**FELIPE RODRIGUES DO PRADO**
**JOÃO PAULO NAKAJIMA PEREIRA**

# MODULARIZAÇÃO DE SOFTWARES

**UNIVERSIDADE DO VALE DO SAPUCAÍ**
**POUSO ALEGRE**
**2015**

**FELIPE RODRIGUES DO PRADO**
**JOÃO PAULO NAKAJIMA PEREIRA**

# MODULARIZAÇÃO DE SOFTWARES

Pesquisa para desenvolvimento do projeto apresentado à disciplina de TCC 1 do curso de Sistemas de Informação como requisito parcial para obtenção de créditos.

Orientador: Me. Márcio Emílio Cruz Vono de Azevedo.

**UNIVERSIDADE DO VALE DO SAPUCAÍ**
**POUSO ALEGRE**
**2015**

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | *Application Programming Interface* |
| BSD | *Berkeley Software Distribution* |
| CSS | *Cascading Style Sheet* |
| CRUD | *Create, Retrieve, Update, Delete* |
| EJB | *Enterprise JavaBeans* |
| HTML | *HyperText Markup Language* |
| HTTP | *HyperText Transfer Protocol* |
| ICC | Inatel *Competence Center* |
| IDE | *Integrated Development Environment* |
| IoT | *Internet of Things* |
| JDBC | Java *Database Connectivity* |
| JEE | Java *Platform, Enterprise Edition.* |
| JMS | Java *Message Service* |
| JPA | Java *Persistence* API |
| JSON | *JavaScript Object Notation* |
| JVM | Java *Virtual Machine* |
| NTT | *Nippon Telegraph and Telephone* |
| OSGi | *Open Services Gateway initiative* |
| REST | *Representational State Transfer* |
| SENAC | Serviço Nacional de Aprendizagem Comercial |
| SQL | *Structured Query Language* |
| SRA | *Systems Research and Applications Corporation* |
| UNIFEI | Universidade Federal de Itajubá |
| UNIVAS | Universidade do Vale do Sapucaí |
| URI | *Uniform Resource Identifier* |
| URL | *Uniform Resource Locator* |
| XHTML | *Extensible HyperText Markup Language* |
| XML | *Extensible Markup Language* |
| W3C | *World Wide Web Consortium* |

# SUMÁRIO

# 1 QUADRO TEÓRICO

Para que se possa desenvolver qualquer solução, é necessário o uso de algumas ferramentas, teorias e tecnologias. Neste capítulo serão apresentadas algumas delas, que serão utilizadas no desenvolvimento desta trabalho, bem como sua utilidade.

## 1.1 Modularização

O conceito de modularização surgiu como uma solução aos antigos *softwares* monolíticos, pois representavam grandes dificuldades de entendimento e manutenção. Esse conceito, sugere que a complexidade do *software* possa ser dividida em partes menores, resolvendo assim problemas separadamente (USP, 2015).

A modularidade é um fator interno no desenvolvimento de *softwares*, que influencia significativamente em fatores externos, como: qualidade, extensibilidade e reusabilidade do mesmo. O desenvolvimento de um sistema modular é realizado através da composição de partes pequenas para formar partes maiores. Essas partes são chamadas de módulos (BORBA, 2015).

Segundo Knoernschild (2012), para que uma aplicação seja modularizada, seus módulos devem ser instaláveis, gerenciáveis (parar, reiniciar e ser desinstalado sem interromper o restante da aplicação), reutilizáveis (utilizar em outros sistemas), combináveis (combinar com outros módulos), não guardar estado e oferecer uma interface clara.

A granularidade de um *software* é definida a partir de como o sistema é dividido em partes. Sendo assim partes maiores tendem a fazer mais operações do que partes menores. Isso influencia diretamente no uso e reúso de código, como demonstra a Figura 1.

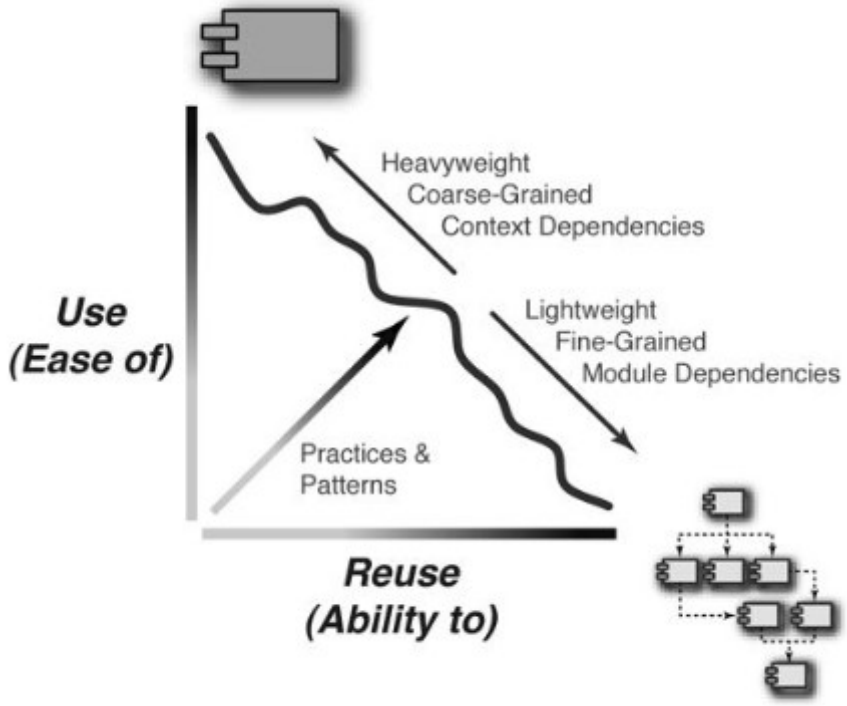


**Figura 1** - Uso e reúso de partes do software. **Fonte**: Java Application Architecture (2012)

Conforme mostra a Figura 1, quanto maior a granularidade dos módulos do sistema, torna-se mais fácil usá-los e mais difícil de reutilizá-los. Do contrário, quanto menor a granularidade, torna-se mais fácil reutilizá-los, e mais difícil usá-los (KNOERNSCHILD, 2012).

Muitos sistemas são desenvolvidos utilizando o modelo de camadas. Neste modelo, também existe a granularidade, pois quanto mais se caminha para baixo da hierarquia de camadas, menor é a granularidade, enquanto as camadas superiores da hierarquia possuem uma granularidade maior. Esse *design* de camadas é demonstrado na Figura 2.

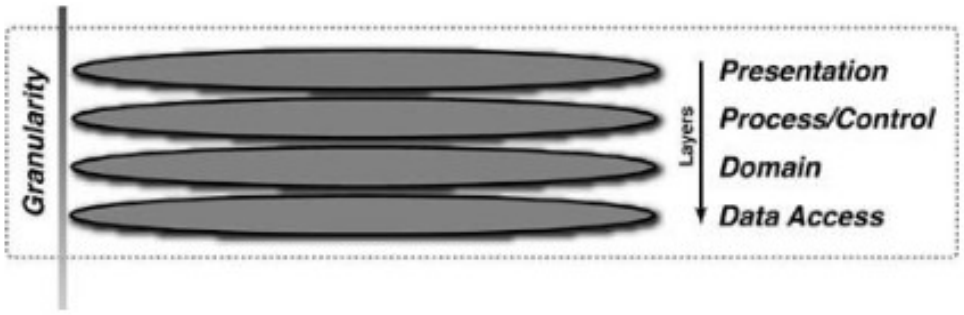


**Figura 2** - Arquitetura de camadas. **Fonte**: Java Application Architecture (2012)

Knoernschild (2012) diz que, diferente da arquitetura de camadas mostrada

anteriormente, a arquitetura de sistemas modulares em Java, através da tecnologia OSGi, propõe um desenvolvimento modular conforme mostra a Figura 3.

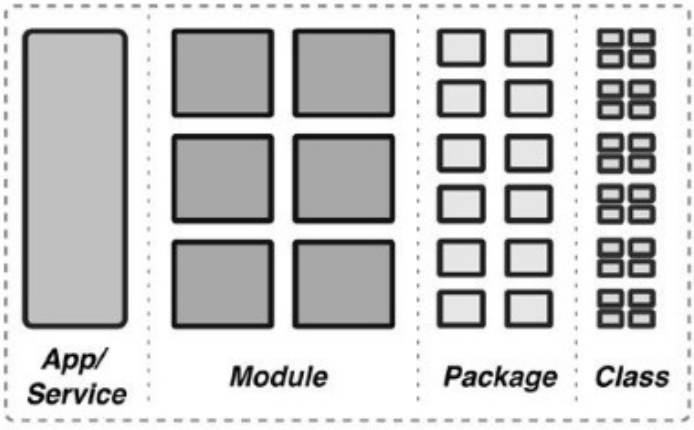


**Figura** 3 - Arquitetura modular OSGi. **Fonte**: Java Application Architecture (2012)

Nesse modelo, o desenvolvedor cria os módulos com classes e pacotes que são disponibilizados para o restante do sistema através de interfaces que podem se tornar serviços para outros módulos.

Um software modular reduz a complexidade, facilita a mudança e resulta em uma implementação mais fácil ao estimular o desenvolvimento paralelo de partes do sistema, resultando numa velocidade maior de desenvolvimento da aplicação (WERLANG; OLIVEIRA, 2006).

Werlang e Oliveira (2006) afirmam que os módulos independentes são mais fáceis de desenvolver, manter e testar. Desta forma, os efeitos secundários provocados por modificações são limitados.

Knoernschild (2012) confirma isso, afirmando que, uma das vantagens da modularidade, é o impacto reduzido que as alterações num *software* modularizado podem causar. A Figura 4, demonstra a representação do impacto em aplicações modulares e não modulares quando acontece alguma alteração no sistema.

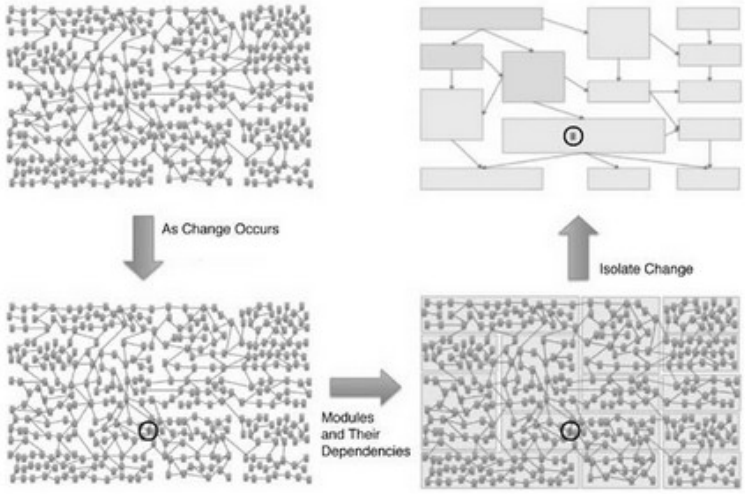


**Figura 4** - Impacto de mudanças. **Fonte**: Java Application Architecture (2012)

Em uma aplicação não modular, a alteração em uma classe pode representar impacto na aplicação toda. Já no sistema modular as alterações representam impacto somente dentro do módulo, já que os mesmos oferecem interfaces bem definidas e suas operações estão encapsuladas, ou seja, não são conhecidas por módulos externos.

O quanto os módulos são independentes é medido levando em consideração dois critérios: coesão e acoplamento. A coesão é uma medida da força funcional e depende do inter-relacionamento entre os elementos que constituem um módulo. Quanto mais forte for este inter-relacionamento, melhor será a coesão. O acoplamento é uma medida da interdependência e depende do volume de elementos que constituem a interface e da forma com que é estabelecida a interface. O acoplamento surge em função do relacionamento existente entre os módulos e é caracterizado pela passagem de controle entre eles (STAA, 2000).

Pressman (1995) estabelece que um módulo coesivo executa uma única tarefa dentro do software. Enquanto ao acoplamento, o nível em um projeto de software deve ser o mais baixo possível, utilizando um número mínimo de informações trocados entre os módulos.

Na modularização deve-se considerar a baixa coesão de forma que o projeto possa ser modificado para conseguir maior independência funcional, já que o acoplamento depende da complexidade de interfaces entre os módulos. A simples conectividade entre os módulos resulta em um software que é mais fácil de entender e menos propenso a propagação de erros pelo sistema (PRESSMAN, 1995).

## 1.2 Especificação OSGi

Grandes aplicações enfrentam um problema conhecido no seu desenvolvimento, a complexidade. Uma maneira de resolver esse problema é quebrar o sistema em partes menores, ou módulos. Devido a sua flexibilidade, a linguagem de programação Java permitiu construir sobre sua plataforma um poderoso sistema de módulos, o OSGi, que oferece suporte à construção de sistemas modulares (FERNANDES, 2009).

A especificação OSGi é mantida pela OSGi Alliance, um consórcio mundial formado por empresas tecnológicas que juntas criam especificações abertas que permitem o desenvolvimento modular de *softwares* através da tecnologia Java. OSGi é um conjunto de especificações, que segue um modelo de desenvolvimento em que as aplicações são dinamicamente formadas por componentes distintos e reutilizáveis (OSGI ALLIANCE, 2015).

A especificação teve início em março de 1999 pela própria OSGi Alliance. Seus principais desafios na época não era desenvolver uma solução para a execução de diferentes versões de um mesmo projeto na mesma aplicação, mas sim de elaborar uma maneira que diferentes componentes que não se conhecem possam ser agrupados dinamicamente sob o mesmo projeto (OSGI ALLIANCE, 2015).

De acordo com OSGi Alliance (2015), essa tecnologia tem como benefício a redução da complexidade do desenvolvimento de modo em geral, como por exemplo, aumento da reutilização de módulos, incremento da facilidade de codificação e teste, gerenciamento total dos módulos, sem a necessidade de reiniciar a aplicação, aumento do gerenciamento da implantação e detecção antecipada de *bugs*.

Os *bundles* – como os módulos são chamados no contexto da OSGi – consomem e/ou disponibilizam serviços. Eles estão organizados de uma maneira que formam a tríade consumidor, fornecedor e registro de serviços, na qual pode-se consumir serviços ou disponibilizá-los. Para a utilização de um novo serviço, deve-se registrá-lo em um catálogo, onde este teria visibilidade por parte de *bundles* externos que consumiriam esses serviços disponibilizados (OSGI ALLIANCE, 2015).

A OSGi Alliance (2015), define que o *framework* OSGi utiliza uma arquitetura de camadas em sua implementação, conforme demonstrado na Figura 5.

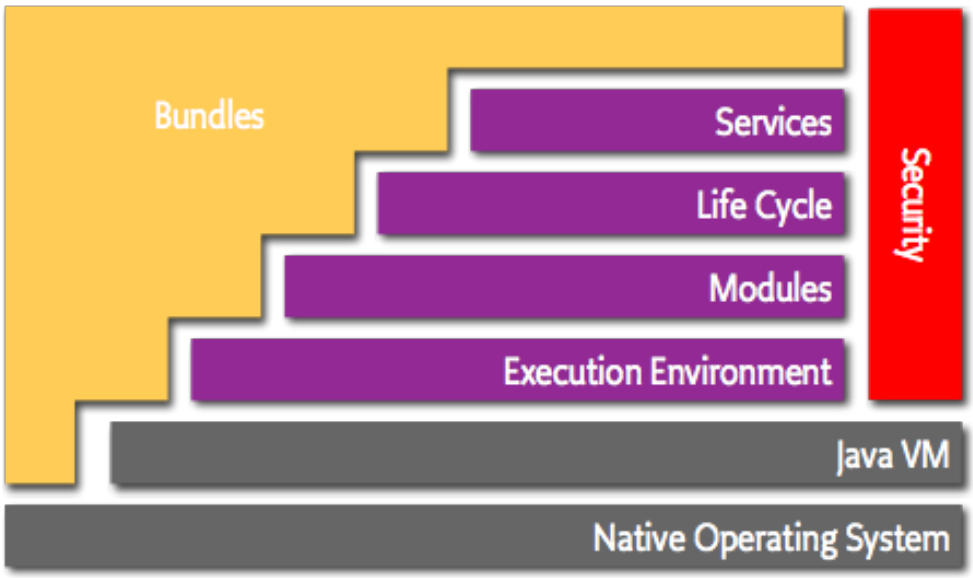


**Figura 5** - Estrutura de camadas. **Fonte**: OSGi Alliance (2015)

As camadas utilizadas pelo *framework* OSGi podem ser assim explicadas:

- **Bundles:** são os componentes (módulos) criados pelos desenvolvedores de softwares;
- **Services:** camada responsável por conectar os *bundles* de forma dinâmica oferecendo um modelo *publish-find-bind*. Esta camada permite registrar e obter os serviços dos módulos;
- **Life Cycle:** camada que provê uma API para instalar, iniciar, parar, atualizar, e desinstalar os *bundles*;
- **Modules:** camada que administra a importação e exportação de códigos de um *bundle*.
- **Execution Environment:** camada que define quais os métodos e classes estão disponíveis em uma plataforma específica;
- **Security:** camada responsável por gerenciar aspectos de segurança do *framework*.

A especificação OSGi define uma camada chamada *Life Cycle* que gerencia o ciclo de vida e provê uma API[1] que permite o desenvolvedor instalar, desinstalar, iniciar, parar e atualizar os *bundles* (BOSSCHAERT, 2012).

Bartlett (2009) afirma que um *bundle* pode passar por vários estados em seu ciclo de vida. Na Figura 6, é demonstrado cada um desses estados e suas funções.

1 API – Abreviação para *Application Programming Interface*.

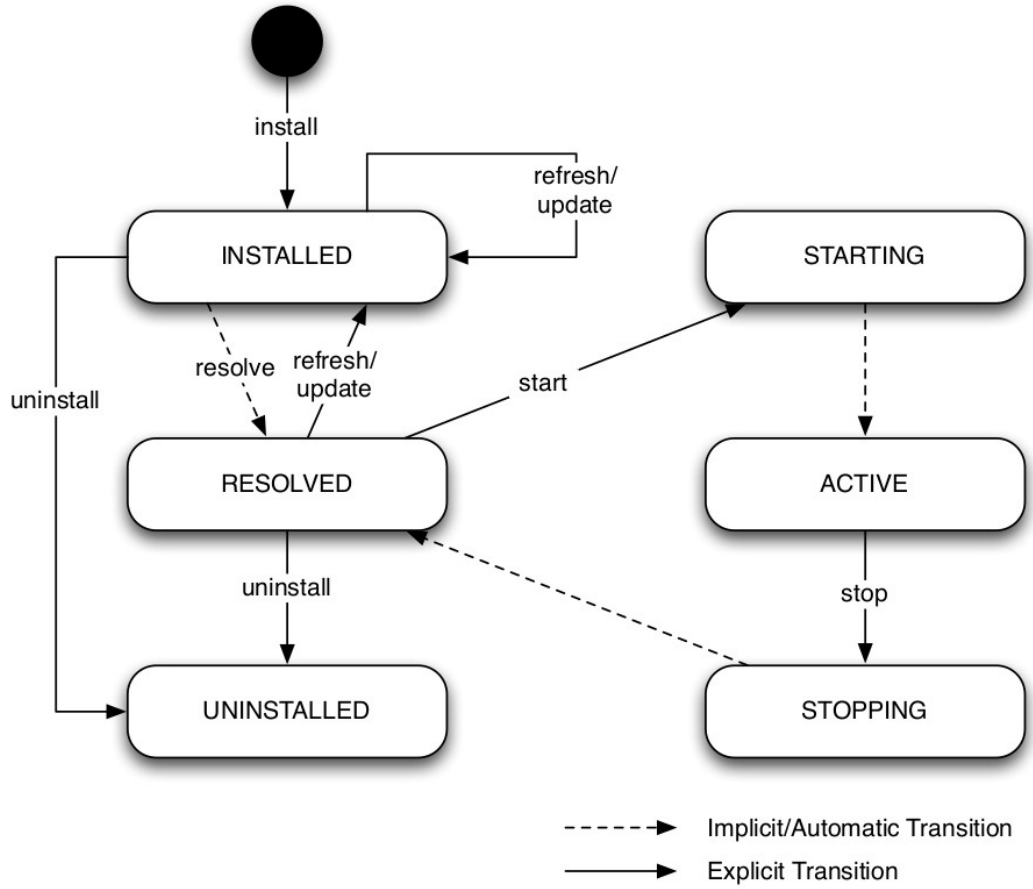


**Figura 6** - Estados dos *bundles*. **Fonte**: OSGi In Practice (2009)

Esses estados e funções podem ser assim explicados:

- **Installed:** o *bundle* é instalado com êxito;
- **Resolved:** avalia se o *bundle* está pronto para ser iniciado ou parado, validando informações dos *bundles*, seus pacotes e versões, e ainda a versão do Java necessária para executar o *bundle*;
- **Starting:** estado apenas de transição, neste estado o método BundleActivator.start() do *bundle* é invocado para que o mesmo seja iniciado. Por ser um estado de transição, nenhum *bundle* fica parado nesse estado;
- **Active:** nesse estado o *bundle* está validado e ativo, somente esperando alguma requisição para ser utilizado;
- **Stopping:** também um estado de transição, parecido com o estado *Starting*, porém nesse estado o método BundleActivator.stop() é chamado para parar o *bundle*, com

isso o mesmo é transferido para o estado *Resolved* novamente;

- **Uninstalled:** quando ocorre a desinstalação do *bundle*, não se pode transitá-lo para nenhum outro estado e este não é mais representado no *framework*. Se o *bundle* for reinstalado, ele assume o papel de um novo *bundle*.

Segundo Fernandes (2009), após a instalação do *bundle*, o mesmo fica armazenado em um meio persistente do *framework* até ser desinstalado explicitamente.

Um aspecto importante é que um *bundle* pode ser executado em qualquer implementação OSGi. Ou seja, um *bundle* desenvolvido e testado em uma implementação pode ser executado em qualquer outra implementação. (LUCENA, 2010).

O *framework* é o centro da especificação da plataforma OSGi, definindo um modelo para o gerenciamento do ciclo de vida da aplicação, registro dos serviços e ambiente de execução (FERNANDES, 2009).

Segundo Bartlett (2009), a OSGi Alliance apenas define uma especificação do *framework*, por isso existem várias implementações do mesmo nos dias atuais, mas as quatro a seguir merecem destaque por serem as principais e terem seu código livre.

- **Equinox:** é o *framework* mais usado atualmente, sua grande popularidade provém de fazer parte do gerenciador de *plug-ins* do Eclipse. Pode ser encontrado em muitos outros *softwares*, como os da IBM Lotus Notes e Websphere Application Server. Encontra-se sob a licença Eclipse Public License (EPL) e implementa a versão 4.1 das especificações OSGi1.
- **Knopflerfish:** é uma implementação bem madura e popular da versão 4.1 da especificação. É desenvolvido e mantido pela empresa Sueca Makewave AB, a qual oferece seu produto tanto em uma versão gratuita sob a licença BSD quanto uma versão comercial com suporte oferecido pela empresa.
- **Felix:** é a implementação mantida pela Apache Software Foundation. Implementa a versão 5 da especificação OSGi e possui foco na compactação e facilidade de incorporação do *bundle* na aplicação. Está sob a licença Apache 2.0.
- **Concierge:** é uma implementação bem compacta e altamente otimizada da especificação versão 3. É mais usado para plataformas com recursos limitados, como por exemplo, aplicações móveis e embarcadas. Se encontra sob a licença BSD[2].

2 BSD – Abreviação para *Berkeley Software Distribution*.

O OSGi especifica um conceito de versionamento de componentes, em que os módulos podem trabalhar com versões diferentes, seguindo uma consistente estrutura de numeração, utilizando três segmentos de números e um segmento alfanumérico, são eles, *major*, *minor*, *micro* e *qualifier*, respectivamente. Exemplo, "1.2.3.beta_1". Esses segmentos também podem ser omitidos, formando por exemplo a versão "1.2". Isso se torna necessário para resolver o problema de incompatibilidade de versões entre módulos (FERNANDES, 2009).

A OSGi Alliance explora também a tecnologia OSGi junto a *Internet of Things*, significado da sigla IoT, que, em português significa internet das coisas. A tecnologia OSGi demostra uma arquitetura modular ágil, incluindo um mecanismo de ciclo de vida dinâmico, que permite a criação de produtos e serviços em diversos setores, como demonstra a Figura 7. Também assegura a gestão remota e interoperabilidade de aplicações e serviços em uma ampla variedade de dispositivos, isso devido à facilidade na componentização de softwares modulares (OSGI ALLIANCE, 2015)


**Figura 7** - OSGi atuando em diferentes setores. **Fonte**: OSGi Alliance (2015)

De acordo com Fernandes (2009), os *Working Groups*, criados pelo conselho administrativo da OSGi Alliance, são subdivisões responsáveis pelo desenvolvimento técnico para cada área de atuação. Estão entre os *Working Groups* a *Core Platform Expert Group* (CPEG), *Mobile Expert Group* (MEG), *Vehicle Expert Group* (VEG), *Enterprise Expert Group* (EEG) e *Residential Expert Group* (REG). Dessa forma, à medida que é necessário explorar e desenvolver algo novo, são criados novas *Working Groups*, para analisar seus requisitos de mercado, como é o caso da nova *Working Group* da subdivisão de IoT.

Segundo Gama (2008) "existe uma curva de aprendizado que não vale a pena e nem faz sentido se você está desenvolvendo aplicações que não precisam das vantagens do OSGi".

## 1.3 Java

A linguagem de programação Java é composta por coleções de classes existentes nas bibliotecas de classe Java, conhecidas como Java APIs. Estas classes são partes que consistem uma aplicação Java, cada classe é composta por métodos, que são responsáveis por realizar tarefas e retornar informações. É possível criar classes para formar uma aplicação Java e também utilizar as coleções de classes da Java API. Portanto, para se desenvolver uma aplicação utilizando essa linguagem, é preciso entender como criar classes próprias que irão compor a aplicação e como trabalhar utilizando as classes da Java API (DEITEL, 2010).

Java foi lançada em 1996 pela Sun Microsystems[3] com a finalidade de desenvolver aplicativos para diferentes plataformas (Windows – Linux – *Web* – *Mobile*). Qualquer dispositivo que possua a JVM[4] é capaz de executar um *software* desenvolvido em Java (CLARO e SOBRAL, 2008).

Java oferece uma plataforma para o desenvolvimento de sistemas de médio a grande porte. Uma das muitas vantagens de utilizá-la é a grande quantidade de bibliotecas gratuitas que podem ser utilizadas em diversos tipos de projetos. Como cada linguagem é apropriada para uma determinada área, a utilização de Java é melhor para o desenvolvimento de aplicações que tenham tendência a crescer (CAELUM, 2015).

Para desenvolver aplicações em Java é necessário instalar um Kit de desenvolvimento,

3 Fabricante de computadores, semicondutores e softwares adquirida pela Oracle Corporation em 2009.
4 JVM – Abreviação para Java Virtual Machine.

o Java *Development Kit* – JDK, o qual pode ser obtido no próprio site da Oracle – empresa mantenedora da plataforma. Ele é composto de compilador, máquina virtual, bibliotecas e utilitários (CAELUM, 2015).

Essa linguagem demonstra ser de imprescindível utilidade para nosso projeto, pois além das vantagens citadas anteriormente, a especificação OSGi é desenvolvida sobre sua plataforma.

## 1.4 Felix

Felix ou Apache Felix é uma implementação da especificação OSGi, que implementa sua versão 5. Atualmente é mantido pela Apache Software Foundation, sob licença Apache 2.0 (FERNANDES, 2009).

Apache (2015) afirma que, o *framework* Felix permite que os códigos desenvolvidos possam funcionar em qualquer outro *container* OSGi, permitindo a troca de implementações sem problemas de compatibilidade. Sua comunidade está sempre se esforçando para que não haja esse problema.

Utilizar uma implementação de fácil utilização no desenvolvimento de uma aplicação, que oferece desde a mais simples base de *software* até a possibilidade de integração total com outros *softwares,* como os servidores de aplicação GlassFish e Jonas, nos auxilia no desenvolvimento do projeto, pois permite, além das facilidades para fazer a aplicação, a opção de migração para outra implementação OSGi (BARTLETT, 2009).

## 1.5 GlassFish

Em 2005, a Sun Microsystems cria o projeto GlassFish com o objetivo principal de criar um servidor de aplicações JEE. Atualmente se encontra na versão 4.1, e é mantido pela Oracle, que adquiriu a Sun em 2010. (GONCALVES, 2010).

Segundo DevMedia (2011), o servidor de aplicações GlassFish possui uma versão

paga e outra de código *Open Source*. A versão livre fornece um servidor de aplicações com o código fonte disponibilizado para *download* no site do GlassFish. Possui integração com diversas IDEs, como NetBeans, Eclipse ou IntelliJ.

O GlassFish é um servidor de aplicação de fácil utilização, além de ser leve e modular. Também fornece armazenamento em *cluster* com alta disponibilidade, console de administração baseado na web, e REST APIs (ORACLE, 2011).

Devido as vantagens apresentadas e a grande integração com o *framework* Felix, a utilização do GlassFish será útil no desenvolvimento de uma aplicação modular para *web.*

## 1.6 RESTful Web Services

Em 2000, o cientista Roy Fielding, apresentou em sua tese de doutorado uma nova forma de integrar sistemas distribuídos, esse novo modo se chamava REST[5] (SANTOS, 2009).

De acordo com Santos (2009), REST (Transferência de Estado Representativo) é um estilo de arquitetura de software para sistemas hipermídia distribuídos, onde utilizamos um navegador web para acessar recursos, mediante a digitação de uma URL[6].

Foi concebido baseando-se em HTTP[7] e devido a isso sua arquitetura é baseada em cliente-servidor e seus serviços não possuem estado (*stateless*). Empregar o uso do protocolo HTTP e de URIs[8] torna as aplicações mais simples, leves e com alta performance (SANTOS, 2009).

Segundo Santos (2009), os recursos dos serviços são obtidos de um servidor web após enviar a requisição, porém o endereço dessa requisição podem ser denominados:

- URL representa o caminho percorrido, separando em diretórios.
- URI são os identificadores de acesso a recursos específicos que utilizando junto à URL aponta para o recurso especifico.

5 REST – Abreviação para *Representational State Transfer.*
6 URL – Abreviação para *Uniform Resource Locator.*
7 HTTP – Abreviação para *HyperText Transfer Protocol.*
8 URI – Abreviação para *Uniform Resource Identifier.*

Uma vez que o recurso é identificado, é necessário definir a ação que irá se tomar utilizando este recurso, em REST estas operações são GET, PUT, POST e DELETE. Essas operações são mapeadas com as operações CRUD (Create, Retrieve, Update, Delete) de um banco de dados.

Conforme Saudate (2013), os recursos e operações são obtidos e salvos através de um tipo de dado, este pode ser XML[9] ou JSON[10], cada um com suas vantagens e características próprias, onde:

- XML é uma linguagem de marcação extensível. Desta forma, conseguimos expressar grande parte das informações utilizando este formato. Entre as características principais dessa linguagem de marcação temos:
  - Tem um e apenas um elemento raiz, porém este é flexível o bastante para ter qualquer nome;
  - Possuem seções específicas para fornecimento de instruções de processamento, prólogo e epílogo, onde estas se localizam antes ou após os dados, respectivamente;
  - Estruturas mais básicas em um XML são as *tags* e os atributos, com isso os dados transportados pela sua aplicação podem estar presentes tanto na forma de atributos como de conteúdo das *tags*.

- JSON é uma linguagem de marcação criada por Douglas Crockford. Tem por principal vantagem o tamanho reduzido em relação ao XML, e devido a isso acaba sendo usado quando o cenário apresenta o consumo de banda como um recurso crítico. Suas principais características são:

  - Um objeto contém zero ou mais membros;
  - Um membro contém zero ou mais pares e zero ou mais membros;
  - Um par contém uma chave e um valor;
  - Um membro também pode ser um *array*.

De acordo com Jersey (2015), a especificação JAX-RS define e regulamenta as regras

9 XML – Abreviação para *eXtensible Markup Language.*
10 JSON – Abreviação para JavaScript *Object Notation.*

do JSR 311, porém para se utilizar das vantagens oferecidas, é necessário usar uma implementação. Com a finalidade de simplificar o desenvolvimento de serviços Web, o *framework* Jersey é uma implementação *opensource* dessa especificação.

Utilizar Jersey se torna de grande vantagem pois ele é muito mais do que a implementação de referência da especificação, este fornece sua API que estende o JAX-RS, oferecendo utilitários adicionais e simplificando o serviço RESTful e o desenvolvimento do cliente da aplicação (JERSEY, 2015).

## 1.7 Maven

O Maven é um projeto de código livre, mantido pela Apache Software Foundation, criado para gerenciar o processo de criação do projeto Jakarta Turbine. Com o lançamento das novas versões, ele passou de uma simples ferramenta de construção para uma ferramenta complexa de gestão de construção de *software* (FILHO, 2008).

Segundo Apache (2015), o Maven possui como principal finalidade o entendimento do estado de desenvolvimento de um *software* em um curto espaço de tempo. Para alcançar este objetivo, é necessário atingir alguns outros, como:

- Facilitar o processo de desenvolvimento;
- Prover um sistema de desenvolvimento uniforme;
- Disponibilizar informações importantes sobre o projeto;
- Prover orientação para atingir melhores práticas de desenvolvimento.

Santos (2008) apresenta um conjunto de modelos conceituais que explicam o funcionamento desta ferramenta:

- **Project object model (POM):** é o componente principal de configurações para o funcionamento, onde o desenvolvedor informa em um arquivo pom.xml as dependências de que necessita, além de outras configurações;
- **Dependency management model:** a gestão de dependência é uma importante fase do

processo de construção do *software*, quando se utiliza projetos e dependências de terceiros, é comum ter muito trabalho e problemas na obtenção e gerenciamento das dependências de nossas dependências. Nesse quesito, o Maven é uma das melhores ferramentas para a construção de projetos complexos, pela sua robustez na gestão dessas dependências;

- **Ciclo de vida de construção e fases:** associado ao POM, existe a noção de ciclo de vida de construção e fases. Onde é criado uma conexão entre os modelos conceituais e os modelos físicos;
- **Plug-ins:** estendem as funcionalidades do Maven.

No desenvolvimento modular é necessário que seja realizado as dependências existentes entre cada módulo do sistema. Controlar tais dependências manualmente demandará um grande esforço e tempo, dessa forma, utilizar o Maven para gerenciar a obter as dependências dos módulos, se torna de suma importância, pois auxilia na redução do tempo de desenvolvimento, implantação e manutenção do *software.*

## 1.8 PostgreSQL

Segundo Neto (2003 apud Souza et al., 2012, p. 2), o PostgreSQL é um Sistema Gerenciador de Banco de Dados Relacional (SGBDR) que está baseado nos padrões SQL[11] ANSI-92, 96 e 99. Possui alta *performance,* fácil administração e utilização em projetos pelos *Database Administrators* (DBAs) e desenvolvedores de *softwares*.

O PostgreSQL teve origem em um projeto chamado de POSTGRES na Universidade Berkeley, na Califórnia (EUA), em 1986. Sua equipe fundadora foi orientada pelo professor Michael Stonebraker com o apoio de diversos órgãos, entre eles o Army Research Office (ARO) e o National Science Foundation (NSF). Atualmente, o SGBD encontra-se em sua versão 9.4 estável, contendo todas as principais características que um SGBD pode disponibilizar (MILANI, 2008).

Apesar da pouca popularidade, o PostgreSQL é um gerenciador de banco de dados veloz, robusto e se encontra na lista dos que mais possuem recursos, além de ser o pioneiro na introdução de vários conceitos objeto-relacionais (STERN, 2003).

11 SQL – Abreviação para *Structured Query Language*.

Seu código é livre e há um grupo responsável pela sua validação. Grandes empresas como a Fujitsu, NTT[12] Group, Skype, Hub.org, Red Hat e SRA[13] são financiadoras do PostgreSQL que, além disso, recebe doações. Ele é utilizado por multinacionais, órgãos governamentais e universidades. Recebeu vários prêmios como melhor sistema de banco de dados *Open Source* (SOUZA et al., 2012).

## 1.9 HyperText Markup Language (HTML)

*HyperText Markup Language*, é o significado da sigla HTML, que, em português, significa linguagem para marcação de hipertexto. Ela foi criada por Tim Berners-Lee na década de noventa tornando-se um padrão internacional. De modo geral, o hipertexto é todo o conteúdo de um documento para *web*, com a característica de se interligar a outros documentos da web através de links presentes nele próprio (SILVA, 2011).

Conforme Mozilla Developer Network (2014), a HTML é uma linguagem de marcação que estrutura o conteúdo de um documento da web. O conteúdo visto ao acessar uma página através do navegador é estruturado e descrito utilizando a HTML, tornando-se a principal linguagem para conteúdo da web mantida pelo *World Wide Web Consortium* ou resumidamente W3C (W3C, 2015).

O W3C é uma comunidade internacional liderada pelo criador da *web* Tim Berners-Lee, é formada por organizações, profissionais e público em geral com o objetivo de conduzir a *web* ao seu potencial máximo, através do desenvolvimento de padrões e especificações (W3C, 2015).

Mesmo sendo uma linguagem destinada à criação de documentos, a HTML não tem como objetivo definir estilos de formatação, como por exemplo, nomes e tamanhos de fontes, margens, espaçamentos e efeitos visuais. Também não possibilita adicionar funcionalidades de interatividade avançada à página. A linguagem HTML destina-se somente a definir a estrutura dos documentos *web*, fundamentando dessa maneira os princípios básicos do desenvolvimento seguindo os padrões *web* (SILVA, 2011).

12 NTT – Abreviação para *Nippon Telegraph and Telephone*.
13 SRA – Abreviação para *Systems Research and Applications Corporation*.

## 1.10 JavaScript

JavaScript é uma linguagem de programação criada pela Netscape em parceria com a Sun Microsystems. Sua primeira versão, definida como JavaScript 1.0, foi lançada em 1995 e implementada em março de 1996 no navegador Netscape Navigator 2.0 (SILVA, 2010).

Segundo Silva (2010), o JavaScript tem como finalidade fornecer funcionalidades para adicionar interatividades avançadas a uma página web. É desenvolvido para ser executada no lado do cliente, ou seja, é interpretada pelo navegador do usuário. Os navegadores possuem funcionalidades integradas para realizar a interpretação e o funcionamento da linguagem JavaScript.

Conforme Mozilla Developer Network (2015), JavaScript é baseado na linguagem de programação ECMAScript, o qual é padronizado pela Ecma International na especificação ECMA-262 e ECMA-402.

Entre suas características está ser uma linguagem leve para a execução, interpretada, assíncrona e baseada em objetos com funções de primeira classe. Funções de primeira classe são funções que podem ser passadas como argumentos, retornadas de outras funções, atribuídas a variáveis ou armazenadas em estrutura de dados. Além de ser executado nos navegadores, o JavaScript é utilizado em outros ambientes, como por exemplo, node.js ou Apache CouchDB (MOZILLA DEVELOPER NETWORK, 2015).

De acordo com Caelum (2015), o JavaScript é responsável por aplicar qualquer tipo de funcionalidade dinâmica em páginas *web*. É uma linguagem poderosa, que possibilita ótimos resultados. Ferramentas como o Gmail, Google Maps e Google Docs são exemplos de aplicações *web* desenvolvidas utilizando JavaScript.

## 1.11 Cascading Style Sheet (CSS)

*Cascading Style Sheet*, é o significado da sigla CSS, que, traduzido para o português, significa folhas de estilo em cascata. Tem a finalidade de definir estilos de apresentação para um documento HTML (SILVA, 2012).

De acordo com a W3C (2015), "*Cascading Style Sheets* (CSS) *is a simple mechanism for adding style* (*e.g.*, *fonts*, *colors*, *spacing*) *to Web documents*"[14].

Tim Berners-Lee considerava que os navegadores eram responsáveis pela estilização de uma página web, até que em setembro de 1994, surge como proposta a implementação do CSS. Em dezembro de 1996, foi lançada oficialmente pela W3C as CSS1. O W3C utiliza o termo nível em vez de versão, tendo dessa maneira CSS nível 1, CSS nível 2, CSS nível 2.1 e atualmente CSS nível 3, conhecida também como CSS3 (SILVA, 2012).

De acordo com Mozilla Developer Network (2015), a CSS descreve a apresentação de documentos HTML, XML ou XHTML[15]. Ela é padronizada pelas especificações da W3C e já contém seus primeiros rascunhos do nível CSS4.

Enquanto o HTML é uma linguagem destinada para estruturar e marcar o conteúdo de documentos na web, o CSS fica responsável por definir cores, fontes, espaçamentos e outros atributos relacionados a apresentação visual da página usando a marcação fornecida pelo HTML como fundamento para aplicação da estilização. Essa separação da marcação e estrutura de um documento do seu estilo de apresentação torna o uso do CSS uma grande vantagem no desenvolvimento de documentos para a web (MAUJOR, 2015).

## 1.12 AngularJS

AngularJS é um *framework opensource* mantido pela Google. Seu objetivo é estruturar o desenvolvimento *front-end* utilizando o modelo de arquitetura *model-view-controller* (MVC). O *framework* associa os elementos do documento HTML com objetos JavaScript, dessa maneira tendo facilitando a manipulação dos objetos (OLIVEIRA, 2013).

De acordo com Schimtz e Lira (2014), AngularJS é um *framework* muito poderoso que possui particularidades e funcionalidades que tornam o desenvolvimento web mais fácil e empolgante. Funciona como uma extensão para o documento HTML, que permite adicionar parâmetros e a interação de forma dinâmica com seus elementos, com isso adicionamos funcionalidades extras sem a necessidade de programar com JavaScript.

14 *Cascading Style Sheet:* folha de estilo em cascata (CSS) é um mecanismo simples para adicionar estilos (por exemplo: fontes, cores, espaçamentos) para documentos web (tradução nossa).
15 XHTML – Abreviação para *Extensible HyperText Markup Language*.

## 1.13 Bootstrap

Bootsrapt é um *framework* criado pelo Twitter que oferece uma base de estilos pronta para a criação de páginas web. É uma ferramenta *opensource*, com um nível alto de maturidade e importância no mercado (MARIE, 2015).

Segundo Costa (2014), o Bootstrap facilita o desenvolvimento de sites responsivos, ou seja, com a utilização deste *framework* a página web pode ser aberta em dispositivos de diferentes tamanhos sem afetar o estilo do seu conteúdo. Possui também vários componentes que podem ser utilizados sem muito trabalho, como por exemplo, Tooltip, Menu-Dropdown, Modal, Carousel, Navbar, entre outros.

Recursos como, *reset* CSS, estilo visual base para maioria das *tags*, ícones, *grids*, componentes, *plug-ins* JavaScript, e responsividade são oferecidos pelo *framework* com o objetivo de um início rápido do projeto sem perda de tempo. (MARIE, 2015).

# REFERÊNCIAS


APACHE. **OSGi Frequently Asked Questions**. 2015. Disponível em http://felix.apache.org/documentation/tutorials-examples-and-presentations/apache-felix-osgi-faq.html. Acesso em 16 de junho, 2015.

APACHE. **What is maven**. 2015. Disponível em http://maven.apache.org/what-is-maven.html. Acesso em 16 de junho, 2015.

APPOLINÁRIO, Fábio. **Dicionário de metodologia:** um guia para a produção do conhecimento científico. São Paulo: Atlas, 2004.

BORBA, Paulo. **Aspectos de Modularização**. 2015. Disponível em http://www.di.ufpe.br/~java/graduacao961/aulas/aula4/aula4.html. Acesso em 21 de junho, 2015.

BARTLETT, Neil. **OSGi In Practice**. 2009.

BOSSCHAERT, David. **OSGi in Depth**. Shelter Island: Manning Publications Co, 2012

CAELUM. **Apostila Desenvolvimento Web com HTML, CSS e JavaScript**. 2015. Disponível em https://www.caelum.com.br/apostila-html-css-javascript/javascript-einteratividade-na-web/. Acesso em 08 de março, 2015.

CAELUM. **Apostila Java e Orientação a Objetos**. 2015. Disponível em http://www.caelum.com.br/apostila-java-orientacao-objetos/. Acesso em 08 de março, 2015.

CLARO, Daniela Barreiro; SOBRAL, João Bosco Mangueira. **Programação em Java**. Santa Catarina: Copyleft Pearson Education, 2008.

COSTA, Gabriel. **O que é bootstrap?** 2014. Disponível em http://www.tutorialwebdesign.com.br/o-que-e-bootstrap/. Acesso em 09 de agosto, 2015.

DEITEL, Harvey Matt; DEITEL, Paul John. **Java How to Program**. 8. ed. Edson Furmankiewicz. São Paulo: Pearson Prentice Hall, 2010.

DEVMEDIA. **Introdução ao Spring Framework**. 2015. Disponível em http://www.devmedia.com.br/introducao-ao-spring-framework/26212. Acesso em 10 de março, 2015.

DEVMEDIA. **Novidades do GlassFish 3.1**. 2011. Disponível em: http://www.devmedia.com.br/novidades-do-glassfish-3-1-artigo-java-magazine-91/21124. Acesso em 19 de junho, 2015.

DURHAM, Alan; JOHNSON, Ralph. A Framework for Run-time Systems and its Visual Programming Language. In: **OBJECT-ORIENTED PROGRAMMING SYSTEMS,**

**LANGUAGES, AND APPLICATIONS**. San Jose, CA, 1996, p. 20-25.

FERNANDES, Leonardo. **OSGi e os benefícios de uma Arquitetura Modular**. 37.ed. 2009. p. 27-35.

GAMA, Kiev. **Uma visão geral sobre a plataforma OSGi**. 2008. Disponível em https://kievgama.wordpress.com/2008/11/24/um-pouco-de-osgi-em-portugues/. Acesso em 09 de março, 2015.

GIL, Antônio Carlos. **Como elaborar projetos de pesquisa**. São Paulo: Atlas, 2002.

GONÇALVES, Antonio. **Beginning Java EE 6 Platform with GlassFish 3**. Nova York: Springer Science+Businnes Media, LCC. 2010.

JERSEY. **Jersey**: RESTful Web Services in Java. 2015. Disponível em https://jersey.java.net/. Acesso em 10 de agosto, 2015.

KIOSKEA. **Condução de reunião**. 2014. Disponível em http://pt.kioskea.net/contents/579-conducao-de-reuniao. Acesso em 16 de abril, 2015.

KNOERNSCHILD, Kirk. **Java Application Architecture:** Modularity Patterns with Examples Using OSGi. Crawfordsville: Pearson Education, 2012.

LIRA, Douglas; SCHMITZ, Daniel. **AngularJS na prática**. São Paulo: Leanpub, 2014.

LUCENA, Fábio Nogueira de. **Introdução ao Equinox**. 2010. Disponível em https://code.google.com/p/exemplos/wiki/equinox. Acesso em 09 de março, 2015.

MADEIRA, Marcelo. **OSGi – Modularizando sua aplicação**. 2009. Disponível em https://celodemelo.wordpress.com/2009/11/12/osgi-modularizando-sua-aplicacao/. Acesso em 09 de março, 2015.

MALCHER, Marcelo Andrade da Gama. **OSGi Distribuído:** deployment local e execução remota. Monografia de Seminários de Sistemas Distribuídos. Pontifícia Universidade Católica do Rio de Janeiro, 2008.

MARIE, Victor. **Bootsrapt e formulários HTML5.** 2015. Disponível em http://www.caelum.com.br/apostila-html-css-javascript/bootstrap-e-formularios-html5/. Acesso em 09 de agosto, 2015.

MAUJOR. **Site sobre CSS e Padrões Web:** Por que CSS?. 2015. Disponível em http://www.maujor.com/index.php. Acesso em 08 de março, 2015.

MAYWORM, Marcelo. **OSGi Distribuída:** Uma Visão Geral. 42.ed. 2010. p. 60-67.

MILANI, André. PostgreSQL: **Guia do Programador**. São Paulo: Novatec Editora, 2008.

Mozilla Developer Network. **CSS**. 2015. Disponível em: https://developer.mozilla.org/pt-

BR/docs/Web/CSS. Acesso em 08 de março, 2015.

Mozilla Developer Network. **HTML**. 2014. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTML. Acesso em 07 de março, 2015.

Mozilla Developer Network. **JavaScript**. 2015. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em 08 de março, 2015.

Mozilla Developer Network. **JavaScript language resources**. 2014. Disponível em https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_Resources. Acesso em 08 de março, 2015.

OLIVEIRA, Eric. **Aprenda AngularJS com estes 5 Exemplos Práticos.** 2013. Disponível em http://javascriptbrasil.com/2013/10/23/aprenda-angularjs-com-estes-5-exemplos-praticos/. Acesso em 09 de agosto, 2015.

OSGI ALLIANCE. **OSGi**. 2015. Disponível em http://www.osgi.org/Main/HomePage. Acesso em 08 de março, 2015.

ORACLE. **Difference between GlassFish Open Source and Commercial Editions**. 2011. Disponível em https://blogs.oracle.com/GlassFishForBusiness/entry/difference_between_glassfish_open_source. Acesso em 20 de junho, 2015.

PRESSMAN, Roger S. **Engenharia de Software**. 1 ed. São Paulo: Makron Books, 1995.

SAUDATE, Alexandre. **REST**: Construa API's inteligentes de maneira simples. São Paulo: Casa do Código, 2013.

SANTOS, Walter dos. **Introdução ao Apache Maven**. Belo Horizonte: Eteg Tecnologia da Informação Ltda, 2008.

SANTOS, Wagner Roberto dos. **RESTful Web Services e a API JAX-RS**. 2009. Disponível em http://www.univale.com.br/unisite/mundo-j/artigos/35RESTful.pdf. Acesso em 08 de agosto, 2015.

SILVA, Maurício Samy. **CSS3:** Desenvolva aplicações web profissionais com uso dos poderosos recursos de estilização das CSS3. São Paulo: Novatec Editora, 2012.

SILVA, Maurício Samy. **HTML5**: A linguagem de marcação que revolucionou a web. São Paulo: Novatec Editora, 2011.

SILVA, Maurício Samy. **JavaScript:** Guia do Programador. São Paulo: Novatec Editora, 2010.

SOUZA, Arthur Câmara; AMARAL, Hugo Richard; LIZARDO, Luis Eduardo O. **PostgreSQL:** uma alternativa para sistemas gerenciadores de banco de dados de código aberto. In: Anais do Congresso Nacional Universidade, EAD e Software Livre, 2012.

STAA, Arndt von. **Programação Modular**: desenvolvendo programas complexos de forma organizada e segura. Rio de Janeiro: Campus, 2000.

STERN, Eduardo Hoelz. **PostgreSQL** - Introducao e Conceitos. 2003. Disponível em http://www.devmedia.com.br/artigo-sql-magazine-6-postgresql-introducao-e-conceitos/7185. Acesso em 10 de agosto, 2015.

USP. **Fundamentos do projeto de software**. 2015. Disponível em http://www.pcs.usp.br/~pcs722/98/Objetos/bases.html. Acesso em 21 de junho, 2015.

W3C. **About W3C**. 2015. Disponível em http://www.w3.org/Consortium/. Acesso em 07 de março, 2015.

W3C. **What is CSS?**. 2015. Disponível em http://www.w3.org/Style/CSS/. Acesso em 08 de março, 2015.