

ETEP - FACULDADE DE TECNOLOGIA   
DE SÃO JOSÉ DOS CAMPOS

PROPOSTA DE SISTEMA DE CATÁLOGO DIGITAL UTILIZANDO METODOLOGIA BASEADA EM TESTES

João Paulo Gomes dos Santos

Trabalho de Conclusão de Curso de Bacharelado em Engenharia da Computação, orientado pelo Prof. Edizon Basseto Júnior.

ETEP Faculdades

São José dos Campos

2011

ETEP - FACULDADE DE TECNOLOGIA DE SÃO JOSÉ DOS CAMPOS

PROPOSTA DE SISTEMA DE CATÁLOGO DIGITAL UTILIZANDO METODOLOGIA BASEADA EM TESTES

João Paulo Gomes dos Santos

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Edizon Basseto Júnior

ETEP Faculdades

São José dos Campos

2011

“A dúvida é o principio da sabedoria”.

Aristóteles

AGRADECIMENTOS

Agradeço principalmente a Deus que me concedeu a capacidade, força e perseverança para o desenvolvimento e conclusão deste projeto, pois sabemos que sem ele não somos nada, e nada, poderíamos fazer.

Também agradeço à minha namorada Ana Cláudia Narumi Kameda e aos familiares, que me apoiaram na elaboração deste projeto. Gostaria de exprimir os mais sinceros agradecimentos e aqui reconhecer a sua importante contribuição.

Aos meus amigos de classe, pelo apoio incondicional e pela grande amizade que têm dedicado e que nunca pouparam esforços para me ajudar. Aos professores, em especial ao professor Edizon Basseto Júnior pela disponibilidade e força de vontade de orientar, sendo os educadores da ETEP Faculdades os que sempre me incentivaram.

Enfim, agradeço as todas as pessoas que contribuíram de forma direta ou indireta para realização do meu trabalho e acreditando na minha capacidade de realizar esse projeto.

RESUMO

Nos últimos anos a importância do mercado de desenvolvimento de software vem aumentando e a concorrência entre as empresas que desenvolvem software aumenta junto, essa empresas buscam entregar software com qualidade e com menores custos para seus clientes. Para atingir tal objetivo foram criadas práticas para otimização do processo de desenvolvimento de software, tendo como objetivo a redução de custos e aumento da qualidade do produto final. Este trabalho mostra uma maneira de combinar princípios ágeis de desenvolvimento de software de modo a se obter uma melhoria no processo de desenvolvimento de software. O trabalho utiliza como base o pensamento *Lean* e práticas de *Extreme Programming* como o *Test Driven Development*.

***Palavras Chave:*** *Lean Thinking; Extreme Programming; Test Driven Development;*

SUMÁRIO

**Pág.**

LISTA DE FIGURAS 8

LISTA DE TABELAS 9

LISTA DE SÍMBOLOS 10

1 INTRODUÇÃO 11

1.1 ESTUDO DE CASO 12

1.2 OBJETIVO DO TRABALHO 13

1.3 JUSTIFICATIVA 13

2 FUNDAMENTAÇÃO TEÓRICA 14

2.1 TESTES DE SOFTWARE 14

2.1.1 TESTES DE UNIDADE 15

2.1.2 TESTES DE INTEGRAÇÃO 15

2.2 TESTES AUTOMATIZADOS 16

2.3 LINGUAGEM DE PROGRAMAÇÃO RUBY 17

2.4 PADRÃO ARQUITETURAL MVC 18

2.5 FRAMEWORK DE DESENVOLVIMENTO RAILS 18

2.6 DESENVOLVIMENTO PARA DISPOSITIVOS MÓVEIS 20

2.6.1 JAVA MICRO EDITION (J2ME) 20

2.6.2 IPHONE 21

2.6.3 ANDROID 21

2.7 METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE 24

2.7.1 CASCATA 24

2.7.2 RATIONAL UNIFIED PROCESS (RUP) 25

2.7.3 PROGRAMAÇÃO EXTREMA (XP) 26

2.7.4 LEAN 27

2.7.5 TEST DRIVEN DEVELOPMENT (TDD) 29

3 METODOLOGIA 31

3.1 METODOLOGIA UTILIZADA NO DESENVOLVIMENTO 32

3.2 DESENVOLVIMENTO DO SISTEMA WEB 33

3.2.1 TECNOLOGIAS UTILIZADAS 34

3.2.2 DESENVOLVIMENTO 36

4 RESULTADOS (estilo Título 1) 37

5 CONCLUSÃO (estilo Título 1) 38

APÊNDICE A (estilo APÊNDICE) 42

ANEXO A - ABREVIATURA DOS MESES (estilo ANEXO) 44

LISTA DE FIGURAS

**Pág.**

[Figura 3.1 – Configuração de página em papel A4. 32](#_Toc294361523)

[Figura A.1 - Diagrama de funcionamento. 40](#_Toc294361524)

[Figura A.2 - Como apresentar uma figura longa 41](#_Toc294361525)

[Figura A.3 - Movimento realocar tarefa. 41](#_Toc294361526)

LISTA DE TABELAS

**Pág.**

[Tabela 3.1 – Botões da barra de ferramentas 32](#_Toc238012854)

[Tabela 1- Abreviaturas 42](#_Toc238012855)

LISTA DE SÍMBOLOS

*f* - Vetor das Forças Aplicadas e Giroscópicas

*E* - Energia Cinética

# INTRODUÇÃO

A Toyota foi berço para o surgimento do pensamento *Lean*. Logo após a segunda guerra mundial, o Japão estava destruído e as empresas, precisando se reerguer, tinham uma produtividade muito baixa e uma grande falta de recursos para produção, pensando nisso que Taiichi Ohno e Shigeo Shingo desenvolveram o pensamento *Lean*, justamente para aumentar e otimizar a produção da empresa (OHNO, 1988).

O modelo utilizado na Toyota se tornou um sucesso, foi adaptado e adotado em muitos outros segmentos, virando referência quando se fala em redução de custos e em otimização da produção (OHNO, 1988).

Em 2003, Mary Poppendieck escreveu o livro *Lean Software Development: An Agile Toolkit* que mostra como o pensamento *Lean* pode ser adaptado para o desenvolvimento de software. Este livro pode ser considerado um marco para as empresas que desenvolvem software, pois com base nele muitas empresas puderam conhecer e aplicar os princípios do pensamento *Lean* e assim conseguiram aumentar seus desempenhos (POPPENDIECK, 2003).

Como um dos principais pilares do pensamento *Lean* temos a redução de desperdício, essa redução de desperdício pode ser alcançada com a adoção de algumas práticas de *Extreme Programming* (POPPENDIECK, 2003)*.*

*Extreme Programming* comtempla uma série de práticas para o sucesso no desenvolvimento de um software, uma dessas práticas é fazer com que o *feedback* do sistema seja rápido.

Para conseguirmos ter o *feedback* suficientemente veloz podemos utilizar uma prática conhecida como *Test Driven Development* (**TDD**), que diz que os testes para uma determinada funcionalidade do software devem ser criados antes mesmo do desenvolvimento da mesma, assim podemos ter resultados praticamente imediatos a respeito do que foi implementado para atender à funcionalidade em questão (FREEMAN, 2009).

Cada vez mais o mercado de desenvolvimento de software está valorizando práticas presentes no *Extreme Programming*, alguns dos principais motivos para essa valorização são a qualidade do resultado e a redução de custos durante o desenvolvimento.

Quando pensamos em **TDD** logo pensamos em criar testes antes de desenvolver e isso nos dá a impressão que estamos aumentando o ciclo de desenvolvimento e vamos gastar mais tempo, segundo (FREEMAN, 2009) o que ocorre é exatamente o contrário. Quando criamos testes para uma funcionalidade somos obrigados a pensar como ela será implementada e assim conseguimos descobrir problemas do nosso design antecipadamente.

Outra vantagem do uso de **TDD** é a possibilidade de agrupar os testes gerados por todas as funcionalidades já implementadas e executá-los em conjunto, garantindo que nenhuma funcionalidade existente foi afetada por novas implementações (FREEMAN, 2009).

Este trabalho tem como objetivo mostrar o uso de **TDD** aplicado em sistema de exemplo, onde poderemos observar as vantagens e a velocidade de feedback proporcionada por essa metodologia de desenvolvimento.

Mais informações sobre o pensamento *Lean*, *Extreme Programming* e **TDD** serão apresentadas nos próximos capítulos.

Para tornar mais claro os benefícios do uso do pensamento *Lean* e de práticas de *Extreme Programming* para o desenvolvimento de software, será utilizado um estudo de caso, que será explicado com mais detalhes no próximo item deste trabalho.

## ESTUDO DE CASO

Cada dia que passa as empresas de tecnologia estão investindo mais em tecnologias móveis.

Recentemente os *tablets* tem ganharam um lugar de destaque entre os dispositivos móveis. Os *tablets* mais modernos são praticamente computadores, fazem tudo que um computador convencional faz, apenas com uma diferença: são bem menores.

Inicialmente os *tablets* eram muito caros, impossibilitando a adoção pelo público. Com o passar do tempo o custo benefício dos aparelhos cresceu muito, fato que aumentou as vendas e criou muitas oportunidades de negócio.

Hoje existem várias empresas que são especialistas em desenvolvimento de software para dispositivos móveis, devido a grande demanda das empresas consumidoras destes aplicativos.

Este trabalho utilizará como estudo de caso um sistema de catálogo digital que tem seu funcionamento descrito abaixo.

A primeira parte do projeto é composta por um sistema web capaz de armazenar os dados do cliente. Depois que o cliente fizer o cadastro de seus dados ele poderá visualizá-los em *tablets*. Os *tablets* serão capazes de sincronizar seus dados com os dados cadastrados no sistema web.

Um exemplo de aplicação deste projeto poderia ser um restaurante, onde os *tablets* podem ser usados como cardápios. E através dos dados que foram cadastrados no sistema web os *tablets* seriam alimentados, reduzindo o custo e o tempo para a atualização dos cardápios.

## OBJETIVO DO TRABALHO

O primeiro objetivo do trabalho é propor uma metodologia para desenvolvimento de software utilizando uma abordagem baseada em testes automatizados, mostrando ferramentas para testes e vantagens do uso de testes no desenvolvimento de software.

O segundo objetivo do trabalho é mostrar a integração de sistemas web com sistemas desenvolvidos para dispositivos móveis.

## JUSTIFICATIVA

Este tema foi escolhido pela necessidade de aumento de qualidade que existe durante o desenvolvimento de software e também pelo a possibilidade de estudar o desenvolvimento para dispositivos móveis, que está em ascensão.

# FUNDAMENTAÇÃO TEÓRICA

## TESTES DE SOFTWARE

Como em qualquer área de desenvolvimento, os testes são essenciais para aumentar a qualidade do produto final, no desenvolvimento de software não é diferente.

A definição de teste de software segundo (MYERS, 2004) diz que teste de software é o processo de executar um software com o intuito de encontrar erros.

Devido à natureza dinâmica dos softwares não é possível garantir que eles fiquem livres de erros (MYERS, 2004), mesmo executando exaustivas baterias de testes.

Segundo (MCGREGOR, 2001) e (GUERRA, 2005), existem vários tipos de teste de software cada um com um propósito específico, alguns dos principais são:

* Teste de unidade: O teste deve ser feito sobre uma unidade do software, ou seja, sobre uma classe ou método. Testes de unidade ajudam a garantir que a unidade a ser testada está desacoplada das outras unidades e coesa. Seu principal objetivo é verificar se a unidade possui o comportamento esperado.
* Teste de Interação: O teste deve ser feito sobre um grupo de classes do software e devem verificar se a troca de mensagens entre as classes ocorre corretamente.
* Teste de Componente: O teste deve ser feito sobre um grupo de classes e devem ser verificadas as entradas e as saídas do componente.
* Teste de Integração: Deve ser feito sobre um grupo de componentes.
* Teste de Sistema: Deve ser realizado utilizando o software como um todo, usando a visão do usuário final. Este teste ajuda na verificação dos requisitos do software.
* Teste de Validação: Assim como o teste de sistema, este teste deve utilizar o software como um todo, tendo como única diferença a necessidade de acompanhamento do cliente durante os testes.

A execução dos testes de forma manual é muito custosa para grandes softwares e assim a execução de todos os testes a cada nova versão do software acaba se tornando inviável para softwares de grande porte (GUERRA, 2005).

A qualidade do software diminui à medida que a qualidade dos testes diminui, ou seja, quanto menor a eficácia dos testes do software menor será a qualidade do produto final, já que a garantia de que tudo esteja funcionando corretamente irá diminuir (GUERRA, 2005).

Nos próximos capítulos serão detalhados os principais tipos de teste utilizados neste trabalho.

### TESTES DE UNIDADE

A unidade fundamental de um sistema orientado a objetos é denominada classe, o teste de unidade de uma classe tem como objetivo principal verificar se a implementação da classe corresponde ao que foi especificado para a mesma (MCGREGOR, 2001).

Garantindo que todas as classes estão cobertas por testes de unidade, seguindo suas especificações, haverá uma grande chance de que qualquer problema no sistema desenvolvido foi causado por erros de integração entre as unidades.

Segundo (MCGREGOR, 2001), o tempo gasto com correção de problemas no sistema é reduzido drásticamente, já que as unidade são coesas e testadas corretamente.

Pode-se perceber uma grande redução no tempo gasto com correção de defeitos no sistema em que os testes unitários são adotados. Outra vantagem é a melhoria no design do sistema, ou seja, os componentes do sistema deve ser coesos e estar desacoplados uns dos outros para que seja possível a criação de testes de unidade (HUNT, 2003).

### TESTES DE INTEGRAÇÃO

Um sistema orientado a objetos é composto por um conjunto de objetos que colaboram entre si para atingir um objetivo. O modo com que estes objetos interagem entre si determina o resultado do sistema (MCGREGOR, 2001).

Um exemplo seria o caso de um sistema em que todas as unidades tem o comportamento correto quando isoladas, mas quando colocadas para interagirem entre si o objetivo do sistema não é atingido, ou seja, existe um problema de integração entre as unidades (MCGREGOR, 2001).

Segundo (MCGREGOR, 2001) o principal objetivo de um teste de integração é garantir que as mensagens enviadas de um objeto para outro sejam executadas de maneira correta.

Antes de realizar testes de integração deve-se garantir que as unidades participantes do teste estão cobertas por testes de unidade.

A segurança para fazer uma atualização em um sistema aumenta a medida que a cobertura de testes de integração aumenta, ou seja, os testes de integração ajudam a garantir que as funcionalidades do sistema estão funcionando corretamente.

Uma outra vantagem dos testes de integração é a garantia de que uma alteração não danificou alguma funcionalidade que estava correta.

## TESTES AUTOMATIZADOS

Os testes são parte importante do desenvolvimento de um sistema, normalmente eles são executados de forma funcional, ou seja, dependem da interação humana para serem realizados, a execução dos testes desta maneira é muito custosa, já que consome muito tempo durante sua execução.

Uma alternativa para reduzir os custos dos testes de software é a automatização da execução dos mesmos, de forma que não seja necessária a interação humana em nenhuma parte deste processo.

Executar uma bateria de testes manuais a cada iteração do sistema se torna inviável à medida que o software cresce (GUERRA, 2005), com o intuído de contornar este problema surgiu o conceito de automação de testes.

Automatizar o processo de testes aumenta a confiabilidade do software a ser desenvolvido e garante maior agilidade no ciclo de desenvolvimento, já que os testes de regressão podem ser executados a cada iteração do ciclo de desenvolvimento (DUSTIN, 2002).

Com a execução dos testes de regressão de maneira automatizada podemos saber rapidamente se qualquer uma das modificações efetuadas causou algum impacto indesejado em outras partes do software (DUSTIN, 2002).

Recomenda-se que a cada alteração no sistema seja criado um caso teste para cobrir a nova situação criada com a alteração em questão , depois de elaborar o caso de teste é necessário adicioná-lo ao conjunto de testes do sistema em desenvolvimento, fazendo com que o conjunto de testes automatizados esteja sempre atualizado.

## LINGUAGEM DE PROGRAMAÇÃO RUBY

A linguagem de programação Ruby foi criada por Yuki hiro Matsumoto, no Japão, no ano de 1995, desde então vem se tornando uma linguagem robusta o suficiente para ser utilizada em sistemas de qualquer natureza (MATSUMOTO, 2001).

Ruby é uma linguagem de programação totalmente orientada a objetos, já que tudo em Ruby é um objeto, sem exceções.

Muitas linguagens de programação incorporaram aspectos de programação orientada a objetos, mas poucas conseguem ser totalmente orientadas a objetos assim como Ruby (MATSUMOTO, 2001).

Como exemplo pode-se citar a linguagem de programação Java, ela é classificada como uma linguagem de programação orientada a objetos, mas existem representações de tipos primitivos (integer, double, byte, char, etc) em Java, ou seja, esses tipos primitivos não são objetos, sendo assim Java não é uma linguagem totalmente orientada a objetos (SIERRA, 2008). Em Ruby até os inteiros são objetos da classe FixNum.

Segundo (MATSUMOTO, 2001), quando a linguagem Ruby foi desenvolvida o principal foco de seu criador era gerar uma linguagem que pudesse aumentar a produtividade dos desenvolvedores de forma fácil, com base nessa necessidade do criador da linguagem, Ruby adquiriu algumas características, descritas abaixo (MATSUMOTO, 2001):

* Programação Interativa: Ruby é uma linguagem de script, ou seja, não é necessário compilar o código. Existe um interpretador para facilitar o desenvolvimento.
* Programação Dinâmica: Praticamente tudo que é feito em Ruby é feito em tempo de execução. Os tipos das variáveis, expressões, classes e definições de métodos são determinados em tempo de execução. Uma característica interessante é a possibilidade de alteração de suas classes em tempo de execução de maneira totalmente dinâmica, garantindo grande flexibilidade no desenvolvimento.
* Sintaxe Familiar: A sintaxe da linguagem Ruby é muito parecida com a sintaxe de linguagens renomadas, como por exemplo, Java, Perl, Python, C/C++, essa característica ajuda na disseminação da linguagem.
* Bibliotecas de classes: Ruby possui uma grande quantidade de bibliotecas que já vem com a distribuição padrão e cobrem um vasto domínio de necessidades, começando pelos tipos básicos (strings, arrays, hashes) e indo até tópicos mais avançados, como programação para recursos de rede e threads. Mesmo possuindo uma grande quantidade de bibliotecas nativas, é possível adicionar novas bibliotecas desenvolvidas por terceiros.
* Portabilidade: Programas escritos em Ruby podem ser rodados em qualquer ambiente computacional que possua um interpretador Ruby, ou seja, é possível criar programas Ruby em um plataforma e migrá-los para outra sem a necessidade de nenhuma modificação.

## PADRÃO ARQUITETURAL MVC

Em 1979, Trygve Reenskaug desenvolveu um padrão arquitetural para desenvolvimento de aplicativos, padrão conhecido como Modelo – Visão – Controlador **(MVC)**.

Como o próprio nome sugere esse padrão arquitetural divide a arquitetura das aplicações em 3 camadas básicas (RUBY, THOMAS e HANSSON, 2010):

* Modelo: O modelo é responsável por representar os estados dos objetos da aplicação. O estado dos objetos pode ser considerado transiente, quando o estado é mantido apenas por algumas interações, ou persistente, quando o estado é grava em um mecanismo de persistência, como os banco de dados. O modelo é muito mais que apenas dados, ele assegura algumas regras de negócios relacionadas com os dados representados por ele.
* Visão: A visão é responsável por gerar a interface com o usuário, baseando-se nos dados do modelo.
* Controlador: Os controladores são responsáveis por coordenar o funcionamento da aplicação, eles recebem os eventos gerados pela visão, interagem com os modelos e respondem aos eventos delegando o fluxo para outras visões.

## FRAMEWORK DE DESENVOLVIMENTO RAILS

Rails é um framework para desenvolvimento de aplicações para web escrito em Ruby. Criado por David Heinemeier Hansson, tem como seu principal objetivo facilitar o desenvolvimento de aplicações para web, levando em consideração alguns aspectos que os desenvolvedores precisam conhecer para começar a desenvolver uma aplicação (RUBY, THOMAS e HANSSON, 2010).

Os principais conceitos utilizados na criação do Rails foram:

* Convenção ao invés de configuração: Principal fundamento do framework, para tudo no Rails existe uma convenção, como por exemplo:
  + Estrutura de diretórios da aplicação;
  + Padrão de nomes dos controladores, sempre seguindo o nome dos modelos;
  + Padrão de nomes das visões, sempre seguindo os nomes das ações dos controladores;
  + Padrão de nomes de tabelas do banco de dados, sempre o nome do modelo no plural;

Conhecer e obedecer as convenções é muito importante para aproveitar todas as facilidades do framework.

* Não se repita: Grande facilidade para reuso de código através de *plugins*. Alguns do *plugins* mais populares para o desenvolvimento Rails são:
  + Devise: É capaz de criar uma estrutura completa para autenticação de usuários;
  + Will Paginate: Serve para paginar os itens nas listagens;
  + PaperClip: Serve para controlar upload de arquivos;

A estrutura interna do Rails é divida nos seguintes componentes (FERNANDEZ, 2010):

* *Action Controller*: Cuida do gerenciamento dos controles das aplicações. Processa as requisições HTTP, extrai os parâmetros e faz o encaminhamento para a ação desejada. Outros serviços provenientes do *Action Controller* são:
  + Gerenciamento de sessões HTTP;
  + Renderização de templates;
  + Redirecionamentos.
* *Action Dispatch*: É responsável pelo roteamento das requisições HTTP.
* *Action View*: É responsável pela geração das respostas para as requisições feitas, por padrão Rails tem suporte a html, xml e json.
* *Action Mailer*: É utilizado para gerenciar o envio e recebimento de e-mails.
* *Active Record*: É a base para os modelos nas aplicações Rails. Algumas de suas características são:
  + Possibilita a independência de uma banco de dados específico;
  + É capaz de fazer buscas avançadas;
  + Possibilita a criação de relacionamentos entre os modelos.
* *Active Support*: Classes utilitárias usadas por todo o framework Rails.

## DESENVOLVIMENTO PARA DISPOSITIVOS MÓVEIS

Segundo (FLING, 2009) o telefone foi uma das maiores invenções da humanidade, ele revolucionou as comunicações, permitindo que as pessoas, mesmo distantes, consigam se comunicar de maneira rápida e fácil.

Hoje em dia os telefones representam dispositivos com muitas funcionalidades, com um telefone é possível fazer ligações, mandar mensagens de texto, navegar na internet, jogar, acessar mapas, ouvir músicas, assistir vídeos e utilizar aplicações em geral.

Graças a evolução do hardware para dispositivos móveis é possível, cada vez mais, adicionar funcionalidades aos telefones atuais.

Com essa evolução dos telefones, o desenvolvimento de aplicativos para plataformas móveis vem se popularizando, algumas das principais tecnologias utilizadas para desenvolvimento de aplicações móveis serão introduzidas a seguir.

### JAVA MICRO EDITION (J2ME)

Java Micro Edition (**J2ME**) é um conjunto de tecnologias e especificações que têm como principal objetivo a criação de uma máquina virtual Java capaz de ser executada em dispositivos com limitações de recursos de hardware, ideal para dispositivos móveis.

Aplicativos escritos usando J2ME podem ser executados em qualquer dispositivo que possua uma máquina virtual Java, como os dispositivos listados abaixo:

* Celular;
* Palm;
* Pager;
* Tablets;

A linguagem de desenvolvimento é puramente Java com limitações em algumas bibliotecas, isso facilita o crescimento da comunidade que desenvolve para esta plataforma (RISCHPATER, 2008).

### IPHONE

O IPhone é um dispositivo criado pela empresa Apple, é considerado um dos telefones mais sofisticados do mundo, sempre em busca da simplicidade e praticidade durante seu uso.

O desenvolvimento para IPhone é feito utilizando uma linguagem de programação chamada Objetctive-C e ferramentas feitas para rodarem apenas em máquinas Apple, ou seja, o desenvolvimento é mais restrito, ao contrário do desenvolvimento para J2ME que é aberto para qualquer plataforma (NEUBURG, 2011).

Segundo (PILONE, 2010) as ferramentas necessárias para o desenvolvimento de aplicações para IPhone são:

* XCode: Ferramenta para gerenciamento e edição de código para IPhone, com ele é possível navegar pelos diretórios do aplicativo, editar o código e depurar o código. É um ambiente de desenvolvimento completo e se integra completamente com os outros componentes necessários para o desenvolvimento de aplicativos para IPhone.
* *Interface Builder*: Ferramenta para a criação das interfaces visuais da aplicação, você pode arrastar e soltar os componentes para criar suas interfaces visuais, facilitando muito o desenvolvimento.
* *Instruments*: Ferramenta utilizada para a coleta de dados de aplicações e otimização das mesmas.
* IPhone *Simulator*: Simulador para testar o aplicativo a ser desenvolvido sem a necessidade de ter um IPhone em mãos para efetuar os testes, reduzindo o tempo de desenvolvimento.

### ANDROID

A plataforma Android foi criada sobre o *kernel* do Linux, ou seja, foi criada baseada num sistema operacional sólido e estável. O Android tem sua arquitetura de sistema como é mostrado na FIGURA 2.1:



Figura 2.1 - Arquitetura do Android. FONTE (BURNETTE, 2010)

Internamente o Android utiliza-se de vários recursos do *kernel* do Linux:

* Gerenciamento de memória;
* Gerenciamento de processos;
* Gerenciamento de Energia;
* Acesso a rede;
* Driver de comunicação com câmera;
* Driver de comunicação com redes sem fio;

A próxima camada acima do *kernel* é composta pelas bibliotecas nativas do sistema, essas bibliotecas são responsáveis por realizar operações básicas para as camadas superiores. Alguns exemplos de componentes presentes na camada de bibliotecas são mostrados a seguir:

* *Surface Manager*: Responsável pelo gerenciamento de janelas, possibilita a criação de diversos efeitos durante o uso do dispositivo.
* Gráficos 2D e 3D: Motor de renderização capaz de gerar gráficos em 2D e 3D em uma única interface visual. É possível utilizar recursos para renderização 3D caso o telefone utilizado seja compatível, possibilitando a execução de jogos com gráficos mais refinados.
* *Codecs* para vídeos e áudio: São responsáveis por interpretar arquivos de áudio e vídeo em diversos formatos, alguns dois mais conhecidos são: AAC, AVC (H.264), H.263, MP3 e MPEG-4.
* Banco de dados interno: Banco de dados SQLite embutido, utilizado para armazenamento persistente de dados.
* Mecanismo para exibição de conteúdo HTML: Para garantir um bom desempenho na exibição de páginas HTML, o Android usa uma biblioteca chamada WebKit, mesma biblioteca utilizada no browser GoogleChrome, Safari, IPhone e celulares Nokia da linha S60.

Juntamente com a camada de bibliotecas existe a camada responsável pela execução do Android, essa camada é composta por uma máquina virtual Java chamada Dalvik, desenvolvida pela Google e otimizada para dispositivos móveis. Todo o código escrito para Android é feito em Java.

Pelo fato da execução das aplicações ocorrer dentro de uma maquina virtual Java é possível acessar grande parte das bibliotecas disponíveis no Java, garantindo um grande ganho de velocidade durante o desenvolvimento.

A próxima camada, logo após a camada de execução, é a camada que possui aplicativos básicos para o funcionamento do telefone, como por exemplo:

* Gerenciador de atividades: Gerencia o ciclo de vida das aplicações, trata de eventos como os que ocorrem quando o usuário recebe uma mensagem de texto ou recebe uma ligação.
* Provedor de conteúdo: Armazena dados que devem ser compartilhados entre os aplicativos, como por exemplo os contatos do telefone do usuário.
* Gerenciador de recursos: Gerencia tudo que é distribuído com os aplicativos que não seja código, como por exemplo imagens, ícones, sons e vídeos.
* Gerenciador de localização: Responsável por capturar e disponibilizar, para as aplicações, dados sobre a localização do dispositivo.
* Gerenciador de notificações: Responsável por exibir alertas disparados pelo o sistema para o usuário, serve como central de comunicação entre o sistema e o usuário.

A última camada na arquitetura de aplicações Android é a camada dos aplicativos que rodam no Android. Essa é a camada pela qual o usuário vai interagir com o sistema, enviando eventos e requisições para os aplicativos. Alguns dos aplicativos que já vem instalados por num sistema Android são:

* Aplicativo para discagem de números de telefone;
* Leitor de email;
* Gerenciador de contatos;
* Navegador de internet;

Através de um aplicativos chamado Android Market é possível instalar dezenas de outros aplicativos no Android.

## METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE

Nesta etapa serão apresentadas algumas metodologias de desenvolvimento mais conhecidas.

### CASCATA

Foi a primeira metodologia para desenvolvimento de software amplamente utilizada pelas empresas. Por volta de 1970 a indústria de software descobriu que o processo de desenvolvimento de software precisava ter uma organização maior, com base nesta necessidade Royce desenvolveu uma metodologia linear para desenvolvimento de software (ROYCE, 1970).

Nesta metodologia o desenvolvimento segue algumas etapas previamente definidas. Segundo (PRESSMAN, 2009), as etapas existentes são:

* Engenharia do Sistema: Etapa inicial dedicada à análise de requisitos de forma ampla, ideal para definir restrições tecnológicas.
* Análise de Requisitos: Levantamento das funcionalidades necessárias para o software. Nesta fase é criado um conhecimento profundo sobre o domínio de negócios do software. Tudo que for levantando deve ser documentado e revisto pelo cliente.
* Projeto: Nesta etapa são tomadas decisões de arquitetura e modelagem respeitando os requisitos levantados nas fases anteriores. O resultado é um documento que representa detalhadamente como cada parte do sistema deve ser implementada.
* Geração de Código: Etapa onde ocorre a codificação do que foi estabelecido na fase de projeto.
* Testes: Nesta etapa deve ser feita a validação de cada funcionalidade do sistema de modo a garantir que o que foi implementado está de acordo com o que foi projetado.
* Manutenção: Etapa destinada à correções nas funcionalidades já implementadas ou destinada à adição de novas implementações.

### RATIONAL UNIFIED PROCESS (RUP)

Rational Unified Process (**RUP**) é a versão do Processo Unificado (**UP**), para desenvolvimento de software, criada pela Rational Corporation (JACOBSON, 1999).

Por muito tempo RUP foi largamente utilizado em grandes aplicações, assim ganhou a fama de que era um processo pesado. Com a crescente utilização de metodologias ágeis para desenvolvimento de software, a Rational Corporation tem seus esforços voltados para mostrar que o RUP é um processo adaptável e flexível (EVANS, 2003).

O processo unificado é dividido em quatro fases:

* Concepção: Etapa dedicada à coleta de requisitos, levantamento dos principais riscos e um levantamento dos esforços.
* Elaboração: Os requisitos são estudados mais profundamente. A arquitetura do sistema é desenvolvida e testada com objetivo de reduzir riscos e obter uma estimativa mais precisa dos esforços. Esta etapa é subdividida em iterações que contêm: levantamento de requisitos, modelagem, programação e testes.
* Construção: Com base na estrutura criada na etapa de elaboração, as implementações do sistemas são iniciadas. Essas implementações são divididas em iterações incluem codificação e testes de aceitação com os clientes.
* Transição: As implementações estão finalizadas e o sistema pode ser entregue para o cliente. Alguns ajustes no sistema são aceitáveis antes da entrega final para o cliente.

### PROGRAMAÇÃO EXTREMA (XP)

Depois de muitos anos trabalhando com desenvolvimento de software, Kent Beck propôs uma metodologia para desenvolvimento de software conhecida como Programação Extrema **(XP)**, também conhecida como *Extreme Programming*.

Com base em sua experiência com SmallTalk, em 1996 Beck publicou seu livro sobre técnicas para programação (BECK, 1996). No mesmo ano da publicação do livro, Beck foi convidado para liderar um projeto muito importante na Chrysler, era um sistema para controlar a folha de pagamento da empresa, este sistema já estava com prazos e custos estourados e ainda não possuía resultados. Com a participação de Martin Fowler e Ron Jeffries, Beck conseguiu alta produtividade para a equipe e eles conseguiram entregar um sistema de excelente qualidade começando do zero e gastando menos tempo do que foi gasto nas tentativas anteriores.

Neste projeto da Chrysler, Beck decidiu utilizar uma série de práticas que foram consideradas eficientes em outros projetos, este foi o início da Programação Extrema. Algumas das práticas utilizadas, que foram impactantes para o aumento da qualidade do produto, são:

* Revisão de código;
* Testes;
* Integração rápida;
* Feedback do cliente;
* Design simples;

Além de aplicar as práticas descritas acima, Beck intensificou o uso destas práticas, como por exemplo:

* Intensificou a revisão de código através da programação em pares;
* Intensificou o uso de testes com testes automatizados;
* Intensificou o feedback do cliente fazendo o cliente mais presente no desenvolvimento do software.

Com base nestes princípios e práticas utilizados, em 1999, Beck publicou um livro (BECK, 1999) que ajudou a popularizar a metodologia em questão. O livro aborda os principais valores da metodologia:

* Comunicação: é fundamental para conseguir atender as necessidades do cliente, a presença do cliente aumenta o feedback e favorece o desenvolvimento de um produto com mais qualidade.
* Simplicidade: deve ser sempre utilizada para evitar que o tempo da equipe seja gasto com atividades que não precisam ser complexas. Além disso a complexidade aumenta a possibilidade de introdução de erros. Quanto mais simples for o código desenvolvimento mais facilmente um desenvolvedor pode compreender o que foi feito e conseguir contribuir.
* Coragem: é comum existir uma resistência em relação às mudanças, mesmo assim as mudanças são necessárias, tendo em vista que podemos não tomar as melhores decisões durante o desenvolvimento de um sistema. Para garantir que o sistema a ser desenvolvido está tomando o rumo correto deve-se ter coragem para mudar e inovar.
* *Feedback*: deve ser constante entre os participantes de um projeto para compartilhar soluções e problemas, aumentando o nível de entendimento de todos sobre o projeto. Quanto mais cedo um problema for detectado e compartilhado mais cedo ele será resolvido e todos saberão com lidar com ele posteriormente.
* Respeito: deve existir entre os participantes de um projeto para que os outros valores possam ser eficazes. A falta de respeito pode comprometer a comunicação e o feedback, reduzindo a transparência entre os membros da equipe.

### LEAN

Na década de 40 a Toyota ainda era uma pequena empresa, que percebeu a possibilidade de crescimento se conseguisse produzir veículos baratos e com qualidade. Taiichi Ohno e Shigeo Shingo receberam o desafio para reduzir os custos na linha de produção da Toyota, para atingir o objetivo proposto, eles propuseram a eliminação de todo desperdício presente desde a fabricação até a entrega do produto. As mudanças para redução de custo foram base para o início do pensamento Lean. Os princípios elementares para a cultura Lean utilizada pela Toyota eram (OHNO, 1988):

* *Just in Time*: Eliminou a necessidade de estocar itens para a produção, eliminado custos e problemas de armazenamento.
* *Stop the line*: uma parada na linha de produção era forçada se um defeito fosse encontrado, evitando a produção de produtos com erros. A linha de produção só retomava o funcionamento normal quando o defeito fosse eliminado.

Mais tarde esses princípios foram estendidos para outras áreas da empresa.

Os princípios Lean são voltados para a redução de custos e podem ser aplicados no desenvolvimento de qualquer tipo de produto. Alguns princípios podem surgir dependendo do contexto do produto a ser desenvolvido, no desenvolvimento de software podemos destacar os seguintes princípios elementares (POPPENDIECK, 2003):

* Elimine o desperdício: de acordo com o criado do pensamento Lean, desperdício é tudo aquilo que não acrescenta valor ao produto na percepção do cliente. Alguns tipos de desperdícios que devem ser evitados durante o desenvolvimento de software são listados abaixo:
  + Funcionalidades incompletas, poderiam ser evitadas através de um planejamento prévio;
  + Excesso de processos, normalmente não acrescentam valor para o cliente final;
  + Antecipação de funcionalidades, pois aumenta a complexidade do sistema desnecessariamente;
  + Troca de tarefas, pois o número excessivo de trocas de contexto reduz a produtividade;
  + Esperar por informações, ter que esperar em alguma etapa do ciclo de desenvolvimento atrasa todo o ciclo;
  + Troca de pessoas entre equipes causa desperdício em razão da perda de conhecimento;
  + Defeitos causam desperdício pois o tempo gasto costuma ser muito grande quando se comparado com o tempo utilizado na prevenção;
* Amplifique o aprendizado: lições devem ser extraídas dos problemas enfrentados e disseminadas entre os membros da equipe, criando uma base de conhecimento para ajudar na evolução e amadurecimento da equipe.
* Entregue rápido: deve-se entregar pequenas partes funcionais do sistema de forma incremental para aumentar a interação com o cliente e aumentar a frequência com que o cliente dá feedbacks.
* Valorize a equipe: as pessoas não devem ser tratadas apenas como recursos, os membros da equipe devem ter seu trabalho reconhecido para conseguirem motivação e assim contribuir para o desenvolvimento do produto de forma efetiva.
* Adicione segurança: deve-se implementar soluções que deixe a equipe segura de que o produto que está sendo desenvolvido tem qualidade. Priorize a automatização de testes ao invés da detecção e correção de defeitos.

### TEST DRIVEN DEVELOPMENT (TDD)

Segundo (FREEMAN, 2009), o desenvolvimento dirigido por testes (**TDD**)é uma ideia simples que diz: escreva testes para o seu código antes escrever o próprio código. Utilizar TDD muda o papel que os testes tem no desenvolvimento de software, ao invés de utilizarmos os testes pra evitar que os clientes tenham problemas ao utilizar o sistema, o levantamento dos testes faz com que a equipe entenda a funcionalidade antes de implementá-la.

TDD é uma prática muito usada em abordagens ágeis para desenvolvimento de software, é umas das principais práticas da Programação Extrema.

O ciclo básico para quem usa TDD consiste em:

* Escrever um teste;
* Escrever o código para fazer o teste passar;
* Melhorar o código garantindo que os testes ainda estão passando;

Este ciclo deve ser repetido a cada mudança que for feita no sistema. A FIGURA 2.2 representa o ciclo descrito anteriormente.



Figura 2.2 – Ciclo básico para TDD. FONTE (FREEMAN, 2009)

A medida que o sistema vai sendo desenvolvido recebemos feedbacks relativos à implementação e design do sistema.

Quando escrevemos testes temos os seguintes benefícios:

* Ajuda a ter uma ideia clara de qual deve ser o próximo passo para o desenvolvimento, o design tendo a ser melhor;
* O desenvolvimento de componentes desacoplados é favorecido, tendo em vista que componentes desacoplados são mais fáceis de se testar;
* O código de teste ser como uma especificação executável do código desenvolvido;
* Ganha-se um conjunto de testes de regressão.

Quando rodamos os testes que foram escritos anteriormente temos os seguintes benefícios:

* Facilidade na detecção de erros, o erro é detectado de forma instantânea, assim que o conjunto de testes é executado;
* Ajuda a saber quando a funcionalidade está terminada, evitando a criação de partes de código desnecessárias para a funcionalidade em questão.

# METODOLOGIA

Neste capítulo, será apresentado como o sistema foi desenvolvido, as ferramentas escolhidas e mais detalhes sobre o estudo.

Como estudo de caso foi desenvolvido um sistema de catálogo digital que possibilita ao usuário cadastrar seus itens e, em seguida, visualizar os mesmos em um dispositivo móvel, celular ou tablet.

Catálogo digital é um termo muito amplo que pode ser utilizado em diversos contextos, como:

* Catálogo de Roupas;
* Catálogo de Remédios;
* Catálogo de Imóveis;
* Catálogos de Automóveis;

Pode-se ver que existem inúmeras aplicações para este tipo de aplicação, este trabalho aborda a situação em que o catálogo digital é utilizado por um restaurante.

O sistema pode ser dividido em dois blocos básicos, de acordo com a FIGURA 3.1:



Figura 3.1 – Diagrama em blocos do sistema

O sistema Web que está representado na FIGURA 3.1 é responsável pelos seguintes itens:

* Controle de usuários;
* Gerenciamento de categorias dos itens;
* Gerenciamento dos itens;
* Prover interface HTTP para comunicação com o sistema mobile;
* Disponibilizar o aplicativos mobile para download;

O sistema Mobile que está representado na FIGURA 3.1 é responsável pelos seguintes itens:

* Fazer a comunicação com o sistema Web para atualizar as categorias e os itens de um usuário;
* Exibir os itens separados por categoria;

O sistema mobile se comunica com o sistema Web através de requisições HTTP para sincronizar os dados cadastrados no sistema.

## METODOLOGIA UTILIZADA NO DESENVOLVIMENTO

A metodologia proposta consiste em uma união dos princípios do pensamento *Lean* e práticas do *Extreme Programming.*

Alguns dos princípios do pensamento *Lean* que foram utilizados são:

* Elimine o desperdício: Este item tem como foco evitar a geração de defeitos no software, já que defeitos geram um desperdício muito grande quando são corrigidos. Neste caso deve-se favorecer o trabalho preventivo utilizando uma abordagem amparada em testes automatizados.
* Amplifique o aprendizado: Através da participação de grupos de discussão sobre as tecnologias utilizadas é possível economizar muito tempo durante a fase de aprendizado de uma tecnologia.
* Adicione Segurança: Este item está relacionado com a segurança que se deve ter ao realizar modificações durante o desenvolvimento do software, um item imprescindível para aumentar a segurança durante o desenvolvimento é a execução automatizada dos testes desenvolvidos.

Além dos princípios *Lean* algumas práticas de *Extreme Programming* também foram adotadas:

* *Test Driven Development*: Sem dúvida esta prática é a mais valiosa, pois:
  + Aumenta a frequência de *feedback* do código desenvolvido;
  + Ajuda a aumentar a coragem para mudanças, graças ao aumento de *feedback* e consequentemente com o aumento da segurança para os desenvolvedores;
  + Faz com que o código seja criado da forma mais simples possível, evitando a criação de código desnecessário;
* Refatoração: É uma prática bem interessante que ajuda a melhoria do código existente. Refatorar é o ato de mudar código existente, o ideal é executar os testes do código modificado e verificar se o que foi alterado não causou impacto e tudo continua funcionando corretamente. Para se usar refatoração é recomendado o uso de *Test Driven Development* para garantir a segurança durante o desenvolvimento.

## DESENVOLVIMENTO DO SISTEMA WEB

O sistema web pode ser dividido nos seguintes casos de uso, segundo a FIGURA 3.2:



Figura 3.2 – Diagrama de Casos de Uso do Sistema Web

O caso de uso “Gerenciar usuários” comtempla todas as funcionalidades relacionadas aos usuários do sistema, listadas a seguir:

* Criação de uma nova conta de usuário;
* Alteração dos dados de um usuário;
* Cancelamento da conta de um usuário;

Assim que o Administrador do Restaurante se cadastrar para utilizar o sistema, ele poderá cadastrar as categorias em que seus itens serão divididos, por meio do caso de uso “Gerenciar Categorias”, com este caso de uso o usuário poderá:

* Cadastrar novas categorias para seus itens;
* Alterar suas categorias já cadastradas;
* Excluir as categorias;
* Visualizar todas as categorias cadastradas;

Após o cadastro das categorias o usuário poderá cadastrar seus itens que devem ser exibidos no dispositivo móvel. Através do caso de uso “Gerenciar Itens” o usuário poderá:

* Cadastrar novos itens;
* Alterar itens;
* Excluir itens;
* Visualizar os itens já cadastrados;

O último caso de uso relacionado com o sistema Web é o caso de uso “Fazer Download do Aplicativo Mobile”, que consiste basicamente em disponibilizar um executável para ser instalado no dispositivo móvel que será utilizado como catálogo.

### TECNOLOGIAS UTILIZADAS

O Sistema Web foi desenvolvido em Ruby, utilizando Rails como framework MVC. A escolha dessa combinação foi motivada pela facilidade e velocidade que ela dá ao desenvolvimento de aplicações guiadas por testes, já que existem muitos plugins e extensões construídas em Ruby para facilitar o desenvolvimento de aplicações que usam práticas como o *Test Driven Development*.

Uma ferramenta chamada RSpec foi utilizada para contribuir com a legibilidade do código de teste. O RSpec é um plugin para Rails que provê muitos facilitadores para serem usados na criação de testes de unidade e de integração.

Além da legibilidade e facilidade do uso do RSpec, quando ele é utilizado podemos construir um conjunto de testes e executá-lo de maneira automatizada com o uso de uma ferramenta chamada Autotest, que é capaz de detectar mudanças no código e iniciar a execução dos testes relacionados com o código que foi modificado. Além de executar os testes relacionados com o código que foi modificado, periodicamente o Autotest dispara a execução de todos os testes do sistema, garantindo a qualidade do código por completo, não se limitando apenas ao código que foi alterado.

Combinando o RSpec com o Autotest podemos garantir que nossos testes estão bem escritos, legíveis e que estão sendo executados continuamente.

Além plugins destinados para uso em códigos de teste, existem plugins para facilitar o desenvolvimento de funcionalidade rotineiras, como por exemplo:

* Devise: Plugin criado por um brasileiro, chamado José Valim. Esse plugin é destinado ao controle do fluxo de autenticação de usuários nas aplicações. Além do controle do processo de autenticação ele também é capaz de criar várias funcionalidades de maneira simples, entre elas:
  + Enviar e-mails de confirmação durante a criação de um novo usuário no sistema;
  + Habilitar rotinas para travar o acesso de um usuário ao sistema, podendo levar em consideração o número de tentativas sem sucesso.
  + Habilitar rotinas para reenviar a senha para os usuários em caso de perda.
* PaperClip: Plugin criado pela empresa Thoughtbot. O PaperClip é um plugin criado para resolver o problema de upload de imagens. Além de tratar o processo de upload de imagens, ele possui algumas funcionalidades interessantes:
  + Define tamanhos de imagem que devem ser aceitos;
  + Redimensiona automaticamente as imagens, evitando a codificação de algoritmos para isso.
  + Pode ser plugado em serviços de armazenamento de arquivos, como o s3 AWS da Amazon.
* Will Paginate: Plugin especializado na preparação de registros para serem paginados na camada de visualização. Possui interface fluente para facilitar a manipulação dos resultados.

### DESENVOLVIMENTO

Foram levantadas quais funcionalidades seriam desenvolvidas, depois de ter as atividades em mãos elas foram priorizadas de acordo com as dependências existentes entre elas.

As primeiras funcionalidades que foram desenvolvidas foram as relacionadas com o gerenciamento de categorias.

Antes de iniciar o desenvolvimento alguns testes de aceitação foram criados para garantir que o resultado do desenvolvimento fosse realmente o que era necessário. Exemplos:

* A tela de cadastro e edição de categoria devem conter uma caixa de texto para entrar com o nome da categoria.
* Quando uma categoria for cadastrada o usuário deve ser redirecionado para a tela de listagem de categorias.
* A partir da tela de listagem de categorias deve-se ter acesso aos links para edição e exclusão das categorias.
* O modelo de Categoria deve ser validado exigindo a presença do nome da categoria antes de qualquer inserção ou alteração.
* Não devem existir categorias repetidas para um usuário.

Depois de definidos os testes de aceitação deve-se iniciar o desenvolvimento. Como este sistema é baseado em testes, deve-se criar o teste referente a cada parte do sistema para validar gradativamente a evolução do sistema.

Neste caso foram criados os seguintes testes:

* Testes para validar se as telas estavam de acordo com o que foi especificado;
* Testes para validar se o modelo estava com as validações corretas;
* Testes para validar se o controlador estava fazendo seu papel e redirecionando as requisições para os locais corretos;

Podemos ver que conseguimos cobrir todas as camadas com testes, garantindo que todo o fluxo está coberto pelos testes.

De forma análoga foram desenvolvidas as funcionalidades relacionadas aos itens, usuários e download do executável para o dispositivo móvel.

# RESULTADOS (estilo Título 1)

Esta é a parte mais importante do trabalho durante a apresentação. Nesta seção o estudante deve discutir os resultados obtidos. É aqui que devem ser feitas comparações entre os resultados esperados e os previstos pela teoria e/ou a partir de outras experiências conhecidas. Anomalias e discrepâncias devem ser exploradas e explicadas em termos físicos e matemáticos. As explicações devem se basear nos gráficos e nas tabelas apresentadas nas seções anteriores. Por fim, é nessa seção que os resultados serão resumidos e discutidos. Maiores detalhes devem ser colocados em apêndice.

# CONCLUSÃO (estilo Título 1)

Parte final do texto, na qual se apresentam as conclusões correspondentes aos objetivos ou hipóteses. Na conclusão, podem-se incluir também recomendações, sugerindo futuros desenvolvimentos sobre o tema. O(s) autor (es) devem manifestar seu ponto de vista sobre os resultados obtidos e sobre o alcance deles. Não se permite a inclusão de dados novos nessa parte.

REFERÊNCIAS (estilo REFERÊNCIA)

Exemplo no caso de Normas

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 14724:** Informação e documentação — Trabalhos acadêmicos — Apresentação. Rio de Janeiro, 2005. 9.p.

\_\_\_\_\_\_\_\_\_\_\_ **NBR 6023**: informação e documentação: referências: elaboração. Rio de Janeiro, 2002. 24 p.

\_\_\_\_\_\_\_\_\_\_\_ **NBR 10520:** e documentação: citações em documentos. Rio de Janeiro, 2002. 7 p.

\_\_\_\_\_\_\_\_\_\_\_ **NBR 6024**: numeração progressiva das seções de um documento. Rio de Janeiro, 2003. 3 p.

Exemplo no caso de Livro

CERVO, Amado Luis; BERVIAN, Pedro Alcino; SILVA, Roberto. **Metodologia Científica**. 6. ed. São Paulo: Prentice Hall, 2007.

GOMES, L. V. N. **Desenhando:** um panorama dos sistemas gráficos. Santa Maria: Ed.UFSM, 1998.

Exemplo no caso de Capítulo de Livro

WILLIAMS, J. W. Flow measurement. In: ROUSE, H. (org.). **Engineering hydraulic**s. New York: John Wiley & Sons, 1950. p. 229-309.

Exemplo no caso artigo em periódico

ABRAMOF, P. G., MIRANDA, C. R. B., BELOTO, A. F. ; UETA, A.Y., FERREIRA, N. G. An investigation of natural oxidation process on stain-etched nanoporous silicon by micro-Raman spectroscopy. **Applied Surface Science**, v. 253, p. p. 7065-7068, 2007.

Exemplo no caso de monografia, dissertação e tese

MIRANDA, R. B. Filmes de diamante nanocristalino infiltrados em substratos de silício poroso através das técnicas CVD/CVI. 2009. Tese (Doutorado em Engenharia e Tecnologia Espaciais) - Instituto Nacional de Pesquisas Espaciais.

Exemplo no caso de artigo de jornal

MOREIRA, T. Debate sobre software livre chega ao celular. **Valor Econômic**o, São Paulo, 04 out. 2004. p. B4.

Exemplo no caso de trabalho em evento

ABRAMOF, P. G., MOTA A.C. Exame de desempenho dos estudantes: uma porposta de avaliação na educação em engenharia. In: XXXV CONGRESSO BRASILEIRO DE EDUCAÇÃO EM ENGENHARIA, Curitiba. Anais do XXXV Congresso Brasileiro de Educação em Engenharia, 2007.

Exemplo de documento disponível na internet

BRASIL, 2002. Conselho Nacional de Educação, Parecer CNE/CES 1362/2001 – **Diretrizes Curriculares Nacionais dos Cursos de Engenharia**. Despacho do Ministro em 22/02/2002, publicado no DOU de 25 de fevereiro de 2002, Seção 1, p 17. Disponível em: <htlm://portal.mec.gov.br/cne/arquivos/pdf/CES 1/2002> Acesso em 16 junho 2008

GLOSSÁRIO (estilo GLOSSARIO)

Área - conjunto de conteúdos (grupos temáticos comuns) que compõem os diferentes campos do saber.

APÊNDICE A (estilo APÊNDICE)

Elemento opcional. O(s) apêndice(s) são identificados por letras maiúsculas consecutivas e pelos respectivos títulos e contem textos explicativos que não fazem parte do texto da monografia mas que foi elaborado pelo autor,.

A resolução das figuras de qualquer publicação deve ser baixa, de forma a serem lidas em qualquer tipo de rede, sem muita demora. Ver exemplos de figuras em Figura A.1, Figura A.2 e Figura A.3.



Figura A. - Diagrama de funcionamento.

Fonte Adaptada de Tourrilhes (2001)



Figura A.2 - Como apresentar uma figura longa



Figura A.3 - Movimento realocar tarefa.

Fonte: Adaptada de Mauri (2003, p. 17).

A mesma notação deve ser utilizada para tabelas apresentadas nos apêndices.

ANEXO A - ABREVIATURA DOS MESES (estilo ANEXO)

Nos anexos são apresentados textos, mapas tabelas ou figuras que não foram criados pelo autor. São elementos opcionais, que são identificados por meio de letras maiúsculas consecutivas, travessão e pelos respectivos títulos. Vinculados ao trabalho para esclarecimento ou documentação, nem sempre da mesma autoria. A numeração de tabelas e figuras leva em consideração a sequência de ocorrência no texto. Veja exemplo da Tabela 1.

Tabela 1- Abreviaturas

|  |  |  |
| --- | --- | --- |
| **Português** | **Espanhol** | **Italiano** |
| janeiro = jan..  fevereiro = fev.  março = mar..  abril = abr.  maio = maio  junho = jun.  julho = jul.  agosto = ago.  setembro = set.  outubro = out.  novembro = nov.  dezembro = dez.  . | enero = ene  febrero = feb  marzo = mar.  abril = abr.  mayo = mayo  junio = jun.  julio = jul.  agosto = ago. septiembre = sep.  octubre = oct.  noviembre =nov. diciembre = dic. | gennaio = gen.  febbraio = feb.  marzo = mar.  aprile = apr.  maggio = mag. giugno = giu.  luglio = lug.  agosto = ago. settembre = set. ottobre = ott. novembre = nov. dicembre = dic. |

|  |  |  |
| --- | --- | --- |
| **Francês** | **Inglês** | **Alemão** |
| janvier = jan.  février = fév.  mars = mars  avril = avr.  mai = mai  juin = juin  juillet = juil.  août = août  septembre = sept. octobre = oct.  novembre = nov.  décembre = déc. | January = Jan.  February = Feb.  March = Mar.  April = Apr.  May = May  June = June  July = July  August = Aug.  September = Sept.  October = Oct.  November = Nov.  December = Dec. | Januar = Jan.  Februar = Feb.  März = März  April = Apr.  Mai = Mai.  Juni = Juni  Juli = Juli  August = Aug.  September = Sept.  Oktober = Okt.  November = Nov.  Dezember = Dez |