LUCAS RAFAEL DA SILVA

NECESSIDADE DE SE UTILIZAR BOAS PRÁTICAS ARQUITETURAIS E PADRÕES DE PROJETO NO DESENVOLVIMENTO DE *WEB SERVICE* BASEADO NA ARQUITETURA *RESTFUL*

BACHARELADO

EM

CIÊNCIA DA COMPUTAÇÃO

FIC – MINAS GERAIS
2016

LUCAS RAFAEL DA SILVA

NECESSIDADE DE SE UTILIZAR BOAS PRÁTICAS ARQUITETURAIS E PADRÕES DE PROJETO NO DESENVOLVIMENTO DE *WEB SERVICE* BASEADO NA ARQUITETURA *RESTFUL*

Monografia apresentada à banca examinadora do Curso de Graduação em Ciência da Computação das Faculdades Integradas de Caratinga como exigência parcial para obtenção do grau de bacharel em Ciência da Computação, sob orientação do professor Msc. Glauber Luis da Silva Costa.

FIC – CARATINGA
2016

LUCAS RAFAEL DA SILVA

NECESSIDADE DE SE UTILIZAR BOAS PRÁTICAS ARQUITETURAIS E PADRÕES DE PROJETO NO DESENVOLVIMENTO DE *WEB SERVICE* BASEADO NA ARQUITETURA *RESTFUL*

Monografia submetida à Comissão examinadora designada pelo Curso de Graduação em Ciência da Computação como requisito para obtenção do grau de Bacharel.

Profª. Msc. Glauber Luis Costa
Faculdades Integradas de Caratinga

Profº. Jonilson Batista Campos
Faculdades Integradas de Caratinga

Profº. Wanderson Miranda Nascimento
Faculdades Integradas de Caratinga

Caratinga, 14/12/2016

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me dado forças quando mais precisei, a toda minha família por sempre me apoiar e a me incentivar nos meus estudos, aos professores e meus colegas de classe por sempre estarem trocando conhecimento, que ao longo dos anos foi fundamental para o crescimento tanto quanto pessoal quanto intelectual, ao meu orientador e amigo de trabalho Glauber Costa por sempre ter acreditado no meu potencial e ter me ajudado muito nesta longa jornada. Também agradeço a minha namorada Nataly por nunca me deixar desistir e sempre me mostrar que eu era capaz mesmo nos momentos mais difíceis. E por último agradeço a todos que um dia me ajudaram nesta jornada.

"Mais poder tem o sábio do que o forte, e o homem de conhecimento, mais do que o robusto".

Salomão

RESUMO

Um dos fatores que interferem na pouca utilização dos padrões de projeto de *software* é que essa atividade não é uma tarefa simples, ela exige um bom planejamento para ser bem sucedida, e muitos desenvolvedores pensam ser perda de tempo e uma queda na produtividade por causa do tempo gasto para padronização do código-fonte, além disso, a falta de conhecimento sobre os padrões faz com que muito tempo seja investido na manutenção do *software* por muitas das vezes não se ter uma maleabilidade.

No entanto, garantir uma arquitetura de qualidade no desenvolvimento de *software* é fator indispensável, devido a constante mudança nos *softwares* atuais. E com a busca pela satisfação dos clientes, a complexidade dos sistemas estão cada vez maiores, assim surgindo a necessidade de investir em meios profissionais para se obter uma reposta rápida às mudanças. Visualizando este cenário, este trabalho teve por objetivo analisar a adequação de padrões de projeto no processo *software* no desenvolvimento de *web service RESTful*, visando a diminuição do tempo gasto em manutenção ou evolução do *software*.

O questionário aplicado foi desenvolvido tendo como base a pesquisa bibliográfica realizada sobre os principais padrões de projeto de *software*, além dos fundamentos sobre o *web service* e REST. Foram coletadas respostas de 46 profissionais da área de Tecnologia da Informação, e os resultados demonstraram que os profissionais têm ciência da importância de se utilizar padrões de projeto de *software*, apesar de grande parte ainda não utilizá-los efetivamente e por consequência não obterem *web services* maleáveis e de fácil evolução.

Palavras-chave: Padrões de projeto de *software*, *Web service*, *RESTful*, Arquitetura de *software*.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

HTML - *HyperText Markup Language* (Linguagem de Marcação de Hipertexto).

HTTP - *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto).

JSON - *JavaScript Object Notation* (Notação de Objetos JavaScript).

REST - *Representational State Transfer* (Transferência de Estado Representacional).

SOAP – *Simple Object Access Protocol* (Protocolo Simples de Acesso a Objetos).

TI – Tecnologia da Informação.

URI - *Uniform Resource Identifier* (Identificador Uniforme de Recursos).

URL - *Uniform Resource Locator* (Localizador Uniforme de Recursos).

WSDL - *Web Services Description Language* (Linguagem de Descrição de Serviços Web).

XML - *Extensible Markup Language* **(**Linguagem Extensível de Marcação Genérica**).**

SUMÁRIO

INTRODUÇÃO

Com o crescimento da internet e com a chegada de novos dispositivos como por exemplo: *tablets, smartphones* e outros, surgiu a necessidade dos mesmos interagirem com as aplicações *web*. Como solução surgiram os navegadores *web* só que em versão *mobile*. Os sites foram adaptados para trabalhar com telas menores devido à esta necessidade, mas mesmo com o acesso via navegador e sites personalizados para trabalhar em telas menores, a usabilidade não era tão boa, além da questão da dependência do acesso contínuo à internet para visualizar e sincronizar as informações providas pelos sites o que ainda nos dias atuais é difícil de se conseguir. Como as aplicações para *desktop* há tempos já haviam passado pelo mesmo problema, no qual era necessário trabalhar de forma *off-line* e depois sincronizar esses dados, de imediato surgiu uma solução, a criação de *web service*.

Um *web service* ou "serviço web" segundo Deitel (2010, p. 1018) "[...] é um componente de *software* armazenado em um computador que pode ser acessado por um aplicativo (ou outro componente de *software*) em outro computador por uma rede".

Então surgiu-se a ideia de criar aplicativos que consumissem *web service,* que disponibilizaria as mesmas informações disponíveis nos sites. Mas as tradicionais arquiteturas que trabalhavam com *web service* como por exemplo o SOAP (*Simple Object Access Protocol*) consumiam uma grande quantidade de dados na rede para se comunicar e, por mais que já fosse bem consolidado no mercado, não iria atender uma necessidade, que era o baixo consumo de banda de rede para essa sincronização de dados, pois esta arquitetura utiliza o XML para comunicação entre cliente e servidor, que é um formato de arquivo com o tamanho muito maior em *bytes* que arquivos JSON (JavaScript Object Notation) por exemplo.

Uma possível solução seria trabalhar com a arquitetura REST (*Representational State Tranfer*) que também trabalha com o protocolo HTTP (*Hypertext Transfer Protocol*) como a SOAP e, como benefício, essa tecnologia tem uma maior flexibilidade na troca de mensagens, pois não impõe restrições ao formato da mensagem, assim suas operações são mais rápidas que a do SOAP, pois o padrão utilizado por este (WDSL) adiciona um *overhead* considerável, tanto por ser em XML quanto por adicionar muitas *tags* de meta-informação em suas mensagens. Além disso o REST foi tese de Doutorado (PHD) escrita por Roy Fielding, mesmo criador do HTTP, ou seja, uma arquitetura criada para se aproveitar ao máximo dos recursos providos pelo protocolo HTTP, e que mesmo sendo recente já apresentava grande potencial.

O REST como vantagem tem uma baixa curva de aprendizado em nível de implementação de código-fonte, além da possibilidade de trabalhar com o formato JSON (também trabalha com XML e TEXT). Mas por ser uma tecnologia nova, ainda não possui padrões bem definidos no mercado, não a tecnologia, mas sim a forma como ela é implementada. Assim podendo gerar web *services* robustos, mas que não são suscetíveis a mudanças, o que pode gerar transtornos, pois esse tipo de tecnologia sofre mutações constantes no seu ciclo de vida e isso pode elevar o custo final e impossibilitar a evolução do *software*.

A proposta deste trabalho é descobrir se a utilização de padrões de projeto de *software* como apoio no desenvolvimento de *web service RESTful*, é uma boa solução para se ter uma maior maleabilidade a mudanças e diminuir os valores gastos com manutenção. Sendo assim um dos principais problemas dessa pesquisa é entender o porquê de muitos profissionais ligados ao desenvolvimento de *web service RESTful* não utilizarem padrões de projeto de *software*.

Pensando nisso, o objetivo desse trabalho é apresentar o ponto de vista dos profissionais de TI, em relação aos a padrões de projeto de *software*, por meio de uma pesquisa científica do tipo qualitativa, que busca responder às hipóteses de que o uso dos padrões no desenvolvimento de *web service RESTful* aumenta a capacidade de evolução e diminuição de gastos com esse tipo de aplicação, contribuindo assim para o desenvolvimento de *softwares* com arquitetura de qualidade e alta maleabilidade.

O questionário utilizado na pesquisa foi dividido em quatro grupos: Perfil do Entrevistado; Experiência com *web services* baseado em *RESTful*; Experiência com padrões de projeto; Relação entre web services baseados em *RESTful* e padrões de projeto. A pesquisa contou com respostas de 46 profissionais da área de TI, que de maneira geral avaliaram positivamente o uso de padrões de projeto de *software* no desenvolvimento de *web service RESTful*.

Este trabalho está estruturado em três capítulos principais. O capítulo 1 descreve o referencial teórico deste trabalho, o qual apresenta os principais conceitos sobre *web service*, tipos e funcionamento. Apresenta também o conceito de *software,* programação orientada a objetos e padrões de projeto, HTTP e a arquitetura REST. O capítulo 2 apresenta a metodologia utilizada para alcançar os resultados. E o capítulo 3 descreve os resultados obtidos com a pesquisa realizada com os profissionais de Tecnologia da Informação. Por fim, são apresentadas as conclusões e sugestões para trabalhos futuros.

1 REFERENCIAL TEÓRICO

O avanço da Tecnologia da Informação vem sendo acompanhado pelo aumento da complexidade no processo de desenvolvimento de *software*, e junto vem uma crescente necessidade por *softwares* mais versáteis e que atendam a todo tipo de necessidade, como por exemplo, funcionar *online*, em desktops, celulares e *tablets*, e ainda que sejam de alta performance e de baixo custo devido à grande concorrência no cenário atual. A criação de *web service* é essencial para construir novas aplicações e atender a essas necessidades.

Nesta seção será descrito o referencial teórico desde trabalho, o qual abordará os principais conceitos sobre *software*, padrões de projeto e técnicas para o desenvolvimento de um *web service* baseado na arquitetura *RESTful*, permitindo assim uma melhor compreensão dos assuntos abordados no capítulo de metodologia.

1.1 SOFTWARE

Um *software* pode ser definido com um programa de computador, algo que automatize tarefas repetitivas que antes eram feitas manualmente, instruções pré-definidas que quando executadas são capazes de realizar algum procedimento desejado em um computador. Também pode ser definido como uma estrutura de dados que é manipulada por um programa a fim de gerar informações ou até mesmo como manuais que descrevem os procedimentos para que um programa execute (PRESSMAN, 2006).

O *software* é algo abstrato e intangível, não é limitado pelas leis da física ou processos de manufatura, por isso podem se tornar algo complexo em um piscar de olhos. Existem vários tipos de *software*, sistemas para uso pessoal, comercial, de alcance local ou até mesmo mundial. Atualmente ele está por toda parte, seja em equipamentos eletrônicos ou serviços do governo, sempre haverá um por trás dos mesmos para controlá-los (SOMMERVILLE, 2011).

Sommerville (2011) afirma que os *softwares* são separados em duas categorias, os genéricos e os fabricados por encomenda. Os genéricos são construídos para atender a maioria dos usuários, por exemplo, um editor de texto onde vários usuários conseguirão utilizá-lo sem

maiores necessidades de funções avançadas. Já os fabricados por encomenda são construídos para atender necessidades específicas de um grupo de usuários, como por exemplo, um *software* que gerencie as finanças de uma determinada empresa.

Os *softwares* podem ser muitos simples ou complexos, mas o importante é compreender que eles servem para resolver problemas, por sua vez, agilizando as tarefas, gerando resultados satisfatórios e entregando resultados com um mínimo de confiança. É importantíssimo compreender os benefícios que o mesmo pode trazer aos seus usuários, contudo esses benefícios serão considerados, mas não serão o foco desde estudo.

1.1.1 Engenharia de Software

A engenharia de *software* surgiu pela necessidade do desenvolvimento de *softwares* profissionais, onde o foco não era mais no desenvolvimento individual, mas em equipe. Ela inclui várias técnicas para essa construção, como especificação de requisitos, projeto e evolução de sistemas, que no desenvolvimento pessoal não são muito importantes ou não são levadas em consideração (SOMMERVILLE, 2011).

No desenvolvimento de *softwares* para uso pessoal, onde possivelmente ninguém mais usará, não é necessário documentar o programa, sua arquitetura, ter um documento de requisitos, documento de utilização do sistema e etc. Mas se ele for produzido para o uso de outras pessoas como, por exemplos, os comerciais, onde uma equipe estará envolvida no desenvolvimento, todos os recursos adicionais além do próprio código fonte serão necessários para entendimento do sistema (SOMMERVILLE, 2011).

> O *software* distribui o produto mais importante da nossa era – a informação. Ele transforma dados pessoais (por exemplo, transações financeiras de um indivíduo) de modo que possam ser mais úteis num determinado contexto; gerencia informações comerciais para aumentar a competitividade; fornece um portal para redes mundiais de informações (Internet) e os meios para obter informações sob todas as suas formas (PRESSMAN, 2011, P. 31).

Sendo assim, para garantir a qualidade os desenvolvedores utilizam da engenharia de *software*, que envolve processos, conjuntos de práticas e várias ferramentas que capacita o

desenvolvimento de sistemas com alta qualidade e de forma profissional. Os processos de engenharia definem técnicas que possibilitam o desenvolvimento dentro do prazo e com o melhor nível de qualidade possível. As práticas ou técnicas fornecem informações técnicas que incluem tarefas de comunicação, análise de requisitos, modelagem, construção, testes e suporte, essas ferramentas fornecem suporte automatizado para o processo e práticas da engenharia de *software* (PRESSMAN, 2011).

Sommerville (2011) afirma que engenharia de *software* é uma disciplina de engenharia que cuida de todos os estágios da produção de um *software*, desde as especificações até a manutenção do mesmo quando já estiver sendo utilizado. Seu conceito a princípio pode parecer bem simples, entretanto, é fundamental que conheçamos bem sua metodologia e técnicas, a fim de compreender como funciona a elaboração de um *software* e ter em mente o quão importante é a utilização da mesma.

1.2 PROGRAMAÇÃO ORIENTADA A OBJETOS

Uma Classe é composta por propriedades e métodos. As propriedades são seus atributos e os métodos são funções que atuam sobre estes atributos. Sendo assim os atributos são variáveis responsáveis por armazenar os dados e os métodos realizar a comunicação entre eles (SANDERS, 2013).

Um objeto é algo distinguível que contém atributos ou propriedades e que se comporta de maneira específica. Cada objeto tem sua própria identidade e é distinguível de outro, mesmo que seus atributos sejam idênticos. Um objeto possui limites nítidos, mesmo que sejam instanciados pela mesma classe (REZENDE, 2005).

Existem definições distintas sobre o que é o desenvolvimento de *software* orientado a objetos. Contudo, a ideia geral sobre o que é a orientação a objetos segundo Mazzola (2010), limita-se a:

- Uma abordagem de modelagem e desenvolvimento que tem por objetivo à construção de sistemas complexos a partir de pequenos componentes individuais;
- Uma técnica de construção de *software* condicionada a uma coleção estruturada de tipos abstratos de dados;

- Um estilo de desenvolvimento de aplicações com alto nível de abstração, com intuito de construir de forma econômica o que imita o mundo real mais fielmente;
- Uma forma de organizar o *software* como um repositório de objetos discretos que incorporam estrutura de dados e comportamentos.

A fabricação de *softwares* orientados a objetos está fortemente acoplada à abstração, encapsulamento, herança e polimorfismo (MAZZOLA, 2010).

A herança e o polimorfismo são grandes auxiliadores no reuso de objetos. Pois através dessas técnicas é possível criar uma nova classe, reutilizar métodos já existentes, além disso, alterar o comportamento dos métodos sem alterar a forma como eles trabalham nas demais classes, sendo possível obter um maior nível de reuso de modo abstrato. É possível utilizar um método sem compreender detalhadamente seu funcionamento (PAULA, 2000).

1.3 PADRÕES DE PROJETO

Padrão de projeto serve para descrever um problema diante de um determinado ambiente junto com a solução para o mesmo, podendo utilizar está solução várias vezes e de diversas formas diferentes (GAMMA, 2010).

Um padrão de projeto pode ser descrito ou catalogado contendo as seguintes informações segundo Gamma (2010):

- Nome do padrão: servida de referência para descrever de forma abstrata o que o padrão de projeto busca tratar, deve ser descrito em uma ou duas palavras. Dar nomes aos padrões de projeto é importante, pois aumenta o nosso vocabulário e, além disso, é de grande ajuda em documentações ou até mesmo em discussões sobre o padrão, pois ao se dar um nome não precisamos relembrar a todo o momento do que estamos tratando.
- Problema: descreve em que situação aplicar o padrão, busca explicar o contexto e o problema, expõe quais são as condições para que o padrão de projeto seja aplicável, pode-se utilizar de classes ou objetos para exemplificar a situação.
- Solução: descreve como é o funcionamento do padrão de projeto através de classes, objetos, relacionamentos, responsabilidades e suas colaborações. A solução pode ser

demonstrada em um contexto abstrato para se exemplificar, mas nunca em algo concreto, pois um padrão busca ser como um gabarito que pode ser aplicado em diferentes situações.

- Consequências: são os pontos positivos e negativos de se aplicar padrões de projeto em um *software*. Geralmente são descritas linguagens de programação que suportam o padrão, nível de complexidade de se aplicar, flexibilidade e reutilização.

Sendo assim, um padrão de projeto nomeia, abstrai e identifica aspectos-chave de um problema e cria uma solução abstrata, útil e reutilizável. Ele identifica classes, objetos, instancias participantes, seus papeis, colaborações e suas responsabilidades. Cada padrão de projeto é focado em resolver um problema em determinado nível do *software*, sendo assim, eles podem ser divididos em categorias como, Padrões de Criação, Padrões Estruturais e Padrões Comportamentais (GAMA, 2000).

1.3.1 Padrões de Criação

Segundo Gamma (2000), os padrões de criação servem para abstrair o processo de instanciação de um objeto, tornam uma aplicação independente de como os objetos são criados, representados e compostos. Podemos aplicar este padrão tanto na instanciação de classes como de objetos, uma classe usa a herança para variar a classe que será instanciada, já em nível de objeto, o mesmo usará outro objeto para sua criação. A necessidade de se utilizar um padrão de criação em um projeto se dá à medida que a aplicação evolui no sentido de depender mais da composição de objetos do que a herança de classes. Ou seja, quando se necessita de um número indeterminado de objetos com características e comportamentos semelhantes.

Padrões de projetos de criação catalogados por Gamma (2000):

- Abstract Factory: Fornece uma interface para criação de famílias de objetos relacionados.
- Builder: Separa a construção de um objeto complexo de sua representação, de modo que o mesmo processe a criação de um objeto, assim, criando vários objetos diferentes, mas com o mesmo processo.

- Factory Method: Define interfaces para criação de objetos, mas permite que as subclasses definam qual será a classe instanciada.
- Prototype: Especifica os tipos de objetos a serem criados usando uma instância prototípica e a partir disto cria novos objetos clonando os objetos existentes.
- Singleton: Garante que uma classe tenha somente uma instância e fornece um ponto global de acesso a mesma.

Sendo assim fica claro que os padrões de projeto de criação são aqueles que focam no processo de instanciação de objetos ocultando seu processo de instanciação (SANDERS, 2013).

1.3.2 Padrões Estruturais

Segundo SANDERS (2013), os padrões estruturais analisam como as classes e objetos se relacionam para formar estruturas maiores. No exemplo de uma classe a mesma pode mudar sua forma ao herdar duas ou mais classes, sendo assim, o mesmo acontece para os objetos criados das classes que obtiveram sua estrutura alterada por uma herança.

Padrões de projetos estruturais catalogados por Gamma (2000):

- Adapter : Converte a interface de uma classe em outra esperada pelo cliente, permitindo o trabalho em conjunto de classes que possuem interfaces diferentes.
- Bridge: Separa a abstração da implementação, onde as duas podem variar de acordo com a situação.
- Composite: Compõe seus objetos em estrutura de árvore, a fim de representar hierarquias, permitindo que os clientes tratem os objetos individuais e composições de objetos uniformemente.
- Decorator: Atribui responsabilidades adicionais a um objeto dinamicamente, fornecendo uma alternativa flexível para subclasses adicionarem funcionalidades.
- Façade: Disponibiliza uma interface para um conjunto de interfaces a um nível mais alto de abstração, tornando os subsistemas mais fáceis de usar.
- Flyweigth: Utiliza-se de compartilhamento a fim de suportar grandes quantidades de

objetos.

- Proxy: Fornece um objeto representante ou marcador de outro objeto, a fim de controlar o acesso ao mesmo.

Os padrões estruturais servem para lidar com a forma como as classes e objetos são compostos para formar estruturas maiores. Ao invés de simplesmente compor interfaces estaticamente como heranças simples, os padrões estruturais lhe fornecem flexibilidade para compor classes em tempo de execução. Sendo assim estes padrões servem para modificar as estruturas das classes e objetos em tempo de execução (GAMA, 2000).

1.3.3 Padrões Comportamentais

Os padrões comportamentais lidam com os algoritmos e a atribuição de responsabilidade entre os objetos. Eles não apenas tratam de classes e objetos mas também como a comunicação será realizada entre eles. Esses padrões servem para tratar fluxos complexos em tempo de execução, assim permitindo que o desenvolvedor se concentre em como os objetos serão interconectados (GAMMA, 2000).

Padrões de projetos comportamentais catalogados por Gamma (2000):

- Chain of Responsibility: Evita o acoplamento de uma requisição com um objeto, passando uma requisição a vários objetos até que um resolva a requisição.
- Command: Encapsula todas as requisições em objetos para parametrizar clientes diferentes, enfileirando as mesmas, criando logs ou até mesmo cancelando-as.
- Interpreter: Recebe uma determinada linguagem, define uma representação gramatical com um interpretador para reconhecer palavras da linguagem. É utilizado em gramáticas simples.
- Iterator: Fornece acesso sequencial a elementos de um conjunto de objetos sem expor sua representação subjacente.
- Mediator: Define um objeto que encapsula a forma como os objetos se interagem. Ele fornece um baixo nível de acoplamento, pois evita que os objetos se referenciem explicitamente uns aos outros, permitindo a variação de interações independentes.

- Memento: Permite tirar uma foto de um objeto em um determinado estado e voltar este estado quando necessário, tudo sem violar o encapsulamento.
- Observer: Cria uma dependência de objetos sendo a mesma de um-para-muitos, onde os objetos ficam em observação, quando um muda todos os outros são notificados e atualizados.
- State: Permite que um objeto altere seu comportamento quando seu estado interno muda, o mesmo pode mudar completamente seu comportamento passando a impressão de ter até mudado de classe.
- Strategy: Encapsula vários algoritmos, encapsula cada um deles e os torna intercambiáveis. Assim permitindo que o algoritmo varie independente dos clientes que o utilizam.
- Template Method: Define a estrutura de um algoritmo em tempo de execução, desprezando a definição de alguns passos para as subclasses, permitindo que as subclasses redefinam etapas de um algoritmo sem mudar a estrutura da mesma.
- Visitor: Representa uma operação a ser executada sobre os elementos da estrutura de um objeto. O mesmo permite definir novas operações sem a necessidade de alteração da classe à qual o objeto pertence.

Os padrões comportamentais tiram o foco das classes, objetos e suas composições, dando ênfase principalmente na forma em que eles se comunicam e interagem para executar uma requisição de um determinado cliente. Sendo assim, estes padrões são melhor compreendidos através de simulações de requisições completas (SANDERS, 2013).

1.4 HTTP

O HTTP é um protocolo utilizado para enviar e receber informações em rede. A versão mais utilizada atualmente é a 1.1, definida pela especificação RFC 2616. Burke (2010) afirma que o funcionamento do protocolo HTTP é muito simples. O cliente envia uma requisição ao servidor contendo informações sobre o método a ser utilizado, a localização do recurso ou método a ser invocado, e um conjunto de variáveis no cabeçalho da mensagem, além de um corpo de mensagem opcional que pode ser qualquer formato, tais como: HTML (HyperText

Markup Language), XML ou JSON, entre outros.

O protocolo HTTP basicamente consiste em requisições e respostas entre clientes e servidores. O cliente pode ser um navegador ou dispositivo que fará a requisição que pode ser chamado de *user agente*, que solicita um determinado recurso (*resource*), enviando um pacote de informações contendo alguns cabeçalhos (*headers*) a um URI ou, mais especificamente, URL. O servidor recebe estas informações e envia uma resposta, que pode ser um recurso ou um simplesmente um outro cabeçalho. A figura 1 ilustra uma requisição GET a URI http://google.com/.

```
GET  HTTP/1.1
Host: google.com
Content-Type: text/html; charset=iso-8859-1
Cache-Control: no-cache
Postman-Token: fa64bb09-a8e0-c730-7856-3ecd6b3a0fd4
```

FIGURA 1 – Funcionamento de uma requisição HTTP.
Fonte: Próprio autor.

Segundo esta requisição, o cabeçalho contém algumas informações que identificam nosso cliente (*host*), e qual o método da requisição (GET), o servidor identifica os itens do cabeçalho que lhe são convenientes e envia uma resposta. Neste exemplo o servidor retorna o cabeçalho de resposta, além de todo o conteúdo HTML da página inicial do Google. A figura 2 representa o retorno da requisição GET ao URI http://google.com/.

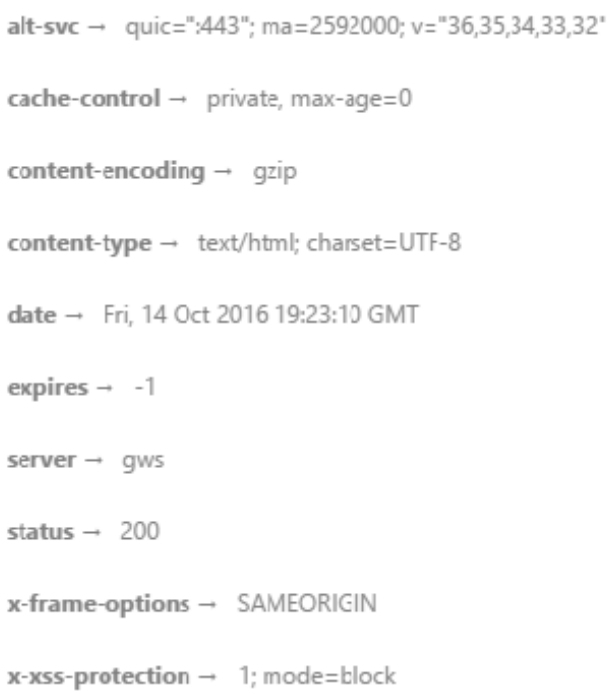

FIGURA 2 – Retorno de uma requisição HTTP.
Fonte: Próprio autor.

Nos cabeçalhos de resposta se pode obter algumas informações muito importantes, dentre elas o código de resposta (Status), que serve para identificar qual o estado da requisição, se obteve sucesso ou não.

O protocolo HTTP trabalha de maneira *stateless*, sendo assim, ele não guarda o estado entre as requisições, ou seja, entre uma requisição e outra ele perde qualquer referência existente. Para persistir informações você precisa utilizar *cookies*, sessões, campos de formulário ou variáveis na própria URI.

1.4.1 Métodos HTTP

Algumas operações fundamentais do HTTP são: GET, POST, DELETE, PUT, HEAD que são descritas a seguir:

- GET: Solicita um determinado recurso. É definido como um método seguro e não deve ser usado para disparar uma ação.
- POST: Envia informações para um recurso onde as informações são utilizadas para criação de um novo registro. Também pode ser utilizado para realizar o processamento de informações.
- DELETE: Solicita a remoção de algum recurso. Em seu cabeçalho especifica qual recurso deseja remover, o retorno da chamada deve ser status 204 caso não exista nenhum recurso.
- PUT: Atualiza um determinado recurso. Assim como o DELETE, ele deve especificar qual é o recurso, caso o recurso não exista, ele poderá criar um.
- HEAD: Retorna informações sobre um recurso. Na prática, funciona semelhante ao método GET, mas sem retornar o recurso no corpo da requisição. Também é considerado um método seguro.

1.4.2 Status HTTP

Toda requisição HTTP recebe um código de resposta que pode ser chamado de status. Com isso é possível identificar qual foi o resultado da requisição em questão.

Existem muitos status divididos em diversas categorias, na especificação você pode ver cada um deles com uma descrição bastante detalhada. A seguir, alguns códigos de status.

- 200 OK: A requisição foi bem sucedida.
- 301 Moved Permanently: O recurso foi movido permanentemente para outra URI.
- 302 Found: O recurso foi movido temporariamente para outra URI.
- 304 Not Modified: O recurso não foi alterado.
- 401 Unauthorized: A URI especificada exige autenticação do cliente. O cliente pode tentar fazer novas requisições.
- 403 Forbidden: O servidor entende a requisição, mas se recusa em atendê-la. O cliente não deve tentar fazer uma nova requisição.
- 404 Not Found: O servidor não encontrou nenhuma URI correspondente.
- 405 Method Not Allowed: O método especificado na requisição não é válido na URI.
- 410 Gone: O recurso solicitado está indisponível, seu endereço atual não é conhecido.
- 500 Internal Server Error: O servidor não foi capaz de concluir a requisição devido a um erro inesperado.
- 502 Bad Gateway: O servidor, enquanto agindo como proxy ou gateway, recebeu uma resposta inválida do servidor a que fez uma requisição.
- 503 Service Unavailable: O servidor não é capaz de processar a requisição, pois está temporariamente indisponível.

1.5 ARQUITETURA DE SOFTWARE

Arquitetura de *software* é uma área que lida com a estrutura dos componentes de dentro de um *software*, onde se englobam os relacionamentos entre qualquer parte do sistema, seus princípios e diretrizes, assim conduzindo o desenvolvimento do projeto e evolução do mesmo

(GARLAN; PERRY, 95).

A arquitetura de *software* é uma subárea da engenharia de *software*, que teve seu surgimento por volta da década de 1980, quando alguns pesquisadores e profissionais da época começaram a apontar a necessidade de se considerar o estado organizacional/arquitetural dos sistemas (SILVA, 2002).

Segundo Sommerville, (2011, p. 105) "O projeto de arquitetura é um processo criativo no qual você projeta uma organização de sistema para satisfazer aos requisitos funcionais de um sistema".

Mendes (2002) afirma que a arquitetura de *software* lida com alguns problemas, por exemplo:

- Seleção de alternativas de projeto;
- Escalabilidade e desempenho;
- Organização e estrutura geral de controle;
- Protocolos de comunicação, sincronização;
- Atribuição de funcionalidade a componentes de projeto.

Mendes (2002) ainda afirma que processos de engenharia de *software* requerem projeto arquitetural de *software*.

1.5.1 Web service

Web service ou "serviço *web"* são aplicações que disponibilizam serviços e funcionam do lado do servidor, servem como uma solução para integração de diferentes aplicações, assim possibilitando uma comunicação entre elas. Um *web service* pode ser construído em qualquer linguagem, assim como as aplicações que irão consumir seus serviços, pois ao se comunicarem eles trocam suas informações em um formato universal, como, por exemplo, o XML (*Extensible Markup Language*), JSON (DEITEL, 2010).

O funcionamento de um *web service* é descrito na Figura 1. Segundo Deitel, (2010, pag. 1019) "O aplicativo cliente envia uma solicitação por uma rede ao host do serviço Web, que processa a solicitação e retorna uma resposta pela rede ao aplicativo".

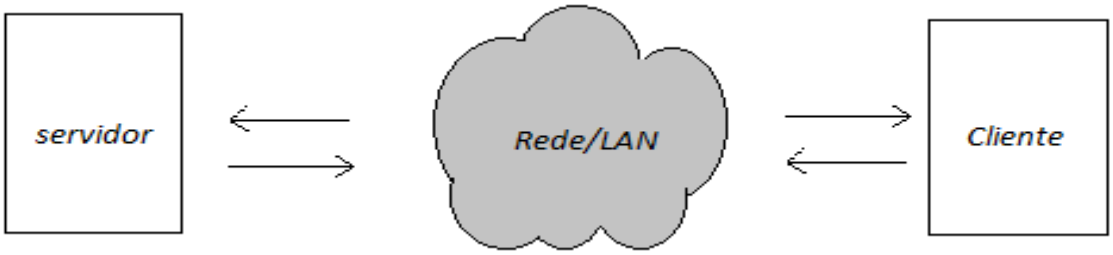

FIGURA 3 – Funcionamento de um *web service*.
Fonte: Próprio autor.

1.5.1.1 SOAP

SOAP é um protocolo que não depende de plataforma e que utiliza o XML (*Extensible Markup Language*), para sua comunicação com outras aplicações sobre o protocolo HTTP. Cada mensagem, tanto enviada como recebida, é empacotada em uma mensagem SOAP com especificações definidas por um WSDL, que é um padrão que utiliza marcações em XML para definir a estrutura da mensagem. Essas mensagens são escritas em XML por facilitar a leitura tanto para computadores como para humanos com conhecimentos em marcação XML, que por sua vez é bem simples, ainda com o benéfico de ser independente de plataforma. O protocolo utilizado é o HTTP pois o firewall, que são paredes de fogo ou barreira de segurança entre um computador e a rede, permitem o trafego HTTP com poucas limitações, o que possibilita enviar e receber mensagens SOAP por meio de conexões HTTP. (DEITEL, 2010).

O funcionamento básico de uma arquitetura SOAP é ilustrada na figura 2, onde o cliente (*Service client*) solicita um serviço (método) de um *web service* SOAP. A solicitação e todos os parâmetros são empacotados em mensagem SOAP e são enviadas ao envelope SOAP (SOAP *libraries* do lado do cliente). Quando está mensagem chega ao servidor (SOAP *libraries* do lado do servidor) ela é analisada pelo WSDL, que é encarregado de realizar as devidas validações da mensagem, onde as mesmas são especificadas em um XML. Passando pela validação, o conteúdo da mensagem é processado. No conteúdo estão especificados o método que o cliente deseja executar e os parâmetros para o método. Com isso o *web service* chama o método desejado e passa seus respectivos parâmetros, enviando a mensagem de resposta em outra mensagem SOAP. Por fim, o cliente analisa a resposta para recuperar os dados da mensagem (DEITEL, 2010).

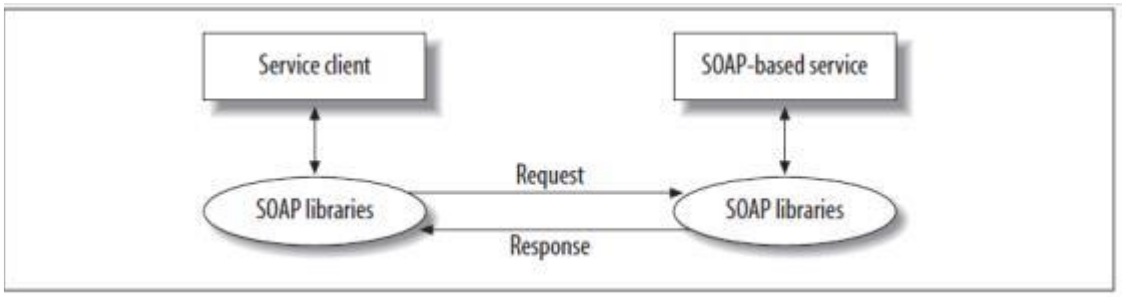

FIGURA 4 – Arquitetura SOAP.
Fonte: Java Web Services Up and Running, Martin Kalin.

1.5.1.2 REST

Segundo Fielding (2000) o REST é um estilo de arquitetura para sistemas de hipermídia distribuídos. Ou seja arquitetura criada para se trabalhar com protocolos HTTP.

REST é uma referência para um estilo arquitetônico de se desenvolver *web service*, onde o conjunto dos serviços criados é chamado de *web service ou* "serviço web". Os serviços criados são implementados seguindo o padrão *web*, por exemplo: todos os métodos implementados seguindo esta arquitetura são identificados por um URI único. Sempre que o servidor recebe uma solicitação o mesmo já sabe exatamente qual serviço deve executar. Além disso as chamadas aos serviços podem ser realizadas tanto em um aplicativo quanto por um navegador de internet. Após a execução destes serviços eles podem ser armazenados no cache dos navegadores, então, uma solicitação do tipo POST pode ter seus resultados armazenados e posteriormente reutilizado, deixando operações subsequentes mais rápidas, pois poderá carregar os dados direto do cache (DEITEL, 2010).

Algumas operações fundamentais do HTTP são: POST, DELETE, GET e UPDATE que servem respectivamente para salvar, deletar, obter e atualizar algum dado. Com os princípios da arquitetura REST é possível construir sistemas mais flexíveis, eficientes, com baixo acoplamento, e com isso ter um sistema que poderá ser facilmente reutilizado.

O REST é uma junção de vários modelos de arquitetura de rede, é um estilo híbrido que pode conter vários modelos arquitetônicos (FIELDING, 2000):

- Cliente/Servidor;
- Stateless;

- Interface Uniforme;
- Cache;
- Code on Demand.

1.5.1.2.1 Cliente/Servidor

Cliente/Servidor é um modelo que visa separar as responsabilidades e é comumente utilizado em sistemas que trabalhão em rede, principalmente os sistemas *Web*. Ele tende a separar a interface de usuário do armazenamento de dados, desta forma o servidor consegue ter um melhor gerenciamento de responsabilidade. No modelo cliente/servidor há um servidor responsável por oferecer um conjunto de serviços que podem ser invocados por aplicações clientes, o servidor disponibiliza seus serviços para serem consumidos por algum cliente independentemente da plataforma ou da linguagem escrita (FIELDING, 2010).

1.5.1.2.2 Stateless

O modelo arquitetônico *Stateless* consiste em não manter o estado entre o cliente e o servidor, assim gerando um desacoplamento entre as aplicações. Em cada requisição o cliente deve enviar todas as informações necessárias junto ao recurso para que esta possa ser processada pelo servidor, com isso o servidor tem uma diminuição de carga de processamento considerável já que os mesmos não precisam armazenar o estado de cada cliente. O estilo *Stateless* é baseado em três importantes conceitos segundo Fielding (2010), que são:

- Confiabilidade: Permite a recuperação de falhas parciais;
- Escalabilidade: Pelo servidor não precisar armazenar o estado de cada requisição, ele pode processar mais pedidos em um tempo menor;
- Visibilidade: Toda requisição contém todas as informações necessárias para que o servidor possa processá-la.

1.5.1.2.3 Interface Uniforme

Interface que define alguns princípios fundamentais como a identificação de recursos, diferentes representações, mensagens auto descritivas e recursos hipermídia (links) (FIELDING, 2010).

1.5.1.2.4 Cache

Permite fazer cache (local de armazenamento temporário) das informações para que não seja preciso o acesso ao banco de dados a todo o momento. É utilizado para otimização de desempenho do sistema (FIELDING, 2010).

1.5.1.2.5 Code on Demand

Código sob demanda é um paradigma de sistemas distribuídos que define a possibilidade e as técnicas de mover um código existente no servidor para a execução no cliente. Um exemplo disso são os códigos Java Script, o servidor pode dividir a carga de processamento com o cliente, o que evita a sobrecarga. Porém, o mesmo quebra alguns princípios de interoperabilidade, com isso, seu uso é opcional (Fielding, 2010).

1.5.1.2.6 Endereçamento

Segundo Burke (2010), o endereço URI representa uma forma de se acessar algum recurso único no sistema, ou seja, assim como recurso todo endereço URL necessita de ser

único. O endereço de um recurso disponível na rede se dá no formato de URI, sendo assim o mesmo é capaz de ser utilizado e manipulado por meio do protocolo HTTP.

1.5.1.2.7 Orientado a Representações

O REST trabalha orientado as representações, pois todo serviço pode utilizar formatos diferentes para troca de informações entre cliente-servidor, desde que o mesmo seja especificado no cabeçalho de requisição com o atributo Content-Type. Algumas das representações mais conhecidas são: json, xml e text.

1.5.1.3 RESTFul

Fielding (2000) descreveu os princípios que fazem da *Web* uma imensa aplicação distribuída, oferecendo diversos serviços e aplicativos através de um conjunto de arquiteturas bem definidas, sendo uma delas o REST. O termo *RESTful* é utilizado para denominar aplicações que seguem os princípios REST, aplicações que implementam seus conceitos.

2 METODOLOGIA

O presente trabalho teve o objetivo de identificar o quão importante é a utilização de padrões de projeto de *software* na construção de um *web service* baseado em *RESTful*, buscando obter um conhecimento aprofundando de quais são os padrões mais utilizados e os impactos de se utilizar padrões neste tipo de aplicação.

A princípio, para se realizar este trabalho foi feita uma pesquisa bibliográfica minuciosa sobre padrões de projeto de *software*, afim de identificar quais eram os padrões mais citados na literatura e suas principais vantagens. Após obter este conhecimento se deu sequência ao trabalho elaborando um questionário estruturado, objetivo e quantitativo com base nos padrões analisados. Sendo assim, o questionário foi disseminado em fóruns e grupos de desenvolvimento de *software* através da internet, a fim de conhecer melhor o perfil e experiência dos profissionais acerca deste assunto.

O questionário citado anteriormente se encontra no APÊNDICE A e será melhor detalhado nas subseções seguintes.

2.1 PÚBLICO ALVO

Como público respondente do questionário, foram escolhidos profissionais relacionados ao processo de produção de *software* e estudantes da área de Tecnologia da informação (TI), com o intuito de identificar o perfil atual dos mesmos e entender o ponto de vista destes em relação ao tema em questão, padrões de projetos de *software*.

Os entrevistados foram abordados para responder o questionário através de *e-mails*, divulgação em grupos e fóruns relacionados sobre desenvolvimento de *software*. O público que foi abordado via *e-mail*, foram especificamente ex-alunos do curso de Ciência da Computação, ministrado pela Faculdades Integradas de Caratinga e profissionais que atuam na área de desenvolvimento de *software*. Em todos meios de comunicação foram enviados um link para acesso ao questionário e uma mensagem que explicava o objetivo do questionário. Os grupos e fóruns citados se encontravam em redes sociais como, Facebook, LinkedIn e Google Plus.

2.2 ELABORAÇÃO DO QUESTIONÁRIO

As questões que compõem este questionário sobre padrões de projeto de *software* em *web service* baseado em arquitetura *RESTful*, foram elaboradas com base no conhecimento obtido através de pesquisas realizadas e uma análise minuciosa sobre os padrões de projeto catalogados em comum. Sendo assim, alguns padrões de projeto foram selecionados, afim de se comparar a sua eficiência em *web service RESTful*. Eles foram selecionados por categoria sendo elas: padrões de criação, padrões de estruturais e padrões comportamentais.

Dos padrões de criação foram encontrados os seguintes padrões em comum: Factory Method, Abstract Factory e Prototype. Para os padrões estruturais os seguintes foram selecionados: Adapter, Decorator e State. E por último foram selecionados os padrões comportamentais: Proxy, Strategy e Chain of Responsability.

Foram formuladas vinte e seis perguntas que foram organizadas em quatro grupos, onde buscaram ser descritas de forma mais simples possível, fornecendo um fluxo de leitura lógico e agradável para o respondente, assim proporcionando uma leitura menos cansativa. O questionário teve um total de seis páginas, a maioria das perguntas foi de caráter obrigatório, mas nem todas precisavam de ser respondidas pois algumas eliminavam perguntas posteriores no questionário.

2.3 O QUESTIONÁRIO

As vinte e seis questões do questionário foram organizadas em 4 grupos, sendo: Perfil do Entrevistado; Experiência com *web service* baseado em *RESTful*; Experiência com padrões de projeto; Relação entre web services baseados em *RESTful* e padrões de projeto. cada um deles teve o objetivo de coletar uma informação em especifico.

O primeiro grupo: Perfil do Entrevistado, buscava obter informações sobre o perfil do entrevistado. Suas perguntas visam obter dados com e-mail, nível de formação, função que o entrevistado executa atualmente e tempo em que trabalhava com o desenvolvimento de *software*.

O segundo grupo: Experiência com *web service* baseado em *RESTful*, serviu para obter informações sobre a experiência do usuário com *web service* baseado em *RESTful*. Suas perguntas obtiveram descobrir o número de envolvidos nos projetos para implementar esse tipo de tecnologia, media de sucesso na construção e implementação dessa tecnologia, media de sucesso na manutenção e escalabilidade dos *web service* implementados e linguagens de programação que foram utilizadas para construção do código-fonte.

O terceiro grupo: Experiência com padrões de projeto, busca entender qual era o nível de maturidade do entrevistado acerca de padrões de projetos, com perguntas como, em qual frequência o mesmo utilizava padrões de projetos na construção de *software*, a importância de se ter um arquitetura bem definida em um projeto, sobre os padrões que foram selecionados quais o mesmo obtinha conhecimento e dos padrões que o mesmo disse que obtinha conhecimento qual era a importância de se aplicá-los na construção de um *web service* baseado em *RESTful*.

O quarto grupo: Relação entre web service baseado em *RESTful* e padrões de projeto, teve o objetivo em coletar informações sobre opinião em relação *web service* baseado em *RESTful* e padrões de projeto, com perguntas como, a relação ao tempo de desenvolvimento de *web service* baseado em *RESTful* pode diminuir utilizando padrões de projeto e se custo com manutenções também poderia diminuir, assim podendo medir a eficiência dos padrões de projeto aplicados a esse tipo de tecnologia.

2.4 COLETA DE DADOS

O questionário foi elaborado com auxílio da ferramenta *Google Docs*, que permite a criação de formulários, compartilhamento *online*, onde o mesmo pode ser acessado por qualquer dispositivo que tenha acesso a internet e navegador *web*, além disso o mesmo disponibiliza acesso aos dados coletados por meio da própria ferramenta ou através da exportação dos dados.

A coleta dos dados da pesquisa durou cerca de quarenta e cinco dias. O questionário foi disponibilizado no dia 09 de setembro 2016 ás 00h00min, ficando disponível até o dia 23 de outubro de 2016 as 23h59m, neste período foram coletadas 47 respostas.

2.5 TRATAMENTO DE DADOS

As respostas obtidas através do formulário foram armazenadas automaticamente em uma planilha gerada pelo *Google Docs*, o que auxiliou na tabulação dos dados. A planilha gerada continha a reposta individual de cada respondente separadamente. Para a manipulação dos dados a planilha foi exportada para edição no *Microsoft Power BI,* onde os dados puderam ser organizados de acordo com o perfil do respondente, sendo assim, possibilitando a análise das repostas e a análise dos resultados através de gráficos com a porcentagem das respostas, possibilitando uma melhor compreensão dos resultados que serão apresentados na seção seguinte. Ao fim os resultados foram confrontados com os padrões de projeto de *software* selecionados, assim possibilitando aferir se os padrões catalogados pelos autores são de alta ou baixa influência no desenvolvimento de um *web service* baseado na arquitetura *RESTful*.

3 RESULTADOS

Nesta seção serão apresentados os resultados obtidos por meio das respostas dos 47 entrevistados que responderam ao questionário. Durante o processo de análise, nenhuma entrevista foi avaliada com duplicidade e uma outra pessoa respondeu o questionário de forma aleatória. Esse respondente afirmou não ter conhecimento em nenhuma linguagem de programação e nenhum conhecimento em REST, mesmo assim afirmou conhecer padrões de projeto, essa resposta foi desconsiderada no tratamento dos dados a seguir, portanto, todos os itens analisados não contêm as respostas do entrevistado em questão. Assim sendo, os resultados apresentados nessa seção são referentes aos 46 participantes restantes.

As respostas exibidas a seguir serão organizadas pelos quatro grupos do questionário, cada um em uma seção, sendo: Perfil do Entrevistado, Experiência com *web service* baseado em *RESTful*, Experiência com padrões de projeto e Relação *web service* baseados em *RESTful* e padrões de projeto.

3.1 PERFIL DO ENTREVISTADO

A entrevista contou com a resposta de 46 pessoas, sendo que todos trabalham diretamente com o processo de desenvolvimento de *software*, 36,96% do público de pessoas tem entre 1 e 3 anos de experiência na área, 23,91% tem mais de 5 anos de experiência, 23,91% tem entre 3 e 5 anos de experiência e 15,22% tem cerca de 1 ano de experiência.

O GRÁFICO 1 mostra a experiência dos entrevistados com desenvolvimento de *software*.

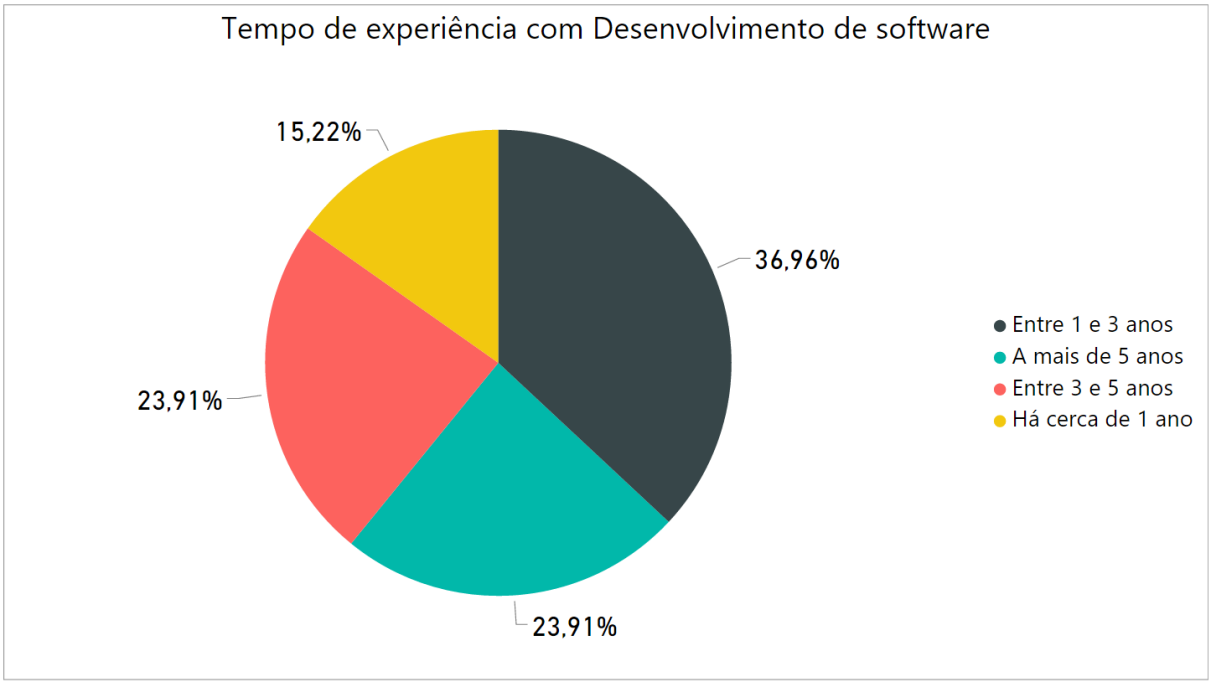

GRÁFICO 1 – Há quanto tempo o entrevistado trabalha com produção de *software*.

Fonte: Próprio autor.

Quando perguntados sobre a formação acadêmica, 71,74% dos entrevistados declararam formação de graduação em andamento ou finalizada, 17,39% são de pós-graduação em andamento ou finalizada, 6,52% estão com mestrado em andamento ou finalizado, 2,17% cursaram ou estão cursando doutorado e 2,17% estão cursando ou cursarão técnico.

O gráfico 2 mostra o nível de escolaridade dos entrevistados.

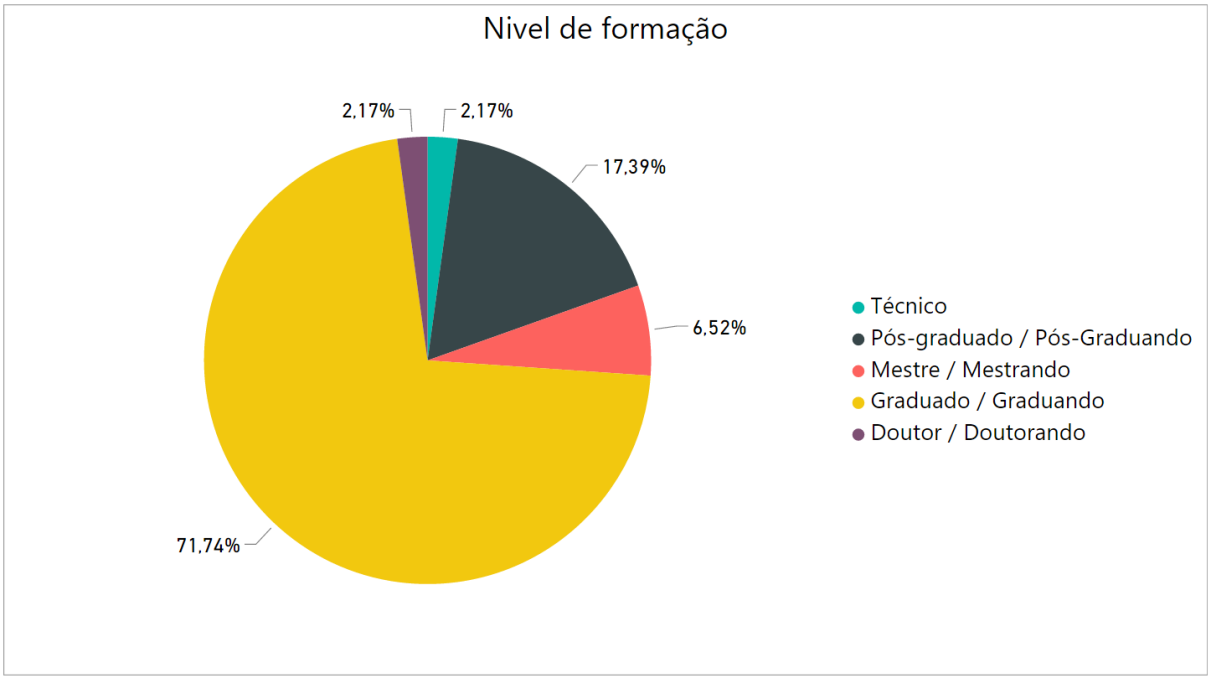

GRÁFICO 2 – Há quanto tempo o entrevistado trabalha com produção de *software*.

Fonte: Próprio autor.

Em relação às funções desempenhadas profissionalmente, 97,83% disseram desempenhar funções ligadas exclusivamente ao processo de desenvolvimento de *software*, sendo que 63,04% responderam ser Programador/Desenvolvedor, 15,22% são analistas de sistemas, 10,87% disseram desempenhar função de Engenheiro de *software*, 4,35% são Arquitetos de *software* e 4,35% responderão ser Gerente de Desenvolvimento. E cerca de 2,17%, disseram desempenhar outra função na área de Tecnologia da Informação (TI), Administrador de Redes.

O gráfico 3 exibe a função que é desempenhada pelos entrevistados.

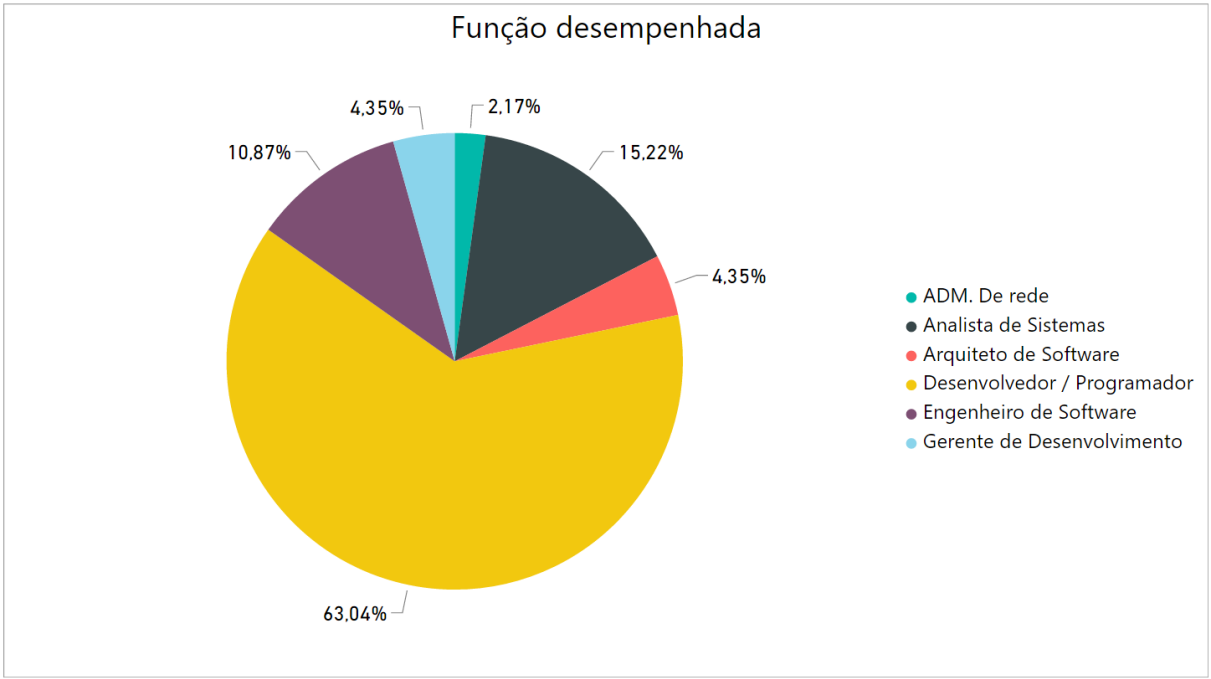

GRÁFICO 3 – Qual a função desempenhada.
Fonte: Próprio autor.

É perceptível notar que o público entrevistado na sua grande maioria são pessoas que atuam como desenvolvedores cerca 63.83%, onde 83.33% destes grupos de desenvolvedores então cursando ou cursaram graduação, e 50% deles tem experiência entre 1 e 3 anos.

3.2 EXPERIÊNCIA COM *WEB SERVICE* BASEADOS EM *RESTFUL*

As questões dessa seção têm por objetivo indagar os entrevistados a respeito de sua experiência com desenvolvimento de *web service* baseados na arquitetura *RESTful*, assim

buscando conhecer a quantidade de pessoas que compunham a equipe, linguagens de programação utilizada, media de sucesso no quesito de implementação de código-fonte, implantação e manutenção desse *software*.

Trabalha ou já trabalhou com *web service* baseados na arquitetura *RESTful*? Essa pergunta foi feita aos entrevistados com o intuito de identificar o nível de maturidade dos profissionais com esse tipo de tecnologia.

O gráfico 4 mostra o nível de maturidade dos entrevistados acerca do desenvolvimento de *web service* baseados na arquitetura *RESTful*.

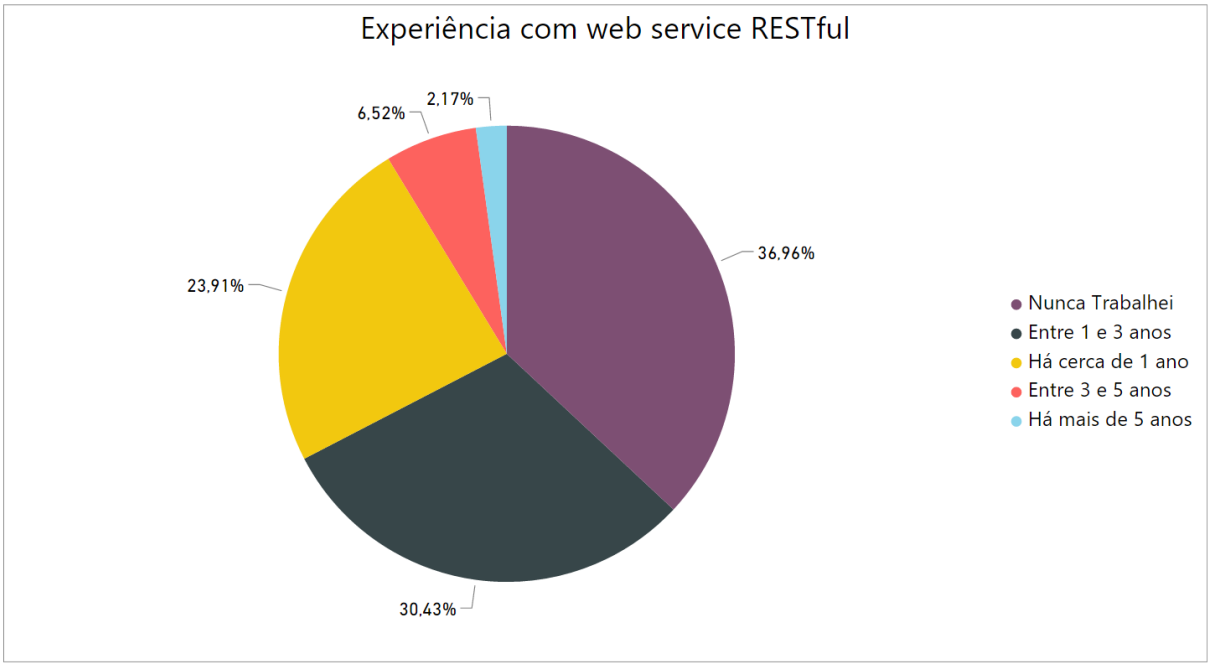

GRÁFICO 4 – Experiência com desenvolvimento de *web service* baseado na arquitetura *RESTful*.
Fonte: Próprio autor.

O gráfico mostra que maioria dos entrevistados já tiverem alguma experiência com a tecnologia *RESTful*, sendo que 36,96% nunca trabalharam com a mesma, sendo assim, a análise do restante das perguntas desta seção será realizada sobre o grupo restante, que é de 63,04%. Ainda sobre este gráfico a grande maioria, 30,43% se declarou com experiência entre 1 e 3 anos com uso da tecnologia, 23,91% disseram que possuem experiência de cerca de 1 ano, 6,52% disseram ter entre 3 e 5 anos de experiência e 2,17% disse ter mais de 5 anos de experiência.

Como foi dito anteriormente as questões descritas a seguir contam apenas com as respostas dos entrevistados que declaram ter experiência com *RESTful* (um total de 29 respondentes).

Nos projetos onde você teve de realizar a implementação de um *web service*, baseados na arquitetura *RESTful*, qual era a média de profissionais envolvidos com o desenvolvimento

do mesmo (gráfico 5). Essa pergunta ajudará nas seções seguintes a identificar se o número de profissionais envolvidos em um projeto é de alta influência na utilização de padrões de projeto de *software* ou não.

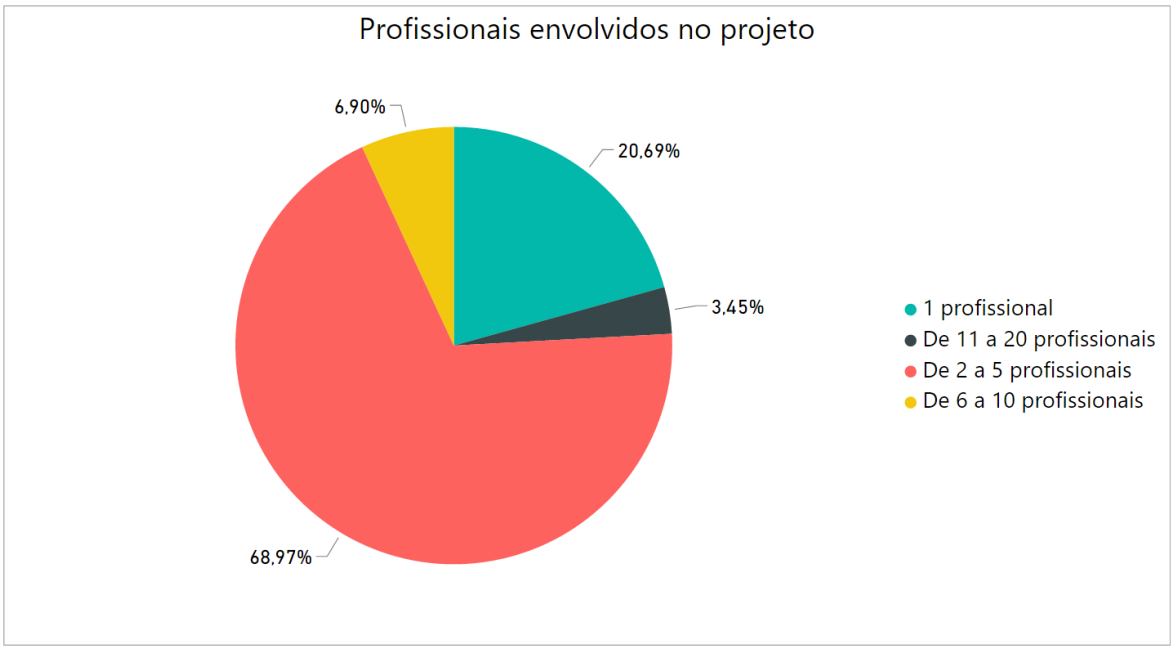

GRÁFICO 5 – Quantidade de profissionais envolvidos no projeto de *web service* baseado na arquitetura *RESTful*.
Fonte: Próprio autor.

Dentre os entrevistados, 68,97% responderão ter trabalhado com equipes de 2 a 5 profissionais, 20,69% disseram trabalhar sozinhos em seus projetos, 6,90% responderam trabalhar com equipe de 6 a 10 profissionais e 3,45% disseram trabalhar com equipe de 11 a 20 profissionais. Sendo assim a maioria dos respondentes declararam realizar seus projetos em equipe.

Os entrevistados também responderam sobre a média de sucesso no quesito de construção/implementação dos *web service* (gráfico 6). Com essa pergunta posteriormente será possível aferir se os padrões de projetos de *software* são de alta influência neste quesito.

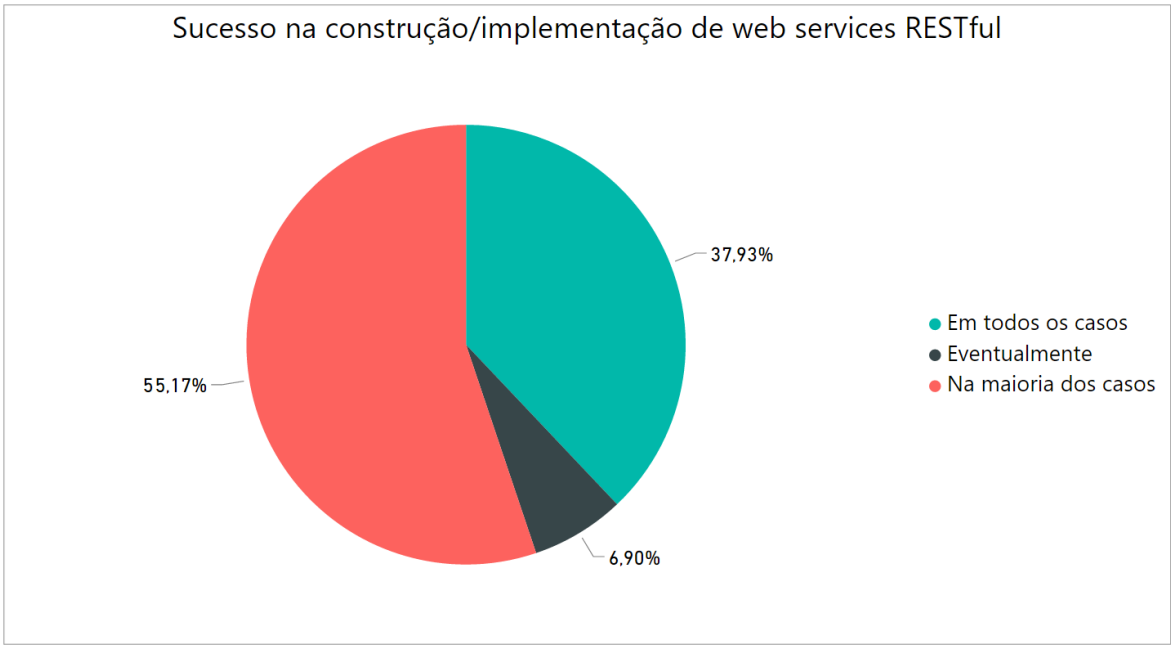

GRÁFICO 6 – Media de sucesso na construção/implementação de *web service* baseado na arquitetura *RESTful*.
Fonte: Próprio autor.

A grande maioria dos entrevistados 55,17% afirmou ter sucesso na construção/implementação de um *web service RESTful* na maioria dos casos, 37,93% disseram ter sucesso em todos os casos e apenas 6,90% responderem ter sucesso eventualmente em seus projetos neste quesito.

Os entrevistados responderam sobre a média de sucesso no quesito de implantação dos *web service* baseados na arquitetura REST (gráfico 7).

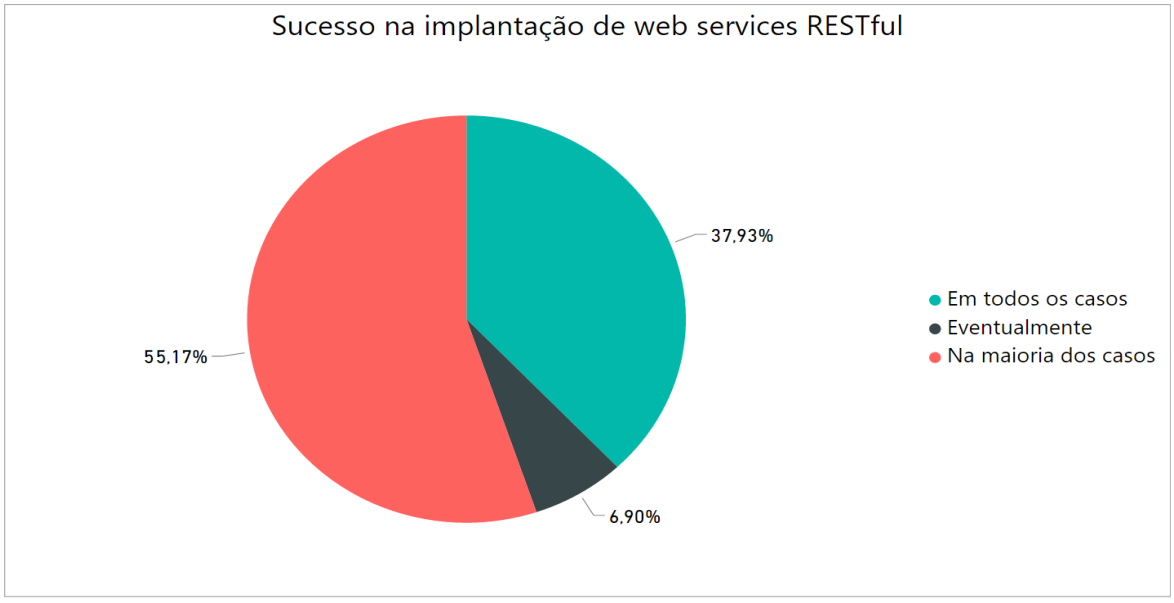

GRÁFICO 7 – Media de sucesso na implantação de *web service* baseado na arquitetura *RESTful*.
Fonte: Próprio autor.

Grande parte dos entrevistados afirmaram que conseguem obter sucesso na maioria dos casos na implantação dos *web servisse,* este grupo representa 55,17%, já 37,93% disseram obter sucesso em todos os casos e 6,90% afirmaram obter sucesso eventualmente no quesito implantação.

Por fim, os entrevistados responderam sobre a média de sucesso no quesito de manutenibilidade dos *web service* (gráfico 8), identificando a complexidade de se continuar evoluindo aplicações *RESTful*.

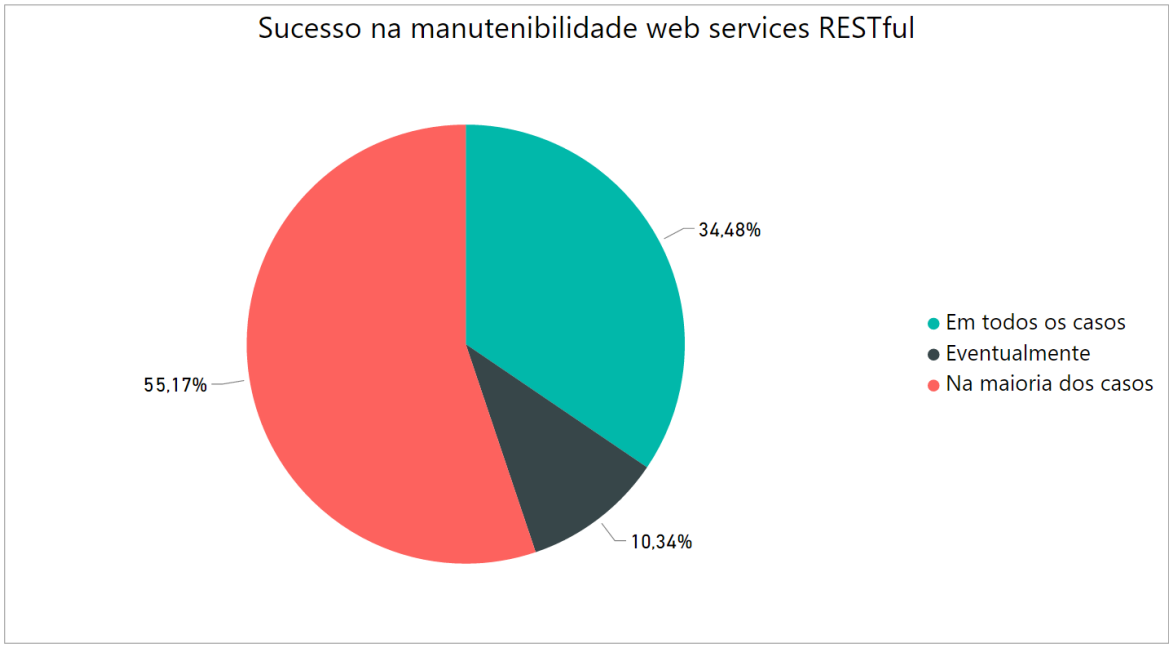

GRÁFICO 8 – Media de sucesso na manutenibilidade de *web service* baseado na arquitetura *RESTful*.
Fonte: Próprio autor.

Como podemos observar os entrevistados mantiveram quase que as mesmas respostas que nas duas perguntas anteriores, o que pode se disser que o funciona de forma continua desde a construção do *software*, dentre os entrevistados 55,17% afirmaram ter sucesso na maioria dos casos, 34,48% disseram ter sucesso em todos os casos e 10.34% afirmaram ter sucesso na manutenibilidade do *software* eventualmente.

Com intuito de identificar se as linguagens de programação que foram utilizadas para o desenvolvimento do *software* influenciam diretamente na utilização de padrões de projeto, a seguinte pergunta foi feita aos entrevistados: Quais linguagens foram utilizadas para implementação dos *web service*? O gráfico 9 mostra quais são as linguagens mais utilizadas para o desenvolvimento das aplicações.

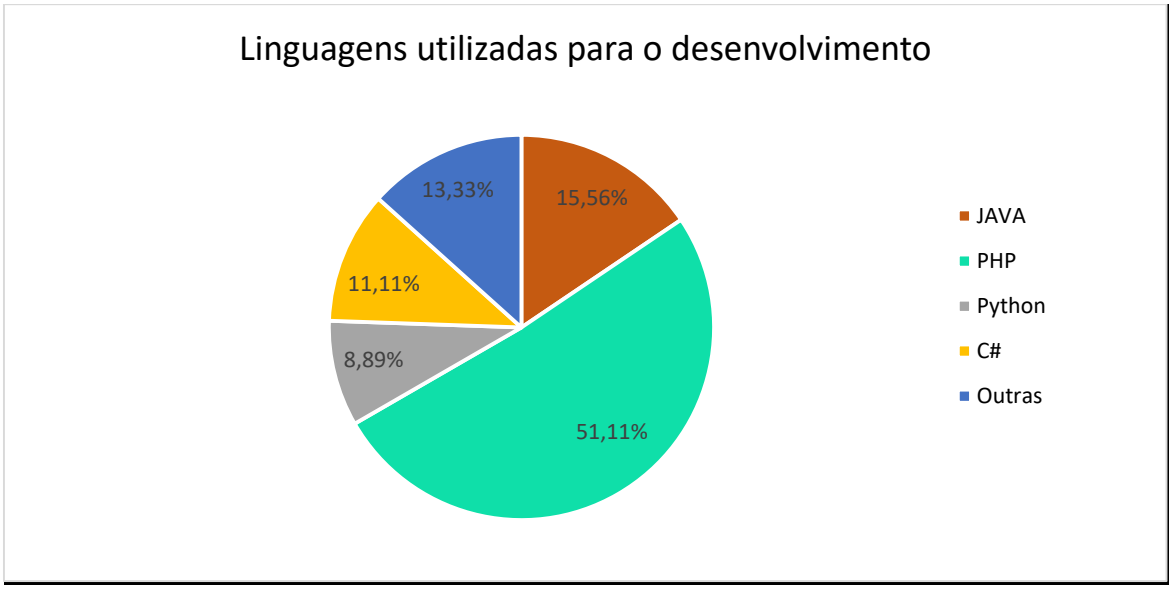

GRÁFICO 9 – Linguagens utilizadas para implementação dos *web service*.
Fonte: Próprio autor.

Do grupo dos entrevistados a linguagem de programação predominante com 51, 11% de utilização para o desenvolvimento é a PHP, em segundo temos o JAVA com 15,56%, seguido de outras com 13,33%, os entrevistados tinham a opção de colocar qual seria a outra linguem, mas nenhum dos entrevistados quis citar qual era a outra linguagem. A linguagem C# obteve um total de 11,11% de utilização e por fim Python com 8,89%.

3.3 EXPERIÊNCIA COM PADRÕES DE PROJETO

As questões desta seção têm por objetivo indagar os entrevistados a respeito da experiência com padrões de projeto de *software*, assim buscando entender se os padrões de projeto de *software* selecionados são realmente importantes para o desenvolvimento de *web service RESTful*. Nesta seção as porcentagens dos gráficos são relacionadas aos 46 entrevistados.

Você utiliza padrões de projeto com que frequência em seus projetos? Essa pergunta foi feita aos entrevistados com o intuito de identificar qual é a familiaridade deles com padrões de projeto de *software*.

O gráfico 10 mostra a frequência em que os entrevistados utilizam padrões de projeto de *software*.

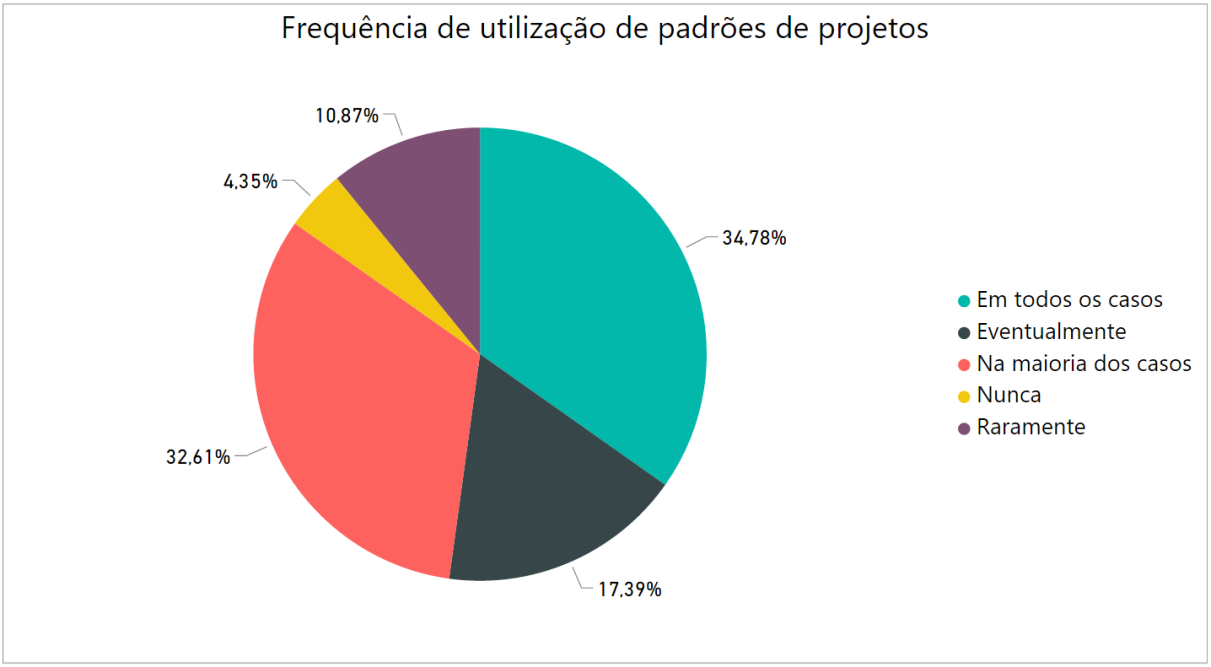

GRÁFICO 10 – Frequência da utilização de padrões de projeto na construção de *software*.
Fonte: Próprio autor.

Dos entrevistados cerca de 34,78% afirmaram utilizar padrões de projeto em todos os casos, 32,61% disseram utilizar na maioria dos casos, assim podemos observar que a maioria do público, cerca de 67,39% utilizam padrões de projeto com alta frequência no desenvolvimento de *software*. Temos também 17,39% dos entrevistos que afirmaram utilizar padrões eventualmente, 10,87% raramente e 4,35% nunca utilizam.

Para você, qual a importância de utilizar padrões de projeto na construção de um *web service*, baseados na arquitetura *RESTful*. Essa questão é fundamental para entender como os entrevistados veem a utilização de padrões de projeto no desenvolvimento deste tipo de aplicação.

O gráfico 11 exibe o grau de importância que os entrevistados apontaram para utilização de padrões de projeto no desenvolvimento de aplicações *RESTful*.

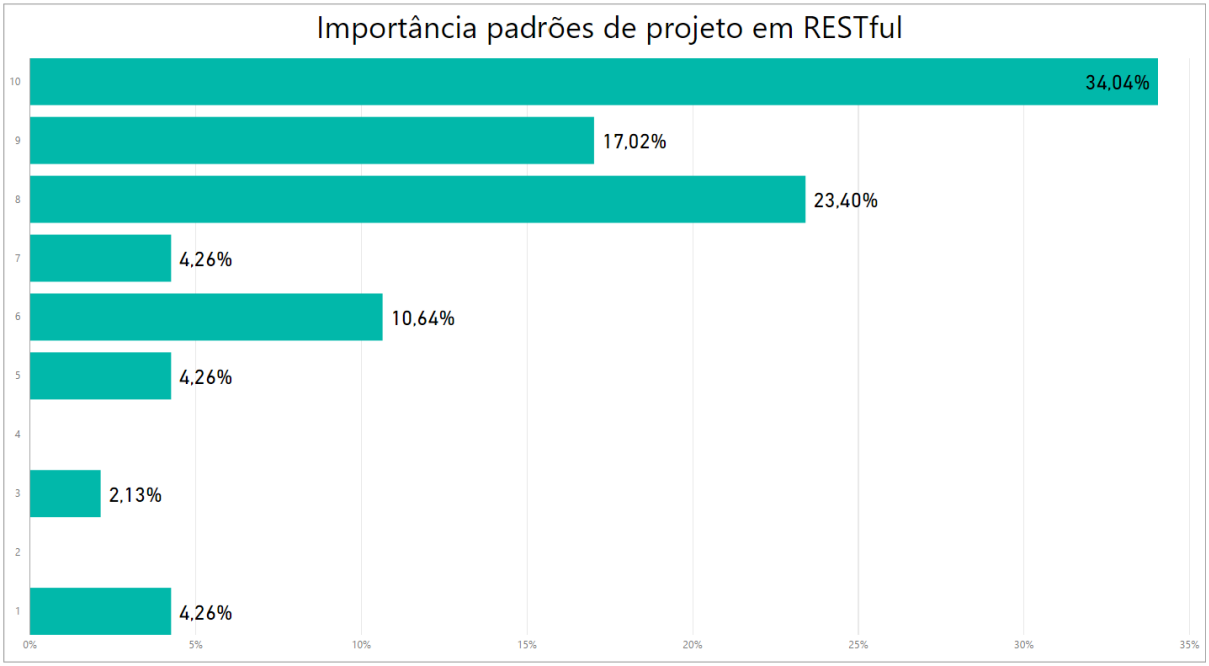

GRÁFICO 11 – Importância da utilização de padrões na construção de *web service RESTful*.
Fonte: Próprio autor.

Se avaliarmos em grupos de importância sendo eles de 1 a 3 baixa, 4 a 7 media e 8 a 10 alta, a maioria dos entrevistados, cerca de 74,46% avaliaram que a utilização de padrões de projeto é fundamental para criação de *web serve* REST, já 19,16% o segundo grupo que avaliou a importância como média, seguido do último grupo cerca 6,39% que jugou como baixa importância.

Para você qual a importância de ter uma arquitetura bem definidas para o desenvolvimento de *web service*, baseados na arquitetura *RESTful* (gráfico 12). Essa pergunta busca entender qual é o nível e organização arquitetural das aplicações que os entrevistados desenvolvem, afim de entender se a utilização de padrões de projeto influencia diretamente na construção de boas arquiteturas.

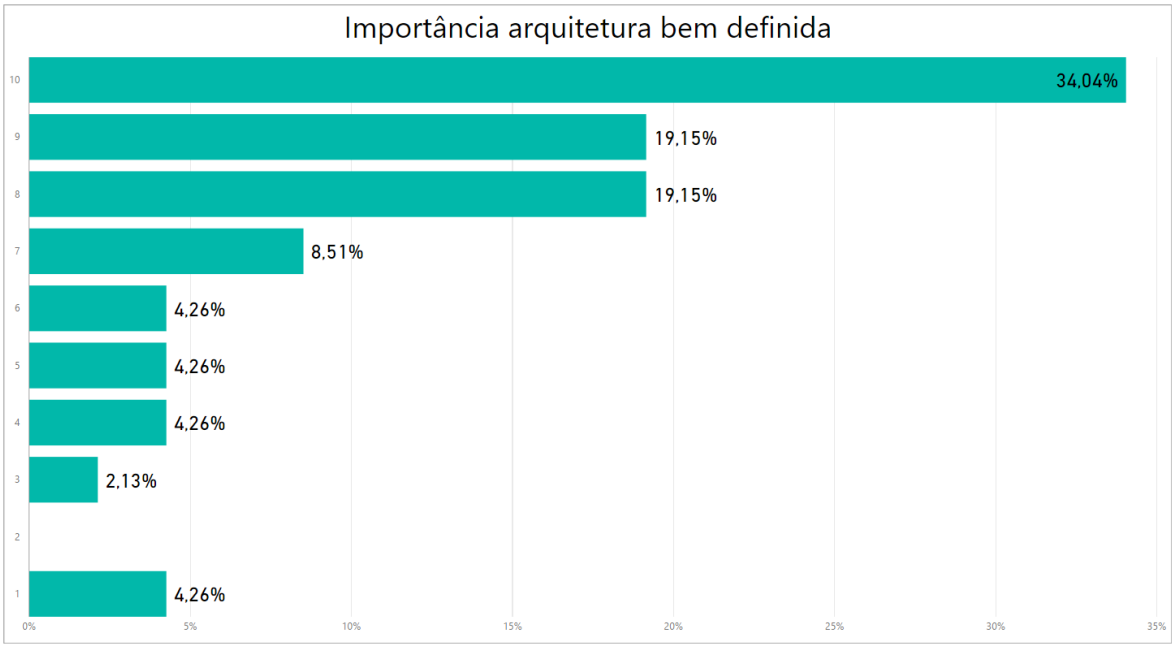

GRÁFICO 12 – Importância da utilização de padrões na construção de *web service RESTful*.
Fonte: Próprio autor.

Seguindo a avaliação em grupos de importância sendo eles de 1 a 3 baixa, 4 a 7 media e 8 a 10 alta, a maioria dos entrevistados, cerca de 72,34% avaliaram que uma arquitetura bem definida é fundamental para criação de *web serve* REST, já 20,93% o segundo grupo que avaliou a importância como média, seguido do último grupo cerca 6,39% que jugou como baixa importância.

É perceptível que a utilização de padrões de projeto podem influenciar diretamente no desenvolvimento de *web service RESTful* com uma boa arquitetura definida, pois ao analisarmos os gráficos cuidadosamente percebemos que em ambos os gráficos 11 e 12, o grupo que avalia como alta a utilização de padrões de projeto e arquitetura bem definida para aplicação foram a grande maioria.

Dos padrões de projetos listados, em quais você tem conhecimento? Essa pergunta foi feita aos entrevistados com o intuito de identificar qual era o conhecimento deles acerca dos padrões de projeto selecionados.

O gráfico 13 exibe quais são os padrões de projeto mais conhecido pelos entrevistados.

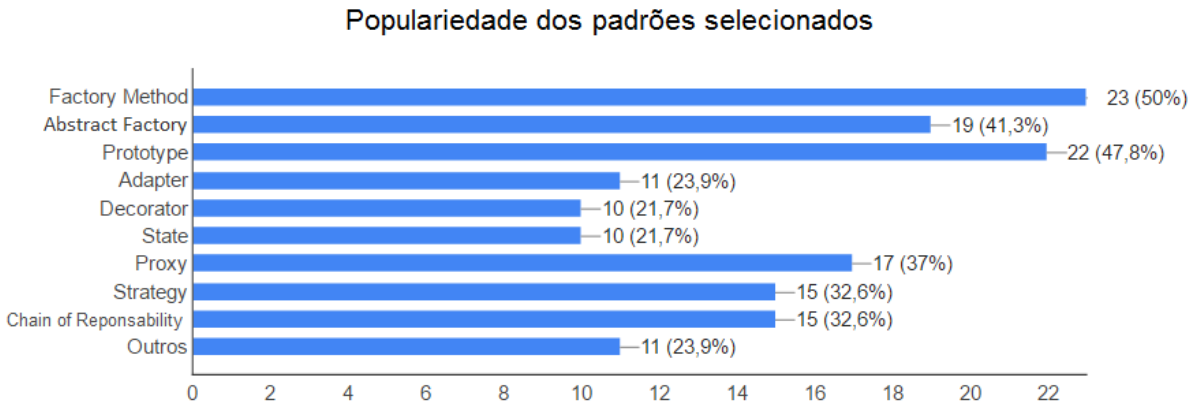

GRÁFICO 13 – Padrões de projetos mais conhecidos entre os entrevistados.
Fonte: Próprio autor.

Do grupo de entrevistados 50,00% conhecem o padrão de projeto Factory Method, seguido do Abstrac Factory com 41,3%, Prototype 47,8%, Adapter 23,9%, Decorator 21,7%, State 21,7%, Proxy 37,7%, Streategy 32,6%, Chain of Responsability 32,6% e por fim outros padrões com 23,9%, a opção outros contendo padrões como Mediator, Composite, Interpreter, Singleton e Service.

Se analisarmos por categorias é perceptível notar que dos padrões estruturais o mais conhecido pelos entrevistados é o Proxy, dos padrões comportamentais temos um empate entre o Chain of Responsability e Strategy, e por fim, do grupo dos padrões de criação, podemos observar que eles são os mais conhecidos entre os entrevistados sendo o mais conhecido deles o Factory Method e o menos conhecido o Abstract Factory.

As próximas questões são relacionadas aos padrões de projeto selecionados com o intuito de apontar a proficiência de utilizá-los em *web service RESTful*. Assim como algumas questões anteriores as próximas serão analisadas e dividas em três grupos de importância sendo eles de 1 a 3 baixa, 4 a 7 media e 8 a 10 alta. Nestas questões todos 46 entrevistados responderam, pois, as questões se basearam nas opiniões sobre o assunto.

Para você qual é a importância de se aplicar o padrão Factory Method em um web *service baseado* na arquitetura *RESTful*? Dos entrevistados 72,34% consideram que a utilização deste padrão de projeto é de alta importância, 21,29% declaram que a importância é media e já 4,26% afirmaram ser de baixa importância.

O gráfico 14 exibe a opinião dos entrevistados que já desenvolveram *web service RESTful* sobre a importância da utilização do Factory Method.

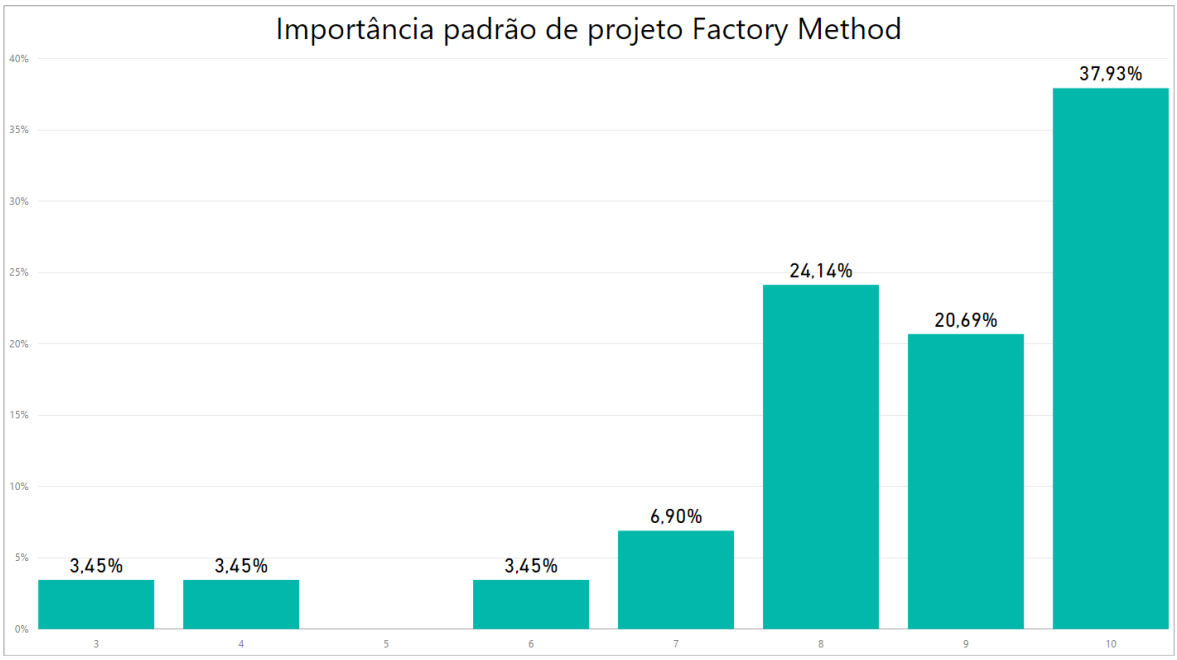

GRÁFICO 14 – Grau de importância da aplicação do padrão Factory Method segundo entrevistados que tem experiência com o desenvolvimento de *web service RESTful*.
Fonte: Próprio autor.

Neste gráfico foi realizado um filtro onde foram selecionados somente os entrevistados que declaram ter tido alguma experiência com *web service RESTful* Gráfico 4, com esse filtro foi possível identificar não somente a opinião do entrevistado mas como também a experiência dele acerca do padrão. Cerca de 82,76% declaram que a utilização do padrão de projeto Factory Method é de importância alta, 13,8% afirmaram ser de importância média e apena 3,45% disseram ser de baixa importância a utilização do padrão. Podemos notar que as repostas tanto dos entrevistados que não tem experiência com *web service RESTful* quanto os que tem experiência foram muito parecidas sobre este padrão de projeto.

Para você qual é a importância de se aplicar o padrão Abstract Factory em um web *service baseado* na arquitetura *RESTful*? Dos entrevistados 51,35% consideram que a utilização deste padrão de projeto é de alta importância, 35,14% declaram que a importância é media e já 13,51% afirmaram ser de baixa importância.

O gráfico 15 exibe a opinião dos entrevistados que já desenvolveram *web service RESTful* sobre a importância da utilização do Abstract Factory.

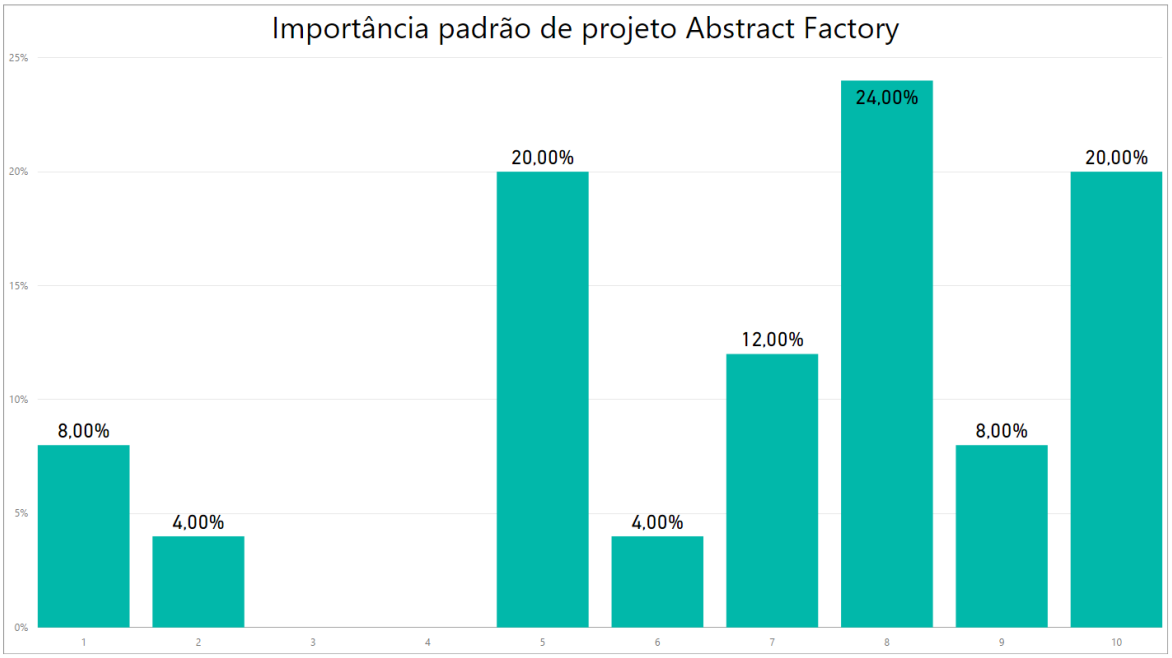

GRÁFICO 15 – Grau de importância da aplicação do padrão Abstract Factory segundo entrevistados que tem experiência com o desenvolvimento de *web service RESTful*.
Fonte: Próprio autor.

Como no gráfico anterior também foi realizado um filtro onde foram selecionados somente os entrevistados que declaram ter tido alguma experiência com *web service RESTful* Gráfico 4, sendo assim cerca de 52,00% declaram que a utilização do padrão de projeto Abstract Factory é de alta importância, 36,00% afirmaram ser de importância média e apena 3,45% disseram ser de baixa importância a utilização do padrão. Podemos notar que as repostas tanto dos entrevistados que não tem experiência com *web service RESTful* quanto os que tem experiência foram muito parecidas sobre este padrão de projeto assim como no caso do padrão Factory Method.

Para você qual é a importância de se aplicar o padrão Prototype em um *web service* baseado na arquitetura *RESTful*? Dos entrevistados 42,11% consideram que a utilização deste padrão de projeto é de alta importância, 42,10% declaram que a importância é media e já 15,79% afirmaram ser de baixa importância.

O gráfico 16 exibe a opinião dos entrevistados que já desenvolveram *web service RESTful* sobre a importância da utilização do Prototype.

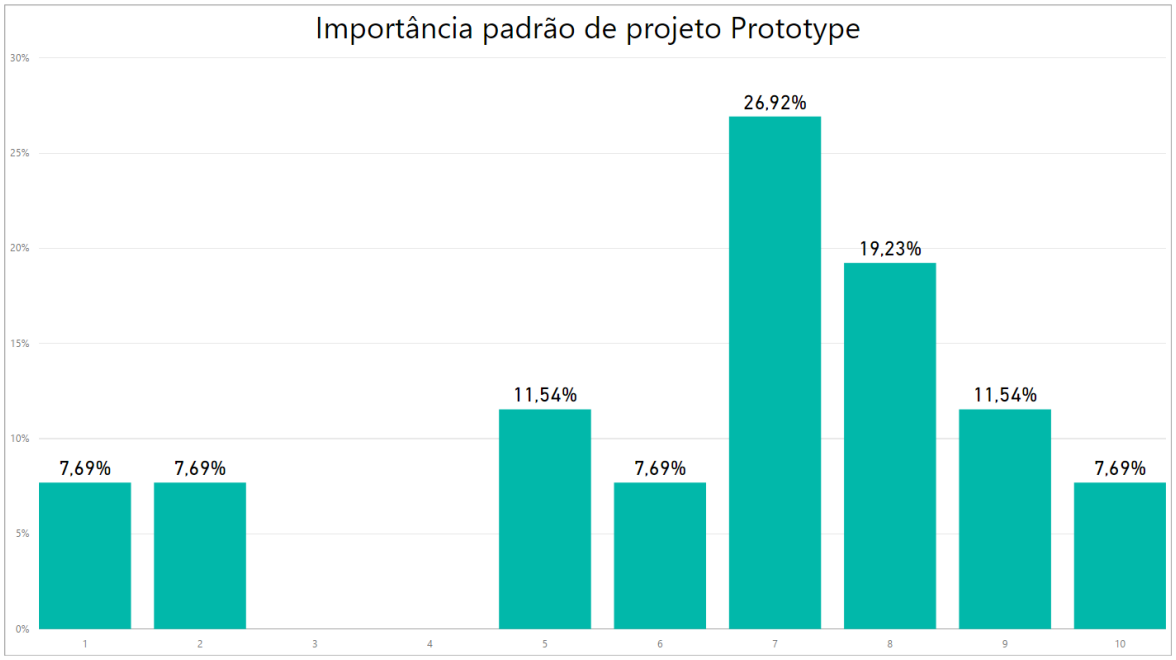

GRÁFICO 16 – Grau de importância da aplicação do padrão Prototype segundo entrevistados que tem experiência com o desenvolvimento de *web service RESTful*.
Fonte: Próprio autor.

Sendo assim 38,45% declaram que a utilização do padrão de projeto Prototype é de alta importância, 46,16% afirmaram ser de importância média e 15,38% disseram ser de baixa importância a utilização do padrão. Assim como nos padrões anteriores podemos notar que as repostas tanto dos entrevistados que não tem experiência com *web service RESTful* quanto os que tem experiência foram muito parecidas sobre este padrão de projeto.

Para você qual é a importância de se aplicar o padrão Adapter em um *web service* baseado na arquitetura *RESTful*? Dos entrevistados 57,14% consideram que a utilização deste padrão de projeto é de alta importância, 31,43% declaram que a importância é media e já 11,42% afirmaram ser de baixa importância.

O gráfico 17 exibe a opinião dos entrevistados que já desenvolveram *web service RESTful* sobre a importância da utilização do Adapter.

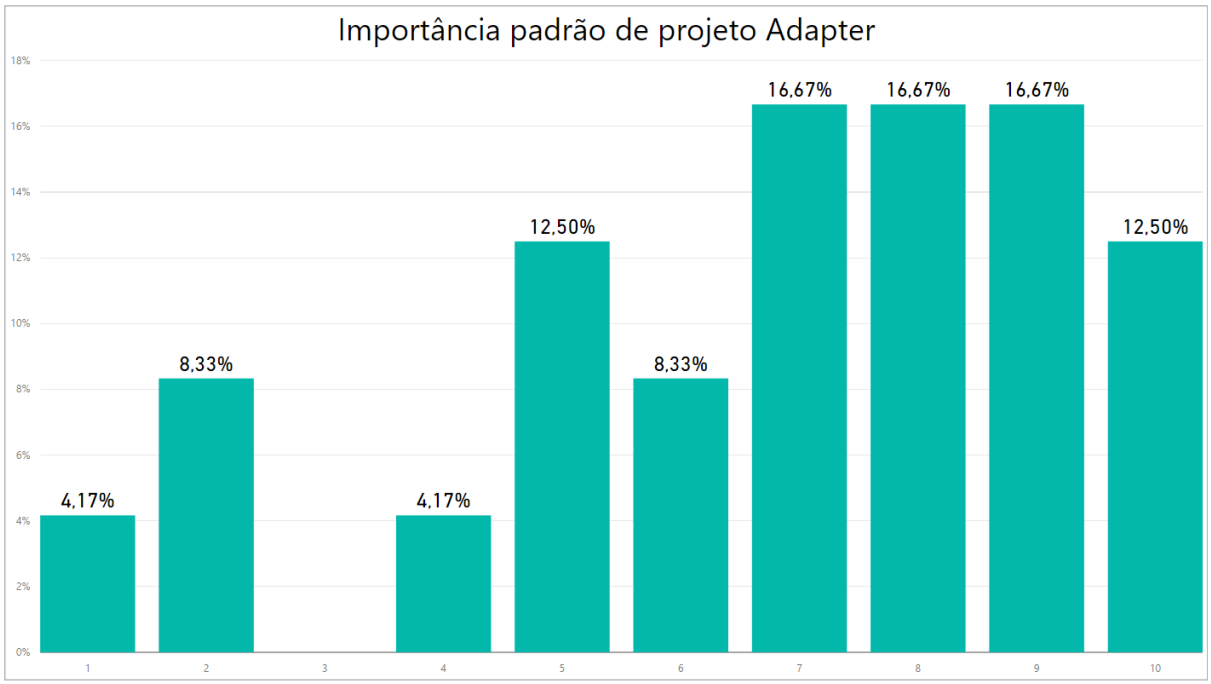

GRÁFICO 17 – Grau de importância da aplicação do padrão Adapter segundo entrevistados que tem experiência com o desenvolvimento de *web service RESTful*.
Fonte: Próprio autor.

Cerca de 45,84% declaram que a utilização do padrão de projeto Adapter é de importância alta, 41,67% afirmaram ser de importância média e 12,50% disseram ser de baixa importância a utilização do padrão. Podemos notar que as repostas dos entrevistados que não tem experiência com *web service RESTful* foram um pouco diferentes quanto aos entrevistados que tem experiência sobre o padrão Adapter, entretanto nos dois grupos a maioria destacou a utilização deste padrão como tendo alta importância a sua aplicabilidade.

Para você qual é a importância de se aplicar o padrão Decorator em um *web service* baseado na arquitetura *RESTful*? Dos entrevistados 33,33% consideram que a utilização deste padrão de projeto é de alta importância, 50,1% declaram que a importância é media e já 16,67% afirmaram ser de baixa importância.

O gráfico 18 exibe a opinião dos entrevistados que já desenvolveram *web service RESTful* sobre a importância da utilização do Decorator.

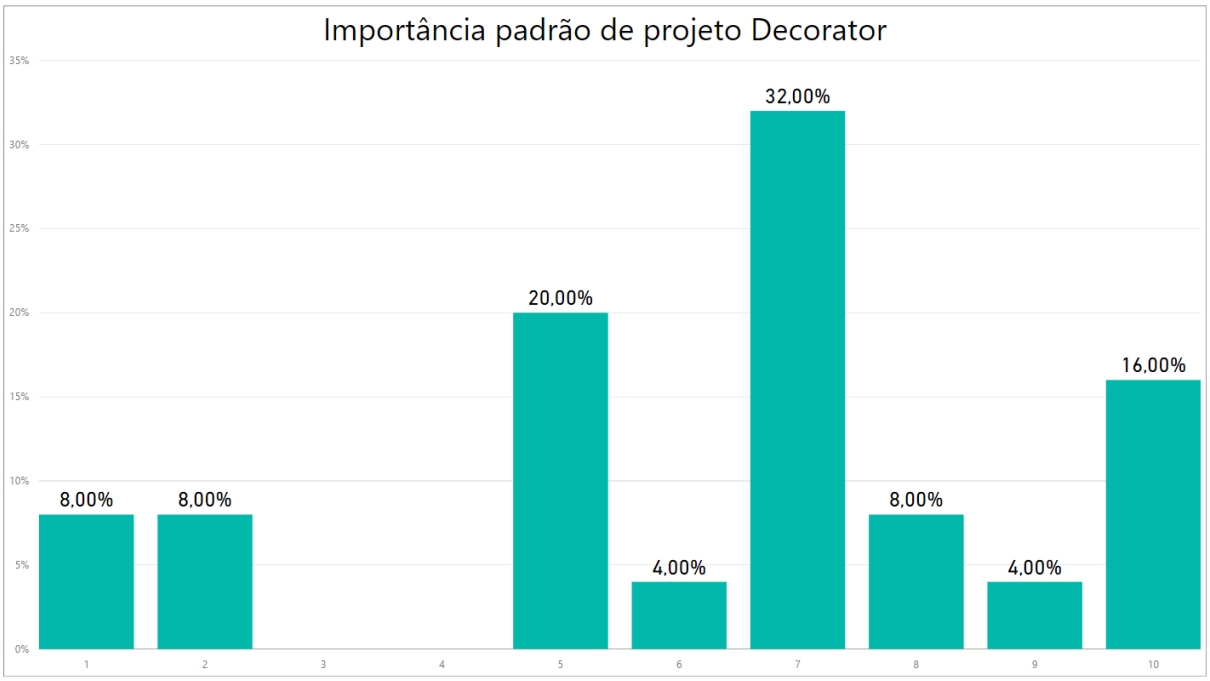

GRÁFICO 18 – Grau de importância da aplicação do padrão Decorator segundo entrevistados que tem experiência com o desenvolvimento de *web service RESTful*.
Fonte: Próprio autor.

Cerca de 28,00% declaram que a utilização do padrão de projeto Decorator é de importância alta, 56,00% afirmaram ser de importância média e 16,00% disseram ser de baixa importância a utilização do padrão. Podemos notar que as repostas tanto dos entrevistados que não tem experiência com *web service RESTful* quanto os que tem experiência foram muito parecidas sobre este padrão de projeto.

Para você qual é a importância de se aplicar o padrão State em um *web service* baseado na arquitetura *RESTful*? Dos entrevistados 47,23% consideram que a utilização deste padrão de projeto é de alta importância, 36,11% declaram que a importância é media e já 16,67% afirmaram ser de baixa importância.

O gráfico 19 exibe a opinião dos entrevistados que já desenvolveram *web service RESTful* sobre a importância da utilização do State.

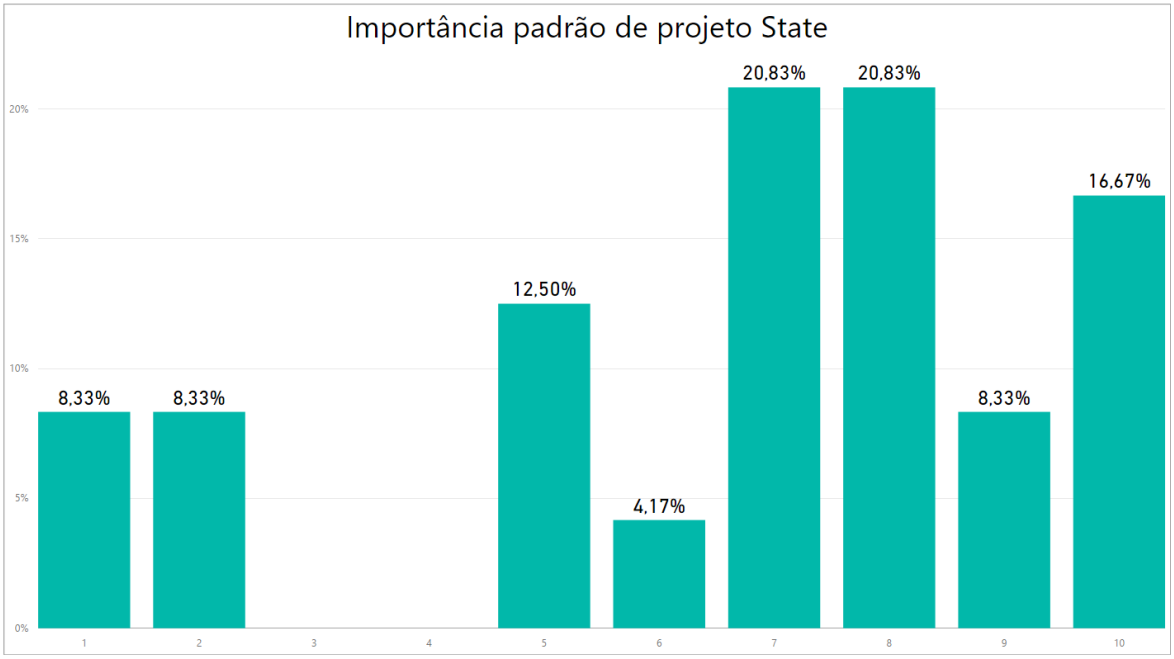

GRÁFICO 19 – Grau de importância da aplicação do padrão State segundo entrevistados que tem experiência com o desenvolvimento de *web service RESTful*.
Fonte: Próprio autor.

Tendo em vista os entrevistados com experiência no desenvolvimento de *web service RESTful* 45,83% destes declaram que a utilização do padrão de projeto State é de importância alta, 37,50% afirmaram ser de importância média e 16,66% disseram ser de baixa importância a utilização do padrão. Podemos notar que as repostas tanto dos entrevistados que não tem experiência com *web service RESTful* quanto os que tem experiência foram muito parecidas sobre este padrão de projeto assim como no caso dos padrões anteriores.

Para você qual é a importância de se aplicar o padrão Proxy em um *web service* baseado na arquitetura *RESTful*? Dos entrevistados 47,23 consideram que a utilização deste padrão de projeto é de alta importância, 36,11% declaram que a importância é media e já 16,67% afirmaram ser de baixa importância.

O gráfico 20 exibe a opinião dos entrevistados que já desenvolveram *web service RESTful* sobre a importância da utilização do Proxy.

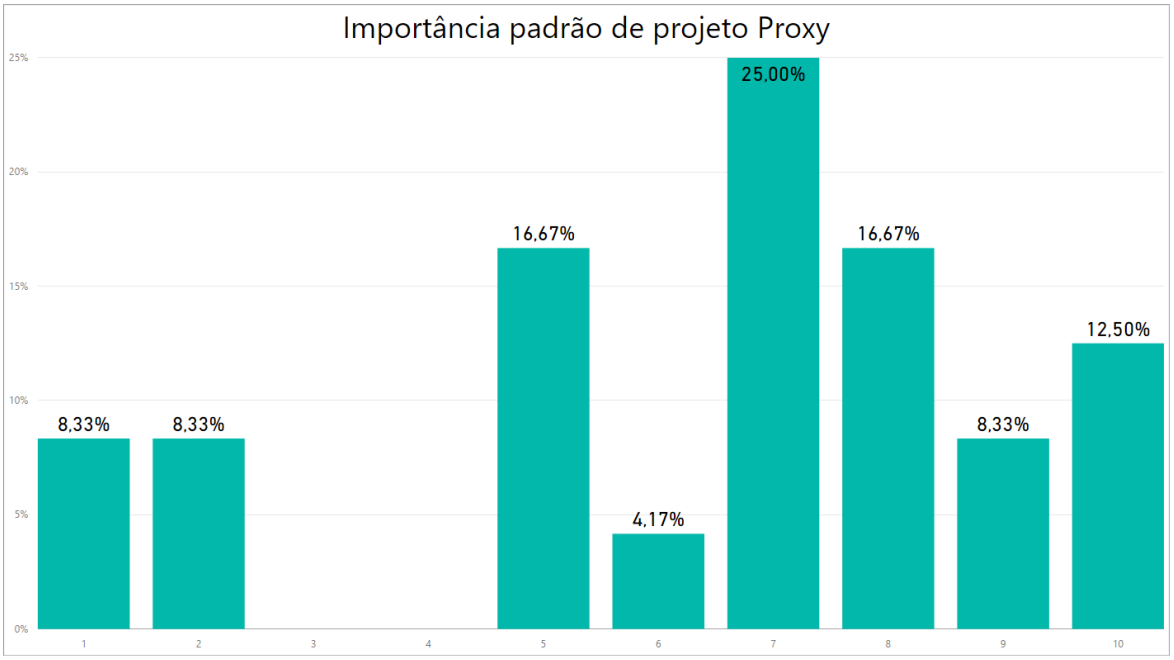

GRÁFICO 20 – Grau de importância da aplicação do padrão Proxy segundo entrevistados que tem experiência com o desenvolvimento de *web service RESTful*.
Fonte: Próprio autor.

Dos entrevistados com experiência no desenvolvimento de *web service RESTful* 37,50% declaram que a utilização do padrão de projeto Proxy é de importância alta, 45,84% afirmaram ser de importância média e 16,66% disseram ser de baixa importância a utilização do padrão. Podemos notar que as repostas dos entrevistados que com experiência com a tecnologia foi um pouco diferente onde a maioria jugou como mediana a importância do padrão de projeto Proxy.

Para você qual é a importância de se aplicar o padrão Strategy em um *web service* baseado na arquitetura *RESTful*? Dos entrevistados 41,66 consideram que a utilização deste padrão de projeto é de alta importância, 41,66% declaram que a importância é media e já 16,67% afirmaram ser de baixa importância.

O gráfico 21 exibe a opinião dos entrevistados que já desenvolveram *web service RESTful* sobre a importância da utilização do Strategy.

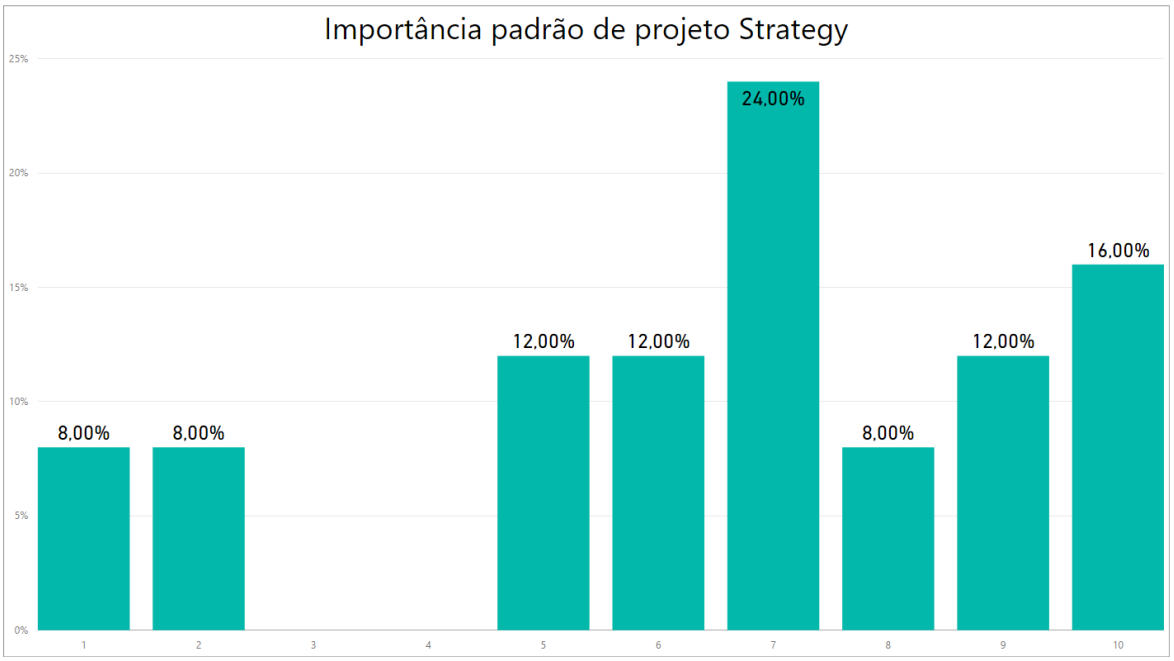

GRÁFICO 21 – Grau de importância da aplicação do padrão Strategy segundo entrevistados que tem experiência com o desenvolvimento de *web service RESTful*.
Fonte: Próprio autor.

Dos entrevistados com experiência no desenvolvimento de *web service RESTful* 36,00% declaram que a utilização do padrão de projeto Strategy é de importância alta, 48,00% afirmaram ser de importância média e 16,00% disseram ser de baixa importância a utilização do padrão. Podemos notar que as repostas dos entrevistados que com experiência com a tecnologia foi um pouco diferente onde a maioria jugou como mediana a importância do padrão de projeto Strategy.

Para você qual é a importância de se aplicar o padrão Chain of Responsibility em um *web service* baseado na arquitetura *RESTful*? Dos entrevistados 45,94% consideram que a utilização deste padrão de projeto é de alta importância, 40,55% declaram que a importância é média e já 13,52% afirmaram ser de baixa importância.

O gráfico 22 exibe a opinião dos entrevistados que já desenvolveram *web service RESTful* sobre a importância da utilização do Chain of Responsibility.

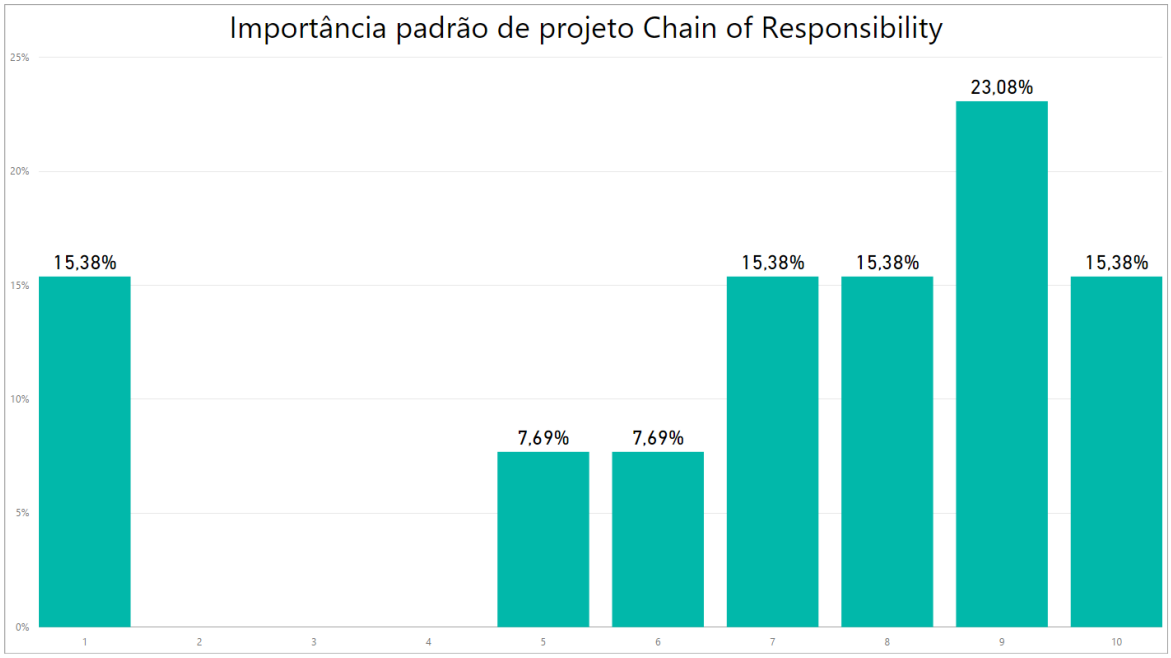

GRÁFICO 22 – Grau de importância da aplicação do padrão Chain of Responsibility segundo entrevistados que tem experiência com o desenvolvimento de *web service RESTful*.
Fonte: Próprio autor.

Dos entrevistados com experiência no desenvolvimento de *web service RESTful* 53,84% declaram que a utilização do padrão de projeto Strategy é de importância alta, 30,76% afirmaram ser de importância média e apenas 15,38% disseram ser de baixa importância a utilização do padrão. Neste gráfico podemos observar que a maioria dos entrevistados com experiência com a tecnologia jugou como alta a importância do padrão de projeto Chain of Responsibility.

Com qual frequência você utiliza frameworks para o desenvolvimento de *web service* baseados em *RESTful* (gráfico 23)? Com essa pergunta foi possível identificar se os entrevistados utilizam padrões de projeto implicitamente, pois em grande parte dos casos os *frameworks* são recheados de padrões de projetos em suas implementações e essas camadas são totalmente ocultadas, o que dá a sensação aos usuários de que não utilização padrões de projetos.

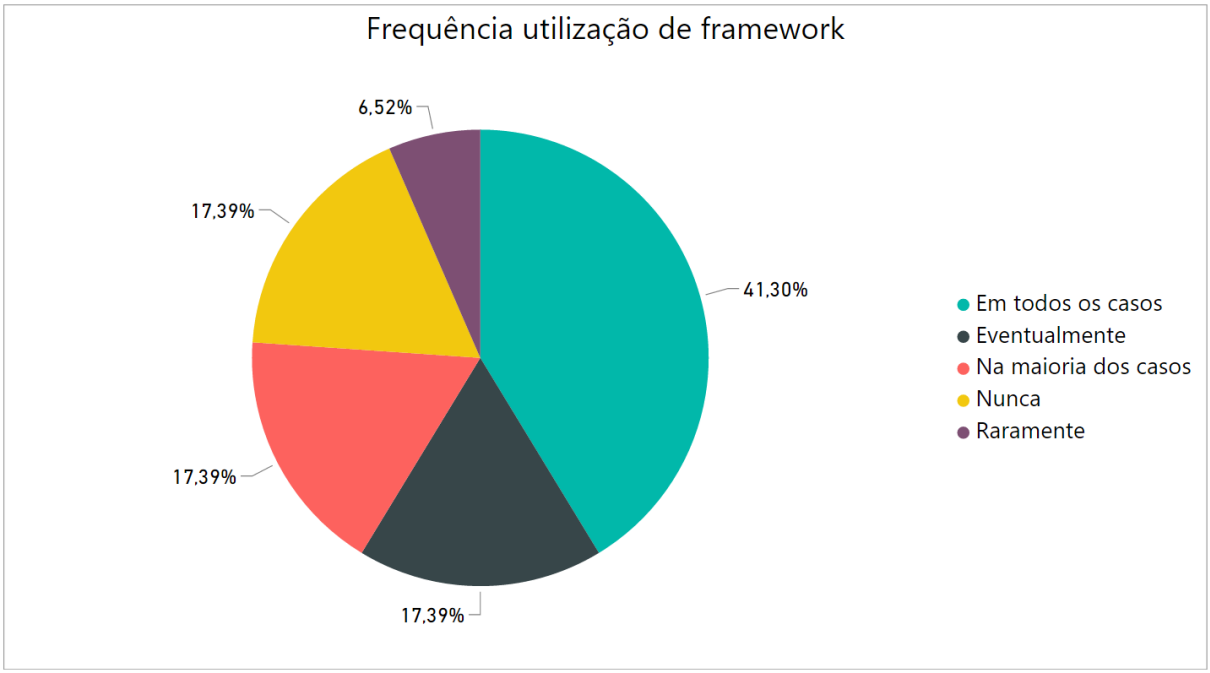

GRÁFICO 23 – Frequência de utilização de frameworks no desenvolvimento de *web service RESTful*.
Fonte: Próprio autor.

Como nos gráficos anteriores os resultados foram divididos em cinco grupos de frequência de utilização, sendo 1 a 2 nunca, 3 a 4 raramente, 5 a 6 eventualmente, 7 a 8 na maioria dos casos e 9 a 10 em todos os casos. Sendo assim 41,30% dos entrevistados afirmaram utilizar *framework* no desenvolvimento em todos os casos, 17,39% na maioria dos casos, 17,39% eventualmente, 6,52% raramente e 17,39% que responderam nunca utilizar. É perceptível notar que a grande maioria dos entrevistados, cerca de 58,69% realiza o desenvolvimento com *frameworks* frequentemente.

3.4 EXPERIÊNCIA COM PADRÕES DE PROJETO E RELAÇÃO *WEB SERVICE RESTFUL* E PADRÕES DE PROJETO

As questões dessa seção têm por objetivo coletar a opinião dos entrevistados a respeito de tempo e valor a certa da construção de *web service RESTful*, assim entendo qual é o real ganho em se aplicar padrões de projetos neste cenário.

Você concorda que um *web service* que foi implementado utilizando padrões de projeto terá um menor gasto com manutenções (gráfico 23)?

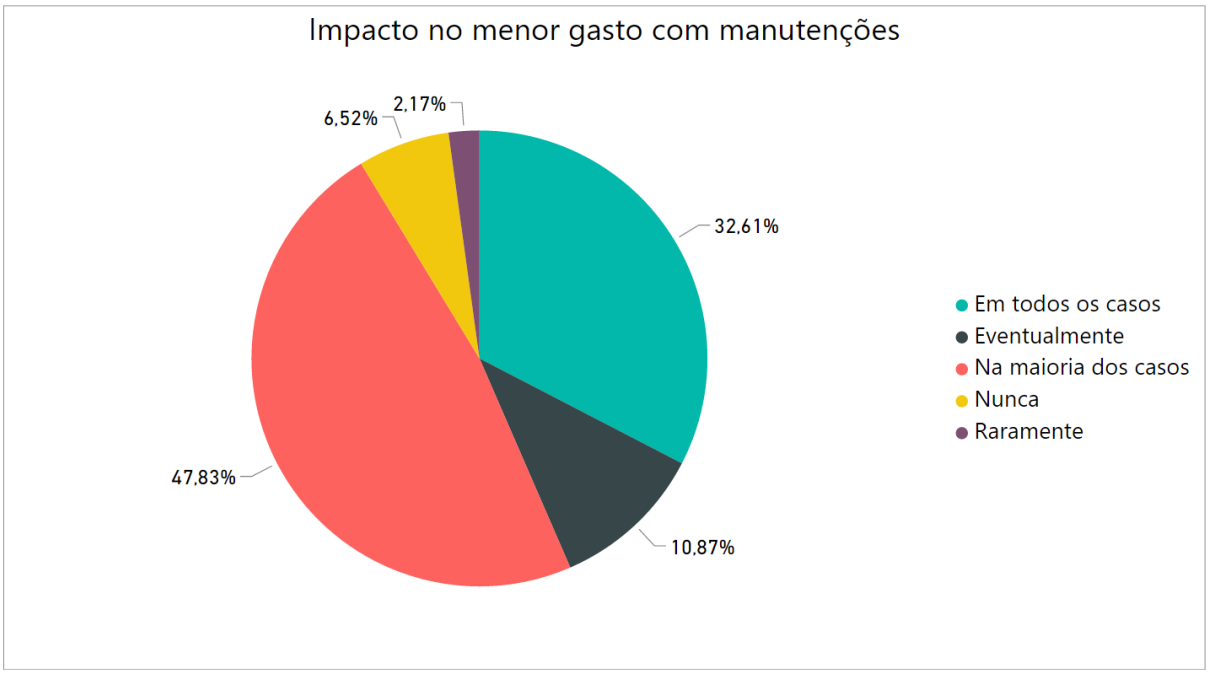

GRÁFICO 23 – Opinião sobre a utilização de padrões em *web service RESTful* terá um menor gasto com manutenções.
Fonte: Próprio autor.

Dos entrevistados 32,61% afirmaram que na maioria dos casos se obtém uma diminuição de gastos com manutenção de *web service RESTful* que foram implementados utilizando padrões de projetos, 47,83% na maioria dos casos, 10,87% eventualmente, 2,17% raramente e apenas 6,52% afirmaram nunca obter essa diminuição de gastos.

Podemos observar que dos entrevistados em sua maioria acreditam que a utilização de padrões de projeto pode diminuir os custos de manutenção da aplicação.

Você concorda que um *web service* implementado utilizando padrões de projetos será desenvolvido em menor tempo (gráfico 24)?

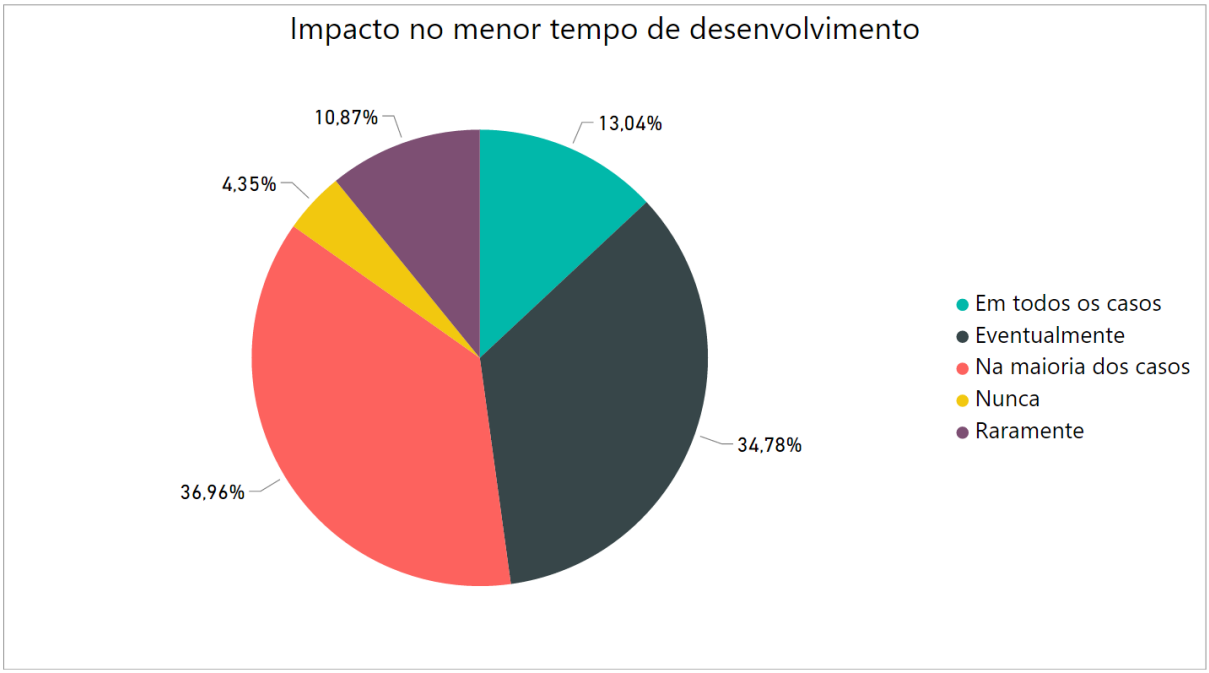

GRÁFICO 24 – Opinião sobre a utilização de padrões em *web service RESTful* terá um menor tempo de desenvolvimento.
Fonte: Próprio autor.

Dos entrevistados apenas 13,04% afirmaram ter uma diminuição de tempo no desenvolvimento de *web service RESTful* que foram implementados utilizando padrões de projetos, 36,96% na maioria dos casos, 34,78% eventualmente, 10,87% raramente e apenas 4,35% afirmaram nunca obter essa diminuição no tempo de desenvolvimento.

Podemos observar que dos entrevistados a maioria acredita que a utilização de padrões de projeto não diminuirá o tempo de desenvolvimento desde tipo de aplicação.

3.5 DISCUSSÃO DOS RESULTADOS

A partir da análise do questionário foi possível observar que a grande maioria dos entrevistados concorda com o fato de que se utilizar padrões de projeto de *software* e ter uma boa arquitetura definida para um *web service RESTful* é importante para a boa qualidade da aplicação o que também segundo Gamma (2000), “os padrões de projeto tornam mais fácil reutilizar projetos e arquiteturas bem sucedidas”.

Segundo Sommerville (2011) “a manutenção detém, aproximadamente, dois terços do orçamento, contra um terço para o desenvolvimento”, os entrevistados concordaram que a

utilização de padrões em *web service RESTful* diminuem os custos com manutenção do *software*, o que pode gerar diminuição considerável no custo total do projeto, já que o custo de manutenção pode chegar até dois terços do valor, se considerarmos apenas desenvolvimento e manutenção do *software*. Entretanto podemos notar que o processo de utilização de padrões de projetos nem sempre irá interferir diretamente na diminuição do tempo de desenvolvimento da aplicação segundo os entrevistados.

Umas das principais dificuldades de se utilizar padrões de projeto é a de falta de profissionais qualificados e com conhecimento sobre padrões. Em sua grande maioria segundo a pesquisa, os entrevistados não possuem conhecimento profundo sobre os padrões de projetos, além de não conhecerem os padrões selecionados nesta pesquisa e nem mesmo outros padrões, segundo o que eles afirmaram ao responderem sobre os padrões de projeto que tinham conhecimento.

Ao serem questionados sobre a importância de se utilizar cada padrão de projeto selecionado em *web service RESTful*, podemos observar que a maioria dos entrevistados que não tinham experiência com *web service RESTful* concordou que a utilização daqueles padrões tinha alto nível de importância de serem utilizados. Assim como os profissionais que tinham experiência no desenvolvimento deste tipo de tecnologia.

Sobre os padrões de projeto que foram selecionados nesta pesquisa, podemos afirmar que os padrões mais conhecidos dentre os entrevistados são os da categoria de criação, sendo o mais conhecido o Factory Method com cerca de 50%. Mesmo que nem todos os padrões selecionados para este trabalho sejam muito conhecidos pelos entrevistados podemos observar que nenhum foi totalmente desconhecido por eles, o que pode nos levar a crer que o método de seleção foi efetivo. Além disso, foi perceptível que a utilização dos padrões de projeto da categoria de criação é maior em *web service RESTful*, podemos interligar isso ao grande volume de objetos de derivações que são criados devido a inúmeras requisições feitas a um *web service*.

CONCLUSÃO

Com a elaboração deste trabalho foi possível constatar que os profissionais de Tecnologia da Informação têm ciência da necessidade de se ter uma boa arquitetura definida para o desenvolvimento de *web service RESTful*, em função da complexidade de se desenvolve-los e das exigências atuais como, alta maleabilidade e progressão do *software*. Com isso podemos notar que os padrões de projeto de *software* são necessários para se alcançar este objetivo.

Os padrões de projeto de *software* geram, ao serem utilizados, uma grande contribuição para diminuição no tempo de manutenção e respectivamente evolução de *web service RESTful*, pois torna-o mais flexível e apto a mudanças, o que neste tipo de aplicação é de extrema importância como foi dito anteriormente. Entretanto é perceptível notar que a não utilização de padrões de projeto está diretamente ligada a falta de conhecimento aprofundado sobre padrões de projeto de *software*.

A utilização de padrões de projeto no desenvolvimento de *web service RESTFul* é uma boa opção se alcançar maleabilidade. Padrões também diminuem os gastos com manutenção e acabam por melhorar a qualidade do *software* e por sua vez auxiliam na definição de uma boa arquitetura. Desta forma, este trabalho oferece contribuição para os profissionais que trabalham com desenvolvimento de *web service RESTful*, à medida que disponibiliza uma visão sobre a importância de padrões de projeto, no intuito de melhorar a arquitetura deste tipo de aplicação, atingindo assim, critérios como maior maleabilidade e menor custo no ciclo de vida do *software*.

TRABALHOS FUTUROS

Como direção para possíveis trabalhos futuros tem se:

- Um estudo de caso comparativo entre *web services*, desenvolvidos com ou sem a utilização de padrões de projeto de software;
- Um estudo mais aprofundado acerca de outros padrões de projeto que não foram selecionados a fim de comparativo;
- A aplicação de um questionário mais aprofundado sobre padrões de projeto de *software* a fim de estender a complexidade de utilizá-los em *web service RESTfull*.

REFERÊNCIAS

DAVID A. Chappell and Tyler Jewell Java Web Services: **Up and Running**, O Reilly Media 1 edition, (2002).

DEITEL, P. **Java Como Programar**. Tradução. São Paulo: Pearson Prentice Hall, 2010.

Erl, T. **SOA**. Tradução. Upper Saddle River, NJ: Prentice Hall, 2008.

Erl, T. **SOA design patterns**. Tradução. Upper Saddle River, NJ: Prentice Hall, 2009.

Erl, T. **Service-oriented architecture**. Tradução. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2005.

EXTENSIBLE Markup Language (XML). Disponível em: http://www.w3.org/XML/. Acessoem: 14 set. 2016.

FIELDING, Roy Thomas Fielding, **Architectural Styles and the Design of Network-based Software Architectures**, Tese de Doutorado, University of california, Irvine, 2000, Disponívelem: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Acessado em agosto de 2016.

GAMMA et al. **Padrões de Projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000.

GARLAN, D. and Perry, D., "**Introduction to the Special Issue on Software Architecture**", IEEE Transactions on Software Engineering, April 1995, p. 269-274.

KREGER, H. eBook – **Web Services Conceptual Architeture** (WSCA 1.0). IBM. 2001.

LIMA, J. **WEB SERVICES (SOAP X REST)**. Disponível em: <http://www.fatecsp.br/dti/tcc/tcc00056.pdf>. Acesso em: 31 mar. 2016.

MAZZOLA, Vitório B..**Engenharia de Software**. Universidade Federal de Santa Catarina, 2010.

MOREIRA, D. **A WEB SOCIAL**. UNIVERSIDADE FEDERAL DE GOIAS – UFG, 2009.

PAULA, Wilson de Pádua. **Engenharia de Software: fundamentos, métodos e padrões**. Rio de Janeiro: LTC, 2003. GAMMA.

PAULA, Wilson de Pádua F..**Manual do Engenheiro de Software**. 2000.

PEREIRA, A. **PADRÕES DE PROJETO: UMA COMPILAÇÃO DOS MAIS UTILIZADOS EM PROJETOS DE SOFTWARE**. Disponível em: <http://professores.dcc.ufla.br/~terra/publications_files/students/2008_faminas_pereira.pdf>. Acesso em: 31 mar. 2016.

REZENDE, Denis A.**Engenharia de Software e Sistemas de Informação**. 3ª edição. Rio de Janeiro: BRASPORT Livros e Multimídia LTDA, 2005.

RFC 2616 - **Hypertext Transfer Protocol** -- HTTP/1.1. Disponível em: <https://tools.ietf.org/html/rfc2616>. Acessoem: 14 out. 2016.

RICHARDSON, L.; Amundsen, M.. **RESTful Web APIs**. Tradução. Sebastopol, Calif.: O'Reilly, 2013.

RICHARDSON, L.; Ruby, S. **RESTful web service**. Tradução. Farnham: O'Reilly, 2007.

SOMMERVILLER, lan. **Engenharia de Software 9° edição**. Tradução. Ivan Bosnic e Kalinka G. de O. Gonçalves. Pearson, 2011.

Web Services Architecture. Disponível em: <https://www.w3.org/TR/ws-arch/>. Acesso em: 27 mar. 2016.

WIEHLER, G. eBook **- Computer - Service Oriented Architecture**. Siemens. 2004.

APÊNDICE A – QUESTIONÁRIO

Este questionário foi elaborado no âmbito de um trabalho de conclusão de curso de Ciência da Computação, o qual busca obter informações para analisar se padrões de projetos influenciam diretamente na construção e escalabilidade de um *web service*.

Os dados recolhidos são confidenciais e serão utilizados apenas para fins acadêmicos. O tempo estimado para o preenchimento deste questionário é de aproximadamente 5 a 10 minutos.

A sua colaboração é muito importante para o meio acadêmico / científico. Será relevante para este estudo que responda a todas as perguntas com seriedade e honestidade.

Desde já agradeço por participar!

*Obrigatório

Perfil do entrevistado

1. E-mail caso queira receber o resultado da pesquisa

_____

Qual seu nível de formação *

a) Técnico
b) Graduado / Graduando
c) Pós-graduado / Pós-Graduando
d) Mestre / Mestrando
e) Doutor / Doutorando
f) Outro: _____

Qual sua Função? *

a) Desenvolvedor / Programador
b) Analista de Sistemas
c) Engenheiro de Software

d) Gerente de Desenvolvimento
e) Arquiteto de Software
f) Outro: _____

Há quanto tempo aproximadamente você trabalha com produção de *software*? *

a) Nunca Trabalhei
b) Há cerca de 1 ano
c) Entre 1 e 3 anos
d) Entre 3 e 5 anos
e) Há mais de 5 anos

Sobre sua experiência com *web service* baseado em *RESTful*

Trabalha ou já trabalhou com *web service* baseado na arquitetura *RESTful*? *

a) Nunca Trabalhei
b) Há cerca de 1 ano
c) Entre 1 e 3 anos
d) Entre 3 e 5 anos
e) Há mais de 5 anos

Nos projetos onde você teve de realizar a implementação de um *web service*, baseados na arquitetura *RESTful*, qual era a média de profissionais envolvidos com o desenvolvimento do mesmo.

a) 1 profissional
b) De 2 a 5 profissionais
c) De 6 a 10 profissionais
d) De 11 a 20 profissionais
e) De 21 a 50 profissionais
f) Mais de 50 profissionais

No quesito de construção/implementação dos *web service*, qual sua média de sucesso?

a) Em todos os casos
b) Na maioria dos casos
c) Eventualmente
d) Raramente
e) Nunca

No quesito de implantação dos *web service*, qual sua média de sucesso?

a) Em todos os casos
b) Na maioria dos casos
c) Eventualmente
d) Raramente
e) Nunca
f)

No quesito de manutenibilidade e escalabilidade dos *web service*, qual sua média de sucesso?

a) Em todos os casos
b) Na maioria dos casos
c) Eventualmente
d) Raramente
e) Nunca

Quais linguagens foram utilizadas para implementação dos *web service*?

- ☐ C#
- ☐ JAVA
- ☐ PHP
- ☐ Python
- ☐ Outras: _____

Sobre sua experiência com padrões de projeto

Você utiliza padrões de projeto com que frequência em seus projetos? *

a) Em todos os casos
b) Na maioria dos casos
c) Eventualmente
d) Raramente
e) Nunca

Para você, qual a importância de utilizar padrões de projeto na construção de um *web service*, baseados na arquitetura *RESTful*. *

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Para você qual a importância de ter uma arquitetura bem definidas para o desenvolvimento de pequenos *web service*, baseados na arquitetura *RESTful*. *

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Dos padrões de projetos listados, em quais você tem conhecimento? *

- ☐ Factory Method
- ☐ Abstract Factory
- ☐ Prototype
- ☐ Adapter
- ☐ Decorator
- ☐ State
- ☐ Proxy
- ☐ Strategy
- ☐ Chain of Responsibility
- ☐ Outro: _____

Para você qual é a importância de se aplicar o padrão FactoryMethod em um *web service* baseado na arquitetura *RESTful*?

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Para você qual é a importância de se aplicar o padrão Abstract Factory em um *web service* baseado na arquitetura *RESTful*?

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Para você qual é a importância de se aplicar o padrão Prototype em um *web service* baseado na arquitetura *RESTful*?

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Para você qual é a importância de se aplicar o padrão Adapter em um *web service* baseado na arquitetura *RESTful*?

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Para você qual é a importância de se aplicar o padrão Decorator em um *web service* baseado na arquitetura *RESTful*?

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Para você qual é a importância de se aplicar o padrão State em um *web service* baseado na arquitetura *RESTful*?

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Para você qual é a importância de se aplicar o padrão Proxy em um *web service* baseado na arquitetura *RESTful*?

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Para você qual é a importância de se aplicar o padrão Strategy em um *web service* baseado na arquitetura *RESTful*?

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Para você qual é a importância de se aplicar o padrão Chain ofResponsibility em um *web service* baseado na arquitetura *RESTful*?

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Com qual frequência você utiliza frameworks para o desenvolvimento de *web service* baseados em *RESTful*? *

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Sobre sua opinião em relação entre *web service* baseado em *RESTful* e padrões de projeto

Você concorda que um *web service* que foi implementado utilizando padrões de projeto terá um menor gasto com manutenções? *

a) Em todos os casos
b) Na maioria dos casos
c) Eventualmente
d) Raramente
e) Nunca

Você concorda que um *web service* implementado utilizando padrões de projetos será desenvolvido em menor tempo? *

a) Em todos os casos
b) Na maioria dos casos
c) Eventualmente
d) Raramente
e) Nunca