, de modo que se possa ter uma ideia do todo – qual será seu aspecto em termos de arquitetura, como as partes constituintes se encaixarão e várias outras características. [...]

**Construção** – Essa atividade combina geração de código (manual ou automatizada) e testes necessários para revelar erros de codificação.

Emprego – O software é entregue ao cliente, que avalia o produto entregue e fornece feedback, baseado na avaliação. (PRESSMAN, 2011, p. 41)

* + 1. PROCESSOS GENÉRICOS

Pressman defende que essas etapas, que ele chama de processos, podem vários fluxos, vejamos na figura 1 esses exemplos.

Figura 1- Fluxos de processo

Fonte: PRESSMAN (2011, p. 54)

* + 1. MODELO DE PROESSO PRESCRTIVO

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)

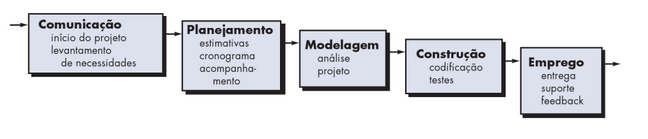
* + - 1. MODELO CASCATA

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.

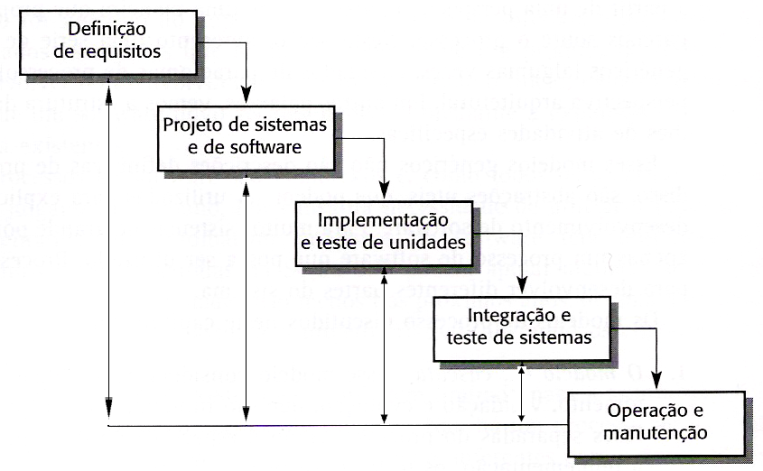
Figura 2- Modelo Cascata - Pressman



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[...]

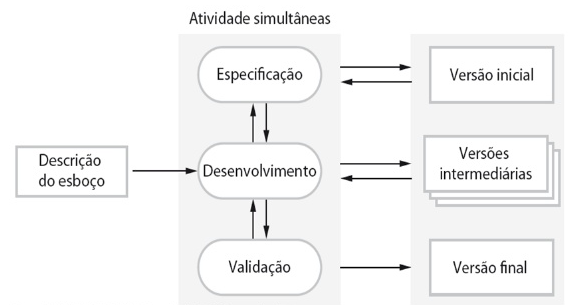
O cliente dever ter paciência. Uma versão operacional do(s) programa(s) não estará disponível antes de estarmos próximo do final do projeto[...] (PRESSMAN, 2011, p. 61)

“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. MODELO INCRMENTAL

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, é mais flexível, pois considera as mudanças de requisitos algo esperado e normal. “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).

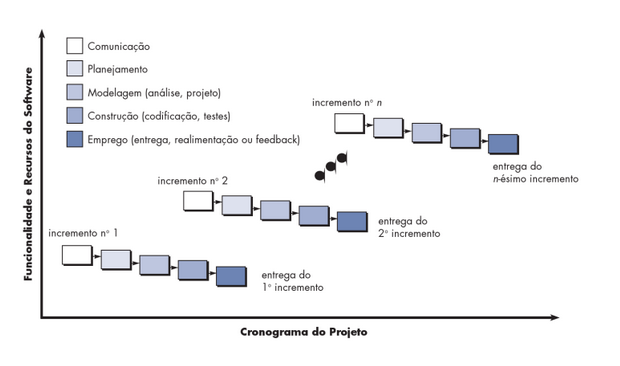
A figura 4 demonstra como o modelo incremental se comporta segundo Sommerville.

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



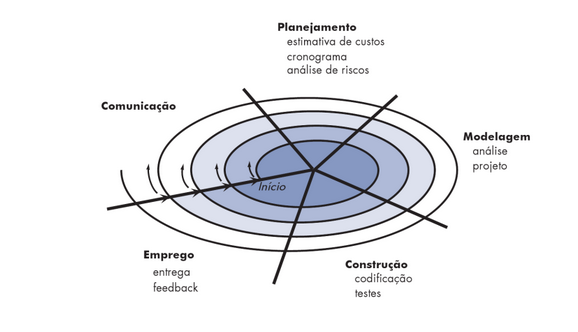
Fonte: PRESSMAN (2011, p. 61)

O modelo incremental se mostra eficaz em muitas situações, porém há algumas específicas em que não é 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. 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).

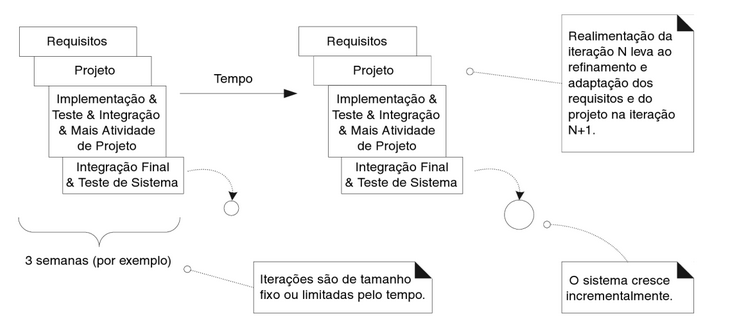
Como nenhum modelo é perfeito, o Modelo Espiral também tem suas limitações, como o fato de exigir avaliação de riscos frequentes, pois se algum risco importante não for detectado e controlado poderão ocorrer sérios problemas.

* + - 1. 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. Iterativo

Figura 7- Desenvolvimento e Evolutivo

Fonte: LARMAN (2007, p. 48)

LARMAN descreve as quatro fases do Processo Unificado:

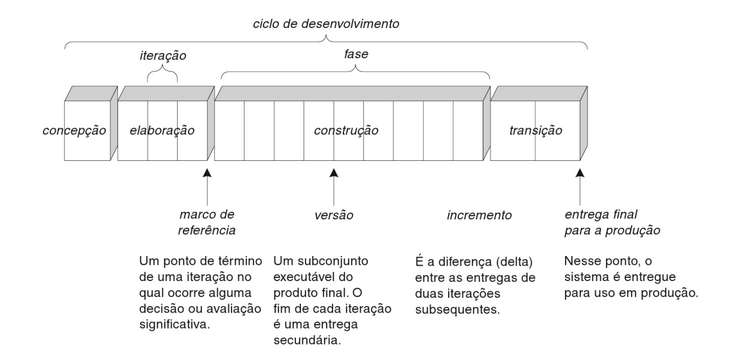
**Concepção** – visão aproximada, casos de negócio, escopo e estimativas vagas.

**Elaboração** – visão refinada, implementação iterativa da arquitetura central, resolução dos altos riscos, identificação da maioria dos requisitos e do escopo e estimativas mais realistas.

**Construção** – implementação iterativa dos elementos restantes de menor risco e mais fáceis e preparação para a implantação.

**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

Fonte: LARMAN (2007, p. 62)

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.

* 1. PARADIGMAS DE PROGRMAÇÃO

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 subprogramas. 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. RUP

O RATIONAL UNIFIED PROCESS (RUP) é um processo de engenharia de software criado para otimizar e apoiar o desenvolvimento orientado objeto, com uma sistemática para facilitar o uso da UML, podendo ter três definições.

* 1. Uma maneira de desenvolvimento de software iterativa e centrada na arquitetura e seguida por casus de uso.
  2. Provê uma estrutura bem definida para o ciclo de vida de um projeto, um processo bem definido e bem estruturado na parte de engenharia de software.
  3. É um produto que oferece para a engenharia de software uma estrutura de processo flexível.

Para a aplicação do RUP não a uma formula definida pois pode ser aplicado de diferentes formas para cada organização e projeto que ele é apresentado.

O RUP alguns elementos principais:

**Papéis:** Um papel define o comportamento com as responsabilidades de determinado individuo ou grupo de indivíduos que trabalham como uma equipe.

**Atividades:** A atividade é uma unidade de trabalho que um indivíduo executa quando está exercendo um determinado papel e produz um importante resultado para a elaboração do projeto.

**Artefatos:** Um artefato é produzido e pode ser modificado e utilizado em um processo.

**Fluxo de Trabalho:** Um fluxo de trabalho é uma sequência de atividades que são executadas para a produção de um resultado valioso para o projeto [Kroll e Kruchten 2003].

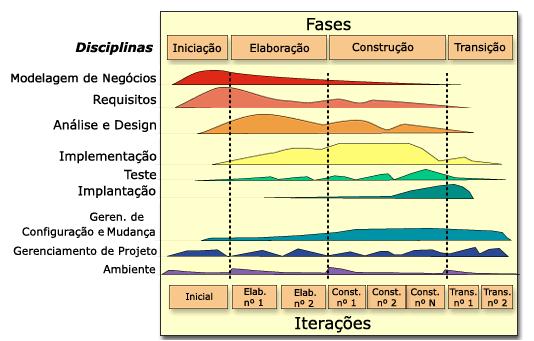
Os fluxos de trabalho do RUP são os Fluxos de trabalho principais, associados com cada disciplina, o Fluxos de trabalho de detalhe, para detalhar cada fluxo de trabalho principal e os planos de iteração, que mostram como a iteração deverá ser executada.

Os fluxos de trabalho podem ser representados por diagramas de sequência, diagramas de colaboração e diagramas de atividades da linguagem de modelagem unificada.

Ao todo o RUP tem nove disciplinas que são divididas em disciplinas do processo e de suporte, as de disciplina são: Modelagem de negócios, requisitos, análise e projeto, implementação, teste e distribuição. As de suporte são: Configuração e gerenciamento de mudanças, gerenciamento de projeto, e ambiente, de acordo com [Kruchten 2003].

Como mostra a figura [Martins 2007] A arquitetura do RUP e constituída por dois eixos horizontal e vertical.

Figura – 9 Eixos do Fluxo de Trabalho do RUP



Disponível em <http://www.sgp-ti.uerj.br/site/metodologia/>

O eixo Vertical representa as disciplinas que agrupam as atividades de maneira lógica naturalmente, e representa o modo estático do processo que é descrito em termos de componentes, disciplinas, atividades, fluxos de trabalho, artefatos e papéis do processo.

O eixo horizontal apresenta o tempo mostrando os aspectos do ciclo de vida do processo à medida que ele se desenvolve. Apresentado o aspecto dinâmico do processo em termos e fases.

O ciclo de vida do RUP constitui-se em 4 fases, são elas:

* Concepção: Define o escopo do projeto.
* Elaboração: Detalha os requisitos e a arquitetura.
* Construção: Desenvolve o sistema.
* Transição: Implanta o sistema.
  1. BANCO DE DADOS RELACIONAL

Um banco de dados serve para fazer o armazenamento de dados e obter as voltas dos resultados com eficiência, a maneira que os dados são organizados ou armazenas e o que torna esse banco relacional ou não. Um banco de dados relacional “RDBMS” ou ainda “Relational Database Management System” define diversos tipos de dados e os armazena em tabelas. Esses dados possuem uma estrutura que se repetirá a cada linha da tabela, como por exemplo em uma tabela normal, onde os relacionamentos os relacionamentos existentes entre as tabelas que são capazes de fazer com que estas mesmas tabelas sejam relacionais no banco de dados.

Antigamente quando o banco de dados ainda não havia sido inventado, lá por volta dos anos 70, existiam outros tipos de bancos de dados, que alguns se utilizavam de hierarquia. Porem depois da invenção dos bancos de dados que conhecemos atualmente muitas empresas tem feito grandes sucessos como IBM e Oracle.

Temos muitos bancos de dados comercias atualmente como o Ingres que foi o primeiro banco de dado relacional usado comercialmente, ainda temos o Progress, Oracre, IBM DB2, Microsoft SQL Server. Fora esses bancos de dados comercias citados ainda temos os que chamamos de livres ou de código aberto, tais como, MySQL, SQLite, MariaDB, PostgresSQL. Todos têm um mecanismo de segurança seja por criptografia ou controle de acesso aos usuários e também podem fazer o processamento de requisições em SQL.

Foi no ano de 1970 que foi criado um tipo de modelo relacional que fazia a gestão de banco de dados e que também era conhecido como a normalização de dados, criado por Ted Codd, esse modelo abrangia 12 leis que fala o que um RDBMS, fora isso o modelo ainda estabelece as propriedades de dados que estão relacionadas, a partir de então apenas os dados que sofreram a normalização que poderiam ser considerados com dados relacionais.

Mas para que serve a normalização afinal? A normalização e um processo que organiza os dados de campos e as tabelas de um determinado banco de dados relacional e seu objetivo e minimizar a independência e redundância no mesmo banco. Esse processo divide as tabelas grandes das pequenas e as que são menos redundantes e ainda faz o relacionamento existentes entre essas mesmas tabelas.

De forma mais clara, e a forma de fazer o isolamento de dados de entrada, remoção e alteração de informações de um determinado campo do banco e que tudo isso possa ser feito apenas em uma tabela e essa informação iria se espalhar pelo resto do banco de dados, tudo de acordo com o que foi definido no relacionamento. Vamos imaginar uma tabela de clientes, o qual alguns clientes podem possuir informações semelhantes, ao lançar os dados na tabela, algumas informações dos clientes agiriam como duplicatas e precisariam ser movidas para outras tabelas e cada uma dessas informações seria dado um ID, assim seria feito uma troca de informação de acordo com ID das tabelas criadas.

Já sabemos então que um sistema de gerenciamento de bando de dados relacional, e um software que faz o controle do armazenamento, da recuperação, da exclusão, segurança e integridade dos dados em um determinado banco de dados. O mecanismo de armazenamento de dados nos permite fazer a persistência de dados e opcionalmente fazer a implementação das funcionalidades. O sistema de banco de dados relacional e utilizado para fazer o armazenamento das informações que são requeridas por certas aplicações que foram construídas pelas tecnologias procedurais, como o COBOL ou FORTRAN, as tecnologias que são orientadas a objetos como o Java e C# e também as tecnologias que se baseiam em componentes como o Visual Basic.

Quando um banco de dados faz o armazenamento em tabelas, essas mesmas tabelas são organizadas em colunas e cada uma dessas colunas armazena um certo tipo de dado seja um dado inteiro, de números reais, strings de caracteres, data, dentre outros. Os dados de apenas uma instancia de uma determinada tabela são armazenadas em uma linha, como por exemplo uma tabela com o nome cliente, essa tabela teria as colunas como numeroCliente, primeiroNome e o sobrenome.

As tabelas possuem um mecanismo de chaves, que serve para identificar uma ou mais colunas existentes na tabela. No exemplo da tabela cliente a chave seria identificada na coluna numeroCliente, existe ainda os índices de uma tabela que serve para melhorar o tempo de acesso a uma determinada tabela, Índice nada mais e do que um meio de buscar informações rapidamente e uma ou mais colunas existentes na tabela, como acontece em um livro onde o índice serve para encontra rapidamente a informação ou a página que queremos.

A maneira mais fácil de utilizar ou manipular dados em um banco de dados que seja relacional e escrever declarações ou códigos em linguagem SQL, códigos esses que servem para fazer alguma alteração no banco ou na tabela onde esses comandos em SQL causaram algum tipo de reação ao banco de dados, comandos mais conhecidos como INSERT, SELECT, UPDATE e DELETE, são bastantes utilizados no Banco de Dados Relacional.

* 1. ARQUITETURA DE SOFTWARE

A arquitetura de software define a forma de organização dos sistemas, no desenvolvimento de um software a definição da arquitetura é a primeira etapa. “O projeto de arquitetura está preocupado com a compreensão de como um sistema deve ser organizado e com a estrutura geral desse sistema” (SOMMERVILLE, 2011).

A arquitetura deve ser bem definida, pois no processo de desenvolvimento de software a refatoração de componentes para atender a mudanças costume ser fácil, mudanças arquiteturais custam um valor altíssimo.

4.5.1 ARQUITETURA EM CAMADAS

Este modelo arquitetural separa o sistema por camadas que se comunicam entre si, esse tipo de arquitetura é ideal quando se deseja ter a forma de visualização dos dados de forma independente do restante do sistema.

Sommerville descreve a arquitetura em camadas em seu livro Engenharia de Software, da seguinte maneira:

Organiza o sistema em camadas com a funcionalidade relacionada associada a cada camada. Uma camada fornece serviços à camada acima dela, assim, os níveis mais baixos de camadas representam os principais serviços[..](SOMMERVILLE, Ian, 2011).

O padrão mais utilizado neste tipo de arquitetura é o modelo-visão-controlador (MVC). Uma das vantagens de se utilizar este modelo segundo Sommerville é que ele “permite que os dados sejam alterados de forma independente de sua representação, e vice-versa” (SOMMERVILLE, 2011).

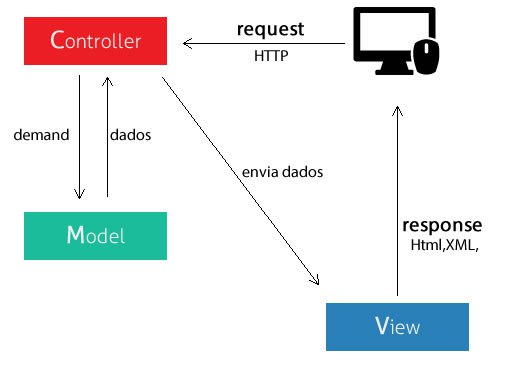


Figura 9 - Padrão MVC para uma aplicação Web (Fonte: <http://tableless.com.br/mvc-afinal-e-o-que/ > Acesso em 28 de maio de 2016)

4.5.2 ARQUITETURA CLIENTE-SERVIDOR

O padrão cliente-servidor é o mais utilizado em sistemas distribuídos, neste padrão você tem uma aplicação servidor que faz o processamento dos dados e os clientes são os usuários que fazem uso destes dados e dos serviços da aplicação estes dados.

Sommerville descreve está arquitetura:

Em uma arquitetura cliente-servidor, a funcionalidade do sistema está organizada em serviços – cada serviço é prestado por um servidor. Os clientes são os usuários desses serviços e acessam os servidores para fazer uso deles (SOMMERVILLE, Ian, 2011).

A principal vantagem desta arquitetura é que os servidores podem ser acessados através de uma rede distribuída.

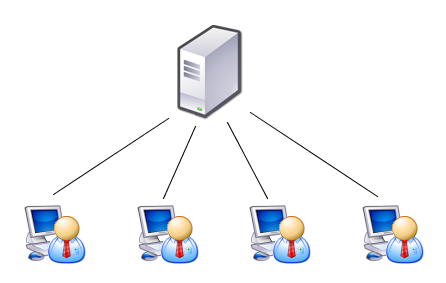


Figura 10 - Modelo Cliente-Servidor (Fonte: <https://arqserv.wordpress.com/2012/03/17/como-funciona-a-arquitetura-cliente-servidor/> Acesso em 28 de maio de 2016)

4.5.3 ARQUITETURA ORIENTADA A SERVIÇOS (SOA)

Este modelo arquitetural e muito utilizado hoje em dia, ele foi criado para que você pudesse acessar uma mesma aplicação de diferentes dispositivos. Está arquitetura é implementada por meio de *Web Services,* serviços web, neste padrão se têm uma aplicação que fica em um servidor remoto e todos os acessos a ela são feitos através de serviços, estes serviços são consumidos por aplicações cliente, que podem ser qualquer aplicação que utilize o protocolo HTTP.

A arquitetura orientada a serviços permite que a aplicação cliente esteja em um computador, notebook, smartphone, tablete ou qualquer dispositivo capaz de enviar e receber mensagens por meio do protocolo HTTP.

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. Para o processo de desenvolvimento será utilizado o Modelo Cascata, que propõe primeiro a documentação do sistema, já que o projeto não é de alta complexidade e como o contato com o cliente não é de extrema necessidade durante o processo de desenvolvimento, será definido os requisitos para o sistema depois os mesmos serão desenhados e organizados segundo os padrões da UML.

Estás escolhas se fundamentam também no fato de nem todos os membros da equipe possuírem experiência no desenvolvimento de sistemas e que o desenvolvimento deverá ser acompanhado pelo tutor.

* 1. AQUITETURA DO SISTEMA

Para solucionar os problemas do desenvolvimento do sistema da drogaria vamos utilizar uma arquitetura em camadas, que seguira o padrão MVC, pois como estamos falando de uma rede isso permitirá que a forma de visualização do sistema não dependa do restante de sua implementação. Também será definida uma parte de serviços, através de *web services*, que servirão para uma integração com outras aplicações que possam consumir os serviços da aplicação.

As camadas do sistema ficaram assim:

* Model – modelo, que definirá todas as regras de negócio da aplicação.
* Controller – controlador, que ficará responsável por realizar a comunicação das visões com o modelo.
* View – visão, camada que definirá as maneiras pelas quais os usuários verão as informações geradas pela aplicação.
* Integração – camada composta por web services, que fornecerão informações para outros sistemas.
  1. JAVA
     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 múltiplas plataformas, 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;
* Múltiplas plataformas;
* Desenvolvimento Orientado a Objetos.
  + 1. 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.
  + 1. O QUE SERÁ UTILIZADO DO JAVA

Para resolver os problemas do desenvolvimento de um sistema para a drogaria 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.

* + 1. 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.

* 1. UML

A Unified Modeling Language ou linguagem unificada de modelagem ou ainda UML, é um tipo de linguagem padrão voltado 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 (Propício para fazer a análise 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 é composta por quatro documentos importantes, infraestrutura de UML, Superestrutura de UML, Linguagem para Restrições de objetos (OCL) e Intercâmbio de diagramas UML.

Infraestrutura de UML – É formada por um conjunto de diagramas UML que forma uma linguagem que e secundária 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 infraestrutura 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 é 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 prática de engenharia de software, voltadas à orientação de objetos. Além disso, é 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 prática e possível fazer uma combinação excelente de modelos de dados, objetos, negócios e componentes. Essa linguagem e muito comum e bastante utilizada.

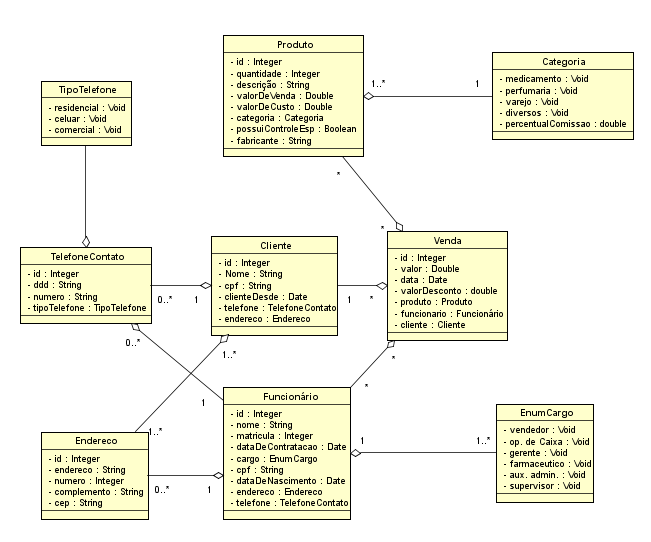
Com a UML, mesmo que possível fazer a representação dos softwares, isso por meio de seus modelos a objetos, essa linguagem não mostra que tipo de trabalho deve ser feito, ela não tem um processo que padroniza como o trabalho deve realmente ser feito. Então assim temos muitos outros objetivos como o que fazer como fazer, quando deve ser feito e porque deve ser feito. Torna-se necessária 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 formam as representações através de gráficos do modelo de um sistema ou software, que torna necessário o uso de combinações para se obter os melhores aspectos e visões do software.

* + 1. DIAGRAMAS ESTRUTURAIS

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

**Diagramas de Classe** –É 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.

Figura 11- Diagrama de Classes

Fonte: Autoria própria

**Diagrama de Objeto** – Este diagrama está 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.

Figura 12- Diagrama de Objeto

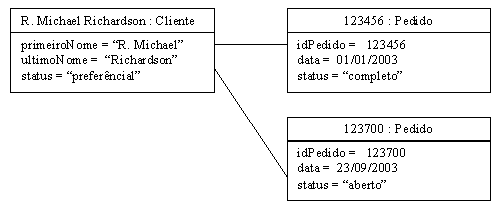
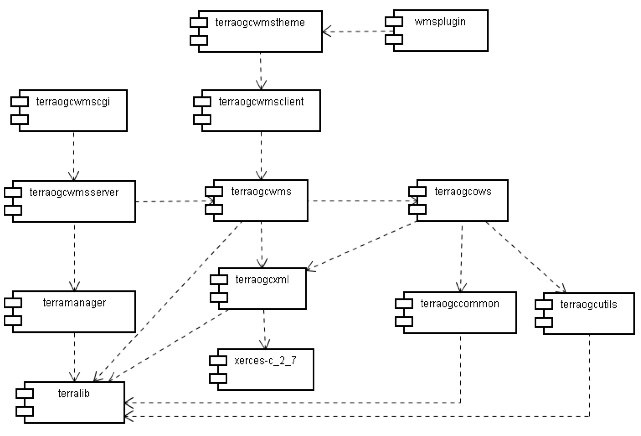


Diagrama de Objeto disponível em <http://homepages.dcc.ufmg.br/~amendes/GlossarioUML/glossario/conteudo/objetos/FigurasDigObjetos/diagramaObjetos.gif>. Acesso em 26 de abril de 2016.

**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.

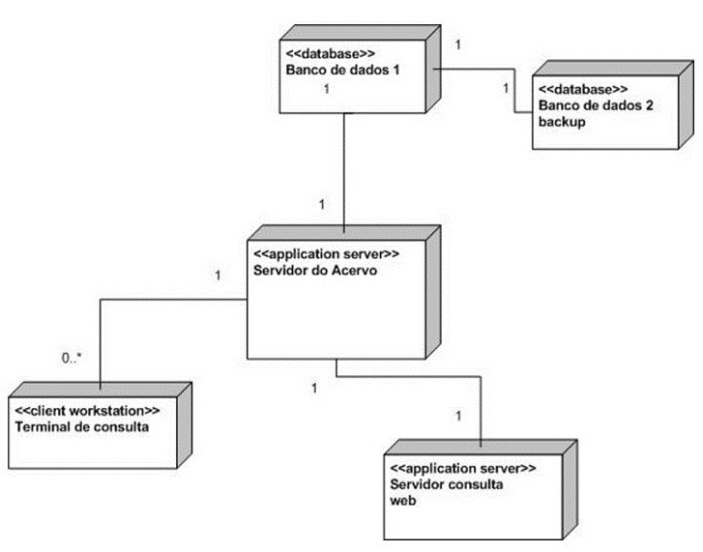
Figura 13- Diagrama de Componentes



Disponível em <https://lh3.googleusercontent.com/lJN4APf5abN6i-OIsKhTfsruo03YQFv4H5ioTEmnLRdeK20oT-3MGag3rShl4jA8mqjsFmFLkU4tuJWgfLS6zwDW2chas72YxImPJobaN6LYC8fryqomaywdxEZZI3B-Ji31BNI>. Acesso em 26 de abril de 2016.

**Diagrama de Implementação** – Faz a modelagem dos relacionamentos entre os recursos de infraestrutura, de artefatos ou de redes do sistema. Ele é 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.

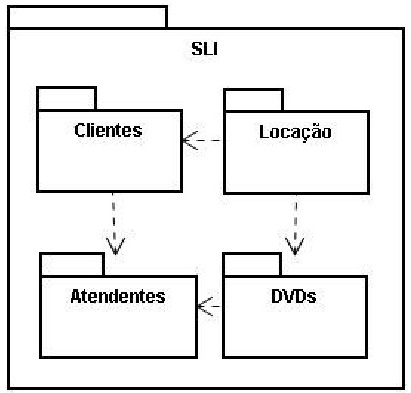
Figura 14- Diagrama de Implantação



Disponível em <http://images.slideplayer.com.br/1/327235/slides/slide\_6.jpg>. Acesso em 26 de abril de 2016.

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

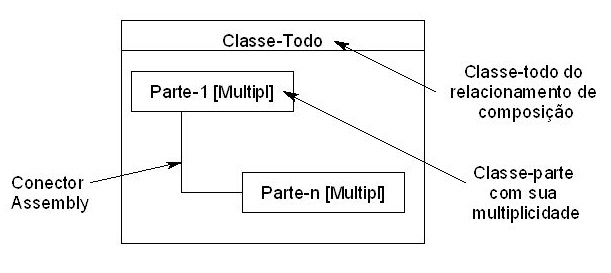
Figura 15- Diagrama de Pacotes



Disponível em <http://image.slidesharecdn.com/umlpacotes-100819163942-phpapp02/95/uml-diagrama-de-pacotes-6-728.jpg?cb=1282236021>. Acesso em 28 de abril de 2016.

**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 instâncias que se ajuda para completar uma determinada tarefa a ser realizada, de tal forma que a mesma instância mostra uma ligação entre as outras instâncias e os seus respectivos papéis a serem cumpridos naquela tarefa.

Figura 16- Diagrama de Estrutura



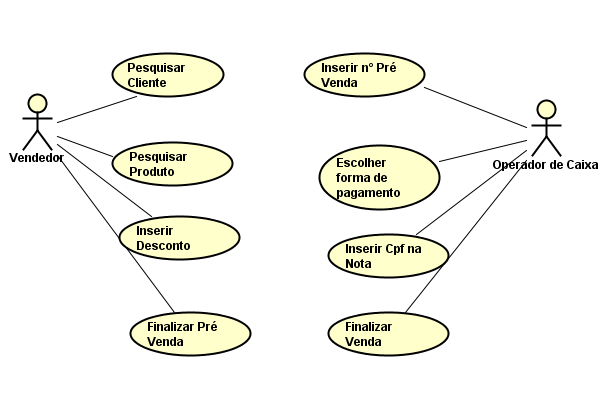
Disponível em <http://images.slideplayer.com.br/10/2843796/slides/slide\_27.jpg>. Acesso em 28 de abril de 2016.

* + 1. DIAGRAMAS COMPORTAMENTAIS

Como o próprio nome já diz, os diagramas comportamentais são aqueles que mostram alguma alteração na classe. Em UML é 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 análise e o levantamento de requisitos de um determinado sistema seja por meio geral ou até mesmo informal em suas distintas fases de projeto.

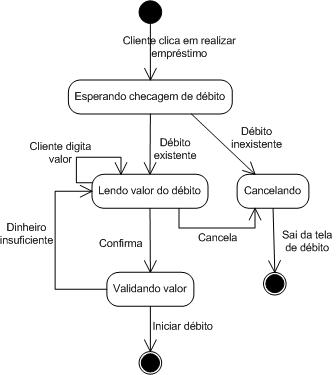
Figura 17- Diagrama de Caso de Uso



Fonte: Autoria própria

**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.

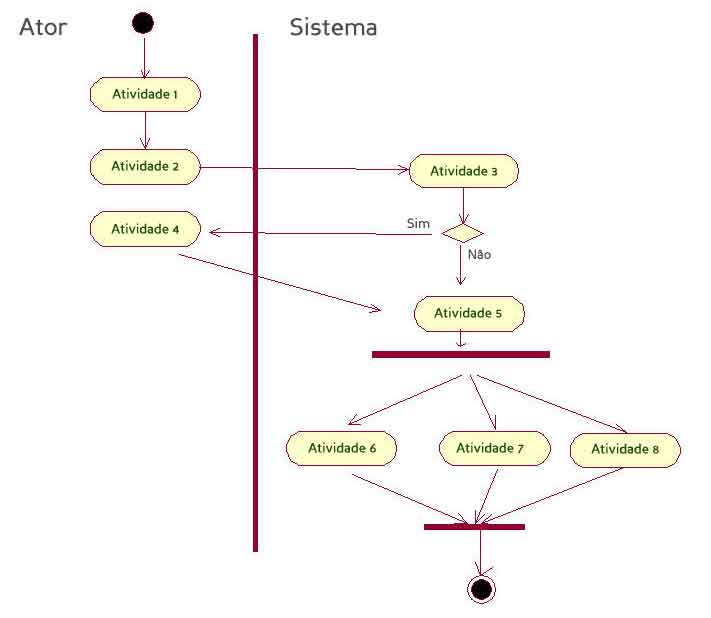
Figura 18- Diagrama de Estado



Disponível em <http://www.micreiros.com/wp-content/uploads/Diagrama-de-Estado.jpg>. Acesso em 28 de abril de 2016.

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

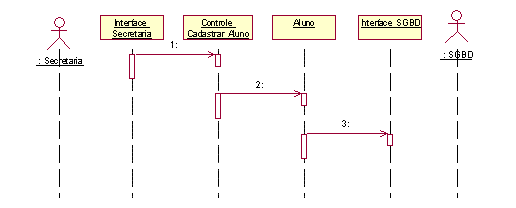
Figura 19- Diagrama de Atividades



Disponível em <http://www.micreiros.com/wp-content/uploads/DiagramaAtividades.jpg>. Acesso em 29 de abril de 2016.

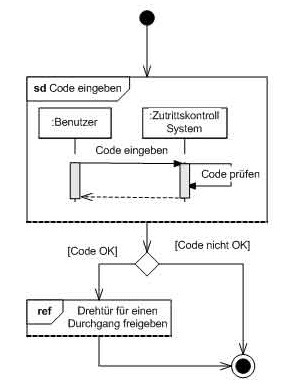
**Diagrama de Interação** – Este diagrama se divide em outros quatro diagramas sendo eles, diagrama de Sequência (mostra em ordem de tempo quando as mensagens são trocadas entre os objetos), Diagrama Geral de Interação (Exibe a variação dos diagramas de atividades que mostra uma visão geral dentro de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

Figura 20- Diagrama de Sequência



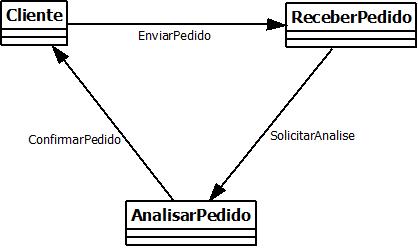
Disponível em <http://www.macoratti.net/vb\_uml2b.gif>. Acesso em 29 de abril de 2016.

Figura 21- Diagrama Geral de Interação



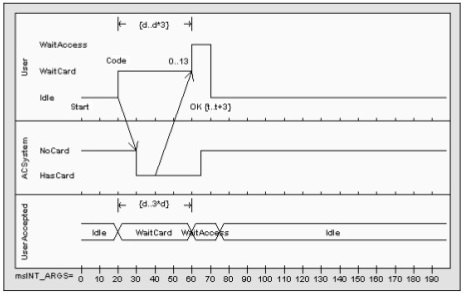
Disponível em <http://nusseagora.dominiotemporario.com/wp-content/uploads/2009/09/Interacao.jpg>. Acesso em 29 de abril de 2016.

Figura 22- Diagrama de Comunicação



Disponível em <https://upload.wikimedia.org/wikipedia/commons/7/72/Diagrama\_de\_Colaboracao.jpeg>. Acesso em 29 de abril de 2016.

Figura 23- 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>. Acesso em 29 de abril de 2016.

* 1. HIBERNATE

Para facilitar a interação entre a aplicação o banco de dados, utilizaremos o framework Hibernate, que por sua eficiência é muito utilizado nas aplicações Java atualmente, “é um framework de persistência que tem como finalidade armazenar objetos em bases de dados relacionais ou fornecer uma visão orientada a objetos de dados relacionais existentes” (BAUER e KING, 2005, p. 19).

O Hibernate facilita muito o desenvolvimento de software, pois evita que códigos SQL (Structured Query Language) fiquem misturados com o código fonte Java, isso evita infortúnios, como ter que reescrever os códigos SQL em caso de uma eventual mudança do SGBD (Sistema Gerenciador de Banco de Dados). Outra vantagem é que paradigma orientado a objetos (Java) e o paradigma relacional (banco de dados) são bastante diferentes, e o Hibernate evita o trabalho o processo de adaptação entre esses paradigmas, fica mais fácil adaptar conceitos do Java como herança, composição, polimorfismo para um banco de dados relacional, e igualmente conceitos de banco de dados relacional como chave primária, chave estrangeira para Java.

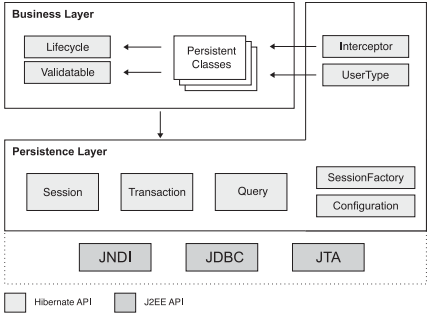
Entretanto, não só apenas vantagens que o Hibernate proporciona, ele não um curinga que deve ser utilizado em qualquer projeto de desenvolvimento de software que utilize Java e um banco de dados relacional, há algumas ponderações que devem feitas para que seja feita a escolha.

Perda de performance, já que há uma camada adicional se compararmos com o JDBC simples, além das configurações das tabelas são lidas na inicialização da aplicação o que pode torná-la mais lenta.

Se aplicação envolver processamento em lote (tarefas processadas uma após a outra), nesta parte da aplicação não dever utilizado Hibernate, pois não é adequado.

Para que a aplicação tenha seu desenvolvimento facilitado e sua performance não seja afetada é importante sempre procurar adequar o framework a aplicação e não o inverso, partes da aplicação podem usar do Hibernate, enquanto partes que possam ter a performance degradada podem usar o JDBC simples.

Estrutura do Hibernate



Fonte: BAUER e KING(2005, p. 37)

Na estrutura do Hibernate há cinco interfaces, chamadas core interfaces que são usadas nas aplicações que usam o Hibernate, elas são responsáveis por armazenar e recuperar objetos na aplicação, abordaremos abaixo uma rápida explicação sobre o funcionamento de cada uma.

* **Session** – esta é a principal interface utilizada pelas aplicações, sua função é criar e destruir sessões no acesso ao banco de dados. É importante ressaltar que session não threadsafe, ou seja, o acesso ao banco de dados pela aplicação deve ser feito por um a thread de cada vez, isso lhe dá a vantagem de tornar o acesso ao banco de dados menos custoso.
* **SessionFactory** – É responsável por retornar para a aplicação uma instância de Session, graças a ela o Hibernate pode acessar múltiplas bases de dados, já que consegue trabalhar com múltiplas threads.
* **Configuration** – esta interface é responsável pela configuração e inicialização do Hibernate, a configuração é possível pois aqui será localizado o documento contendo as informações, o que tornará possível a inicialização do Hibernate.
* **Transaction** – como o nome sugere, esta interface é responsável pelo controle das transações no Hibernate, a sua utilização é opcional, o desenvolvedor pode controlar as transações na própria infraestrutura da aplicação.
* **Query** – esta interface permite a realização e controles das queries que podem ser escritas em HQL (Hibernate Query Language) - que é a linguagem de consulta orienta a objetos do hibernate - ou ainda no SQL nativo do SGBD

1. CRONOGRAMA

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ATIVIDADE** | **2016** | | | | | **2016** | | | | |  | |
| **JAN** | **FEV** | **MAR** | **ABR** | **MAI** |  | **AGO** | **SET** | **OUT** | **NOV** | **DEZ** | |
| 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 |  |

1. 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;

**O básico sobre UML** <https://docs.kde.org/trunk4/pt\_BR/kdesdk/umbrello/uml-basics.html#about-uml> Acesso em 24 de Abril de 2016;

**UML na Engenharia de Software** <http://www.infoescola.com/engenharia-de-software/uml/> Acesso em 24 de Abril de 2016;

**Os principais Diagramas de UML** <https://www.profissionaisti.com.br/2011/07/os-principais-diagramas-da-uml-resumo-rapido/> Acesso em 25 de Abril de 2016;

**Wikipédia UML** <http://pt.wikipedia.org/wiki/UML> Acesso em 25 de Abril de 2016;

**Artigo sobre UML**< http://projetos.inf.ufsc.br/arquivos\_projetos/projeto\_721/artigo.tcc.pdf> Acesso em 27 de Abril de 2016;

**UML** <http://www.macoratti.net/uml\_vb.htm> Acesso em 27 de Abril de 2016;

**Diagramas Estruturais** <http://micreiros.com/uml-e-os-diagramas-estruturais/> Acesso em 28 de Abril de 2016;

**Diagramas Comportamentais** <http://micreiros.com/os-diagramas-comportamentais-da-uml/> Acesso em 28 de Abril de 2016;

LUCKOW, Décio Heinzelmann; MELO, Alexandre Altair de. **Programação Java para a Web**. Aprenda a desenvolver uma aplicação financeira pessoal com as ferramentas mais modernas da plataforma Java 1ª ed. Novatec. 2010;

JANDL JUNIOR, Peter, **JAVA Guia do Programador –** Atualizado para o Java 8 3ª Edição – Novatec. 2015;

Banco de dados relacionais <<http://www.devmedia.com.br/bancos-de-dados-relacionais-artigo-revista-sql-magazine-86/20401>> Acesso em 25/05/2016;

O que e um banco de dado relacional <<https://elias.praciano.com/2013/09/o-que-e-um-banco-de-dados-relacional/>> Acesso em 25/05/2016;

Banco de dados relacional <<https://pt.wikipedia.org/wiki/Banco_de_dados_relacional>> Acesso em 25/05/2016;

Hibernate Advantages and Disadvantages <<http://www.javatutsworld.com/hibernate/hb_advantage_disadv.html>> Acesso em 24/05/2016;

BAUER, Christian e KING, Gavin. **Hibernate in Acion.** 2005. Greenwich. Manning.