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: Ms. 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*
EJB	*Enterprise JavaBeans*
HTML	*HyperText Markup Language*
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
JVM	Java *Virtual Machine*
NTT	*Nippon Telegraph and Telephone*
OSGi	*Open Services Gateway initiative*
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
XHTML	*Extensible HyperText Markup Language*
XML	*Extensible Markup Language*
W3C	*World Wide Web Consortium*

SUMÁRIO

INTRODUÇÃO

O desenvolvimento de um *software* não envolve apenas métodos de programação. Para desenvolvê-lo é necessário preocupar-se com planejamento, engenharia, metodologia e tecnologia a ser utilizada. Esses fatores influenciam na manutenção, atualização e expansão do mesmo. Dessa forma, definir tais fatores é fundamental para que o *software* seja flexível a mudanças. Pensando dessa maneira, será demonstrado um modelo de desenvolvimento modular, utilizando a especificação OSGi.

OSGi, abreviação para *Open Services Gateway Initiative*, possibilita o desenvolvimento de *softwares* em módulos, disponibilizando o gerenciamento dos mesmos e facilidades na manutenção e expansão da aplicação.

Segundo Fernandes (2009), um sistema modular possui algumas propriedades. Deve ser autocontido, os módulos podem ser incluídos, retirados, instalados ou desinstalados. Outra propriedade é ter alta coesão. Um módulo deve cumprir apenas sua finalidade, ou seja, deve fazer somente as funções que lhe foram atribuídas. O baixo acoplamento é outra propriedade muito importante. Um módulo não precisa se preocupar com implementações de outros módulos que interagem com ele, além de permitir alterá-lo sem a necessidade de atualizar os outros.

Muitos *softwares* são desenvolvidos de maneira semelhante à modularização, divididos em partes, tendo cada parte uma responsabilidade. Porém não são realmente modularizados, ou seja, não atendem ao conceito de modularização como descrito acima. Dois exemplos de projetos de *softwares* que foram desenvolvidos utilizando a especificação OSGi e que atendem a definição de modularização, são, IDE[1] Eclipse, ferramenta de desenvolvimento de softwares e o GlassFish, servidor de aplicações JEE[2].

O uso da modularização, com certeza, traz grandes benefícios para o desenvolvimento e manutenção de um *software*. Poder parar parte de uma aplicação para fazer uma manutenção ou poder instalar novas funcionalidades, garantindo que todas as outras partes restantes continuem funcionando normalmente, seria uma característica notável da aplicação.

Fernandes (2009) forneceu uma visão geral sobre OSGI, mostrando seus benefícios e

1 IDE – Abreviação para *Integrated Development Environment.*
2 JEE – Abreviação para Java *Platform*, *Enterprise Edition*.

salientando a importância da plataforma Java possuir um melhor suporte à modularidade, até demonstrando com um exemplo simples as premissas e vantagens do OSGi.

Mayworm (2010) demonstra a tecnologia OSGi no contexto de aplicações distribuídas, permitindo a disponibilização de seus serviços remotamente, integrando com diferentes *frameworks* de *middleware* para o desenvolvimento de aplicações empresariais.

Malcher (2008) apresenta um modelo de componentes adaptável para se usar em ambientes distribuídos. Também analisa alguns aspectos, como modelo de distribuição, transparência, descoberta de novos módulos disponíveis, desempenho e *performance* dos mesmos. Afirma ainda que para se escolher entre um modelo de distribuição – *deployment* local ou execução remota – é necessário analisar o contexto e o objetivo da aplicação, pois cada um se adapta melhor a determinadas situações e ambientes de execução.

Portanto, após despertar um grande interesse pelo modelo de desenvolvimento modular, será realizado estudos sobre a especificação OSGi, e ainda, o desenvolvimento de uma aplicação modular que demonstre tal modelo de desenvolvimento.

2 OBJETIVOS

Os objetivos desta pesquisa serão demonstrados a seguir.

2.1 Objetivo geral

- Demonstrar o modelo de desenvolvimento modular utilizando a especificação OSGi para aplicações empresariais.

2.2 Objetivos específicos

Para atingir o objetivo geral será apresentado os seguintes objetivos específicos:

- Pesquisar as melhores práticas e *frameworks* que contribuem para a produtividade no desenvolvimento modularizado;
- Desenvolver uma aplicação que exemplifique o modelo de desenvolvimento modularizado através da especificação OSGi;
- Obter através dos resultados uma conclusão sobre as vantagens do modelo de desenvolvimento modular.

3 JUSTIFICATIVA

Diante dos vários segmentos e constantes mudanças empresariais, novos *softwares* são desenvolvidos e precisam estar preparados para acompanhar as modificações que ocorrem nos processos de uma empresa. Dessa forma o desenvolvimento separado em módulos seria uma solução para atender empresas de diferentes setores em vez de criar um *software* para cada ramo empresarial. Além disso a utilização de módulos em um sistema torna mais flexível sua manutenção, pois a mesma é realizada nos módulos, não afetando o restante da aplicação.

O trabalho tem relevância na visão acadêmica diante da aprendizagem da tecnologia OSGi, além de agregar conceitos sobre o desenvolvimento de softwares de forma modularizada. E ainda, por ser pouco utilizada no mercado devido à sua complexidade de aprendizagem, este trabalho visa reunir informações e disponibilizar uma documentação com práticas de programação e vantagens que essa tecnologia pode oferecer.

Desenvolvedores de *softwares* procuram sempre a utilização de novas práticas e tecnologias produtivas. O modelo de desenvolvimento modular através da tecnologia OSGi oferece diversos benefícios que ajudam na expansão e manutenção de uma aplicação. Isto também se torna útil para as empresas, pois para novas funcionalidades seriam criados módulos, e a manutenção do sistema seria realizada somente no módulo específico, sem a necessidade de parar o restante do sistema.

Como a adoção dessa nova prática e tecnologia proporciona a reutilização de código, consequentemente não haverá a necessidade do desenvolvimento repetido de funcionalidades, economizando tempo, energia elétrica e recursos computacionais, algo que se tornou de grande valor atualmente, visando o desenvolvimento harmonioso e sustentável entre o ser humano e a natureza.

4 QUADRO TEÓRICO

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

4.1 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 a seguir.

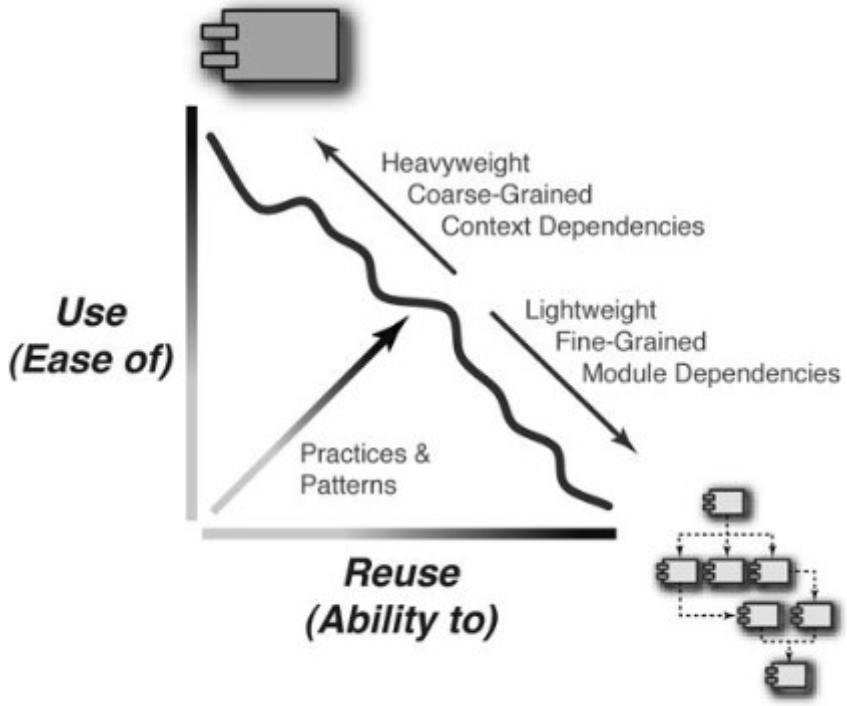

Figura 1 - Uso e reúso de partes do software. Fonte: Java Application Architecture (2012)

Conforme mostra a Figura 1, quanto maior a granularidade de uma parte do sistema, é mais fácil usá-la e mais difícil de reutilizá-la. Do contrário, quanto menor a granularidade de uma parte, é mais fácil reutilizá-la, e mais difícil usá-la (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 abaixo.

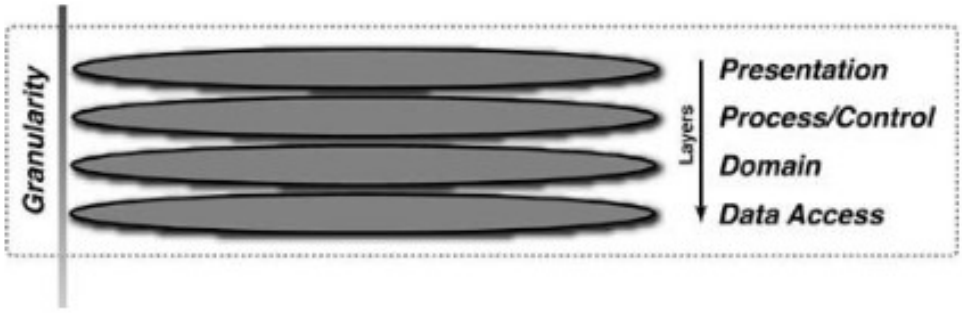

Figura 2 - Arquitetura de camadas. Fonte: Java Application Architecture (2012)

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 a seguir.

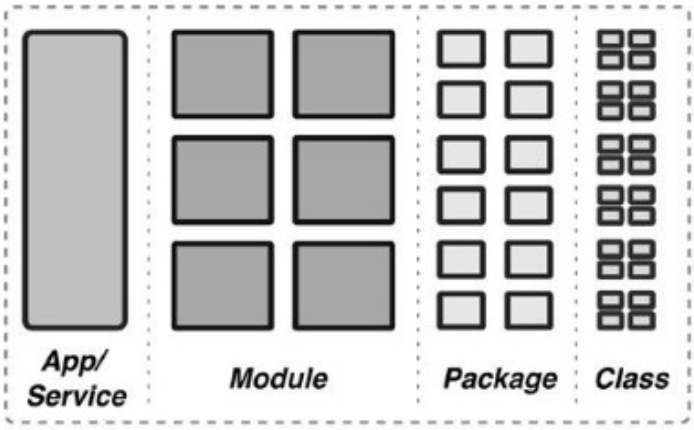

Figura 3 - Arquitetura modular OSGi. Fonte: Java Application Architecture (2012)

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.

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

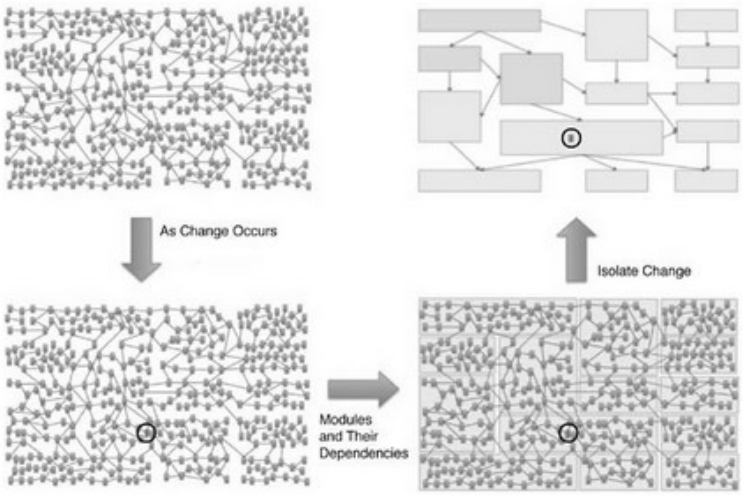

Figura 4 - Impacto de mudanças. Fonte: Java Application Architecture (2012)

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.

4.2 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 liberação 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 especificação OSGi define uma camada chamada *Life Cycle* que gerencia o ciclo de vida e provê uma API[3] 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 5, a seguir, é demonstrado cada um desses estados e suas funções.

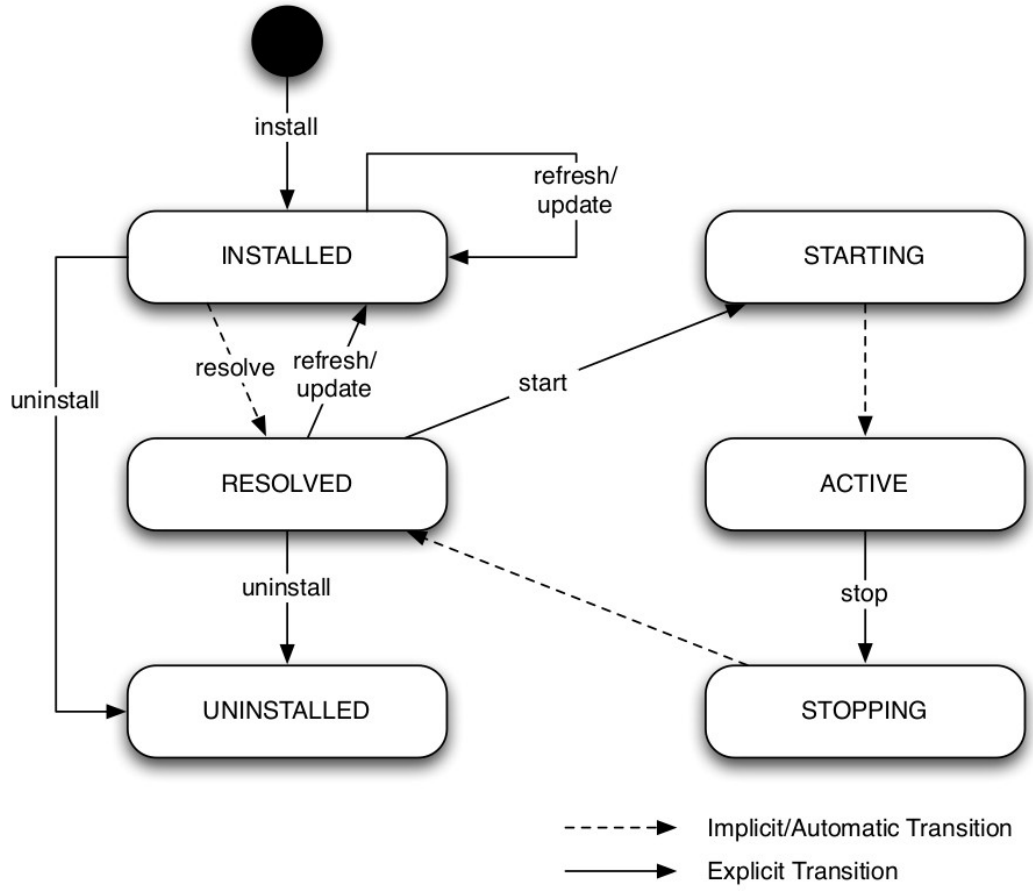

Figura 5 - Estados dos *bundles*. Fonte: OSGi In Practice (2009)

3 API – Abreviação para *Application Programming Interface*.

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).

A OSGi Alliance (2015), define que o *framework* OSGi utiliza uma arquitetura de camadas em sua implementação, conforme demonstrado na Figura 6 a seguir.

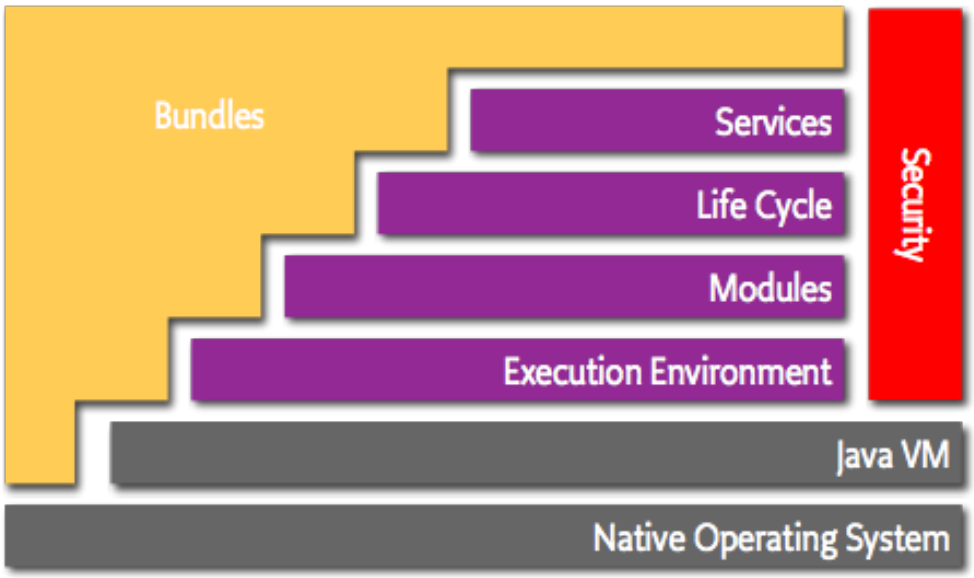

Figura 6 - Estrutura de camadas. Fonte: OSGi Alliance (2015)

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*.

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[4].

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 a seguir. 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)

4 BSD – Abreviação para *Berkeley Software Distribution*.

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”.

4.3 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[5] com a finalidade de desenvolver aplicativos para diferentes plataformas (Windows – Linux – *Web* – *Mobile*). Qualquer dispositivo que possua a JVM[6] é capaz de executar um *software* desenvolvido em Java (CLARO; 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, 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.

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

4.4 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 auxiliará no desenvolvimento do projeto, pois permite, além das facilidades para fazer a aplicação, a opção de migração para outra implementação OSGi (BARTLETT, 2009).

4.5 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.*

4.6 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.

Filho (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.*

4.7 Spring Framework

Spring Framework disponibiliza um modelo de programação e configuração para o desenvolvimento de aplicações corporativas para a plataforma Java, com o objetivo de estruturar a aplicação de maneira que o desenvolvedor tenha foco na lógica de negócios em nível de aplicativo, enquanto o *framework* gerencia implementações específicas (PIVOTAL SOFTWARE, 2015).

De acordo com DevMedia (2015), o Spring é um *framework Open Source* que foi criado por Rod Johnson no ano de 2002, com o objetivo de simplificar o desenvolvimento utilizando a plataforma Java, possibilitando a criação de *softwares* que só poderiam ser criados anteriormente através de EJB's[7].

Referente às funcionalidades, destacam-se como principais: a injeção de dependências, programação orientada a aspectos, arquitetura MVC[8] para *web*, *framework* de serviços *web* RESTFul e suporte para JDBC[9], JPA[10] e JMS[11] (PIVOTAL SOFTWARE, 2015).

O Spring Framework é integrado com a plataforma OSGi através do Spring Dynamic Modules que gerencia o ciclo de vida, controla e permite exportar e importar serviços OSGi de forma transparente (PIVOTAL SOFTWARE, 2015).

7 EJB – Abreviação para *Enterprise JavaBeans*.
8 MVC – Abreviação para *Model View Controller.*
9 JDBC – Abreviação para Java *Database Connectivity*.
10 JPA – Abreviação para Java *Persistence* API.
11 JMS – Abreviação para Java *Message Service.*

4.8 Hibernate

Segundo Bauer e King (2005), Hibernate é um *framework* de persistência, que possui como propósito fornecer uma visão orientada a objetos do banco de dados relacional, ou seja, permite persistir e gerenciar objetos Java para tabelas existentes no banco de dados, de forma simplificada.

Sua configuração é realizada em arquivo com extensão XML[12], que contém informações do banco de dados. Além disso, as classes em Java são mapeadas através de anotações definidas e entendidas pelo *framework*.

Utilizar-se dos benefícios oferecidos pelo Hibernate para o desenvolvimento de uma aplicação que utilize banco de dados relacional e linguagem orientada a objetos é de grande estima, pois segundo Durham e Johnson (1996), no desenvolvimento de *softwares*, o *framework* auxilia no desenvolvimento e organização do projeto. Tipicamente, o *framework* pode incluir programas de apoio, bibliotecas de código, linguagens de *script* e outros *softwares* para ajudar a desenvolver e juntar diferentes componentes do seu projeto.

4.9 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[13] ANSI-92, 96 e 99, possui alta *performance* e fácil administração e utilização em projetos por *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).

12 XML – Abreviação para *Extensible Markup Language*.
13 SQL – Abreviação para *Structured Query Language*.

Seu código é livre e há um grupo responsável pela sua validação. Grandes empresas como a Fujitsu, NTT[14] Group, Skype, Hub.org, Red Hat e SRA[15] 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).

4.10 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* (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).

14 NTT – Abreviação para *Nippon Telegraph and Telephone*.
15 SRA – Abreviação para *Systems Research and Applications Corporation*.

4.11 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.

4.12 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*"[16].

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[17]. 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).

4.13 Test-Driven Development (TDD)

Test-Driven Development, é o significado da sigla TDD, que, em português, significa desenvolvimento guiado por testes. É sem dúvida uma das práticas mais populares utilizadas no desenvolvimento de softwares, com a ideia de escrever os testes antes mesmo de escrever o código de programação do *software* (ANICHE, 2014).

Automatizar os testes de um *software* é a atividade de submeter o código desenvolvido para outro *software* testá-lo, e assim obter o apontamento de erros no código e indicação das falhas para futura correção.

Qualidade do software, fácil manutenção e evolução são vantagens que se conquista testando um *software* (VAZ, 2003).

Conforme Aniche (2012), quando a produtividade é medida através do número de

16 Folha de estilo em cascata (CSS) é um mecanismo simples para adicionar estilos (por exemplo: fontes, cores, espaçamentos) para documentos web (tradução nossa).

17 XHTML – Abreviação para *Extensible HyperText Markup Language*.

linhas de código escrito por dia, o rendimento será menos produtivo. Todavia, se produtividade for a quantidade de linhas de código final sem defeitos escritos por dia, provavelmente o rendimento será melhor ao usar testes automatizados.

Segundo Vaz (2003), para facilitar a execução das rotinas dos testes automatizados foram criados *frameworks* como o JUnit, que através da instalação de seu plug-in na ferramenta de desenvolvimento Eclipse, disponibiliza uma infraestrutura para se realizar os testes do código.

Dentre as atividades de um processo de desenvolvimento, as atividades de testes têm uma importância fundamental para a garantia de qualidade do software que está sendo desenvolvido, a qual é aplicada no decorrer de todo o projeto.

5 QUADRO METODOLÓGICO

Neste capítulo serão abordados os caminhos definidos para conduzir a pesquisa. Serão apresentados o tipo de pesquisa, seu contexto, bem como os participantes, o orçamento, o cronograma, os instrumentos e os procedimentos para o desenvolvimento da pesquisa.

5.1 Tipo de pesquisa

Pesquisa é um processo desenvolvido com o objetivo de obter respostas para indagações propostas, através de conhecimentos existentes e a utilização de métodos, técnicas e procedimentos científicos. Uma pesquisa se faz necessária quando não existem repostas suficientes que satisfaçam a resolução de problemas (GIL, 2002).

Para atingir os objetivos desta pesquisa, será desenvolvida uma pesquisa de abordagem aplicada, a qual é utilizada quando se desenvolve um produto real, com uma finalidade prática, que pode ser aplicado em determinado contexto. Conforme aponta Appolinário (2004), pesquisas aplicadas têm o objetivo de resolver problemas ou necessidades concretas e imediatas.

Aplicando esses conceitos e utilizando a pesquisa de forma aplicada por agregar maiores vantagens e mostrar melhor os resultados obtidos, será desenvolvido um *software* modularizado com o objetivo de facilitar sua manutenção e aplicação em empresas de diferentes ramos, demonstrando assim, a tecnologia OSGi.

5.2 Contexto de pesquisa

Cada vez mais *softwares* são utilizados em empresas, indústrias, computadores pessoais, *web* e dispositivos móveis. Estes são desenvolvidos por meio de práticas e tecnologias existentes que auxiliam na sua criação, porém a não utilização de tais ferramentas

torna o seu desenvolvimento e manutenção um processo desgastante e trabalhoso.

Esta pesquisa demonstra a utilização do *framework* OSGi, que possibilita o desenvolvimento do *software* em módulos, oferecendo melhor organização na sua estrutura, vantagens na manutenção e maior facilidade na expansão do mesmo, já que é construído em módulos.

Softwares modularizados oferecem maior flexibilidade para atender empresas de diferentes áreas. Através do OSGi, os módulos podem ser desenvolvidos de forma independente, sendo assim, o *software* pode disponibilizar módulos específicos para cada empresa. Dessa maneira será desenvolvida uma aplicação que pode ser adotada por empresas de diferentes segmentos no mercado, bastando apenas que exista um módulo que atenda a necessidade específica da área.

O objetivo desse projeto é trazer as metodologias, engenharias, técnicas e tecnologias para o âmbito de *softwares* comerciais e prover uma base para a facilitação na manutenção e expansão do mesmo.

5.3 Participantes

Participam deste projeto os acadêmicos do curso de bacharelado em Sistemas de Informação da Universidade do Vale do Sapucaí – UNIVAS, os alunos Felipe Rodrigues do Prado e João Paulo Nakajima Pereira sob a orientação do professor Márcio Emílio Cruz Vono de Azevedo.

Felipe Rodrigues do Prado, no ano de 2010, formou-se em técnico em informática pelo Serviço Nacional de Aprendizagem Comercial (SENAC) e atualmente é estagiário no Inatel *Competence Center* (ICC), atuando como desenvolvedor de *software* para soluções empresariais.

João Paulo Nakajima Pereira exerce a profissão de analista de suporte na empresa Automação e Cia, a qual presta serviços de atendimento ao cliente e suporte em *softwares* de automação de empresas.

Márcio Emílio Cruz Vono de Azevedo é bacharel em Engenharia Elétrica com ênfase em Eletrônica pelo Instituto Nacional de Telecomunicações – INATEL, Mestre em Ciência da

Computação pela Universidade Federal de Itajubá – UNIFEI, é professor nas instituições de ensino INATEL na disciplina de Orientação a Objetos e na UNIVAS nas disciplinas de Engenharia de Software, Linguagem de Programação e Sistemas Distribuídos. É diretor de desenvolvimento de software na empresa TM Tecnologia da Informação Ltda.

O interesse dos participantes nesta pesquisa é conhecer e demonstrar o potencial de um sistema modularizado de atingir novas áreas empresariais, analisando os benefícios e desafios que poderão ser encontrados, além de obter mais conhecimentos sobre o modelo de desenvolvimento modular, permitindo assim, como consequência final o alcance de maior vantagem competitiva para as empresas, pois estas usufruiriam de menor custo, tanto financeiro quanto de tempo de manutenção do *software*, propiciando maior confiabilidade e disponibilidade do mesmo.

5.4 Instrumentos

Para desenvolver esta pesquisa será necessária a realização de reuniões entre os participantes, para a obtenção e organização das informações, divisão das tarefas e desenvolvimento do *software*, pois segundo Kioskea (2014), as reuniões são um meio para partilhar, num grupo de pessoas, um mesmo nível de conhecimento sobre um assunto ou um problema e para tomar decisões coletivamente. Além disso, decisões tomadas coletivamente, com representantes das diferentes entidades interessadas, serão mais facilmente aceitas por todos.

Os dados serão obtidos a partir de livros, artigos, documentações e reuniões *online* com pessoas que possuem conhecimento e experiência nesse tema. Após aplicados estes instrumentos será possível analisar e organizar as informações.

As reuniões entre os participantes possuem o objetivo de planejar a execução pesquisa, tomar decisões quanto ao desenvolvimento do *software*, bem como as tecnologias e metodologias que serão utilizadas.

5.5 Procedimentos

A ideia deste trabalho é elaborar um *software* modularizado que permita a fácil manutenção e expansão, auxiliando no suporte e futura ampliação do mesmo. Desta forma, para o desenvolvimento desta pesquisa, tornou-se necessário a distribuição dos procedimentos citados a seguir:

- Desenvolvimento de protótipos;
- Definir o *software* a ser desenvolvido para demonstrar as funcionalidades e vantagens de um *software* modularizado;
- Levantar dos requisitos;
- Modelar e elaborar o banco de dados;
- Mapear o banco de dados seguindo a metodologia objeto relacional;
- Fazer o *design* e criar as *interfaces* de interação com o usuário;
- Implementar a lógica do negócio;
- Estruturar e ligar funcionalmente os módulos do sistema;
- Documentar o projeto;
- Testar o *software*.

Após a conclusão desses procedimentos deverá ser feita a análise para a verificação do funcionamento e desempenho da tecnologia no âmbito de aplicações empresariais, averiguando os benefícios e o ponderando a utilização dessa prática de desenvolvimento de *softwares*.

5.6 Orçamento

A Tabela 1 demonstra o planejamento dos gastos com o projeto.

ORÇAMENTO DETALHADO DO PROJETO			
1. RECURSOS MATERIAIS			
1.1 Material Permanente: (equipamentos, livros, máquina fotográfica e gravadores, *softwares*, equipamentos de informática, etc.)			
Descrição do Material	Quantidade	Valor (unidade – em reais)	Total R$
Livros	1	52,11	52,11
Subtotal			52,11
1.2 Material de Consumo: (Papéis necessários para impressões, cartuchos de tinta para impressora, pastas, etc.)			
Descrição do Material	Quantidade	Valor (unidade – em reais)	Total R$
Material de papelaria			20,00
Subtotal			20,00
2. SERVIÇOS: (cópias, encadernações, impressos gráficos, despesas de locomoção e estadia, etc.)			
Descrição do Material	Quantidade	Valor (unidade – em reais)	Total R$
Capa dura	2	50,00	100,00
Cópias	50	0,10	5,00
Locomoção	100	2,75	275,00
Impressões	500	0,10	50,00
Subtotal			430,00
3. RESERVA TÉCNICA/ Despesas Operacionais (10% no total do dispêndio)			
Reserva	1	50,21	50,21
Subtotal			50,21
TOTAL	Valor previsto R$	Reserva de gastos	Total
	502,11	50,21	552,32

Tabela 1 - Orçamento do projeto

Após a análise dos objetivos do projeto, foi visto que serão necessários livros para estudo, passagens de ônibus para reuniões, incluso também gastos com material de papelaria, impressões, xerox e encadernações, para assim possibilitar um melhor projeto de pesquisa.

5.7 Cronograma

Os cronogramas demonstrados na Tabela 2 e 3 mostram como serão realizadas as etapas do desenvolvimento da pesquisa.

5.7.1 Primeiro Semestre de 2015

Mês / Tarefas	DEZ	JAN	FEV	MAR	ABR	MAI
Orientação do Pré-projeto	X					
Formulação do Pré-projeto		X				
Pesquisas dos itens do pré – projeto		X				
Fechamento do pré-projeto			X			
Entrega do pré-projeto			X			
Orientação da Introdução, Objetivos e Justificativa			X			
Formulação da Introdução, Objetivos e Justificativas			X			
Fechamento da Introdução, Objetivos e Justificativas			X			
Entrega da Introdução, Objetivos e Justificativas			X			
Estudo da tecnologia OSGi				X		
Orientação do Quadro Teórico				X		
Formulação do Quadro Teórico				X		

Entrega do Quadro Teórico				X		
Orientação do Quadro Metodológico				X	X	
Formulação do Quadro Metodológico					X	
Entrega do Quadro Metodológico					X	
Revisão do projeto para entrega					X	
Entrega dos projetos para qualificação						X
Bancas de qualificação de Projetos						X
Orientações finais dos projetos						X

Tabela 2 - Cronograma do Primeiro Semestre de 2015

5.7.2 Segundo Semestre de 2015

Mês **Tarefas**	**JUN**	**JUL**	**AGO**	**SET**	**OUT**	**NOV**	**DEZ**
Desenvolvimento de protótipos	X						
Estudo de *frameworks* e ferramentas	X						
Desenvolvimento do *software* modularizado	X	X	X				
Atualização da pesquisa			X	X			
Análise e discussão de resultados				X			
Pré-banca				X			
Redação final do TCC					X	X	
Defesa pública						X	
Acertos finais para capa dura						X	X
Entrega da capa dura							X

Tabela 3 - Cronograma do Segundo Semestre de 2015

Estas etapas têm em vista serem seguidas minuciosamente, sabendo que poderão sofrer alterações.

REFERÊNCIAS

ANICHE, Maurício. **Test-Driven Development:** Teste e Design no Mundo Real. São Paulo: Casa do Código, 2012.

ANICHE, Maurício. **TDD**. 2014. Disponível em http://tdd.caelum.com.br/. Acesso em 11 de março, 2015.

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.

BAUER, Christian; KING, Gavin. **Hibernate in action**. Greenwich: Manning Publications Co., 2005.

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.

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.

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

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.

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

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.

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.

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.

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.

Pivotal Software. **Preface**. 2015. Disponível em http://docs.spring.io/osgi/docs/current/reference/html/preface.html. Acesso em 10 de março, 2015.

Pivotal Software. **Spring Framework**. 2015. Disponível em http://projects.spring.io/springframework/. Acesso em 10 de março, 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.

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.

VAZ, Rodrigo Cardoso. **JUnit – Framework para testes em Java**. Palmas: Centro Universitário de Palmas, 2003.

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.