MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21c/2018/05.21.12.09-TDI

ANÁLISE COMPARATIVA DE ACOPLAMENTO LÓGICO ENTRE COMPONENTES DE FRAMEWORKS WEB

Eduardo Pereira de Sousa

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Eduardo Martins Guerra, aprovada em 30 de maio de 2018.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3R66UN2>

INPE
São José dos Campos
2018

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21c/2018/05.21.12.09-TDI

ANÁLISE COMPARATIVA DE ACOPLAMENTO LÓGICO ENTRE COMPONENTES DE FRAMEWORKS WEB

Eduardo Pereira de Sousa

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Eduardo Martins Guerra, aprovada em 30 de maio de 2018.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3R66UN2>

INPE
São José dos Campos
2018

Dados Internacionais de Catalogação na Publicação (CIP)

Sousa, Eduardo Pereira de.

So85a Análise comparativa de acoplamento lógico entre componentes de frameworks web / Eduardo Pereira de Sousa. – São José dos Campos : INPE, 2018.

xxii + 79 p. ; (sid.inpe.br/mtc-m21c/2018/05.21.12.09-TDI)

Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2018.

Orientador : Dr. Eduardo Martins Guerra.

1. Engenharia de software. 2. Mineração de repositórios de software. 3. Acoplamento lógico. 4. Frameworks para aplicações web. I.Título.

CDU 004.41

Aluno (a): *Eduardo Pereira de Sousa*

Título: "ANÁLISE COMPARATIVA DE ACOPLAMENTO LÓGICO ENTRE COMPONENTES DE FRAMEWORKS WEB"

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de *Mestre* em *Computação Aplicada*

Dr. Gilberto Ribeiro de Queiroz

Presidente / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

Dr. Eduardo Martins Guerra

Orientador(a) / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

Dr. Lamartine Nogueira Frutuoso Guimarães

Membro da Banca / INPE / SJCampos - SP

( ) Participação por Video - Conferência

Dr. Ricardo Terra Nunes Bueno Villela

Convidado(a) / UFLA / Lavras - MG

(X) Participação por Video - Conferência

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 30 de maio de 2018

"Tudo o que você vê, o que ouve ou vivencia de qualquer jeito que seja é especifico para você. Você cria um universo ao percebê-lo, então tudo no universo que percebe é especifico para você".

Douglas Adams
em *"O Guia do Mochileiro das Galáxias"*, 1979

A meus pais ***José Roberto*** *e* ***Sônia****, minha esposa* ***Simone*** *e nossos filhos* ***Enzo*** *e* ***Laís****.*

AGRADECIMENTOS

O desenvolvimento deste trabalho não teria sido possível sem o apoio de diversas pessoas às quais devo meus sinceros agradecimentos:

A meu orientador, Dr. Eduardo Martins Guerra, pela valiosa orientação, pelos muitos conselhos, ensinamentos e incentivo em cada etapa deste trabalho.

A Coordenação do curso de Computação Aplicada do Instituto Nacional de Pesquisas Espaciais, pela oportunidade de realização deste curso e por todo apoio na realização deste trabalho.

Ao Dr. Maurício Finavaro Aniche e ao Dr. Gustavo Ansaldi Oliva, pelo valiosos conselhos e ensinamentos na área de Mineração de Repositórios de Software e Análise do Acoplamento Lógico.

Ao Instituto Federal de São Paulo pelo apoio e incentivo ao desenvolvimento deste trabalho por meio das políticas de incentivo a capacitação.

Aos professores do curso de Computação Aplicada e demais professores do INPE por todos os ensinamentos.

A minha família pelo apoio e compreensão durante as muitas viagens para São José dos Campos e os vários dias e noites de trabalho em casa.

Aos membros da banca examinadora, por todas as contribuições para melhoria deste trabalho.

RESUMO

Frameworks são peças-chave no desenvolvimento de aplicações Web, provendo aos desenvolvedores um conjunto de abstrações e uma arquitetura de referência para a construção dessas aplicações. Os métodos de inversão de controle e extensão utilizados pelos *frameworks*, no entanto, tornam a arquitetura das aplicações extremamente ligada a arquitetura do próprio *framework*. Entender o acoplamento esperado de acordo com o tipo de *framework* utilizado é importante tanto para subsidiar seu uso em novas aplicações quanto para análises em aplicações existentes. O objetivo deste trabalho é realizar um estudo em aplicações Web desenvolvidas com diferentes tipos de *frameworks*, analisando o acoplamento entre componentes com diferentes papéis arquiteturais. Neste estudo, foram considerados *frameworks* do tipo *request-based*, *component-based* e *rich-internet-application*. A análise baseou-se em técnicas de mineração de repositórios de software com o objetivo de explorar o acoplamento lógico, ou evolutivo, entre os componentes. É proposto ainda um conjunto de heurísticas para identificação de papéis arquiteturais durante o processo de mineração, para permitir a identificação de classes com diferentes características. Nos resultados são comparados os dados de acoplamento lógico obtidos para os papéis arquiteturais entre os diferentes tipos de *framework*, evidenciando o acoplamento inerente a cada abordagem arquitetural. Destaca-se entre os resultados obtidos o maior acoplamento entre componentes do *backend* e *frontend* para a abordagem *component-based*, enquanto as demais abordagens demonstram uma maior coesão desses componentes.

Palavras-chave: Engenharia de Software. Mineração de Repositórios de Software. Acoplamento Lógico. Frameworks para Aplicações Web.

COMPARATIVE ANALYSIS OF LOGICAL COUPLING BETWEEN WEB FRAMEWORK COMPONENTS

ABSTRACT

Frameworks are key components in Web application development, providing developers with a set of high level abstractions and a reference architecture for building them. Inversion of control and component extension are the main methods of design reuse employed by Web frameworks, these methods, however, create strong bonds between application and framework architecture. Due to this bond, the understanding of the inherent coupling by the type of framework is very important on both to subsidize its use in new applications and to analyze its impact on existing applications. The goal of this work is to perform a study on Web applications developed with different types of frameworks, analyzing the coupling between components with different architectural roles. In this study were considered three types of frameworks: request-based, component-based, and rich-internet-application. The analysis was based on mining software repository techniques with the objective of exploring the logical coupling between components of different architectural roles. We also propose a set of heuristics to identify architectural roles during the mining process to allow the identification of classes with different characteristics. In our results we compare the logical coupling data obtained for the architectural roles between the different types of framework, evidencing the inherent coupling to each architectural approach. Among the results, we highlight the greater coupling between backend and frontend components for the component-based approach, while the other approaches demonstrate a greater cohesion of these components.

Keywords: Software Engineering. Mining Software Repositories. Logical Coupling. Web Application Framework.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

AJAX	–	Asynchronous Javascript and XML
API	–	Application Programming Interface
CGI	–	Common Gateway Interface
CSS	–	Cascading Style Sheets
DAO	–	Data Access Object
HTML	–	HyperText Markup Language
JPA	–	Java Persistence API
MVC	–	Model-View-Controller
MSR	–	Mining Software Repositories
REST	–	Representational State Transfer
RIA	–	Rich Internet Application
SPA	–	Single Page Applications
UML	–	Unified Modeling Language
XML	–	Extensible Markup Language
WAF	–	Web Application Framework

SUMÁRIO

Pág.

1 INTRODUÇÃO

Frameworks são peças fundamentais no desenvolvimento de aplicações modernas, pois permitem o reúso de um conjunto de componentes e padrões bem estabelecidos (JOHNSON, 1997). A utilização de *frameworks* específicos para certos domínios de aplicação permite que detalhes de implementação sejam substituídos por abstrações de mais alto nível (FAYAD; SCHMIDT, 1997).

A busca por técnicas que permitam a reutilização de código de maneira eficiente não é algo novo na Engenharia de Software. Neste sentido, os *frameworks* são mais um passo na busca que vem desde a criação dos primeiros paradigmas de programação estruturada (DIJKSTRA et al., 1972), passando pela criação de técnicas de *design* estruturado (STEVENS et al., 1974), e a criação de paradigmas de programação de alto nível, como a programação orientada a objetos (RUMBAUGH et al., 1991).

A reutilização de código no processo de desenvolvimento pode ocorrer em diferentes níveis. A orientação a objetos possibilita a reutilização por meio de recursos como herança, composição e polimorfismo, os quais permitem a criação de componentes de funcionalidade geral que podem ser especializados posteriormente. *Frameworks*, por sua vez, são ferramentas que permitem o reúso não apenas do código em si, mas também do *design* das aplicações ao preconizar o uso de determinadas arquiteturas e padrões (JOHNSON, 1997).

No desenvolvimento de aplicações para Web, o uso de *frameworks* tornou-se por si só um padrão *de facto*. Os *frameworks* são tão importantes para esse tipo de software, que em muitos casos, o nome do *framework* é utilizado para descrever a tecnologia com a qual a aplicação foi desenvolvida ao invés da linguagem de programação. Por exemplo, ao perguntar para um desenvolvedor o que ele utilizou para implementar determinada aplicação Web, sua resposta será algo como *"Rails"*, *"Django"* ou *"Spring"*, ao invés de *"Ruby"*, *"Python"* ou *"Java"*.

A larga adoção dos *frameworks* Web pode ser justificada, em certo ponto, pela complexidade do ambiente no qual as aplicações Web são executadas, com diversos componentes interdependentes distribuídos entre cliente e servidor. A interface da aplicação exibida ao usuário no navegador Web é gerada de forma dinâmica pelos componentes executados no servidor, no entanto, no navegador Web os componentes da aplicação não se limitam a uma simples exibição do estado da aplicação, mas em muitos casos são também responsáveis por controlar as alterações decorrentes das ações do usuário, repassando-as aos componentes executados no servidor.

Além da complexidade inerente ao ambiente das aplicações Web, os *frameworks* permitem que as aplicações Web se adéquem com pouco esforço a determinados padrões arquiteturais, como os baseados em camadas. Segundo Fowler (2002), dividir um software complexo em camadas pode ser benéfico, pois: (i) facilita o entendimento individual das camadas, (ii) permite a substituição das camadas sem grande impacto nas demais, (iii) reduz a dependência entre componentes de camadas distintas, (iv) simplifica a padronização e (v) permite sua reutilização com outros serviços de alto nível.

Os *frameworks* Web desenvolvidos na linguagem Java são em sua maioria baseados no padrão de camadas *Model-View-Controller*, ou simplesmente MVC (KRASNER; POPE, 1988), porém, diferem em diversos aspectos, como, na forma de processar as requisições recebidas, no encapsulamento de seus componentes e na geração da interface com o usuário. Essas especificidades permitem que os *frameworks* Web desenvolvidos na linguagem Java sejam classificados em cinco categorias distintas: *Request-Based*, *Component-Based*, *Hybrid*, *Meta* e *RIA-Based* (SHAN; HUA, 2006).

Apesar da grande importância dos *frameworks* para o desenvolvimento de aplicações Web, são raros os estudos que levam em conta as abordagens arquiteturais propostas por cada um deles. Esses estudos poderiam trazer uma melhor compreensão dos *frameworks* para os desenvolvedores de aplicações, subsidiando a escolha da melhor ferramenta para cada tarefa, além de colaborar no aperfeiçoamento das abordagens para construção de *frameworks*.

Dentre os pontos que carecem de investigação, é particularmente interessante a análise do acoplamento inerente a cada arquitetura de *framework*, com destaque especial para o acoplamento entre componentes executados no cliente (*frontend*) e aqueles executados no servidor (*backend*). Acoplamento é a medida de força dos relacionamentos entre os componentes do software. Um alto nível de acoplamento está diretamente relacionado com uma maior complexidade e custo de manutenção do software (STEVENS et al., 1974; OMAN; HAGEMEISTER, 1992).

Um dos pontos que dificulta essa investigação, é a especificação em diferentes linguagens dos componentes executados no cliente e no servidor. Dependências em software multi-linguagem são pouco estudadas e impõe um conjunto próprio de desafios (KONTOGIANNIS et al., 2006). Além dos diferentes domínios, nas aplicações Web temos um conjunto de linguagens de diferentes paradigmas, desde linguagens imperativas, como Java e JavaScript, até linguagens declarativas utilizadas para configuração e criação de interfaces visuais, como XML, HTML e CSS.

Nas aplicações Web temos dois conjuntos de artefatos com linguagens e características distintas: aqueles que são executados no navegador Web do cliente, chamado de *frontend*, e aqueles executados pelo servidor, chamado *backend*. No entanto, esses dois conjuntos de artefatos formam uma única aplicação e por isso estabelecem diversas relações de dependência entre si, as quais implicam no acoplamento entre esses artefatos. Em aplicações convencionais esse acoplamento pode ser verificado por meio das relações definidas no código-fonte, como chamadas a métodos de diferentes objetos, no entanto, as relações entre artefatos do *frontend* e *backend* se estabelecem pela comunicação entre eles, a qual nem sempre pode ser prontamente identificada no código-fonte. A falta de informações sobre o acoplamento desses artefatos não permite aos desenvolvedores reconhecer prontamente os pontos da aplicação mais suscetíveis a mudanças decorrentes dessas dependências.

Dessa forma, faz-se necessária a utilização de um conjunto distinto de técnicas para investigação do acoplamento entre os componentes das aplicações Web. Neste trabalho, propõe-se a utilização da mineração de repositórios de software (MSR) em busca de dependências evolucionárias, ou dependências lógicas (GALL et al., 1998). Esse tipo de dependência está ligada ao histórico de desenvolvimento do software e pode ser identificada por meio de alterações realizadas simultaneamente em um conjunto de artefatos de software (ZIMMERMANN et al., 2005).

A partir desse contexto investiga-se a formação de dependências lógicas em aplicações desenvolvidas com diferentes categorias de *frameworks* Web para a linguagem Java. Avaliando e comparando o acoplamento entre diferentes componentes[1] arquiteturais, com enfase nas relações formadas entre os componentes da aplicação executados no cliente (*frontend*) e aqueles executados no servidor (*backend*). Essa investigação permitirá formular um conjunto de hipóteses sobre a forma como os diferentes componentes se relacionam nas abordagens arquiteturais investigadas, o que servirá de base para uma investigação manual do código-fonte desses componentes.

A investigação manual do código-fonte tem por objetivo verificar as origens e as possíveis implicações de cada uma das hipóteses levantadas, subsidiando uma compreensão abrangente dessas arquiteturas e de seus mecanismos de acoplamento. Com isso tem-se por objetivo prover subsídios para os desenvolvedores de aplicações e *frameworks* Web, permitindo (i) a escolha dos *frameworks* mais adequados a deter-

[1]Neste trabalho utiliza-se o termo componente, ou componente arquitetural em um sentido mais amplo do que a definição formal de componentes de software. Este termo foi aplicado no sentido de arquivos que compõe a arquitetura da aplicação, sem necessariamente encapsular um conjunto independente de funcionalidades como um componente de software tradicional.

minadas tarefas e equipes de desenvolvimento, (ii) a compreensão da arquitetura dos *frameworks* e sua possíveis deficiências que possam levar a inconformidades arquiteturais e (iii) a melhoria dos *frameworks* baseadas na observação das possíveis fraquezas levantadas por esse estudo.

1.1 Motivação

Novas aplicações são lançadas na Web diariamente, desenvolvidas nas mais diversas linguagens, usando inúmeras metodologias de desenvolvimento e com os mais variados propósitos. Apesar disso, grande parte dessas aplicações tem como fator comum o uso de *frameworks* como base para seu desenvolvimento, o que torna essencial a realização de estudos que avaliem seu impacto sobre as aplicações desenvolvidas.

O INPE possui entre seus projetos diversas aplicações Web, as quais são utilizadas para prover serviços ao público, realizar aquisição de dados e também para divulgar as atividades do instituto. Muitas dessas aplicações foram desenvolvidas na linguagem Java com o uso de *frameworks* para aplicações Web, como é o caso das aplicações desenvolvidas pelos programas EMBRACE (Estudo e Monitoramento Brasileiro Clima Espacial), LEONA (Rede Colaborativa na América Latina para a Investigação de Eventos Luminosos Transientes e Emissões de Alta Energia de Tempestades) e dos diversos sistemas de software que compõe a plataforma Web do CPTEC (Centro de Previsão e Estudos Climáticos). O estudo das características desses *frameworks* pode ajudar a subsidiar decisões quanto ao desenvolvimento e manutenção dessas aplicações, além de prover parâmetros para comparação das características dessas aplicações com diversas outras analisadas neste trabalho.

Diferentes *frameworks* possuem arquiteturas distintas que podem influenciar no acoplamento entre os componentes da aplicação. O campo da análise de impacto em mudanças de software possui um grande conjunto de técnicas e estudos que podem colaborar na compreensão do impacto oriundo do uso de cada *framework*. No entanto, os estudos de análise de impacto em mudanças de software abordam os componentes de software de maneira geral, sem levar em conta o contexto no qual esses componentes se inserem ou a arquitetura do projeto.

Ao analisar o acoplamento entre diferentes componentes de aplicações Web, é importante conhecer p papel arquitetural desempenhado por estes componentes. Estudos recentes afirmam que os desenvolvedores de software de código-aberto utilizam em média quatro diferentes linguagens em seus projetos (KARUS; GALL, 2011). Compreender o acoplamento entre componentes de diferentes linguagens e sua co-evolução

constitui por si só um grande passo no entendimento das aplicações Web. *Acredita-se que esse entendimento pode ser enriquecido quando considerado o contexto no qual os componentes são criados e mantidos.*

Os resultados dessa investigação podem contribuir significativamente para dois grupos distintos de desenvolvedores: aqueles que desenvolvem *frameworks*, e aqueles que os utilizam. Os desenvolvedores de *frameworks* poderão utilizar os resultados obtidos para embasar decisões de *design*, visando reduzir pontos de acoplamento que possam causar quebras de compatibilidade com versões anteriores do *framework* no caso de mudanças (GUERRA et al., 2009).

Por outro lado, os usuários de *frameworks* podem se beneficiar ao conhecer os pontos de acoplamento inerentes a cada *framework*, facilitando o estabelecimento de regras de conformidade arquitetural específicas para o *framework* utilizado, e também simplificando o processo de depuração das aplicações ao permitir uma compreensão mais ampla das relações entre seus componentes.

1.2 Objetivo

Analisar a evolução dos componentes de aplicações Web desenvolvidas com diferentes frameworks, com objetivo de compreender e comparar o acoplamento entre eles, com especial ênfase às relações entre os que estão implementados em linguagens diferentes e entre os que executam no cliente e no servidor.

1.3 Metodologia de Pesquisa

Os subsídios iniciais ao desenvolvimento deste estudo foram dados a partir de um levantamento bibliográfico abrangendo eventos nacionais e internacionais nas áreas de Engenharia de Software, como o Simpósio Brasileiro de Engenharia de Software (SBES), *Workshop* de Visualização, Evolução e Métricas (VEM) e a Conferência Internacional em Mineração de Repositórios de Software (MSR).

Este levantamento bibliográfico proveu o arcabouço teórico para o desenvolvimento do estudo preliminar que o seguiu, o qual foi utilizado para avaliar a pertinência do método de identificação do acoplamento lógico por regras de associação (ZIMMERMANN et al., 2005) a uma análise baseada no contexto de papéis arquiteturais (ANICHE et al., 2016).

O passo posterior ao estudo preliminar foi a definição das abordagens arquiteturais, ou tipos de *frameworks*, que seriam analisados neste trabalho. Optou-se pela esco-

lha dos tipos que possuem os representantes mais populares entre os *frameworks* para aplicações Web na linguagem Java, essa escolha foi baseada em relatórios da indústria de desenvolvimento de software (ZERO TURNAROUND, 2016).

Com um melhor entendimento das técnicas e ferramentas necessárias para o desenvolvimento deste trabalho passou-se ao processo de seleção dos projetos para análise, neste ponto optamos pelo uso de projetos de código aberto hospedados na plataforma GitHub, tendo como critério o uso dos *frameworks* selecionados anteriormente.

Por fim, foram extraídos os dados de acoplamento lógico entre os papéis arquiteturais dos projetos, esses dados foram agrupados conforme o tipo de *framework* utilizado pelo projetos para que pudessem ser analisados e comparados entre si à luz das questões de pesquisa definidas no desenvolvimento deste trabalho. Essa análise subsidiou a construção de hipóteses que foram então verificadas por meio da análise manual do código-fonte das aplicações.

1.4 Relevância

O desenvolvimento de aplicações para Web é uma área em franca expansão e desenvolvimento, o surgimento quase que diário de novas ferramentas e técnicas não permite uma avaliação sistemática de todos esses novos recursos, no entanto, é essencial aprofundarmos o entendimento de ferramentas já bem estabelecidas, visto que essas servem de modelo para os novos projetos.

Nesse sentido, entender e comparar o impacto de diferentes arquiteturas implementadas por *frameworks*, permitirá subsidiar decisões futuras quanto ao seu uso e seu desenvolvimento. Além de ampliar o ferramental existente para a execução de estudos que possam considerar as características individuais de cada arquitetura.

O estudo do acoplamento, por sua vez, se dá por essa ser uma métrica fundamental no desenvolvimento de aplicações. O controle sobre os níveis de acoplamento permite que as aplicações sejam mantidas a longo prazo, enquanto seu descuido é causa fundamental do aumento na complexidade do aplicação e consequentemente dos seus custos de manutenção, o que pode levá-la a uma obsolescência precoce.

Por meio deste estudo, pretende-se subsidiar desenvolvedores de aplicações e de *frameworks* Web para que os primeiros possam escolher as melhores ferramentas para o tipo de trabalho a ser realizado, e para que os últimos tenham subsídios ao tomar decisões de *design* no desenvolvimento dos *frameworks* de modo a reduzir o acoplamento intrínseco das aplicações.

O INPE como desenvolvedor de *frameworks* e aplicações para Web poderá se beneficiar diretamente do desenvolvimento e dos resultados deste estudo, uma vez que o processo de mineração de repositórios de software provê um olhar analítico sobre esses repositórios, permitindo avaliar a adequação destes às melhores práticas da indústria. Além disso, os resultados deste estudo irão abranger um amplo conjunto de projetos de software em diferentes níveis de maturidade, os quais permitirão uma análise comparativa das aplicações desenvolvidas pelo Instituto.

1.5 Contribuição

Apesar de não trazer uma nova técnica no que concerne a análise de impacto, a proposta deste trabalho não encontra similares na literatura, uma vez que a maioria dos trabalhos que se dedicam a esse campo o fazem sem levar em consideração o tipo de software analisado, focando nas técnicas e ferramentas, e em seus resultados sobre sistemas de software de caráter geral (LI et al., 2013). Neste trabalho, é proposta a utilização de técnicas de identificação do acoplamento lógico para analisar projetos desenvolvidos com diferentes *frameworks* Web, comparando os resultados obtidos em função da arquitetura utilizada por cada *framework*.

A técnica de análise de impacto por meio de dependências lógicas utilizada neste estudo foi inicialmente proposta com a finalidade de buscar uma melhor decomposição dos componentes da aplicação, visando melhorias no acoplamento e coesão com base em métricas não estruturais (GALL et al., 1998). Esse tipo de trabalho está no cerne da motivação para essa pesquisa, porém, serão tratados exclusivamente de projetos de software baseados em *frameworks* para Web considerando na análise as diversas camadas e papéis arquiteturais dos componentes desses projetos.

A análise do acoplamento com base no contexto arquitetural dos componentes da aplicação é um método recente e pouco explorado dentro da mineração de repositórios de software, o estudo preliminar baseou-se em um método para a definição de valores de referência em métricas de software na detecção de *"bad smells"* (ANICHE et al., 2016), porém, essa abordagem foi ampliada ao serem acrescentados novos *frameworks* e arquiteturas em nossa análise comparativa, além da criação de uma ferramenta e um conjunto de heurísticas para a classificação desses papéis.

Zimmermann et al. (2005) utilizam o acoplamento lógico para avaliar seu desempenho na identificação do acoplamento em diferentes níveis, como, arquivos, classes e métodos. Focando na avaliação do desempenho dessa técnica como meio de navegação pelo código-fonte e prevenção de erros oriundos de dependências não estruturais.

No trabalho de Maffort et al. (2016), é apresentado um método heurístico baseado na análise histórica de versões de um projeto para detecção de violações de conformidade arquitetural. Esse trabalho baseia-se em um conjunto de heurísticas para detecção de relações arquiteturais esperadas e não identificadas (ausências) e relações não permitidas identificadas (divergências). Apesar das semelhanças no uso de heurísticas para detecção de conceitos arquiteturais, a decomposição dos projetos analisados em subsistemas baseia-se um em conjunto de regras definidas por meio de expressões regulares por alguém com conhecimento do projeto, enquanto neste trabalho é proposta uma heurística ampla e comum para detecção desses subsistemas, ou papéis arquiteturais, em aplicações baseadas em *frameworks* Web.

Alguns estudos focam na reconstrução da arquitetura do software por meio de técnicas de engenharia reversa (CANFORAHARMAN; PENTA, 2007). Essa abordagem constitui uma importante ferramenta para compreensão do software, permitindo a criação de documentos de *design*, como diagramas UML, a partir do software existente. Este estudo diferencia-se dessas abordagens pois seu objetivo não está na reconstrução da arquitetura por si, mas na análise do acoplamento gerado por diferentes arquiteturas de *frameworks*.

No trabalho de Thummalapenta e Xie (2008) é proposta uma ferramenta cujo objetivo é facilitar a identificação de *hotspots* em *frameworks* para permitir um maior reúso de código. Essa ferramenta atua como um buscador de código-fonte, realizando a mineração de repositórios que utilizam um determinado *framework* em busca de exemplos da sua utilização. Similar ao proposto neste trabalho, esse estudo utiliza técnicas de mineração de repositórios sobre *frameworks*, porém, seu objetivo não é analisar a estrutura destes *frameworks*, mas identificar pontos de extensão que podem ser utilizados por desenvolvedores de software.

Neste trabalho, são utilizadas algumas técnicas já consagradas na mineração de repositórios de software em busca de um novo ponto de vista sob o qual a aplicação pode ser analisada e visualizada, agrupando seus componentes em papéis arquiteturais para uma melhor compreensão das relações entre esses papéis na arquitetura das aplicações Web. Esse panorama permite uma compreensão de mais alto nível das estruturas da aplicação, com ênfase nas características inerentes ao *framework* utilizado pela aplicação.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção são apresentados alguns dos principais conceitos e ferramentas utilizados no desenvolvimento deste trabalho, é apresentado o conceito de aplicações e *frameworks* para Web, são expostos os desafios para a investigação de sistemas de software multilinguagem, um panorama geral das técnicas de mineração de repositórios de software e uma breve introdução às técnicas de análise de impacto e dependências de software, com principal ênfase no acoplamento lógico e na técnica de detecção deste por meio de regras de associação.

2.1 Aplicações web

Uma definição ampla de aplicação Web é dada por Conallen (1999), segundo o qual, uma aplicação Web pode ser definida como um sistema onde as entradas do usuário, por meio de atividades de navegação ou interação com páginas Web, modificam o estado representado pela aplicação executada no servidor. Essa definição estabelece que uma aplicação Web é um software composto de regras de negócio estabelecidas no *backend* e representadas por um *frontend* executado no navegador Web.

Essa definição foi concebida em um contexto embrionário para as aplicações Web, no qual surgiam e começavam a se consolidar as primeiras linguagens e *frameworks* voltados para o desenvolvimento dessas aplicações. A rudimentalidade dessa definição a torna inadequada para explicar as diversas facetas do desenvolvimento Web contemporâneo, com aplicações inteiramente baseada no navegador ou aplicações executadas como programas convencionais, mas cuja parte da interface e de suas regras de negócio é carregada sob demanda a partir de servidores Web.

As aplicações Web contemporâneas tem se diversificado, ganhando mercado em áreas tradicionalmente dominadas por aplicações para *desktop*, como no caso das suítes de aplicativos de escritório. Essa expansão das aplicações Web se deve em muito a adoção da tecnologia AJAX, ou *Asynchronous Javascript and XML*, possibilitando que a interface das aplicações Web tenham níveis de interatividade similares aos das aplicações *desktop* (ZEPEDA; CHAPA, 2007).

Com o uso da tecnologia AJAX ocorreu uma mudança no paradigma tradicional de desenvolvimento de aplicações Web. Tradicionalmente, as aplicações Web utilizam um modelo de comunicação baseado em requisição e resposta, onde cada nova resposta gera uma página Web completa. Com o modelo Ajax as comunicações não necessariamente geram novas páginas, podendo atualizar segmentos parciais da pá-

gina por meio da manipulação da estrutura que representa os elementos da página em memória, conhecida como *Document Object Model* (DOM).

A partir dessa mudança de paradigma novos métodos e tecnologias tem sido adotadas com a finalidade de equiparar as aplicações Web em usabilidade, recursos e performance àquelas executadas localmente no computador do usuário. Tornou-se popular o conceito de *Single Page Applications*, que são aplicativos Web compostos de uma única página Web, a qual é atualizada de forma assíncrona com o uso de AJAX para refletir o estado da aplicação, sem a necessidade de renderizar páginas completas a cada requisição. Como notado, esse modelo tem substituído algumas aplicações *desktop*, com notáveis vantagens, uma vez que sua disponibilização através da Web permite novos modelos de negócio na comercialização do software, além de ciclos de atualização mais rápidos e eficientes (TURNER et al., 2003).

2.2 *Frameworks* para aplicações Web

No trabalho de Johnson e Foote (1988) os *frameworks* são definidos como sistemas de software incompletos que podem ser estendidos com funcionalidades específicas da aplicação que os utiliza. Em linguagens orientadas a objetos estes são formados por um conjuntos de classes que representam um *design* abstrato para um determinado domínio de aplicação. Os *frameworks* diferem de bibliotecas de software pois não são invocados pela aplicação que deseja utilizar suas funcionalidades, mas se mesclam a ela por meio da extensão e composição de suas estruturas.

O *design* dos *frameworks* de aplicação é caracterizado pela inversão de controle, ou seja, o fluxo de controle é estabelecido pelo *framework*, que invoca as funcionalidades específicas da aplicação quando necessário (FAYAD; SCHMIDT, 1997).

Essas técnicas de especialização em conjunto com a inversão de controle permitem o reuso não apenas do código, mas também da arquitetura da aplicação. Dessa forma os *frameworks* podem ser vistos como uma extensão natural da programação orientada a objetos, ao contribuir com o reuso de código em um nível arquitetural por meio do *design* opinativo em relação as abstrações e a arquitetura da aplicação (JACOBSEN et al., 1999).

A arquitetura de camadas MVC (*Model-View-Controller*) é largamente utilizada no desenvolvimento de aplicações e *frameworks* para Web (FOWLER, 2002). Essa arquitetura foi originalmente proposta como um *framework* para desenvolvimento de aplicações na linguagem *SmallTalk* (KRASNER; POPE, 1988).

Em seu livro, Fowler (2002), destaca que a principal característica dessa arquitetura é a separação entre a apresentação e o modelo, de forma que os componentes da apresentação possam depender do modelo, mas não o contrário. Assim, os controladores serviriam como uma interface entre apresentação e modelo, desacoplando-os.

2.2.1 Classificação dos *frameworks*

Shan e Hua (2006) propõe que os *frameworks* para aplicações Web podem ser classificados em cinco diferentes categorias segundo sua arquitetura: *Request-Based*, *Component-Based*, *Hybrid*, *Meta* e *Rich Internet Application*. Essa definição, entretanto, já apresenta determinado grau de obsolescência, não permitindo um enquadramento completo dos diferentes tipos de *framework* para aplicações Web existentes atualmente, isto pode ser visto pela falta de representantes modernos para categorias como *Hybrid* e *Meta*, ou mesmo pela dificuldade em enquadrar nessas categorias *frameworks* específicos para construção de APIs e *Web Services*.

Apesar dessa possível defasagem entre as definições e os *frameworks* modernos, apresentaremos as cinco categorias propostas em linhas gerais, delimitando as características que permitem identificar seus representantes:

- Request-Based

  Essa classificação compreende os *frameworks* que utilizam um conjunto de abstrações bastante próximas da especificação CGI (W3C, 2009). Nesses *frameworks*, as URLs e métodos HTTP são mapeados diretamente para classes e métodos, chamados de *controllers* e *actions*, respectivamente.

  O desenvolvedor ao implementar uma *action* tem acesso a um conjunto de abstrações que representam os detalhes da requisição HTTP recebida, como cabeçalhos, *cookies* e dados enviados junto à requisição. É responsabilidade da aplicação gerar a resposta que será enviada ao cliente, utilizando um conjunto de abstrações que remonta a estrutura da resposta.

  Grande parte dos *frameworks* Web na linguagem Java pertencem a essa categoria, como *Struts*, *Grails* e *Spring MVC*.

- Component-Based

  Nesse tipo de *framework* os detalhes das requisições e respostas são ocultos do desenvolvedor por um conjunto de abstrações que representam componentes do *frontend*. Esses componentes, que podem mapear elementos da página HTML, encapsulam toda a lógica da aplicação por meio da resposta

a eventos gerados nos componentes. Os *frameworks Component-Based* utilizam um conjunto de abstrações mais próximo do desenvolvimento de aplicações para *desktop*, com um fluxo de controle baseado na resposta a eventos. Nessa categoria temos *frameworks* como *Java Server Faces*, *Tapestry*, *Wicket* e *GWT*.

- Hybrid

  Essa abordagem reúne aspectos dos *frameworks Request-Based* e *Component-Based*, a razão para isso é obter a reusabilidade e as abstrações de alto nível dos componentes, sem a perda do controle da arquitetura de requisições e respostas natural às aplicações Web. Essa arquitetura possui poucos representantes modernos, como o *framework RIFE*.

- Rich Internet Application

  Os *frameworks Rich Internet Application* transferem grande parte das responsabilidades da aplicação para o *frontend*, por meio de páginas HTML "enriquecidas" com outras tecnologias que permitam maior interatividade com o usuário e a aplicação das regras de negócio da aplicação diretamente no *frontend*.

  As páginas Web enriquecidas tem um longo histórico de *frameworks* e tecnologias que foram aplicadas para tornar a experiência na Web mais dinâmica para o usuário, no entanto, muitas dessas tecnologias caíram em desuso nos anos recentes em favor de outras que permitissem uma melhor padronização e compatibilidade com os navegadores Web. Com isso, tornaram-se populares *frameworks* desenvolvidos com a linguagem JavaScript, como AngularJS e React, que quando integrados a *frameworks* voltados para o *backend* permitem o desenvolvimento de aplicações Web completas. Nesse caso, a aplicação Java no servidor se resume a um conjunto de serviços, geralmente implementados como uma API, que são consumidos pela aplicação cliente. As interações entre cliente e servidor são marcadas pela transferência de dados em formatos como XML ou JSON. Nessa categoria em geral são utilizados *frameworks* como o próprio Spring MVC ou Struts, integrados a *frameworks* JavaScript para composição do *frontend*.

- Meta

  Frameworks Meta possuem um conjunto básico de serviços necessários às aplicações Web, enquanto mantém uma estrutura altamente extensível.

Esse tipo de *framework* serve como esqueleto para a composição da aplicação por meio de outros *frameworks* especializados. O *framework Keel* representa essa categoria.

2.3 Software Multilinguagem

As aplicações Web são compostas de artefatos desenvolvidos em diferentes linguagens, cada qual voltada a um domínio específico. Enquanto o uso de diversas linguagens traz vantagens significativas, pois pode-se utilizar a ferramenta mais adequada para cada tarefa. Este tipo de software traz consigo um conjunto próprio de desafios, visto que as mudanças realizada em componentes de uma determinada linguagem se propagam pelos componentes escritos nas demais, por meio de dependências nem sempre explícitas (KONTOGIANNIS et al., 2006).

Em 1998 pelo menos 1/3 dos sistemas de software eram desenvolvidos com duas ou mais linguagens de programação e cerca de 10% eram desenvolvidos com três ou mais linguagens (JONES, 1998). Um estudo mais recente tendo como objeto 22 projetos de software de código aberto, demonstrou que os desenvolvedores destes sistemas de software lidam em média com 4 diferentes linguagens durante o processo de desenvolvimento (KARUS; GALL, 2011).

Este novo paradigma constitui desafios à Engenharia de Software, uma vez que, além das dependências explícitas entre os componentes de cada linguagem, deve-se também observar as dependências não explícitas, formadas por componentes de diferentes linguagens. A identificação dessas dependências é uma tarefa singular, uma vez que a interface entre as linguagens depende da forma de interação dos componentes implementados (LINOS et al., 2003). Além disso, é importante notar que diferentes linguagens possuem paradigmas e abstrações completamente distintos, o que dificulta ainda mais a identificação de dependências entre elas (FERREIRA, 2014).

2.4 Mineração de repositórios de software

A Mineração de Repositórios de Software é uma área da Engenharia de Software que busca lançar luz sobre o processo de desenvolvimento e sua evolução por meio de estudos empíricos (KAGDI et al., 2007).

Segundo a definição de Kagdi et al. (2007) os repositórios de software são qualquer fonte de informação sobre o processo de desenvolvimento do software, desde sistemas de controle de versão do código-fonte até listas de discussão entre os desenvolvedores.

Os estudos baseados na mineração de repositórios de software atendem a uma ampla gama de propósitos. Dentre os objetivos destes estudos podemos citar, a verificação do impacto de mudanças nos artefatos do software por meio de dependências lógicas (ZIMMERMANN et al., 2005), a predição da ocorrência de *bugs* e falhas no software (MOSER et al., 2008), ou a compreensão da dinâmica de equipes de desenvolvimento de software livre (BIRD et al., 2007).

Apesar do destaque recente, a mineração de repositórios de software vem sendo explorada há bastante tempo. Os trabalhos iniciais nessa área são estudos empíricos que tem como alvo sistemas de software proprietários desenvolvidos por entidades governamentais, acadêmicas e da industria. Ainda nos anos 1980, Basili e Perricone (1984) desenvolveram uma extensa investigação sobre a correlação entre a complexidade do software e sua suscetibilidade a erros. Essa investigação baseou-se em relatórios preenchidos pelos desenvolvedores durante a implementação de mudanças no software, coletados em um período de 33 meses (BASILI; PERRICONE, 1984).

As dificuldades inerentes ao trabalho de Basili e Perricone (1984), como o acesso limitado a repositórios de software e a dificuldade na extração manual de dados destes repositórios, foram por muitos anos os principais fatores limitantes ao progresso desse campo de estudo (HASSAN, 2008). Os trabalhos recentes em mineração de repositórios de software contam com mais subsídios para sua realização por conta da popularização dos sistemas de software de código aberto, bem como, pelo uso de sistemas de controle de versão de código-fonte modernos, que simplificam a obtenção e processamento das informações de desenvolvimento do software.

2.5 Impacto em mudanças de software

Alterações realizadas em componentes do software durante o processo de desenvolvimento ou manutenção podem iniciar um efeito cascata, induzindo alterações em outros componentes para que o software se mantenha consistente. Essas alterações secundárias, por sua vez, podem levar à necessidade de novas alterações em outros componentes. A extensão deste efeito é determinada pelos níveis de acoplamento dos componentes do software (YAU et al., 1978).

Enquanto algumas dessas alterações colaterais podem ser rapidamente identificadas por ferramentas de análise estática do código-fonte, como é o caso das dependências estruturais entre objetos (BRIAND et al., 1999), outros tipos de relacionamento podem não se apresentar de forma explícita, e a desatenção às mudanças necessárias nesses artefatos pode levar a obsolescência precoce desses componentes (GALL et al., 1998).

Para superar essa deficiência na identificação de possíveis acoplamentos "não estruturais" foram propostas diferentes abordagens, as quais foram classificadas por Bavota et al. (2013) em 4 diferentes categorias:

- Acoplamento Estrutural

  O acoplamento estrutural é dado pelas relações explícitas entre os componentes do código-fonte da aplicação, sua identificação é em geral realizada por meio da análise estática do código-fonte (HITZ; MONTAZERI, 1995). Uma classe possui um acoplamento estrutural com outra classe se invocar métodos ou utilizar propriedades explicitamente definidas naquela classe.

- Acoplamento Dinâmico

  As relações criadas somente em tempo de execução são chamadas de acoplamento dinâmico, e tem origem no uso de técnicas como reflexão, polimorfismo e ligações dinâmicas. Este tipo de acoplamento é identificado durante a execução do software por meio da instrumentação do seu código-fonte ou do seu ambiente de execução (YACOUB et al., 1999).

- Acoplamento Semântico

  As relações semânticas entre os diferentes componentes de um software dão origem ao acoplamento semântico, àquele formado entre componentes que não possuem uma relação estrutural explícita, porém, relacionam-se por aspectos comuns no papel desempenhado por eles dentro do software. As técnicas para verificação deste tipo de acoplamento são geralmente baseadas em análise léxica, agrupando componentes que compartilham um vocabulário comum (POSHYVANYK et al., 2009).

- Acoplamento Lógico

  As dependências lógicas, ou acoplamento lógico, tem origem no processo de desenvolvimento do software e são identificadas por alterações que ocorrem simultaneamente, seja um conjunto de alterações armazenadas simultaneamente no sistema de controle de versão, ou alterações relacionadas a correção de um *bug* ou implementação de uma *feature* (ZIMMERMANN et al., 2005).

2.6 Identificação do acoplamento lógico

Gall et al. (1998) introduziu o conceito de acoplamento lógico ao demonstrar a deficiência das técnicas existentes (FENTON, 1996; OFFEN; JEFFERY, 1997) na iden-

tificação de dependências evidenciadas pelo histórico de alterações do software. Seu estudo baseou-se no histórico de alterações realizadas durante 20 versões de um software de telecomunicações, verificando a existência de módulos deste software que eram comumente alterados em conjunto.

Os estudos de Ying et al. (2004) e Zimmermann et al. (2005) trouxeram uma nova perspectiva sobre o acoplamento lógico. Esses estudos demonstraram a aplicabilidade dos algoritmos de regras de associação na identificação do acoplamento lógico. Obtendo bons resultados na predição de artefatos passíveis de alteração dada uma alteração inicial em outro artefato.

A origem deste tipo de acoplamento durante o processo de desenvolvimento foi objeto de estudo de Oliva et al. (2011), onde verificou-se que principais fontes para este tipo de acoplamento foram alterações relacionadas a *cross-cutting concerns*, ou seja, alterações em atividades secundárias da aplicação cuja implementação se espalha por todo o software (ex. a escrita de logs de atividade), e alterações em componentes semanticamente relacionados, como classes pertencentes a um mesmo nível arquitetural.

2.7 Regras de associação

Dada a natureza evolutiva do acoplamento lógico, as técnicas mais comuns para sua detecção baseiam-se na mineração de repositórios de software, com destaque ao uso da técnica de mineração de dados conhecida como regras de associação.

Regras de associação são utilizadas para determinar a frequência com que dois ou mais itens aparecem associados em um determinado conjunto de transações. Essa técnica possui diversas aplicações, sendo a mais popular o seu uso em mecanismos de recomendação que permitam determinar a probabilidade de um cliente comprar determinado item baseado nos itens que ele adicionou ao seu carrinho de compras. No contexto deste estudo as regras de associação são utilizadas para determinar a frequência com que certos artefatos de software são alterados simultaneamente.

As regras de associação são geradas a partir de um conjunto de transações T, tal que $T = \{t_0, t_1, t_2, \ldots, t_n\}$. Cada transação é composta de uma combinação de itens do conjunto $I = \{i_0, i_1, i_2, \ldots, i_k\}$. Dessa forma, uma transação individual pode ser expressa como $t_0 = \{i_0, i_1, i_2, i_3\}$. Utilizando um algoritmo de regras de associação, como Apriori, pode-se obter a partir do conjunto de transações T um conjunto de regras R, onde cada regra r_p tem a forma $x_1 \Rightarrow x_2$, tal que $x_1 \subseteq I$ e $x_2 \subseteq I$

e $x_1 \cap x_2 = \emptyset$ (AGRAWAL; SRIKANT, 1994). Assim, no contexto deste estudo as transações são representadas por *commits* do sistema de controle de versão, onde cada transação é composta de um conjunto de artefatos que foram modificados naquele *commit*.

Na regra de $x_1 \Rightarrow x_2$ o item x_1 é chamado de antecedente, e o item x_2 é chamado de consequente. Como notado por ZIMMERMANN et al., as regras de associação devem ser interpretadas como uma probabilidade dada pelas evidências fornecidas em T, dessa forma o símbolo $\Rightarrow$ na regra de associação não pode ser lido como uma implicância absoluta, mas como a possibilidade de x_2 ocorrer dado que x_1 ocorre, sendo determinada pelas medidas de suporte e confiança das regras.

O suporte de uma determinada regra de associação $r = x_1 \Rightarrow x_2$ é dado pelo número de transações em T que contém simultaneamente os itens dos conjuntos x_1 e x_2. O suporte pode ser absoluto, expresso pelo número de transações, ou relativo, expresso como um percentual ou fração do número total de transações.

Enquanto o suporte é capaz de informar o quão frequente é uma regra dentro do conjunto de transações, a confiança diz respeito a força de uma determinada regra de associação. Dessa forma a confiança é determinada pelo número relativo de ocorrências de x_2 no subconjunto de transações onde x_1 ocorre.

Exemplo: Dado o conjunto de itens $I = \{page.jsp, style.css, Controller.java, Entity.java\}$, e as transações $t_0 = \{page.jsp, style.css\}$, $t_1 = \{page.jsp, Controller.java\}$, $t_2 = \{style.css, Entity.java\}$ e $t_3 = \{page.jsp, style.css, Controller.java, Entity.java\}$. Podemos derivar a regra de associação $r_0 = (\{page.jsp\} \Rightarrow \{Entity.java\})$, com suporte relativo $suporte(r_0) = 1/4$, uma vez que em apenas 1 transação do total de 4 temos a ocorrência simultânea dos itens $page.jsp$ e $Entity.java$, a confiança dessa regra por sua vez pode ser expressa como $confianca(r_0) = 1/3$, pois $page.jsp$ ocorre em 3 transações, enquanto $Entity.java$ ocorre em apenas 1 dessas 3 transações. Importante notar que enquanto o suporte é o mesmo para $r_0 = (\{page.jsp\} \Rightarrow \{Entity.java\})$ e $r_1(\{Entity.java\} \Rightarrow \{page.jsp\})$, a confiança dessas regras não é a mesma, sendo $confianca(r_1) = 1/2$, pois $Entity.java$ ocorre em 2 transações, enquanto $\{page.jsp, Entity.java\}$ ocorre em somente 1 transação.

3 ESTUDO DO ACOPLAMENTO LÓGICO EM PROJETOS SPRING MVC

Neste capítulo, é apresentado um estudo prévio ao desenvolvimento do estudo comparativo. Este estudo permitiu o desenvolvimento das ferramentas e técnicas utilizadas posteriormente na análise comparativa entre os *frameworks*, avaliando alguns dos aspectos mais relevantes para o desenvolvimento do trabalho, como a classificação dos papéis arquiteturais e a extração das regras de associação para análise do acoplamento lógico.

Este estudo foi desenvolvido tendo como objeto um conjunto de projetos baseados no *framework* Spring MVC. Os resultados foram analisados sob a ótica de um conjunto de questões de pesquisa preliminar voltadas para a avaliação da arquitetura de projetos desenvolvidos com o *framework* Spring MVC.

3.1 Objetivo

Este estudo prévio tem como objetivo principal caracterizar as aplicações que utilizam o *framework* Spring MVC com relação às suas dependências lógicas, para compreender o impacto do uso deste *framework* sobre o acoplamento entre componentes de diferentes papéis arquiteturais.

Além do objetivo principal, este estudo deverá avaliar a aplicação do método de busca por dependências lógicas nos papéis arquiteturais das aplicações investigadas, permitindo verificar a pertinência dessa técnica ao estudo comparativo posterior.

3.2 Questões de pesquisa

Por meio desse estudo pretende-se responder as seguintes questões de pesquisa preliminar (QPP):

- **QPP1** - Quais componentes arquiteturais do *framework* Spring MVC são mais sensíveis a mudanças nas páginas da aplicação?

- **QPP2** - O acoplamento entre páginas e componentes é uniforme entre os projetos que utilizam o *framework* Spring MVC?

- **QPP3** - Como se dá o acoplamento entre os demais componentes dos projetos que utilizam o *framework* Spring MVC?

3.3 Metodologia

Para desenvolvimento deste estudo foram avaliados projetos de software livre que possuem como base o *framework* Spring MVC. A escolha deste *framework* se deu por sua popularidade e pelo fato de ele utilizar um conjunto de papéis arquiteturais bem definidos e facilmente identificáveis.

A seleção dos projetos utilizados nesse estudo, bem como o método de identificação dos papéis arquiteturais foi extraída da tese de Doutorado de Aniche (2016). Optou-se pela utilização deste *dataset* por se tratar de um conjunto de dados recente, composto por um grande número de projetos para os quais algumas condições de interesse deste estudo se aplicam, como, possuir um histórico de desenvolvimento com pelo menos 500 *commits* e pelo menos 10 classes com papéis arquiteturais definidos pelo *framework*. O *dataset* obtido no trabalho original era composto de 120 projetos, porém, devido a indisponibilidade de seis destes projetos, o *dataset* utilizado neste estudo ficou reduzido para 114 projetos.

Tabela 3.1 - Ocorrência dos itens nos projetos analisados.

Item	Projetos	%
Controller	114	100.00
JSP	95	83.33
Component	89	78.07
HTML	88	77.19
Service	81	71.05
Entity	65	57.02
Repository	55	48.25
HTM	9	7.89
JSPX	5	4.39

Fonte: Produção do autor.

A identificação dos papéis arquiteturais ligados ao *framework* Spring MVC seguiu o método definido no trabalho de Aniche (2016), apontando como papéis arquiteturais as classes que possuem as seguintes anotações: *@Controller*, *@Service*, *@Entity*, *@Repository* e *@Component*.

Esses papéis, no entanto, pertencem apenas às camadas *Model* e *Controller* das aplicações, sendo necessário um método alternativo na identificação dos componentes da camada *View*. Com esse intuito, utilizou-se as extensões de nome de arquivo

htm, *html*, *jsp* e *jspx* para identificar páginas HTML e JSP como integrantes dessa camada.

Conforme a Tabela 3.1, dentre os papéis arquiteturais analisados somente os *Controllers* estão presentes nos 114 projetos, sendo os demais distribuídos em subconjuntos desses projetos.

O método utilizado para identificação das dependências lógicas é similar ao proposto por Zimmermann et al. (2005), obtendo-as por meio da aplicação do algoritmo de regras de associação *Apriori* sobre o conjunto de *commits* extraído de cada um dos projetos.

3.4 Desenvolvimento

O processo de mineração dos repositórios de código-fonte pode ser dividido em uma sequência de passos: *(i)* obtenção de cópias dos repositórios dos sistemas de controle de versão, *(ii)* extração dos nomes de arquivos alterados em cada *commit* dos projetos, *(iii)* extração do conteúdo dos arquivos Java e classificação do papel arquitetural, *(iv)* mapeamento das alterações para tipo de arquivo/papel arquitetural e remoção de duplicidades, *(v)* execução do algoritmo Apriori para geração das regras de associação, *(vi)* análise das regras de associação obtidas.

Os passos *(ii)*, *(iii)* e *(iv)* foram automatizados com a criação de uma ferramenta para extração dos dados do sistema de controle de versão *Git*. Essa ferramenta analisa de forma sequencial todos os *commits* contidos no repositório do sistema de controle de versão extraindo uma lista de arquivos alterados em cada *commit* com seu conteúdo.

Em seguida os arquivos Java são analisados em busca de anotações que permitam determinar seu papel estrutural. As classes contidas nesses arquivos foram rotuladas conforme a Tabela 3.2, e os demais arquivos foram rotulados por sua extensão.

A entrada para o algoritmo *Apriori* é dada por um conjunto de transações compostas de um ou mais itens por transação, dessa forma cada *commit* extraído no passo *(ii)* representa uma transação distinta. Os arquivos modificados em cada *commit*, por sua vez, são representados pelo rótulo de seu papel arquitetural ou por sua extensão de nome de arquivo como itens da transação, porém, uma vez que diversos itens podem possuir um mesmo rótulo, é necessário remover as duplicidades entre os itens ao final do passo *(iv)*.

Após a etapa *(iv)*, as alterações realizadas em cada projeto são representadas por arquivos no formato CSV (valores separados por vírgula), onde cada linha no arquivo representa uma transação e os registros presentes em cada linha representam os itens dessa transação.

Tabela 3.2 - Rótulo dos papéis arquiteturais do *framework* Spring MVC

Anotação	Rótulo
@Controller	java:controller
@Service	java:service
@Entity	java:entity
@Repository	java:repository
@Component	java:component
-	java:other

Fonte: Produção do autor.

As regras de associação são obtidas por meio da execução do algoritmo *Apriori* sobre as transações. A implementação do algoritmo *Apriori* utilizada neste estudo foi realizada e descrita por Borgelt (2003), essa implementação é largamente reconhecida e utilizada em diversas bibliotecas e aplicativos de mineração de dados, como no pacote *arules* da linguagem R (HAHSLER et al., 2007; BORGELT, 2003). Essa implementação inclui um aplicativo de linha de comando que pode ser utilizado para obtenção das regras de associação a partir de arquivos contendo valores separados por vírgula.

A implementação em linha de comando do algoritmo *Apriori* permite a customização de sua execução por meio de uma série de parâmetros. Para este estudo, foram utilizados parâmetros com objetivo de remover filtros de valores mínimos de suporte e confiança (*-s0 -c0*), definir o tipo de saída (*-tr*) e ajustar o número de itens por regra de associação (*-n2 -m2*). As regras de associação foram obtidas a partir de um conjunto de dados contendo 117.551 transações pertencentes aos 114 projetos analisados.

Após a execução do passo *(v)*, as saídas correspondentes a cada projeto são consolidadas em um único arquivo composto de sete campos: *(project)* nome do projeto, *(consequent, antecedent)* itens consequente e antecedente da regra de associação, *(support, con_support, ant_support)* suporte da regra, do item consequente e do item antecedente, e *(confidence)* confiança da regra de associação.

A Tabela 3.3 exibe o número de transações nas quais ocorrem os papéis arquiteturais definidos pelo *framework* Spring MVC, páginas HTML e JSP, arquivos Java não classificados (*java:other*) e arquivos XML (*xml:other*). Fica claro que o componente arquitetural com maior presença nas transações, ou seja, com maior número de modificações, são os *Controllers*, seguidos dos componentes do tipo *Service*.

Tabela 3.3 - Número de transações que contém itens de interesse do estudo.

Item	Transações	
java:component	9.017	7,67%
java:controller	20.699	17.61%
java:entity	5.736	4,88%
java:repository	6.342	5,40%
java:service	17.056	14,51%
java:other	65.851	56,02%
htm:other | html:other	9.356	7,96%
jsp:other | jspx:other	16.182	13,77%
xml:other	33.720	28,69%

Fonte: Produção do autor.

Dentre as transações envolvendo arquivos JSP e HTML, os arquivos JSP estão presentes em 13,77% das transações, enquanto os arquivos HTML são encontrados em 7,96% delas. Outro ponto que chama a atenção deste estudo é o grande número de transações com itens não classificados e com arquivos XML, as quais denotam a necessidade de uma análise mais aprofundada para identificação do papel arquitetural destes itens.

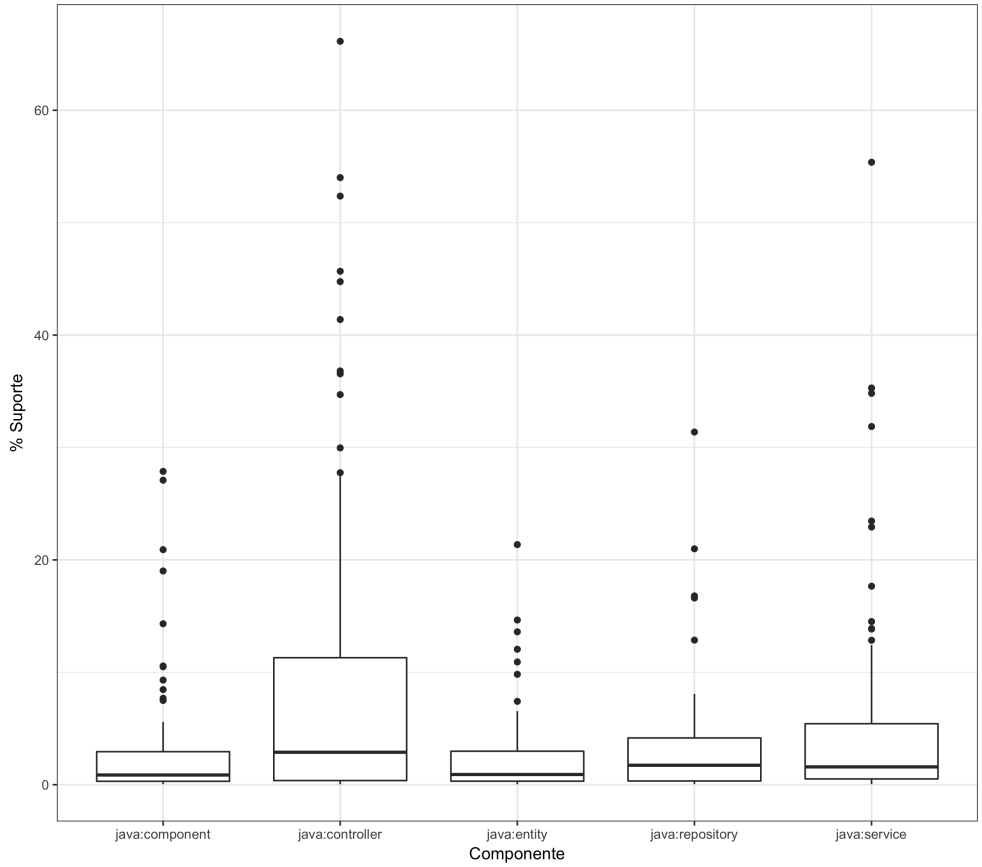
3.5 Resultados e análise dos dados

Nesta seção são apresentados os dados obtidos pelo uso da técnica de mineração por regras de associação. Em seguida, esses dados são analisados segundo as questões de pesquisa preliminar propostas na Seção 3.2.

As regras de associação formadas entre páginas da camada *View* e componentes das demais camadas arquiteturais podem ser analisadas segundo duas dimensões específicas, o suporte e a confiança. O suporte permite verificar o quão comum são essas associações em relação ao total de modificações realizadas nos componentes do projeto, já a confiança permite determinar a força dessas associações dentre o total de modificações em páginas.

O *box-plot* exibido na Figura 3.1 mostra a distribuição do suporte nas associações entre páginas e demais componentes arquiteturais. Nesse gráfico é possível verificar que essas associações compõe uma pequena fração do total das alterações realizadas nos projetos, sendo a associação mais comum aquela verificada entre páginas e *controllers*.

Figura 3.1 - Suporte na associação entre páginas e componentes arquiteturais.



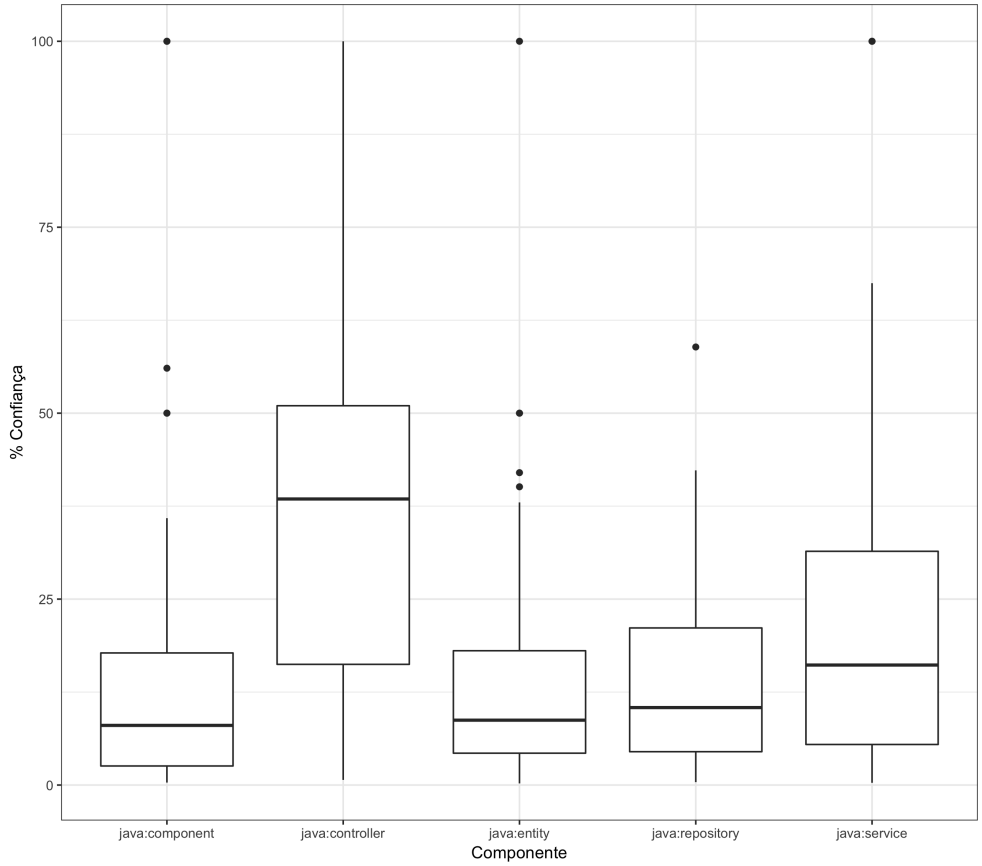
Fonte: Produção do autor.

A confiança das associações entre páginas e componente arquiteturais é exibida na Figura 3.2. Novamente, os *controllers* destacam-se em relação aos demais componentes, com metade dos projetos possuindo valores de confiança entre 16,2% e 51%. Esses valores de confiança denotam que grande parte das mudanças em páginas acarretam em mudanças nos componentes *controller*.

Além da associação entre páginas e demais componentes, é importante analisar a forma como se dá o acoplamento entre os componentes das camadas *Controller*

e *Model*, os dados de suporte e confiança dessas associações são apresentados na Tabela 3.4 e na Figura 3.3, respectivamente.

Figura 3.2 - Confiança na associação entre páginas e componentes arquiteturais.



Fonte: Produção do autor.

A Tabela 3.4 exibe a distribuição do suporte nas associações entre os componentes das camadas *model* e *controller* ordenados de forma decrescente pelo valor de sua mediana. A associação com maior valor mediano é formada entre *controllers* e *services*. Destaca-se o fato das três associações com maior valor médio envolverem o componente *service*, enquanto as três associações com menor valor médio envolvem o componente arquitetural *component*.

A Figura 3.3 traz um panorama geral da confiança nas associações entre os diferentes componentes, as colunas do gráfico representam os antecedentes, enquanto as linhas representam os consequentes. Destaca-se nesse gráfico a confiança das associações que possuem como consequentes os componentes *controller* e *service*, cujos

Tabela 3.4 - Suporte das associações entre componentes arquiteturais.

Componente A	Componente B	Min	Q1	Mediana	Q3	Max
java:controller	java:service	0.07	1.40	6.22	14.32	55.38
java:repository	java:service	0.24	1.09	5.03	11.72	34.16
java:entity	java:service	0.07	0.55	3.32	8.30	23.63
java:controller	java:repository	0.10	0.79	2.81	8.97	38.72
java:controller	java:entity	0.05	0.93	2.62	6.79	24.03
java:component	java:controller	0.05	0.72	2.45	5.31	26.34
java:entity	java:repository	0.16	0.51	2.39	6.81	31.38
java:component	java:service	0.04	0.93	2.13	7.38	28.89
java:component	java:entity	0.10	0.52	2.01	3.41	14.56
java:component	java:repository	0.20	0.45	1.09	3.73	14.74

Fonte: Produção do autor.

níveis de confiança estão acima dos demais. Por outro lado a confiança das associações quando esses componentes se encontram no antecedente não apresenta níveis significativamente diferentes dos demais componentes.

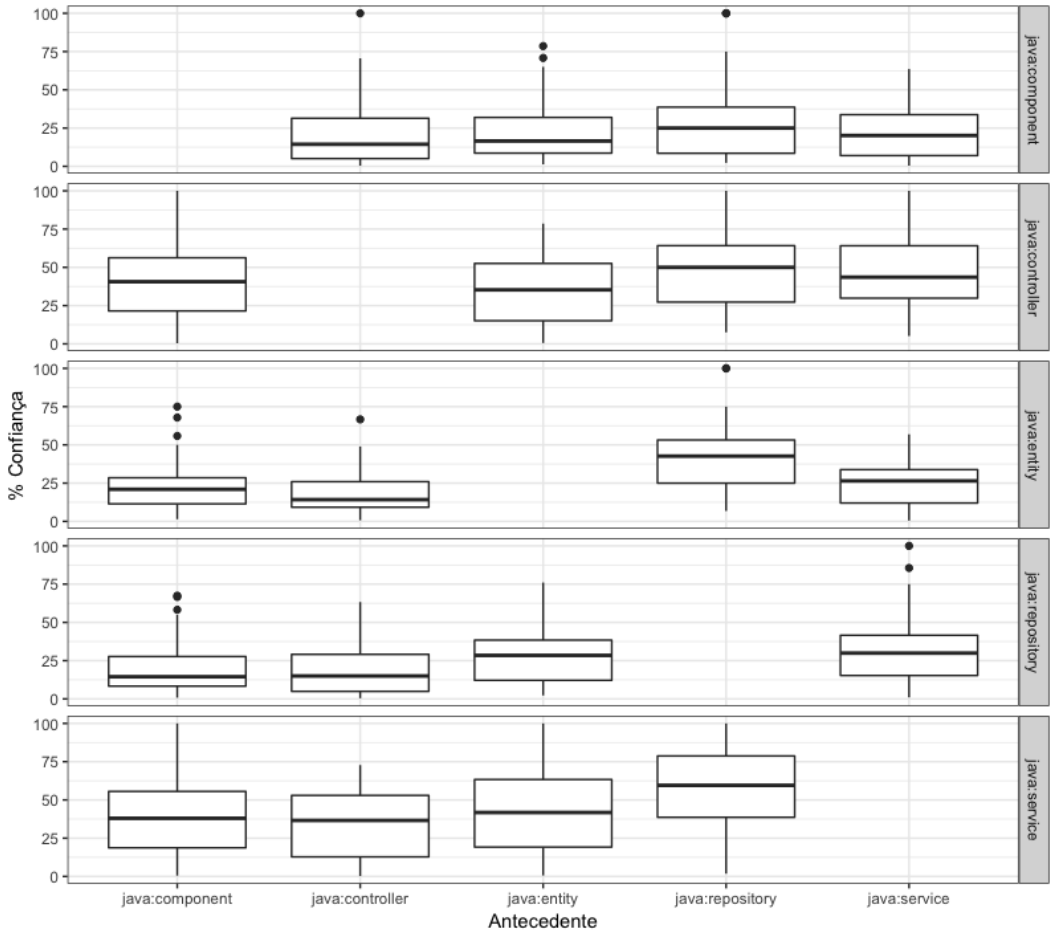
3.5.1 Resposta às questões de pesquisa preliminar

Nesta seção são analisados os dados obtidos à luz das questões de pesquisa preliminar definidas anteriormente, analisando a arquitetura das aplicações desenvolvidas com o *framework* Spring MVC com relação ao acoplamento entre os componentes do *backend* e do *frontend* das aplicações.

(QPP1) - Quais componentes arquiteturais do *framework* Spring MVC são mais sensíveis a mudanças nas páginas da aplicação? Os resultados obtidos por meio das regras de associação permitem concluir que os componentes do tipo *controller* são os mais sensíveis as mudanças ocorridas nas páginas da camada *view*. Essa sensibilidade pode ser verificada pelos seguintes aspectos: *(1)* o suporte mediano da associação entre páginas e *controllers* é consideravelmente mais alto do que aquele encontrado com outros componentes arquiteturais, *(2)* a confiança mediana das associações entre *controllers* e páginas é 2x mais alta do que a encontrada nos demais componentes.

Enquanto a confiança *(2)* provê uma indicação da força das associações entre *controllers* e páginas, e portanto da sensibilidade desses componentes, os valores superiores de suporte *(1)* dessa associação, demonstram que ela é por ampla margem a mais comum entre os componentes arquiteturais e páginas.

Figura 3.3 - Confiança na associação entre componentes arquiteturais.



Fonte: Produção do autor.

(QPP2) - O acoplamento entre páginas e componentes é uniforme entre os projetos que utilizam o *framework* Spring MVC? Os projetos analisados possuem valores de suporte entre páginas Web e componentes arquiteturais distribuídos de maneira bastante compacta, com até 75% das associações distribuídas abaixo dos 10% de suporte. Enquanto esse pode ser um bom indicador da homogeneidade do acoplamento, é preciso salientar a ocorrência de um grande número de *outliers* nessa métrica.

Apesar disso, os valores de confiança entre as associações de páginas e componentes arquiteturais possui uma distribuição um pouco mais ampla, no entanto com um número bastante inferior de *outliers*.

Esses resultados mostram a existência de um padrão para o acoplamento entre páginas Web e componentes arquiteturais, porém, uma investigação aprofundada acerca

das origens dos valores mais dispares poderia ser de grande valia para investigação de possíveis grupos de aplicações com características semelhantes.

(QPP3) - Como se dá o acoplamento entre os demais componentes dos projetos que utilizam o *framework* Spring MVC? Dentre as associações entre componentes arquiteturais, aquelas com maior valor de suporte estão relacionadas aos *services*, enquanto os *components* estão menos associados a alterações em outros componentes arquiteturais. O mesmo ocorre com relação a confiança dessas associações, nas quais o componente *service* possui um valor acima dos demais quando tratado como consequente da associação.

Esse resultado demonstra que os *services* são os componentes mais sensíveis a alterações em outros componentes, particularmente aos *controllers*, *repositories* e *entities*. Isso pode indicar que os *services* atuam como componente intermediário entre as alterações em páginas e os componentes de persistência de dados.

3.5.2 Validade dos resultados

Os resultados obtidos neste estudo, enquanto capazes de corroborar o uso da técnica de dependências lógicas, devem ser melhor investigados antes de qualquer conclusão sobre o acoplamento entre páginas e componentes nos projetos investigados. O resultado obtido não difere do esperado por uma breve análise da arquitetura do framework Spring MVC, porém, alguns fatores dificultam conclusões mais aprofundadas sobre esses resultados.

Um dos pontos que carecem de uma investigação mais aprofundada é com relação as associações formadas por outros componentes da camada *view*, como *script* e *folhas de estilo*. Além disso, foi possível verificar neste estudo a existência de um grande número de artefatos do formato XML, que podem ter papel de configuração para os artefatos Java ou mesmo para os artefatos da camada *view*.

A identificação dos componentes arquiteturais neste estudo seguiu a técnica proposta por Aniche (2016), apesar de representar de maneira bastante precisa os componentes arquiteturais do *framework* Spring MVC, essa técnica leva a um grande número de arquivos sem identificação do seu papel arquitetural. Dessa forma, é importante que essa técnica seja estendida de forma a identificar outros arquivos que não possuem anotações do *framework* Spring MVC.

A granularidade da classificação dos componentes arquiteturais, e demais artefatos, deve ser melhor explorada. Neste estudo agrupou-se as alterações dos *commits* se-

gundo sua extensão de nome de arquivo, ou seu papel arquitetural na aplicação. Essa classificação, porém, poderia ser melhor explorada com a definição de outros papéis que abrangessem conjuntos de arquivos de propósito similar, como folhas de estilo CSS e arquivos que podem ser usados para geração de folhas de estilo, como Less e Sass, ou então, arquivos JavaScript e arquivos que podem ser *transpilados* para essa linguagem, como TypeScript e CoffeScript, ou mesmo arquivos XML segundo o seu propósito na aplicação.

4 COMPARAÇÃO DO ACOPLAMENTO LÓGICO ENTRE *FRAMEWORKS* WEB

Neste capítulo, é descrito o estudo comparativo realizado entre diferentes tipos de *frameworks* para aplicações Web na linguagem Java. Nas seções seguintes apresentamos a motivação para este estudo, as questões de pesquisa que nortearam o seu desenvolvimento e a metodologia utilizada na seleção dos *frameworks*, dos projetos e na obtenção dos dados de classificação dos papéis arquiteturais e do acoplamento lógico.

4.1 Motivação

O estudo preliminar apresentado no Capítulo 3 teve por objetivo a análise da arquitetura inerente aos projetos desenvolvidos com o *framework* Spring MVC, bem como a verificação do método de análise do acoplamento lógico entre componentes de software. A partir da experiência obtida no estudo preliminar foi desenvolvido este estudo que aborda um contexto mais amplo, comparando as características arquiteturais de projetos desenvolvidos com três categorias de *frameworks* distintas, dada sua grande popularidade no desenvolvimento de aplicações Web, são elas: *request-based*, *component-based* e *rich-internet-application*.

Para uma efetiva comparação dos *frameworks* e arquiteturas analisadas foi definido um novo conjunto de papéis arquiteturais de caráter geral. Esse conjunto de papéis identificou componentes com características semânticas e funcionais semelhantes entre os *frameworks*, abstraindo os detalhes de implementação de cada um deles.

A heurística utilizada para determinação dos papéis arquiteturais durante o estudo preliminar, apesar de bastante precisa, acarreta em um grande número de componentes não identificados, além de ser específica ao *framework* Spring MVC.

Faz-se necessário, portanto, utilizar um conjunto mais amplo de heurísticas, baseadas em outros tipos de marcadores que permitam determinar com razoável clareza o papel desempenhado pelos componentes de cada aplicação.

Dessa forma, foi necessário o desenvolvimento de uma ferramenta e de um conjunto de heurísticas capazes de identificar os papéis arquiteturais entre os diversos projetos e *frameworks* analisados.

4.2 Questões de Pesquisa

Para atender ao objetivo deste estudo foram definidas as seguintes questões de pesquisa (**QP**):

- **(QP1)** O acoplamento entre componentes do *backend* e do *frontend* é semelhante para *frameworks Request-Based* e *Component-Based*?
- **(QP2)** A abordagem *Rich-Internet-Application* (RIA) altera o acoplamento entre componentes do *backend* e *frontend* quando comparada a abordagem *Request-Based*?
- **(QP3)** Como o acoplamento entre componentes do *backend* é afetado pelas abordagens *Request-Based*, *Rich-Internet-Application* e *Component-Based*?
- **(QP4)** Como se dá o acoplamento de componentes do *frontend* ou *backend* com componentes de configuração, como arquivos XML?

4.3 Metodologia

As questões de pesquisa apresentadas neste estudo visam comparar o acoplamento entre componentes de software desenvolvidos com base em diferentes tipos de *frameworks* e abordagens arquiteturais, nesse contexto faz-se necessária a seleção de um conjunto de *frameworks* capaz de atender ao escopo determinado pelas questões de pesquisa, as quais abrangem a comparação de três categorias distintas de *frameworks*, sendo: *(a) Request-Based*, *(b) Componet-Based* e *(c) Rich-Internet-Application*.

A escolha dos *frameworks* cujos projetos seriam objeto deste estudo baseou-se no relatório de panorama da linguagem Java publicado pela empresa Zero Turnaround (2016). Assim, foram selecionados para este estudo os *frameworks* Spring MVC e Java Server Faces (JSF). Estes dois *frameworks* são os representantes mais populares em suas categorias, sendo o JSF um *framework Component-Based*, enquanto o *framework* Spring MVC pode ser utilizado como *Request-Based* quando usado de forma independente, ou *Rich-Internet-Application* quando em conjunto com bibliotecas de *frontend* JavaScript.

Para a categoria *Rich-Internet-Application* foram selecionados projetos que combinam o *framework* Spring MVC com o *framework* JavaScript AngularJS, visto a

grande popularidade dessa combinação. Apesar do uso de um único *framework*, essas categorias têm abordagens arquiteturais bastante diferentes, uma vez que grande parte das funcionalidades do *frontend* é transferida para a aplicação JavaScript executada no navegador.

Um dos aspectos fundamentais para o desenvolvimento deste trabalho foi a busca e obtenção de projetos desenvolvidos com esses *frameworks*, dessa forma, optou-se por utilizar como fonte de busca a plataforma de hospedagem de repositórios de software GitHub, a qual concentra grande parte dos projetos de software livre desenvolvidos atualmente.

As ferramentas providas pela plataforma GitHub, no entanto, não são suficientes para a realização de uma busca criteriosa por projetos que atendam aos objetivos deste trabalho. Assim, foi necessária a utilização da ferramenta Google BigQuery, a qual permite realizar consultas na linguagem SQL sobre amplos volumes de dados em tempo muito inferior aos mecanismos de banco de dados tradicionais.

Essa busca foi realizada sobre um conjunto de dados públicos providos pelos responsáveis pela ferramenta Google BigQuery e pela plataforma GitHub. Esses dados compreendem todos os repositórios de software-livre hospedados no site GitHub, para os quais estão disponíveis diversos metadados compreendendo os repositórios, seus *commits*, desenvolvedores e arquivos, além do conteúdo dos arquivos presente na versão mais recente de cada repositório.

Após a busca, os projetos foram filtrados para garantir que se tratassem de aplicações Web baseadas nos *frameworks* e arquiteturas desejadas, desses projetos foram então selecionados aqueles com maior número de *commits* e uma cópia local dos repositórios foi realizada para posterior análise do acoplamento lógico.

A geração das regras de associação para cada um dos projetos foi realizada em três etapas distintas: (a) classificação dos arquivos em papéis arquiteturais, (b) extração dos *commits*, (c) processamento das regras de associação.

Para a etapa (a) foi desenvolvida uma ferramenta própria, disponibilizada como software de código-aberto[1], a qual permite realizar essa classificação segundo um conjunto de heurísticas configuráveis baseadas em metadados do arquivo e em seu conteúdo, essas heurísticas abordam características como: nome e extensão de nome de arquivo, caminho no sistema de arquivos, ou no caso de arquivos de código-

[1] https://github.com/edupsousa/arch-miner

fonte Java, o tipo contido pelo arquivo, importações de tipos e pacotes, herança de classes, interfaces implementadas, métodos e propriedades de classes, entre outras características.

A extração dos *commits* (b) foi realizada por meio de *scripts* os quais utilizam a ferramenta Git para listar os arquivos modificados em cada commit, armazenando em um arquivo de texto separado por vírgulas a identificação do commit e os caminhos para os arquivos alterados.

A etapa final do processo de geração das regras de associação (c) foi realizada com o cruzamento dos dados obtidos nas etapas anteriores, os quais foram utilizados como entrada para o algoritmo de regras de associação Apriori, a exemplo do Capítulo 3 utilizamos aqui o utilitário de linha de comando desenvolvidos por Borgelt (2003). Ele gera como saída um conjunto de regras de associação individual para cada um dos projetos analisados.

A análise dessas regras foi realizada com auxílio da linguagem de programação R, por meio da qual os dados foram agrupados e analisados conforme o tipo de *framework* utilizado por cada projeto, tendo por base as questões de pesquisa apresentadas na Seção 4.2. A análise numérica dos dados de acoplamento lógico permitiu a elaboração de um conjunto de hipóteses sobre a arquitetura de cada uma das abordagens analisadas.

As hipóteses baseadas na análise numérica foram então investigadas por meio da análise manual do código-fonte. Para cada uma das questões de pesquisa foram selecionados aleatoriamente 100 commits de cada abordagem arquitetural. Em seguida foram analisadas as alterações realizadas nos componentes arquiteturais classificados por este estudo, com objetivo de avaliar as hipóteses levantadas na análise numérica e aprofundar a compreensão da arquitetura dos diferentes tipos de *framework* analisados. A investigação manual do código-fonte permitiu uma melhor compreensão das origens das diversas hipóteses de acoplamento apresentadas, bem como das implicações para a manutenção do software a longo prazo.

4.4 Seleção dos Projetos

Os projetos analisados neste estudo foram obtidos a partir da plataforma GitHub, sua seleção, no entanto, foi realizada por meio da ferramenta Google BigQuery a qual possui um repositório de dados públicos contendo informações sobre projetos de software-livre hospedados no GitHub. Utilizando-se dessa ferramenta foi realizada

uma busca por projetos na linguagem Java que contivessem em seu código-fonte importações de anotações relacionadas ao uso dos *frameworks* JSF e Spring MVC, essas anotações são:

- **javax.faces.bean.ManagedBean** - Componente do *framework* JSF responsável pelo gerenciamento dos componentes visuais.
- **org.springframework.stereotype.Controller** - Componente do *framework* Spring MVC responsável pelo controle das requisições e respostas.
- **org.springframework.web.bind.annotation.RestController** - Componente do *framework* Spring MVC responsável pelo controle das requisições e respostas com funcionalidades específicas para APIs.

Os projetos que utilizam a anotação *RestController* podem ser utilizados para a construção de APIs REST, ou seja, esses projetos podem ser compostos somente de um *backend* a ser utilizado por outras aplicações. Esse tipo de software foge ao escopo deste estudo por não possuir componentes responsáveis pela exibição da aplicação ao usuário. Para evitar que esse tipo de projeto fosse incluído na seleção, optou-se por selecionar apenas aqueles que contivessem a biblioteca AngularJS, responsável pela construção de elementos do *frontend* das aplicações Web.

Além dos critérios descritos, os projetos obtidos foram ainda filtrados para remoção daqueles que utilizam simultaneamente os *frameworks* Spring MVC e JSF, ou que contenham simultaneamente anotações *Controller* e *RestController* para o *framework* Spring MVC. Esse processo de seleção resultou em um total de 1726 projetos, distribuídos conforme a Tabela 4.1.

Tabela 4.1 - Quantidade de projetos por *framework*.

Framework/Estilo	**Projetos**
JSF/Component-Based	225
Spring MVC/Request-Based	1.432
Spring MVC/RIA	69

Fonte: Produção do autor.

O próximo passo foi a seleção dos projetos com o maior número de *commits*, essa seleção teve por objetivo obter projetos mais maduros e com um maior número de

transações para geração das regras de associação, visto que as métricas de suporte e acoplamento são relativas ao número total de *commits* do projeto e ao número de *commits* onde um determinado item é modificado, respectivamente.

Por fim estabeleceu-se como critério de corte a existência de um mínimo de 50 *commits* por repositório. Esse critério possibilitou a seleção de 30 repositórios da abordagem RIA, e dessa forma foram selecionados outros 30 repositórios para cada uma das demais abordagens, selecionando aqueles com maior número de commits.

Os 90 repositórios obtidos foram então manualmente verificados, e a cada um deles foi atribuída uma das seguintes categorias:

- **Aplicação** - Aplicação Web completa desenvolvida com os *frameworks* Spring MVC ou JSF, total de 65 projetos.
- ***Framework*** - Software incompleto, o qual deve ser estendido para sua utilização, total de 8 projetos.
- **Biblioteca** - Biblioteca de funcionalidades que podem ser adicionadas a outros projetos de software, total de 6 projetos.
- **Múltiplo** - Repositórios contendo diversas aplicações independentes, total de 11 projetos.

Após a classificação, foram removidos os projetos rotulados em outras categorias além de Aplicação, restando 65 projetos distribuídos entre os *frameworks* conforme a Tabela 4.2.

Tabela 4.2 - Total de projetos selecionados por *framework*/estilo arquitetural.

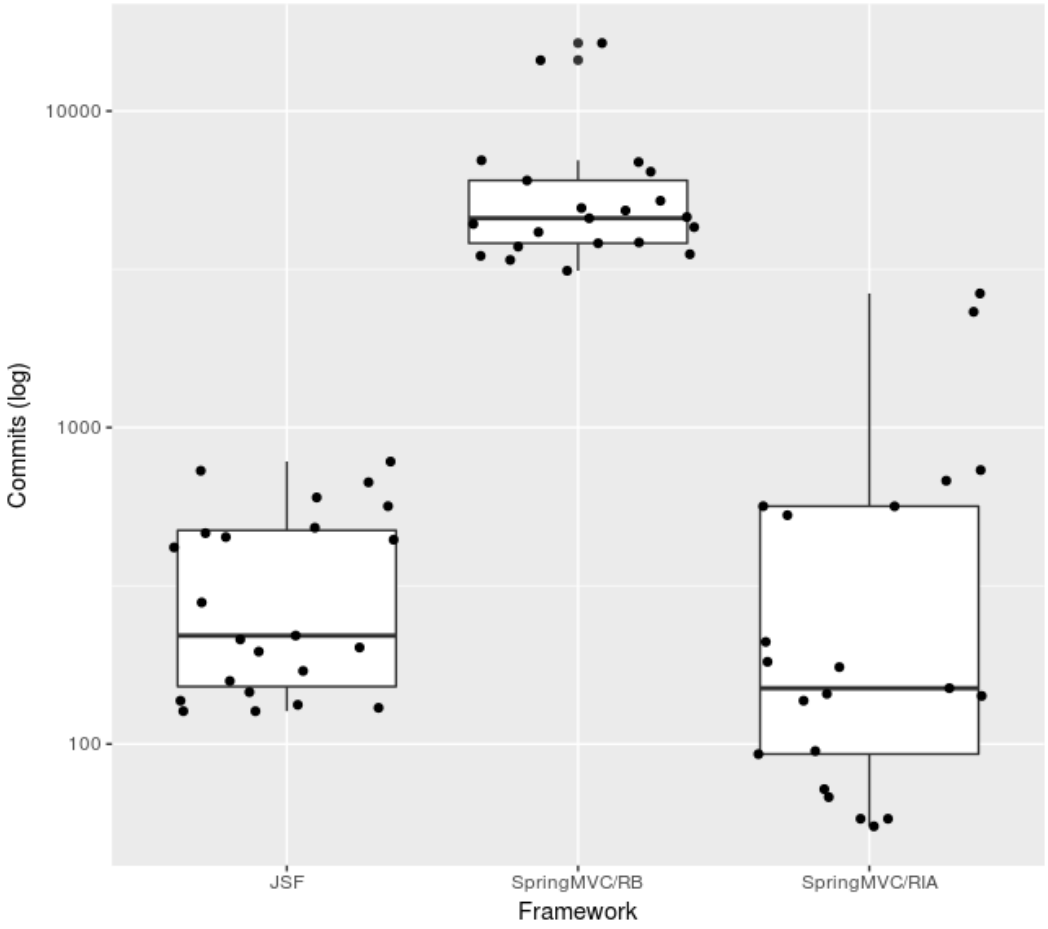
Framework/Estilo	**Projetos**
JSF/Component-Based	23
Spring MVC/Request-Based	21
Spring MVC/RIA	21

Fonte: Produção do autor.

Como visto na Tabela 4.1 a oferta de repositórios para cada um dos *frameworks* é bastante distinta, com um grande número de projetos desenvolvidos utilizando a

abordagem *Request-Based*. Essa grande diferença na oferta de repositórios se reflete nas características dos projetos selecionados.

Figura 4.1 - Distribuição do número de *commits* por *framework*.



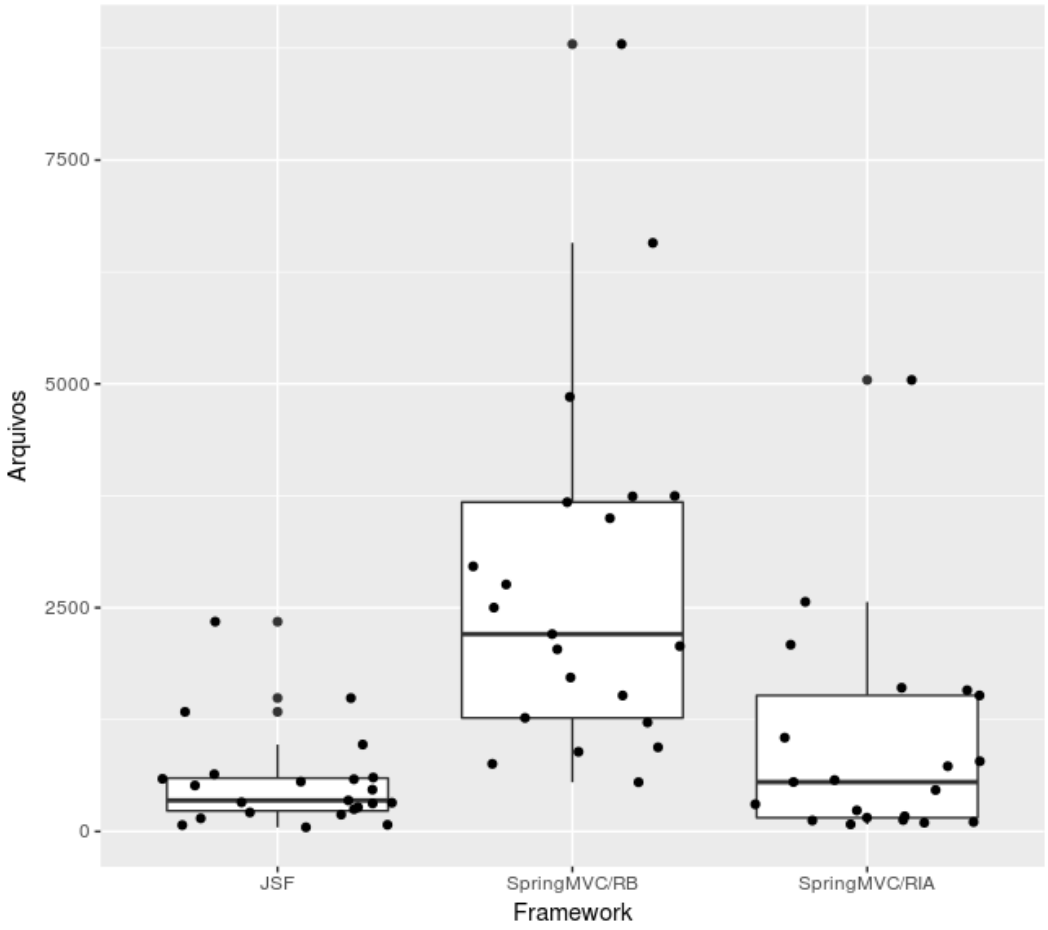
Fonte: Produção do autor.

A Figura 4.1 exibe o número de *commits* nos repositórios dos projetos desenvolvidos com cada um dos *frameworks* analisados. Percebe-se que os projetos desenvolvidos com o *framework* Spring MVC/*Request-Based* possuem um número de *commits* bastante superior aos demais *frameworks*, com 50% dos repositórios contendo entre 3.822 e 6.036 *commits*. O *framework* JSF, por sua vez, possui 50% dos repositórios distribuídos entre 152 e 474 *commits*, enquanto o *framework* Spring MVC/RIA tem 50% dos repositórios distribuídos entre 93 e 564 *commits*.

A distribuição no número de arquivos entre os repositórios de projetos que utilizam cada *framework*, como visto na Figura 4.2, guarda semelhanças com a dis-

tribuição do número de *commits*, uma vez que novamente os repositórios do *framework* SpringMVC/Request-Based possuem um maior número de arquivos, o que pode ser atribuído a maior popularidade, ou maturidade, dos projetos desenvolvidos com essa abordagem.

Figura 4.2 - Distribuição do número de arquivos por projeto separado por *framework*.



Fonte: Produção do autor.

No entanto, ainda que a maturidade dos repositórios, com relação ao número de *commits*, possa influenciar a distribuição do número de arquivos, é possível supor que as características estruturais dos projetos desenvolvidos com cada um dos *frameworks* tenha papel relevante nessa distribuição, uma vez que os projetos desenvolvidos com JSF possuem uma distribuição no número de arquivos mais compacta do que aqueles desenvolvidos com Spring MVC/RIA, ainda que os últimos possuam um menor número médio de *commits*.

4.5 Investigação dos Papéis Arquiteturais

Após a seleção dos projetos, o próximo passo para o desenvolvimento deste estudo foi a classificação dos arquivos de cada projeto segundo seu papel na arquitetura da aplicação Web, de forma similar ao realizado no estudo do Capítulo 3. No entanto, dado o contexto deste estudo novas heurísticas foram adicionadas, e os papéis arquiteturais específicos do *framework* Spring MVC foram generalizados para abranger componentes com papéis similares entre os tipos de *framework* analisados.

No trabalho de Aniche et al. (2016), o qual serviu de inspiração para a construção dessas heurísticas, foi obtido um índice aproximado de 17,5% de arquivos Java identificados como papéis arquiteturais, neste trabalho obteu-se com um conjunto distinto de projetos índices variando entre 14,6% de arquivos identificados para o *framework* Spring MVC/*Request-Based* e 39,7% para o *framework* Spring MVC/-RIA, já os projetos que utilizam o *framework* JSF tiveram um total de 20% dos arquivos Java identificados como papéis arquiteturais.

4.5.1 Papéis Arquiteturais

Uma aplicação Web pode ser dividida em dois conjuntos de componentes distintos, os quais compõe seu *backend* e seu *frontend*. Os componentes do *backend* são executados pelo servidor e geralmente são responsáveis pela persistência dos dados da aplicação, execução das regras de negócio e geração de componentes do *frontend* de forma dinâmica. Já os componentes do *frontend* são executados, ou exibidos, pelo navegador Web do usuário e tem funcionalidades voltadas para exibição e interação com o usuário.

Para efeito deste estudo os artefatos das aplicações Web analisadas foram separados em oito papéis arquiteturais distintos, quatro deles pertencentes ao *backend* e especificados na linguagem Java, três deles pertencentes ao *frontend*, e um único cuja utilização pode ser feita tanto no *backend* quanto no *frontend*.

Os papéis definidos para o *backend* foram:

- Endpoint: Ponto de contato entre a aplicação *backend* e o *frontend*, responsável por interpretar e responder às requisições ou eventos do *frontend*. É a interface entre a aplicação executada no *backend* e o *frontend*.
- Service: Componentes responsáveis pela implementação das regras de negócio da aplicação.

- Entity: Representação das entidades de dados, as quais representam os registros armazenados em bancos de dados e dados utilizados pelas regras de negócio da aplicação.
- DAO: Encapsula interface de persistência da aplicação, responsável pela recuperação, inserção e atualização dos dados.

Na aplicação *frontend* foram definidos os seguintes papéis:

- Página: Páginas HTML estáticas e *templates* para geração dinâmica de páginas na linguagem JSP.
- Script: Arquivos de código-fonte na linguagem JavaScript e outras linguagens usadas para geração de JavaScript, como TypeScript ou CoffeeScript.
- Estilo: Folhas de estilo na linguagem CSS, ou em linguagens utilizadas para geração dinâmica de CSS, como Sass ou Less.

E por fim, o seguinte papel pode ser encontrado tanto no *backend* como no *frontend*:

- Configuração: Arquivox XML, JSON, YAML utilizados para configuração de componentes do *backend* ou *frontend*.

4.5.2 Heurísticas para Classificação do Papel Arquitetural

Os arquivos dos projetos analisados foram classificados com base nos papéis arquiteturais definidos anteriormente utilizando uma ferramenta desenvolvida especificamente para este trabalho. Essa ferramenta utiliza um conjunto configurável de heurísticas para classificar os arquivos da aplicação em componentes arquiteturais. Para o desenvolvimento deste trabalho foi implementado um conjunto de heurísticas tendo em vista a classificação dos componentes dos tipos de *frameworks* analisados, porém, essa ferramenta pode ser facilmente estendida pela adição de novas heurísticas ou pela criação de novas configurações para as heurísticas existentes.

Neste trabalho foram utilizados dois grupos distintos de heurísticas, o primeiro deles é aplicável a qualquer tipo de arquivo pois baseia-se nos metadados do arquivo, como:

- Expressões regulares no caminho do arquivo.

- Expressões regulares para o nome do arquivo.
- Extensões de nome de arquivo específicas.

O segundo grupo de heurísticas é aplicável a arquivos de código-fonte Java, e se utiliza das ferramentas de desenvolvimento Java do Eclipse, conhecidas como Eclipse JDT[2], para analisar o código-fonte por características como:

- Anotações em Classes e Interfaces.
- Anotações em Propriedades e Métodos.
- Herança de Classes ou Implementação de Interfaces.
- Nome da Classe ou Interface.
- Importação de Pacotes ou Tipos.
- Nome do Pacote.
- Estrutura da Classe (Quantidade de métodos, propriedades e construtores).

Para classificar as versões mais recentes dos arquivos nos repositórios dos projetos selecionados foi utilizado o *framework* RepoDriller[3], o qual permite navegar pelos arquivos de diferentes de *commits* em repositórios Git.

A seguir são descritas as heurísticas utilizadas para a classificação dos arquivos entre os papéis arquiteturais. As heurísticas para classificação dos papéis do *frontend* e configurações são baseadas em informações sobre o nome e o caminho do arquivo, enquanto as heurísticas para classificação dos papéis do *backend* são em sua maioria baseadas no código-fonte dos arquivos Java.

As heurísticas utilizadas para classificação dos papéis do *frontend*, as quais são baseadas na extensão de nome de arquivo, permitem classificar os arquivos utilizados para renderização de páginas Web estáticas e dinâmicas, *scripts* nas linguagens JavaScript e em linguagens usadas para geração de código JavaScript, como TypeScript e CoffeeScript, além de folhas de estilo CSS e linguagens para geração dinâmica de folhas de estilo, como Sass e Less, a seguir são exibidos os critérios utilizados nessa heurística:

[2]https://www.eclipse.org/jdt/
[3]https://github.com/mauricioaniche/repodriller

- Página
  - Extensão de Nome de Arquivo
    - htm
    - html
    - xhtml
    - jsp
    - jspx
- Script
  - Extensão de Nome de Arquivo
    - js
    - ts
    - coffee
- Estilo
  - Extensão de Nome de Arquivo
    - css
    - sass
    - scss
    - less

Para classificar os arquivos do papel Configuração foram utilizadas heurísticas baseadas no nome do arquivo, sua extensão e diretório no qual está armazenado. Essas heurísticas classificam de forma geral arquivos que possuem extensões de tipos de arquivo de configuração conhecidos, como XML e JSON, e também arquivos de configuração do sistemas de sistemas auxiliares ao processo de desenvolvimento, como sistemas de *build* automatizado e controle de versão. A seguir são exibidos os critérios utilizados:

- Configuração
  - Extensão de Nome de Arquivo
    - xml
    - json
    - properties

        * yml
        * config

A identificação dos papéis arquiteturais do *backend* foi baseada na análise estática do código-fonte de arquivos na linguagem Java, o método mais comum para a classificação desses papéis foi o uso de anotações de código em classes, interfaces, métodos e propriedades. O papel DAO foi o único entre os papéis do *backend* cuja classificação baseou-se em outros métodos além das anotações, sendo utilizados também métodos baseados em herança e importação de tipos.

Os arquivos identificados com o papel DAO (*Data Access Objects*) são responsáveis por implementar a interface com a camada de persistência da aplicação, recuperando, inserindo, atualizando e excluindo informações no mecanismo de persistência utilizado. As heurísticas utilizadas nesse caso focam nas anotações correspondentes a esse papel no *framework* Spring Data, além de importações do tipo *Query*, utilizado na implementação de consultas ao banco de dados com JPA (*Java Persistence API*).

- DAO
    - Anotação em Declaração de Classe
        * org.springframework.stereotype.Repository
    - Anotação em Declaração de Interface
        * org.springframework.data.rest.core.annotation.RepositoryRestResource
    - Herança em Interface (interface *extends* interface)
        * org.springframework.data.repository.Repository
        * org.springframework.data.repository.CrudRepository
        * org.springframework.data.mongodb.repository.MongoRepository
        * org.springframework.data.repository.PagingAndSortingRepository
        * org.springframework.data.jpa.repository.JpaRepository
    - Importação de Tipo
        * javax.persistence.Query

Os artefatos classificados como *Endpoints* são utilizados como ponto de contato entre os componentes do *backend* e do *frontend* da aplicação, o papel primário dos artefatos classificados com esse tipo é tratar no *backend* as requisições e as respostas para o *frontend* da aplicação, as seguintes configurações foram usadas para essa classificação:

- Endpoint
  - Anotação em Declaração de Classe
    - javax.servlet.annotation.WebFilter
    - javax.servlet.annotation.WebServlet
    - javax.faces.bean.ManagedBean
    - org.springframework.stereotype.Controller
    - org.springframework.web.bind.annotation.RestController

A representação dos dados dos mecanismos de persistência é feita pelo papel *Entity*, as heurísticas apresentadas abaixo são anotações de mecanismos comuns de persistência como Spring Data e JPA, utilizadas para anotar as classes que desempenham esse papel.

Foi utilizada também a anotação *Id*, usada para anotar a propriedade que tem papel de chave-primária para o tipo de dados representado, evitando que classes que utilizam anotações específicas para outros mecanismos de persistência deixasse de ser classificadas:

- Entity
  - Anotação em Declaração de Classe
    - javax.persistence.Entity
    - org.neo4j.ogm.annotation.NodeEntity
    - org.springframework.data.gemfire.mapping.annotation.Region
    - org.springframework.data.mongodb.core.mapping.Document
  - Anotação em Declaração de Propriedade
    - org.springframework.data.annotation.Id

O papel *Service* é utilizado para classificar os componentes responsáveis por implementar as regras de negócio da aplicação. Foram utilizadas heurísticas para identificar componentes relacionados aos *frameworks* porém com anotações de uso geral.

Essas anotações indicam a implementação de funcionalidades específicas ao domínio da aplicação, porém, acessadas por meio dos mecanismos de inversão de controle dos *frameworks* utilizados, os parâmetros para essas heurísticas foram:

- Service
  - Anotação em Declaração de Classe
    - org.springframework.context.annotation.Configuration
    - org.springframework.stereotype.Service
    - org.springframework.stereotype.Component
    - javax.ejb.Stateless
    - javax.ejb.Stateful
  - Anotação em Declaração de Interface
    - javax.ejb.Remote
    - javax.ejb.Local

Conforme a Tabela 4.3 foram analisados 212.979 arquivos, dos quais aproximadamente 36,4% foram classificados segundo as heurísticas utilizadas. Dentre os 63,6% de arquivos não classificados, pouco mais da metade deles são arquivos Java não classificados pelas heurísticas, enquanto os demais pertencem a tipos de arquivo que estão além do escopo deste trabalho.

Tabela 4.3 - Número de artefatos analisados pelas heurísticas.

Artefato	**Total**	**% do Total**
Classificados - Java	14.935	7,0%
Classificados - Outros Arquivos	63.042	29,6%
Classificados - Total	**77.437**	**36,6%**
Não Classificados	**135.542**	**63,6%**
Total	**212.979**	

Fonte: Produção do autor.

Os artefatos classificados, em sua maioria fazem parte do *frontend* da aplicação, com destaque para os *scripts* que perfazem um total de 39,7% dos artefatos analisados. A prevalência deste tipo de artefato é explicada em grande parte pela presença de inúmeras bibliotecas de terceiros junto ao código-fonte da aplicação *frontend*. A Tabela 4.4 exibe uma contagem do total de artefatos classificados em cada um dos papéis arquiteturais definidos por esse estudo.

A presença de um número significativo de arquivos não identificados é esperada, em especial de arquivos de código-fonte Java. Esses arquivos em muitos casos re-

presentam classes utilitárias, cujo papel arquitetural não pode ser bem definido de acordo com as especificações de cada um dos *frameworks* analisados. Além desses, as aplicações concentram um grande número de arquivos com papéis que vão além do escopo deste trabalho. Apesar da contribuição apresentada neste trabalho por meio das heurísticas de classificação de papéis arquiteturais fica clara a existência de grande espaço para melhoria desse método em estudos que possuam uma visão mais ampla dos papéis desempenhados por cada arquivo.

Tabela 4.4 - Número de artefatos classificados.

Aplicação	Papel	Artefatos	%
Backend	DAO	1.938	2,5%
Backend	Endpoint	3.252	4,2%
Backend	Entity	2.247	2,9%
Backend	Service	7.498	9,7%
Backend	**Subtotal**	**14.935**	**19,3%**
Frontend	Página	11.639	15,0%
Frontend	Script	30.755	39,7%
Frontend	Estilo	5.714	7,4%
Frontend	**Subtotal**	**48.108**	**62,1%**
Configuração		14.394	18,6%
Configuração	**Subtotal**	**14.394**	**18,6%**
Total		**77.437**	

Fonte: Produção do autor.

4.6 Transações

Para o processo de extração das regras de associação, e posterior análise das dependências lógicas, foram extraídos os caminhos dos arquivos alterados em cada *commit* dos repositórios analisados. Essa lista de arquivos foi posteriormente processada e os caminhos de arquivo foram substituídos pelos papéis arquiteturais identificados na Seção 4.5.1. Por fim, as transações contendo os papéis arquiteturais foram sanitizadas com a remoção dos papéis duplicados, permitindo a posterior geração das regras de associação utilizando o algoritmo *Apriori*.

Foram extraídas 101.294 transações, no entanto, a Tabela 4.5 mostra que a grande maioria dessas transações afeta um único papel arquitetural, enquanto, alterações que afetem simultaneamente cinco ou mais papéis arquiteturais ocorrem em um número inexpressivo de transações.

O grande número de transações com um único papel arquitetural pode ser explicado por dois cenários bastante distintos: (i) como uma prática do desenvolvedor ao realizar *commits* em arquivos individuais de forma semelhante ao realizado em sistemas de controle de versão como CVS, ou como (ii) alterações auto-contidas, onde a mudança em um arquivo ou em um conjunto de arquivos de mesmo papel arquitetural não se propaga para os demais papéis. A presença relativamente maior de alterações auto-contidas nos projetos do Spring MVC/RB, os quais tem um maior número de *commits*, servem de indicativo para o segundo cenário.

Tabela 4.5 - Transações por número de papéis arquiteturais envolvidos.

Papéis	**Spring MVC/RB**		**Spring MVC/RIA**		**JSF**	
1	62981	70,95%	4490	57,30%	2679	57,06%
2	16724	18,84%	1931	24,64%	1158	24,66%
3	6000	6,76%	751	9,58%	514	10,95%
4	2101	2,37%	368	4,70%	231	4,92%
5	701	0,79%	179	2,28%	80	1,70%
6	180	0,20%	59	0,75%	22	0,47%
7	59	0,07%	40	0,51%	6	0,13%
8	17	0,02%	18	0,23%	5	0,11%
Total	**88763**		**7836**		**4695**	

Fonte: Produção do autor.

A Tabela 4.6 mostra o número de transações nas quais cada papel arquitetural é modificado. Chama atenção nesses valores o grande número de modificações sofridas pelos componentes do papel Configuração, os quais são modificados em até 40% das transações nos projetos baseados no *framework* JSF. Esses dados apontam ainda para um grande número de mudanças em Páginas para o *framework* JSF, presente em 53% das transações. Os dados mostram ainda que ambas as abordagens arquiteturais baseadas no *framework* Spring MVC guardam muitas semelhanças, com uma distribuição mais próxima no percentual de modificações de alguns papéis, como *Endpoints* e Páginas.

Neste capítulo, foi discutido o processo de classificação dos papéis arquiteturais e de extração do acoplamento lógico, as heurísticas de classificação dos papéis arquiteturais utilizadas neste trabalho e os dados obtidos no processo de classificação para cada um dos *frameworks*. Também foi apresentadas uma breve discussão em relação às melhorias já realizadas e possíveis melhorias no processo de classificação.

Tabela 4.6 - Transações por papel arquitetural.

Papel	**Spring MVC/RB**		**Spring MVC/RIA**		**JSF**	
Backend/DAO	4634	5,2%	964	12,3%	244	5,2%
Backend/Endpoint	15531	17,5%	1348	17,2%	1551	33,0%
Backend/Entity	3828	4,3%	784	10,0%	580	12,4%
Backend/Service	21395	24,1%	2494	31,8%	479	10,2%
Frontend/Página	24143	27,2%	2797	35,7%	2490	53,0%
Frontend/Script	19695	22,2%	2169	27,7%	423	9,0%
Frontend/Estilo	5779	6,5%	516	6,6%	408	8,7%
Configuração	32962	37,1%	2678	34,2%	1900	40,5%
Total Transações	**88763**		**7836**		**4695**	

Fonte: Produção do autor.

Foram extraídas as regras de associação referentes ao acoplamento lógico e as regras obtidas foram consolidadas segundo o tipo de *framework* utilizado. Um sumário das regras obtidas foi apresentado para uma melhor caracterização destes dados. No próximo capítulo os resultados obtidos são discutidos sob a luz das questões de pesquisa, com uma análise quantitativa dos resultados em resposta a essas questões.

5 RESULTADOS

Este capítulo foi guiado pelas questões de pesquisa apresentadas no Capítulo 4 para analisar os resultados obtidos pela análise do acoplamento lógico. Utilizou-se os dados obtidos para responder às questões de pesquisa e posteriormente apresentar um conjunto de resultados relevantes que fogem ao escopo dessas questões. Após a análise dos resultados do estudo comparativo são discutidas possíveis ameaças à validade dos resultados e as técnicas utilizadas para mitigar essas ameaças.

5.1 Questões de pesquisa

As questões de pesquisa elaboradas no Capítulo 4 abordam o acoplamento entre os componentes de diversos papéis arquiteturais, enfatizando a comparação desse acoplamento entre os tipos de *framework*. Nesta seção são analisados os dados obtidos com o intuito de responder a essas questões.

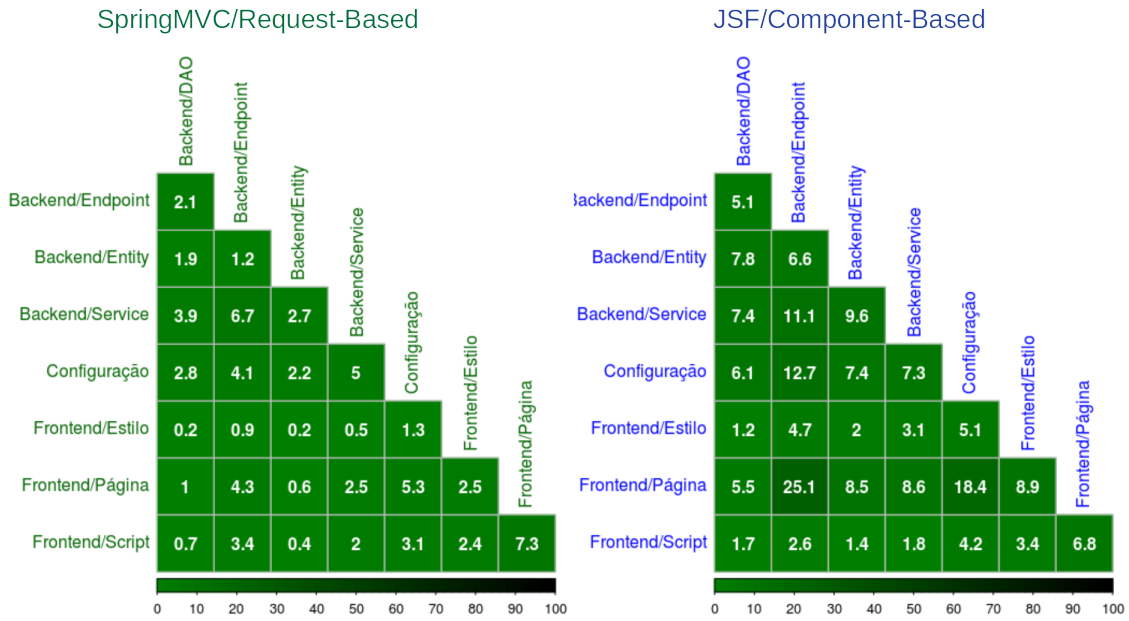
Para responder às questões de pesquisa foram analisados os valores médios de suporte e confiança para os projetos desenvolvidos com os diferentes tipos de *framework* abordados. Esses valores são apresentados em figuras com um formato matricial onde cada célula representa o valor de suporte ou confiança entre os componentes da linha e coluna onde a célula está inserida. A matriz representativa dos valores de suporte é composta somente do segmento inferior esquerdo, uma vez que os valores de suporte independem da direção da associação. Os valores de confiança, por sua vez, são apresentados em matrizes compostas de dois segmentos nos quais os componentes antecedentes da relação são indicados pelas linhas e os consequentes pelas colunas, uma vez que há dois valores distintos de confiança para cada par de componentes, de acordo com o antecedente e consequente da associação.

5.1.1 (QP1) - O acoplamento entre componentes do *backend* e do *frontend* é semelhante para *frameworks Request-Based* e *Component-Based*?

A Figura 5.1 exibe o suporte médio nas associações entre componentes de projetos desenvolvidos com *frameworks Request-Based* e *Component-Based*. Nota-se que em relação ao suporte médio, os projetos desenvolvidos com a abordagem *Component-Based* apresentam valores significativamente superiores em algumas associações quando comparados aos projetos desenvolvidos com a abordagem *Request-Based*.

Em ambos os tipos de *framework* verifica-se que o ponto de maior acoplamento entre o *frontend* e o *backend* se dá, como esperado, entre Páginas e *Endpoints*.

Figura 5.1 - Comparação do suporte médio entre as abordagens *Request-Based* e *Component-Based*.



Valores percentuais de suporte.

Fonte: Produção do autor.

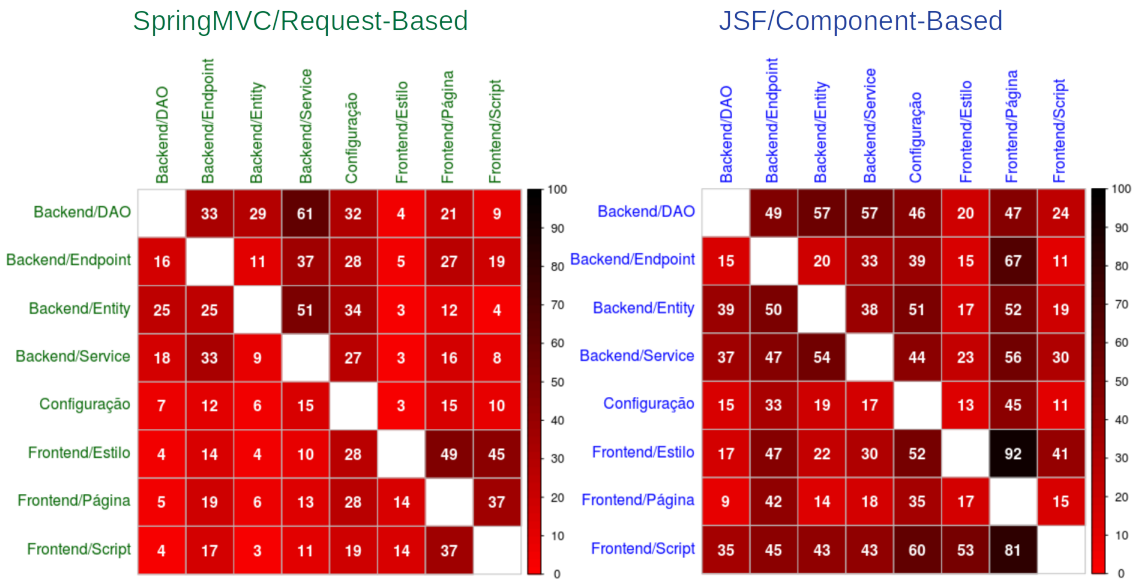
Entretanto, o acoplamento entre esses dois componentes é superior em projetos desenvolvidos com a abordagem *Component-Based*, com uma média de 25% das transações afetando estes dois papéis arquiteturais, este valor é quase 10x superior à associação entre *Endpoints* e *Scripts* nessa mesma abordagem arquitetural.

Nos projetos desenvolvidos com a abordagem *Request-Based* os níveis de suporte na associação entre *Endpoints* e Páginas, e entre *Endpoints* e *Scripts*, são bem próximos um do outro, com 4,3% e 3,4% de média, respectivamente. Chama a atenção nessa abordagem os baixos níveis de suporte nas outras associações entre papéis do *backend* e *frontend*, com valores iguais ou inferiores a 1%.

Os valores de confiança apresentados na Figura 5.2 representam a quantidade de alterações no antecedente e consequente relativa ao total de alterações no antecedente, ou seja, o quão comum são as alterações no consequente dado uma mudança no antecedente. A partir destes valores é possível verificar que a abordagem *Request-Based* apresenta um maior nível de coesão nos componentes pertencentes ao *backend* e ao *frontend* com relação a abordagem *Component-Based*. Isso é constatado pelos

valores mais altos nos quadrantes superior esquerdo e inferior direito da matriz, os quais representam a confiança nas associações entre componentes do *backend* e do *frontend*, respectivamente.

Figura 5.2 - Comparação da confiança média entre as abordagens *Request-Based* e *Component-Based*.



Valores percentuais, antecedentes nas linhas e consequentes nas colunas.

Fonte: Produção do autor.

Nos projetos desenvolvidos com a abordagem *Component-Based* percebe-se que mudanças na maioria dos papéis arquiteturais acabam acarretando em alterações nos papéis Página e *Endpoint*, sendo que alterações em componentes como *Services*, *Entities* e *DAOs*, acarretam em mudanças nas páginas de maneira bastante similar, com valores de confiança na faixa dos 50%. Já as mudanças nos *Endpoints* acarretam em mudanças nas Páginas em cerca de 67% dos casos.

A análise numérica dos dados de acoplamento lógico entre componentes do *backend* e *frontend* em projetos desenvolvidos com as abordagens *Request-Based* e *Component-Based*, em conjunto com a análise manual do código-fonte, permitiu a construção e verificação das seguintes hipóteses:

- Páginas e *Endpoints* formam o principal ponto de contato entre o *frontend* e *backend* das aplicações Web, no entanto, o nível de acoplamento entre

esses elementos é bem diferente entre as abordagens, com a abordagem *Component-Based* mostrando um nível de acoplamento mais elevado entre esses papeis arquiteturais.

- A abordagem *Request-Based* mostra uma maior coesão para os componentes do *frontend* e do *backend*, enquanto na abordagem *Component-Based* existe uma divisão mais fraca entre os papéis do *frontend* e *backend* com um acoplamento significativo entre Páginas e componentes das camadas de dados, como *DAOs* e *Entities*.

- As alterações no *frontend* são em sua maioria acarretadas por mudanças no *backend*, ou seja, o *frontend* é mais dependente e sujeito às mudanças no *backend* do que o contrário, em ambos os tipos de *framework*.

- No *frontend* da abordagem *Request-Based* o acoplamento com o *backend* ocorre de forma semelhante para Páginas e *Scripts*, isso pode indicar uma divisão de responsabilidades entre *backend* e *frontend* na implementação das regras de negócio da aplicação, com algumas tarefas como validação de entradas sendo implementadas em JavaScript.

O maior acoplamento entre Páginas e *Endpoints* na abordagem *Component-Based* chama a atenção não apenas na análise numérica, sendo bastante observado na análise do código-fonte das aplicações que utilizam essa abordagem. Sua origem se dá na arquitetura dos *Endpoints* nessa abordagem, os quais tem seus métodos e propriedades acessados pelas páginas no momento de sua renderização, criando diversos pontos de contato entre esses dois componentes. Os trechos de Código 5.1 e 5.2 exibem o código adicionado a um *Endpoint* e uma Página, mostrando o mecanismo comum de acoplamento entre esses componentes na abordagem *Component-Based*.

A arquitetura da abordagem *Component-Based* está na origem do menor nível de coesão observado entre os componentes pertencentes ao *backend* e ao *frontend*. Nessa abordagem arquitetural tem-se um nível de abstração bastante alto sobre aspectos inerentes a arquitetura Web, inclusive sobre a divisão do aplicação entre *backend* e *frontend*. As abstrações criadas por essa abordagem buscam recriar um ambiente mais próximo do encontrado no desenvolvimento de aplicações tradicionais, com funcionalidades compartimentalizadas em componentes e métodos do *backend* invocados diretamente por ações do usuário nas páginas. Assim, não se tem uma divisão clara, com componentes cujas responsabilidades e dependências permeiam igualmente o *backend* e o *frontend* da aplicação.

Código 5.1 - *Endpoint* na abordagem *Component-Based*.

```
...
@ManagedBean
public class ErrorMB {
...
    public String getServer() {
      return server;
    }
...
```

Fonte: (AURELIANO, 2015)

Código 5.2 - Página na abordagem *Component-Based*.

```
...
<div class="col-md-8">
    <h:outputText id="server" value="#{errorMB.server}"/>
</div>
...
```

Fonte: (AURELIANO, 2015)

Na abordagem *Request-Based*, o mecanismo de renderização das páginas é responsável por passar à página os dados necessários para sua renderização, o que é feito por meio de objetos de caráter geral, como vetores ou mapas, ou por meio de objetos com representações específicas dos dados para as páginas. O Código 5.3 exibe esse mecanismo, onde o mecanismo de renderização do *framework* recebe um objeto do tipo *ModelAndView*, o qual descreve a página a ser renderizada "*boxlets/top10*" e os dados que serão utilizados no processo de renderização, no caso um mapa armazenado na variável "*params*".

Quanto ao maior número alterações no *frontend* desencadeadas por alterações no *backend*, verificou-se que esse aspecto não está ligado a origem da alteração em si, mas ao maior volume de alterações no *frontend* que não afetam aspectos do *backend*, como mudanças no layout das páginas e outros aspectos visuais. Essas alterações auto-contidas entre os elementos do *frontend* faz com que a métrica de confiança nas relações entre *frontend* e *backend* acabe por ser mais alta quando elementos do *backend* estão presentes no antecedente desse acoplamento. Assim, tem-se um amplo volume de mudanças estruturais com impacto no *backend* e *frontend* simultaneamente, e um volume consideravelmente alto de mudanças auto-contidas nos elementos do *frontend*.

A abordagem *Request-Based* mostrou ainda um nível semelhante de acoplamento dos *Endpoints* com Páginas e *Scripts*. Na análise do código-fonte foi verificado um número significativo de alterações que afetam os três papeis simultaneamente, entre-

tanto, na maior parte dos casos as alterações nos *Scripts* estão diretamente ligadas as alterações nas *Páginas*, e essas por sua vez são bastante impactadas pelos *Endpoints*. Assim, não se observou uma dependência direta significativa entre *Scripts* e *Endpoints*, sendo os *Scripts* responsáveis quase que exclusivamente por lidar com aspectos da visualização das páginas, como exibição e ocultação de elementos, transições e validação da entrada do usuário.

Código 5.3 - *Endpoint* na abordagem *Request-Based*.

```
...
+@Controller
+public class TopTenBoxlet extends AbstractBoxlet {
...
+   @RequestMapping("/top10.boxlet")
+   protected ModelAndView getData(HttpServletRequest request) {
...
+       Map<String, Object> params = new HashMap<>();
+       params.put("messages", list);

+       return new ModelAndView("boxlets/top10", params);
...
```

Fonte: (MAXCOM, 2018)

Código 5.4 - Página na abordagem *Request-Based*.

```
...
+<ul>
+   <c:forEach items="${messages}" var="message">
+       <li>
+           <c:url value="${message.url}" var="msg_link">
+           <c:if test="${message.pages == 1}">
+               <c:param name="lastmod" value="${message.lastmod.time}"/>
+           </c:if>
...
+   </c:forEach>
+</ul>
...
```

Fonte: (MAXCOM, 2018)

As diferenças encontradas entre as abordagens *Request-Based* e *Component-Based* são bastante significativas, apesar de ambas as implementações se basearem na arquitetura de camadas MVC. Assim, ambas as abordagens possuem vantagens significativas naquilo que se propõe a realizar, como no caso do alto nível de abstração oferecido pela abordagem *Component-Based*, ou no maior nível de controle sobre os mecanismos das requisições Web oferecidos pela abordagem *Request-Based*.

Entretanto, foram verificadas em ambas as abordagens aspectos sensíveis, os quais são inerentes a arquitetura de cada uma delas. Esses aspectos, quando ignorados, estão sujeitos a abusos que podem levar a deterioração da arquitetura.Um dos aspectos

mais sujeitos a abuso na abordagem *Component-Based* é justamente sua abstração sobre as responsabilidades da aplicação voltadas ao *frontend* e ao *backend*.

O mecanismo por meio do qual as páginas obtém os dados necessários para sua renderização na abordagem *Component-Based*, faz com que muitas vezes os desenvolvedores passem diretamente para as páginas objetos ligados aos mecanismos de persistência, o que acarreta na execução de consultas em banco de dados e aplicação de regras de negócio invocadas diretamente pelas páginas.

Outro ponto de abuso comum nessa abordagem está na implementação de acesso a dados e de regras de negócio diretamente nos *Endpoints*, assumindo assim responsabilidades que seriam de *DAOs* e *Services*.

Na abordagem *Request-Based*, por outro lado, os abusos dados pela má divisão de responsabilidades são menos comuns. Entretanto, nessa abordagem foram verificados alguns pontos críticos dados pela duplicidade de configurações entre *backend* e *frontend*, principalmente no que diz respeito as URLs utilizadas pela aplicação. Na maioria os projetos analisados essas URLs são duplicadas entre *backend* e *frontend*, o que acarreta em um acoplamento entre componentes que em muitos casos não pode ser identificado de forma trivial.

Um ponto de acoplamento potencialmente danoso à arquitetura *Request-Based* está na passagem de objetos com baixo valor semântico, como mapas e vetores, entre *Endpoints* e Páginas para a renderização dessas últimas. Essa prática, apesar de adequada a renderização de páginas mais triviais, acarreta em dificuldades para a identificação das dependências entre *backend* e *frontend*, dificultando a manutenção da aplicação.

Um exemplo dessa prática é exibido no Código 5.3, onde um mapa contendo texto como chave, e objetos como valor, é passado para a Página exibida no Código 5.4. A Página por itera sobre a propriedade *messages* desse mapa, acessando propriedades dos objetos contidos por ela sem qualquer informação sobre o seu tipo ou suas propriedades e métodos. Esse tipo de prática pode levar a erros que somente serão detectados por meio de testes ou durante a execução do sistema.

Assim, o que se verifica é que a abordagem *Request-Based* possui vantagens significativas sobre a abordagem *Component-Based* no que diz respeito a divisão de responsabilidades entre os componentes da aplicação. Essas vantagens, entretanto, são acompanhadas de um menor nível de abstração sobre os mecanismos de funci-

onamento das aplicações Web, acarretando em uma maior transparência e responsabilidade para o desenvolvedor da aplicação no tratamento de requisições. Essas responsabilidades quando descuidadas, por sua vez, podem levar a dificuldades de manutenção e afetar a vida útil da aplicação, dadas as maiores dificuldades nessa abordagem em se identificar os pontos de acoplamento entre *backend* e *frontend*, muitas vezes criados simplesmente pelo uso de sequências de texto semelhantes.

5.1.2 (QP2) - A abordagem *Rich-Internet-Application* (RIA) altera o acoplamento entre componentes do *backend* e *frontend* quando comparada a abordagem *Request-Based*?

Os valores de suporte para a associação entre papéis da abordagem *Request-Based* e da abordagem RIA (*Rich-Internet-Application*) são exibidos na Figura 5.3. O principal aspecto que se pode verificar com relação a abordagem RIA é o nível de suporte semelhante para a associação entre Páginas e *Scripts* com os papéis arquiteturais do *backend*, os quais variam entre entre 5,5% e 7,7%.

Na abordagem RIA, assim como nas demais, a maior associação entre o *frontend* e o *backend* se dá por meio do papel *Endpoint*, no entanto, nesta abordagem não há uma diferença significativa no suporte entre os elementos do *frontend* e *backend*.

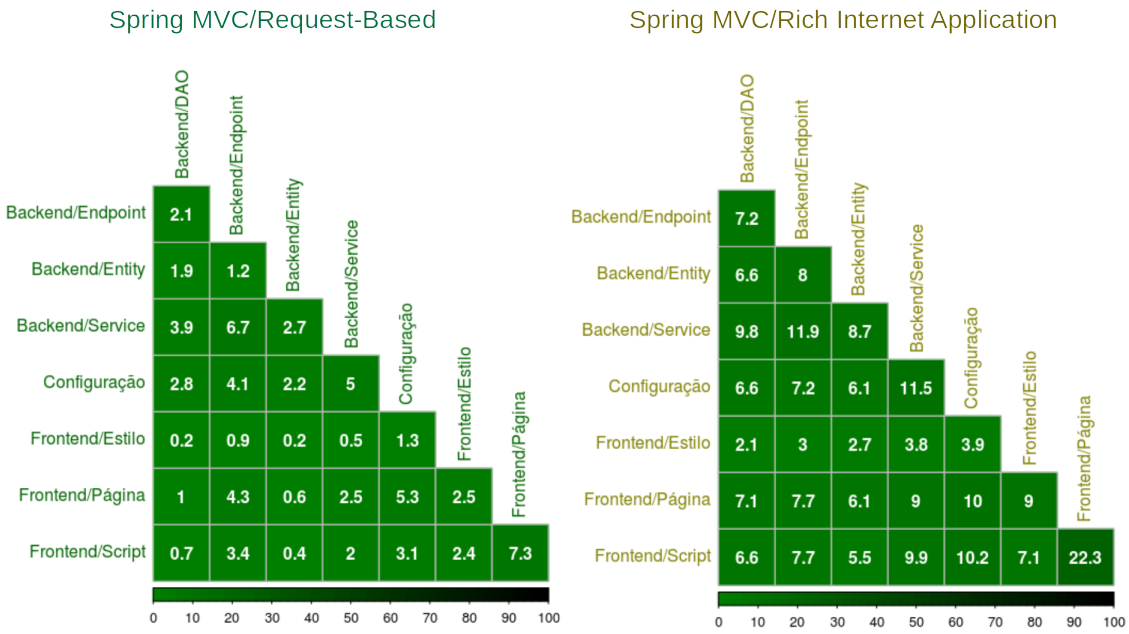
Dado o amplo uso da linguagem JavaScript na abordagem RIA, é esperado que haja um maior acoplamento entre *Scripts* e elementos do *backend*, porém, percebe-se nessa abordagem um grande aumento no suporte das associações entre Páginas e *Scripts*, o que denota um grande acoplamento entre os elementos de visuais do *frontend* e as regras de negócio e demais componentes implementados por meio da linguagem JavaScript.

Na comparação dos valores de confiança para os tipos de *framework Request-Based* e *Rich-Internet-Application* exibida na Figura 5.4, pode-se perceber algumas semelhanças entre ambas as abordagens, como a coesão nas associações de papéis pertencentes ao *frontend* e *backend*. Entretanto, os níveis de confiança para a abordagem RIA são de maneira geral mais elevados do que os apresentados pela abordagem *Request-Based*. Na abordagem RIA, a confiança nas relações entre papéis do *backend* e *frontend* apresenta níveis bastante similares para a maioria das relações, denotando um acoplamento esparso entre esses papéis.

Um dado bastante peculiar do acoplamento na abordagem RIA em relação a abordagem *Request-Based* é o alto nível de confiança nas relações que tem como antecedente

o papel Estilo e como consequente papéis do *backend* da aplicação, entre os projetos analisados as alterações no papel Estilo tem proporcionalmente maior impacto sob elementos do *backend* do que alterações em Páginas ou *Scripts*.

Figura 5.3 - Comparação do suporte médio entre as abordagens *Request-Based* e *Rich-Internet-Application*.



Valores percentuais de suporte.

Fonte: Produção do autor.

Os dados obtidos com relação ao acoplamento entre *frontend* e *backend* nas abordagens *Request-Based* e *Rich-Internet-Application* permitem concluir que:

- Ambas as abordagens possuem boa coesão entre os papéis que compõe o *backend* e o *frontend*, entretanto, enquanto na abordagem *Request-Based* a maior parte do acoplamento com o *frontend* se dá pelo componente *Endpoint*, na abordagem RIA este acoplamento é mais disperso entre os papéis do *backend*.

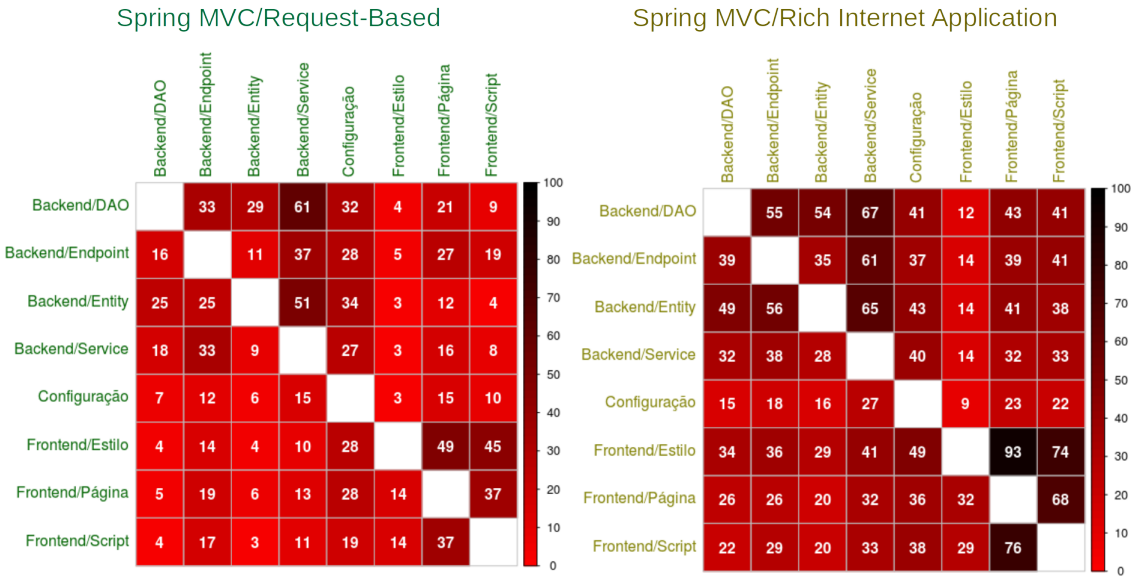
- O papel Estilo possui um nível maior de responsabilidades na abordagem RIA. Na abordagem *Request-Based*, as alterações realizadas nesse papel possuem pouco impacto sobre o *backend*, entretanto, na abordagem RIA o

backend é afetado de forma significativa por alterações no papel Estilo.

- Para ambas as abordagens, as alterações no *backend* da aplicação tem um maior impacto sobre o *frontend* do que o contrário.

A investigação do código-fonte dos projetos desenvolvidos com as abordagens *Request-Based* e *Rich-Internet-Application* mostra que essas abordagens possuem arquiteturas bastante similares, com suas principais diferenças se dando justamente no relacionamento entre os componentes do *backend* e do *frontend* da aplicação.

Figura 5.4 - Comparação da confiança média entre as abordagens *Request-Based* e *Rich-Internet-Application*.



Valores percentuais, antecedentes nas linhas e consequentes nas colunas.

Fonte: Produção do autor.

Na análise numérica da abordagem *Rich-Internet-Application* observou-se um acoplamento mais elevado de *Scripts* com componentes do *backend* além dos próprios *Endpoints*. A análise do código-fonte mostrou que esse maior acoplamento com os demais elementos é na verdade fruto de uma acentuada redução nas responsabilidades dos *Endpoints*, os quais nessa abordagem não tem por responsabilidade a renderização de páginas. Assim, em muitos casos, a responsabilidade do *Endpoint* é de simplesmente receber a requisição do cliente, repassando-a aos serviços responsáveis para em seguida converter a resposta desses serviços para o formato apropriado ao

cliente. Dessa forma, mudanças na representação dos *Services*, *Entities* e *DAOs* tem impacto direto sobre o *frontend*, sem que haja qualquer mudança na implementação original do *Endpoint*.

As listagens de Código 5.5, 5.6 e 5.7 mostram uma alteração com acoplamento entre componentes do *backend* e *frontend* na abordagem RIA, na qual o componente *Entity* do *backend* se acopla diretamente aos componentes Página e *Script* do *frontend*, sem que haja qualquer impacto no componente *Endpoint*.

Código 5.5 - *Entity* na abordagem *Rich-Internet-Application*.

```
...
@Document(indexName = "medicament_db", type = "medicaments")
public class Medicament {
...
+   private String indicationsTherapeutiques;
+
+   public String getIndicationsTherapeutiques() {
+       return indicationsTherapeutiques;
+   }
+
+   public void setIndicationsTherapeutiques(String indicationsTherapeutiques) {
+       this.indicationsTherapeutiques = indicationsTherapeutiques;
+   }
...
}
...
```

Fonte: (ELLIXO, 2017)

Código 5.6 - *Script* na abordagem *Rich-Internet-Application*.

```
...
$http
    .get('http://localhost:8080/api/v1/medicaments/' + $routeParams.codeCIS)
    .then(function(resp) {
...
+       $scope.indicationsTherapeutiques = $sce
            .trustAsHtml(medicament.indicationsTherapeutiques);
...
```

Fonte: (ELLIXO, 2017)

Código 5.7 - *Página* na abordagem *Rich-Internet-Application*.

```
...
+<p ng-if="medicament.indicationsTherapeutiques" ng-bind-html="indicationsTherapeutiques"></p>
...
```

Fonte: (ELLIXO, 2017)

Com relação ao maior acoplamento do papel Estilo com componentes do *backend*, foram verificados poucos casos onde esses componentes apresentam um acoplamento direto, como no caso de compartilharem o identificador de um elemento exibido nas

páginas. Apesar de numericamente o acoplamento entre esses componentes apresentar valores mais significantes na abordagem *RIA*, a maior parte das alterações envolvendo o papel Estilo e componentes do *backend* o faz simplesmente por conta de mudanças em páginas relacionadas a esses componentes.

Portanto, muitos dos aspectos positivos e negativos da abordagem *Request-Based* estão presentes, e em certo ponto acentuados, na abordagem *RIA*, uma vez que nessa abordagem se tem uma divisão muito mais clara entre *backend* e *frontend*. A duplicação de código-fonte entre *backend* e *frontend* é um desses aspectos acentuados, uma vez que muitas entidades e regras de negócio estão representadas em ambas as pontas da aplicação. Além disso, as responsabilidades do desenvolvedor são acentuadas, visto que este passa a ter uma responsabilidade ainda maior pelas comunicações entre *frontend* e *backend*.

Por outro lado, a divisão mais clara entre *backend* e *frontend* serve de catalisador para a criação de soluções que reduzam as duplicações de código e configurações vistas na abordagem *Request-Based*. Essas soluções baseiam-se na criação de contratos e na decrição dos pontos de contato entre *backend* e *frontend*, as APIs, permitindo que URLs, nomes de campos e outras informações que seriam duplicadas pelo *frontend* sejam obtidas de forma dinâmica pela aplicação.

5.1.3 (QP3) - Como o acoplamento entre componentes do *backend* é afetado pelas abordagens *Request-Based*, *Rich-Internet-Application* e *Component-Based*?

Com relação ao suporte nas associações entre componentes do *backend*, exibido nas Figuras 5.1 e 5.3, todas as abordagens possuem como associação mais comum àquela formada entre *Endpoints* e *Services*, com valores médios de suporte variando entre 6,7% para a abordagem *Request-Based* e 11,9% para a abordagem *Rich-Internet-Application*. As demais associações entre papéis do *backend* possuem valores de suporte bastante próximos uns dos outros, o que indica uma divisão de tarefas bastante semelhante entre os elementos do *backend* para todas as abordagens analisadas.

Quando analisada a confiança no acoplamento entre *Endpoints* e *Services*, o mais comum entre as abordagens analisadas, vê-se que as alterações realizadas nos *Services* acarretam em mudanças nos *Endpoints* com mais frequência na abordagem *Component-Based*, enquanto nas demais abordagens são os *Endpoints* que possuem maior impacto sobre os *Services*.

As alterações em componentes do papel DAO são as que possuem maior impacto sobre os demais componentes do *backend*, principalmente sobre componentes do papel *Service*. O papel *Entity* também possui grande impacto sobre os *Services* nas abordagens *Request-Based* e *RIA*, no entanto na abordagem *Component-Based* seu impacto fica abaixo dos demais componentes.

Com relação aos resultados apresentados pode-se salientar:

- O acoplamento entre os papéis que compõe o *backend* ocorre de forma semelhante entre as abordagens analisadas, indicando uma divisão semelhante de funcionalidades entre as camadas de persistência de dados, regras de negócio e controle do *frontend*.
- De forma geral os *Endpoints* são mais impactados por mudanças nos demais componentes, do que fontes de impacto para os demais. Isso indica que a maior parte das mudanças surge por alterações nos mecanismos de persistência e nas regras de negócio da aplicação, e são propagadas para os *Endpoints* e posteriormente para o *frontend*.

Dessa forma, as diferentes abordagens arquiteturais possuem uma distribuição semelhante no acoplamento entre os papéis arquiteturais do *backend*, indicando uma divisão semelhante de responsabilidades entre esses componentes. No entanto, a forma como se dá esse acoplamento possui diferenças significativas entre as abordagens analisadas.

Na abordagem *Component-Based*, os *Endpoints* são afetados de forma semelhante por todos os demais papéis do *backend*, indicando que este papel atual como consumidor dos serviços dos demais papéis. Já na abordagem *Request-Based*, os *Services* causam maior impacto sobre os *Endpoints* e são, por sua vez, bastante afetados pelos demais papéis, o que indica a utilização do papel *Service* como uma camada intermediária entre os *Endpoints* e os papéis voltados para representação e persistência de dados. Já na abordagem RIA os Endpoints são diretamente afetados por *Entities* e DAOs, denotando um acoplamento direto com as camadas de dados, o que pode ser atribuído à implementação das regras de negócio diretamente no frontend da aplicação.

A análise do código-fonte mostra que a divisão de responsabilidades entre os componentes do *backend* em cada uma das abordagens possui diferenças pontuais, mas significativas, que acabam por afetar a forma como esses componentes se acoplam.

Na abordagem *Request-Based*, é prática comum o uso de entidades específicas para a representação dos dados exibidos pela página, essa abordagem faz com que os *Endpoints* acabem por se acoplar fortemente aos demais componentes do *backend*, agindo como um intermediador entre esses componentes e o *frontend*.

Na abordagem *Component-Based* tem-se uma relação mais próxima entre os *Endpoints* e as *Páginas*, com os *Endpoints* fornecendo dados para as *Páginas* de acordo com a demanda apresentada por elas durante o processo de renderização por meio de chamadas a métodos do *Endpoint*. Essa arquitetura acaba por implicar em uma divisão menos clara das responsabilidades dos componentes do *backend*, e em alguns dos casos observados em uma concentração excessiva de responsabilidades nos componentes do papel *Endpoint*.

A análise do código-fonte das aplicações mostra que os componentes do *backend* das diferentes abordagens, com exceção dos *Endpoints*, possuem implementações bastante semelhantes, em muitos casos compartilhando as mesmas bibliotecas de persistência e representação de dados nos papéis *DAO* e *Entity*. As principais diferenças encontradas se dão pelo papel *Endpoint*, o qual possui uma implementação semelhante, mas diferentes níveis de responsabilidade entre as abordagens *Request-Based* e *Rich-Internet-Application*, e uma implementação bastante diferente na abordagem *Component-Based*.

Os efeitos das diferenças entre as abordagens se dão principalmente pela divisão mais tênue entre as responsabilidades de cada papel arquitetural na abordagem *Component-Based*, onde em diversos casos observou-se a implementação de consultas a banco de dados diretamente nos *Endpoints*. Já a abordagem *Request-Based* possui responsabilidades mais claras entre os papeis do *backend*, com uma ligeira tendência a sobrecarga dessas responsabilidades nos *Endpoints*, a qual é praticamente inexistente na abordagem *Rich-Internet-Application*.

5.1.4 (QP4) - Como se dá o acoplamento de componentes do *frontend* ou *backend* com componentes de configuração, como arquivos XML?

Com relação ao suporte, ou seja, a associação mais comum, tem-se um resultado semelhante em termos dos componentes do *backend* e *frontend* para a abordagem *Request-Based* e *Component-Based*, onde *Endpoints* e Páginas tem o maior grau de associação com os componentes de configuração. A abordagem *Rich-Internet-Application*, entretanto, possui um nível de suporte mais alto para a associação com *Services* e Scripts, sendo esse último pouco superior a associação com Páginas.

A confiança dessas associações, por sua vez, mostra que as páginas são bastante sensíveis a mudanças no papel Configuração para todas as abordagens analisadas, no entanto, no *backend* há uma maior sensibilidade para *Endpoints* na abordagem *Component-Based*, e *Services* nas demais abordagens. As *Entities* são o papel do *backend* que acarretam no maior número de mudanças no papel Configuração entre todas as abordagens analisadas, enquanto no *frontend* há uma grande variação, com *Scripts* para a abordagem *Component-Based*, Páginas e Estilo para a abordagem *Request-Based*, e Estilo para a abordagem *Rich-Internet-Application*.

Nessa questão os arquivos de configuração são tratados de maneira bastante ampla e genérica, visto que as heurísticas utilizadas se baseiam amplamente no tipo de arquivo, como XML ou YAML, sem considerar o seu conteúdo. Entretanto, fica claro nessa análise que o uso dado aos arquivos de configuração é bastante diversificado e geralmente mais associado ao *framework* utilizado do que à abordagem arquitetural em si.

Para responder a essa questão de pesquisa devem ser levados alguns aspectos relevantes extraídos da análise quantitativa em consideração:

- O *framework* JSF, por exemplo, apresenta uma associação bastante elevada entre Configuração, Páginas e *Endpoints*, o que pode ser explicado pelo uso de arquivos XML no mapeamento do endereço (URLs) de seus *servlets*. Já no *framework* Spring MVC esse tipo de mapeamento comumente realizado por meio de anotações, diretamente no código fonte da aplicação.
- Já na abordagem RIA ocorre uma grande associação das Configurações com *Scripts* e *Services*, que pode ter origem na combinação entre os *frameworks* Spring MVC e AngularJS, sendo usados para mapeamento dos pontos de contato entre o *frontend* e o *backend* e na descrição das entidades de dados entre o *frontend* e o *backend* da aplicação.
- Um ponto comum a todos os *frameworks* está no acoplamento entre as Configurações e as *Entities*, cuja origem está no uso de mecanismos de persistência que utilizam XML para descrever as entidades de dados armazenadas por eles.

A análise do código-fonte mostra que as diferentes abordagens analisadas possuem aplicações semelhantes para os arquivos XML, assim como outros tipos de arquivo de configuração. Algumas das aplicações comumente encontradas para esses arquivos

são a (i) declaração de dependências em bibliotecas externas, (ii) internacionalização das mensagens e interfaces da aplicação, (iii) configuração de mensagens e níveis de *log* da aplicação e (iv) configuração de repositórios de persistência de dados.

As principais diferenças no uso destes arquivos entre as diferentes abordagens se dão por aplicações específicas dos *frameworks* analisados, como a configuração de *servlets* no *frameworks JSF* e pelo uso de bibliotecas específicas, como JPA e Hibernate para configuração dos mecanismos de persistência.

As listagens de Código 5.8 e 5.9, mostram um tipo de acoplamento comumente encontrado entre as diversas abordagens, onde um arquivo XML é utilizado para configuração do mecanismo de persistência de dados atrelado a um componente do tipo *Entity*.

Código 5.8 - Arquivo XML de configuração do provedor de dados Hibernate.

```
...
+   <persistence-unit name="DataFlow">
+       <provider>org.hibernate.ejb.HibernatePersistence</provider>
+       <jta-data-source>java:jboss/datasources/PostgreSQLDS</jta-data-source>
+       <class>com.arjuna.databroker.data.jee.store.DataFlowEntity</class>
...
```

Fonte: (RISBIC, 2016)

Código 5.9 - Entidade de Representação de Dados Configurada por XML.

```
...
+@Entity
+public class DataFlowEntity implements Serializable
+{
+   private static final long serialVersionUID = -396262545228429102L;
...
```

Fonte: (RISBIC, 2016)

Um ponto interessante notado na análise do código-fonte é o baixo compartilhamento desses arquivos de configuração entre elementos do *backend* e *frontend* das aplicações, o que só é feito de forma mais efetiva na abordagem *Component-Based*. Sendo usado como mecanismo de configuração para vincular eventos ocorridos nas Páginas a *Endpoints* no *backend* da aplicação.

Assim, nessa questão, temos entre os tipos de *framework* analisados diferentes abordagens na tentativa de remover do código-fonte da aplicação a descrição de recursos utilizados por elas, como URLs e modelos de dados.

No entanto, percebe-se que essas abordagens incorrem em um alto acoplamento desses componentes de configuração com o código-fonte da aplicação, o que pode

justificar abordagens mais próximas do código-fonte, como anotações, em alguns casos.

5.2 Análise Complementar

Nesta seção são analisados alguns pontos relevantes levantados durante o desenvolvimento deste trabalho mas cujo escopo foge às questões de pesquisa definidas. A análise desses pontos é relevante pois oferece novas perspectivas sobre os dados analisados e a oportunidade de estudos futuros.

Um dos pontos que merece destaque nessa análise, diz respeito a popularidade dos frameworks e abordagens arquiteturais utilizados. Como exposto no Capítulo 4, durante o processo de seleção dos projetos visualizou-se uma grande discrepância entre o número de projetos qualificados em nosso processo de seleção para cada um dos tipos de *framework* abordados.

A abordagem *Request-Based* possui um número de projetos 6x maior do que a abordagem *Component-Based* e 20x maior do que a abordagem *Rich-Internet-Application*. Os impactos dessa discrepância são discutidos na seção de ameaças à validade, os números encontrados, porém, servem de balizador para a realização de novos estudos sobre as abordagens arquiteturais utilizadas pelos diversos *frameworks*.

Os resultados encontrados em termos de suporte e confiança nas associações possuem algumas distinções claras de acordo com a abordagem arquitetural utilizada. Esses valores podem ser utilizados para comparação e análise individual de projetos, servindo como referência para a verificação das características próprias a cada abordagem. Também podem servir de parâmetro para a construção de regras de conformidade arquitetural específicas para cada abordagem, permitindo uma verificação constante de possíveis desvios na arquitetura dessas aplicações.

Na análise comparativa evitou-se uma comparação direta dos valores de suporte e confiança entre as abordagens encontradas, dadas as possíveis diferenças decorrentes dos diferentes níveis de maturidade entre os projetos de cada *framework*, entretanto, os níveis de acoplamento encontrados entre as abordagens foram consideravelmente menores para a abordagem *Request-Based*. Essa diferença se justifica quando levamos em consideração o fato de 70% das transações afetarem um único papel arquitetural nessa abordagem, mostrando uma maior coesão dos papéis arquiteturais nessa abordagem.

Outro ponto interessante se dá com relação a abordagem *Rich-Internet-Application*, na qual pela divisão de responsabilidades entre *frontend* e *backend*, esperava-se um menor acoplamento entre seus papéis arquiteturais. Entretanto, constatou-se que nessa abordagem o acoplamento se dá de forma mais acentuada entre papéis diversos do *frontend* e *backend*, ou seja, pela divisão de responsabilidades os componentes do *frontend* acabam por ter um maior acoplamento com papéis do *backend* voltados ao tratamento de regras de negócio e persistência de dados, caracterizando uma possível quebra na arquitetura de camadas MVC.

Além dos resultados obtidos na verificação do acoplamento lógico, é importante ressaltar àqueles obtidos na classificação dos papéis arquiteturais. Foram analisados dois diferentes *frameworks* com três abordagens arquiteturais distintas, destes, aproximadamente 80% são parte do *frontend* ou arquivos de configuração, tendo sido classificados em sua totalidade por heurísticas baseadas no caminho do arquivo, seu nome ou sua extensão. Já no *backend*, quase todos os papéis arquiteturais foram classificados por meio de anotações em classes, interfaces ou métodos, apenas o papel DAO requereu o uso de outras heurísticas além das anotações para sua classificação. No caso desse papel arquitetural, as heurísticas baseadas em herança foram responsáveis por cerca de 20% das classificações, e aquelas baseadas em importações de tipos e pacotes identificaram outros 22% dos componentes desse papel arquitetural.

O acoplamento com arquivos de configuração foi analisado sob a perspectiva da arquitetura das aplicações em nossas questões de pesquisa, porém, como notado anteriormente esse uso é em muitos casos um subproduto do *framework* utilizado, podendo haver diferentes *frameworks* com uma mesma abordagem arquitetural que fazem uso de mecanismos distintos. Nos resultados destacou-se a utilização desses arquivos de configuração em dois pontos distintos, no mapeamento entre os *Endpoints* da aplicação e as URLs, e também na definição da camada de persistência de dados. As alterações em arquivos de configuração são bastante próximas para todas as abordagens analisadas, variando entre 34% para a abordagem *Rich-Internet-Application* e 40% para a abordagem *Component-Based*.

5.3 Ameaças à Validade

Os estudos empíricos baseados na mineração de repositórios de software encontram um conjunto de desafios, principalmente no que concerne a possibilidade de generalização dos resultados obtidos. Neste estudo tentamos mitigar esse aspecto por meio da seleção de um conjunto abrangente de aplicações Web para cada um dos tipos de *framework* analisados. No entanto, tem-se em consideração que conclusões mais

gerais sobre a arquitetura inerente a cada tipo *framework* só poderiam ser estabelecidas por meio da análise de um maior número de *frameworks* para cada tipo, além da inclusão de projetos de código proprietário na análise.

Ainda em relação ao processo de seleção dos projetos, um dos aspectos de maior preocupação foi a seleção de projetos com um número significativo de *commits*, para que as regras de associação obtidas pudessem representar de forma significativa o acoplamento entre os artefatos de cada projeto. Porém, a seleção de projetos obtida possui projetos com quantidades de *commits* bastante diferente, o que pode indicar grandes diferenças em seus níveis de maturidade. Essa ameaça é ainda mais evidente quando se compara a distribuição no número de *commits* para cada tipo de *framework* analisado. Para mitigar os efeitos dessas diferenças evitou-se a comparação direta entre os percentuais de suporte obtidos para cada tipo de *framework* dando uma maior enfase na análise da distribuição dos valores entre os componentes de cada *framework* analisado.

Com relação ao processo de identificação dos papéis arquiteturais chama a atenção o número significativo de arquivos Java e outros tipos de arquivo não classificados em nossas heurísticas. Nesse ponto é importante ressaltar que as heurísticas definidas tem como foco um conjunto de artefatos intimamente ligados a arquitetura dos tipos de *framework* analisados. Porém, em muitos pontos, a estrutura dos projetos vai além do escopo deste trabalho, como o uso de arquivos que não fazem parte da composição básica de uma aplicação Web, ou mesmo pelo uso de classes ou interfaces na linguagem Java que tem papel utilitário, ou implementam funcionalidades que não possuem ligação direta com o *framework*. Dessa forma, é esperada a ocorrência de um número significativo de artefatos não classificados. Por outro lado, as associações vistas nos resultados corroboram as relações arquiteturais dos papéis identificados.

Apesar das ameaças encontradas, os dados obtidos são consistentes com a arquitetura dos tipos de *framework* analisados, e as associações entre os componentes quando comparadas com os demais componentes do mesmo tipo de *framework* permitem concluir por diferenças significativas entre os *frameworks*, razão pela qual os resultados obtidos podem ser extrapolados para outros projetos que utilizam as mesmas abordagens arquiteturais.

6 CONCLUSÕES

O objetivo desta dissertação foi a comparação do acoplamento em aplicações Web desenvolvidas com diferentes tipos de *framework*. Por meio deste trabalho foi possível reconhecer e interpretar algumas características arquiteturais inerentes aos tipos de *framework* utilizados por aplicações Web.

Para atingir os objetivos deste trabalho, explorou-se o uso de técnicas de mineração de repositórios de software para identificação do acoplamento lógico. Essa técnica permite identificar o acoplamento entre diferentes artefatos de uma aplicação por meio da análise de seu histórico de modificações.

A avaliação dessa técnica foi realizada por meio de um estudo preliminar, no qual foram analisados 114 projetos desenvolvidos com o *framework* Spring MVC. Neste estudo, classificou-se os artefatos que compõe cada um dos projetos de acordo com seu papel na arquitetura da aplicação, utilizando como base para essa classificação o tipo de arquivo e as anotações de código presentes nas classes da linguagem Java. Por meio desse estudo preliminar, foi possível compreender como ocorre na prática algumas das convenções arquiteturais definidas pelo modelo MVC, além de avaliar a capacidade do acoplamento lógico em reconhecer as particularidades da arquitetura das aplicações inerente ao *framework* Spring MVC.

Com os resultados obtidos no estudo preliminar, foi possível verificar algumas melhorias que deveriam ser realizadas para a execução do estudo principal no que diz respeito a classificação dos artefatos em papéis arquiteturais. Dessa forma, foi estabelecido um novo conjunto de papéis e novos tipos de heurísticas que permitissem identificar e classificar artefatos de papéis arquiteturais similares entre os tipos de *framework* analisados.

Foram então selecionados três tipos de *frameworks* Web para a análise comparativa, sendo eles *request-based*, *component-based* e *rich-internet-application*. Esses tipos de *frameworks* foram representados, respectivamente, por projetos desenvolvidos com os frameworks Spring MVC, Java Server Faces e Spring MVC em conjunto com o *framework* JavaScript AngularJS.

Os projetos analisados foram obtidos da plataforma GitHub e selecionados com o auxílio da ferramenta Google BigQuery, onde foram filtrados projetos que utilizassem os *frameworks* selecionados. Desses, foram selecionados os projetos com maior número de *commits*, totalizando 65 projetos.

Após a seleção dos projetos executou-se o processo de identificação dos papéis arquiteturais e posterior extração das regras de associação, indicativas do acoplamento lógico entre os componentes dos diversos papéis.

As regras de associação obtidas para cada projeto foram agrupadas de acordo com o tipo de *framework* utilizado, e os valores agregados do suporte e confiança para cada uma das associações foi analisado com relação aos valores das demais associações de cada tipo de *framework*.

Na análise comparativa os valores de suporte e confiança para cada um dos *frameworks* foi analisado de maneira direta, pela relação entre os valores obtidos para cada um dos *frameworks*, e indireta, pela comparação das relações entre os valores de cada *framework*.

Nos resultados obtidos, foi verificado um maior acoplamento entre componentes do *backend* e *frontend* em *frameworks component-based*, assim como um maior acoplamento entre páginas e arquivos de configuração. Nos *frameworks request-based* e *rich-internet-application* verificou-se uma maior coesão entre componentes do *frontend* e *backend*.

Por meio dos resultados obtidos neste trabalho, foi possível verificar a adequação da técnica de análise do acoplamento lógico na exploração da arquitetura e do acoplamento entre artefatos de diferentes tipos e diferentes linguagens. A utilização dessa técnica abre um novo panorama para o desenvolvimento de estudos empíricos em projetos que apresentem como característica um acoplamento não explícito entre seus artefatos, permitindo um melhor entendimento das relações entre estruturas antes ignoradas por estudos focados em técnicas tradicionais de análise do acoplamento.

Os resultados indicam ainda que apesar das diferentes categorias de *frameworks* possuírem arquiteturas bastante distintas, seus diferentes componentes podem ser abstraídos em papéis arquiteturais de alto nível, os quais permitem comparar e entender o funcionamento dessas arquiteturas, subsidiando decisões de desenvolvimento sobre a arquitetura do software e permitindo a implementação de restrições arquiteturais específicas para a categoria de *framework* utilizada.

Os estudos de novos *frameworks* para desenvolvimento Web, independente da linguagem utilizada, podem ser subsidiados pelas técnicas aplicadas neste trabalho, além da possibilidade de uso dos valores de suporte e confiança encontrados neste

trabalho como uma referência para a comparação de novas categorias de *frameworks* ou mesmo de aplicações individuais quanto a adequação ao modelo arquitetural do *framework* utilizado por ela.

No contexto do desenvolvimento de aplicações para Web, este estudo apresenta um amplo panorama de características arquiteturais que permeiam as principais abordagens de *frameworks* para esse tipo de desenvolvimento. As abordagens analisadas vão desde as altas abstrações da abordagem *Component-Based*, a qual permite o desenvolvimento de aplicações com pouco ou nenhum conhecimento das engrenagens que formam uma aplicação Web, até os níveis mais próximos da própria especificação CGI (W3C, 2009), por meio do total controle sobre o mecanismo de requisição e resposta da abordagem *Rich-Internet-Application*.

Cada uma dessas abordagens possui grande valor em suas características, entretanto, relatórios técnicos (ZERO TURNAROUND, 2016) e a própria disponibilidade de repositórios encontrados neste estudo, mostram que a abordagem *Request-Based* se estabeleceu em popularidade sobre a abordagem *Component-Based*. Isso demonstra a preferência dos desenvolvedores por um conjunto de abstrações mais sútil, dando ao desenvolvedor a oportunidade de desenvolver suas próprias abstrações sobre o conjunto oferecido pelo *framework*.

A abordagem *Rich-Internet-Application* vai um passo adiante ao separar a aplicação *frontend*, de seu *backend*, representando assim duas aplicações distintas em suas plataformas, linguagem e código-fonte. Entretanto essas aplicações distintas estão profundamente ligadas em sua semântica, suas regras de negócio e suas representações de dados. Essa distinção de aplicações, por outro lado, oferece a oportunidade da utilização de técnicas e ferramentas conceitualmente distintas que melhor se adaptem a cada aplicação. É o que se vê nas versões mais recentes de *frameworks* de *frontend* como Angular.JS e React, os quais utilizam uma arquitetura baseada em componentes no *frontend*, mas lidam com *frameworks* baseados em requisições no *backend*.

6.1 Contribuições

Entre as contribuições deste trabalho pode-se citar:

- Uma metodologia para aplicação de técnicas de mineração de repositórios de software na análise do acoplamento lógico entre grupos de artefatos arquiteturalmente semelhantes.

- Um conjunto de papéis arquiteturais baseado em abstrações de alto nível, que podem ser utilizados para classificar artefatos de software com funcionalidades semelhantes entre diferentes tipos de aplicações e *frameworks* para Web.

- Um conjunto de heurísticas que podem ser exploradas para classificação de artefatos de repositórios de software de acordo com suas características estruturais, como importações, herança, anotações, métodos e propriedades. A implantação dessas heurísticas está disponível[1] como software de código-aberto na plataforma GitHub.

- Um conjunto de valores agregado por tipo de *framework* para as métricas de suporte e confiança no acoplamento lógico entre componentes de diferentes papéis arquiteturais, que pode ser utilizado para a comparação e análise de aplicações desenvolvidas com novos tipos de *framework*.

6.2 Trabalhos Futuros

Este trabalho optou por analisar os tipos de *framework* mais comumente utilizados entre as aplicações Web desenvolvidas com a linguagem Java.

No entanto, mesmo nesse contexto, muitas outras possibilidades podem ser exploradas. Exemplos são quanto aos *frameworks* analisados para cada tipo, combinações com *frameworks* e bibliotecas na linguagem JavaScript, ou mesmo quanto a análise de papéis arquiteturais mais específicos.

Na classificação dos papéis arquiteturais, as heurísticas utilizadas podem ser ampliadas para caracterização de novos papéis arquiteturais e para abranger um conjunto mais amplo de componentes de diferentes *frameworks*, ou mesmo pela criação de heurísticas puramente estruturais, agnósticas ao *framework* utilizado.

A aplicação da técnica de identificação do acoplamento lógico a um conjunto de componentes baseados no seu papel arquitetural é uma das contribuições originais desse estudo.

Isso abre caminho a utilização de novas análises baseadas no contexto ao qual os componentes da aplicação estão inseridos, permitindo novos olhares sobre estudos anteriores nos quais esse contexto não é levado em consideração.

[1] https://github.com/edupsousa/arch-miner

Apesar do foco no papel arquitetural como contexto para aplicação da análise comparativa, trabalhos futuros podem se basear em diferentes aspectos, como características de classes que extraem seus metadados de anotações versus classes puramente baseadas em código, ou configuradas a partir de componentes externos, como arquivos XML ou banco de dados.

REFERÊNCIAS BIBLIOGRÁFICAS

AGRAWAL, R.; SRIKANT, R. Fast algorithms for mining association rules in large databases. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 20, 1994. **Proceedings...** San Francisco, USA: Morgan Kaufmann, 1994. p. 487–499. 17

ANICHE, M.; TREUDE, C.; ZAIDMAN, A.; DEURSEN, A. V.; GEROSA, M. A. Satt: tailoring code metric thresholds for different software architectures. In: INTERNATIONAL WORKING CONFERENCE ON SOURCE CODE ANALYSIS AND MANIPULATION, 2016. **Proceedings...** Raleigh, USA: IEEE, 2016. p. 41–50. 5, 7, 39

ANICHE, M. F. **Context-based code quality assessment**. Tese (Doutorado em Ciência da Computação) — Universidade de São Paulo, São Paulo, 2016. 20, 28

AURELIANO. **Verbum-domini source code repository**. 2015. Disponível em: <https://github.com/aureliano/verbum-domini>. 53

BASILI, V. R.; PERRICONE, B. T. Software errors and complexity: an empirical investigation. **Communications of the ACM**, v. 27, n. 1, p. 42–52, 1984. 14

BAVOTA, G.; DIT, B.; OLIVETO, R.; PENTA, M. D.; POSHYVANYK, D.; LUCIA, A. D. An empirical study on the developers perception of software coupling. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2013. **Proceedings...** San Francisco, USA: IEEE, 2013. p. 692–701. 15

BIRD, C.; GOURLEY, A.; DEVANBU, P.; SWAMINATHAN, A.; HSU, G. Open borders? immigration in open source projects. In: INTERNATIONAL WORKSHOP ON MINING SOFTWARE REPOSITORIES, 2007. **Proceedings...** Minneapolis, USA: IEEE, 2007. p. 6. 14

BORGELT, C. Efficient implementations of apriori and eclat. In: WORKSHOP ON FREQUENT ITEMSET MINING IMPLEMENTATIONS, 2003. **Proceedings...** Melbourne, FL: IEEE, 2003. 22, 34

BRIAND, L. C.; WUST, J.; LOUNIS, H. Using coupling measurement for impact analysis in object-oriented systems. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 1999. **Proceedings...** Oxford, UK: IEEE, 1999. p. 475–482. 14

CANFORAHARMAN, G.; PENTA, M. D. New frontiers of reverse engineering. In: FUTURE OF SOFTWARE ENGINEERING, 2007. **Proceedings...** Minneapolis, USA: IEEE, 2007. p. 326–341. 8

CONALLEN, J. Modeling web application architectures with uml. **Communications of the ACM**, v. 42, n. 10, p. 63–70, 1999. 9

DIJKSTRA, E. W.; DAHL, O.-J.; HOARE, C. A. R. **Structured programming**. London, UK: Academic Press, 1972. 1

ELLIXO. **Medicament db source code repository**. 2017. Disponível em: <https://github.com/Ellixo/MedicamentDB>. 59

FAYAD, M.; SCHMIDT, D. C. Object-oriented application frameworks. **Communications of the ACM**, v. 40, n. 10, p. 32–38, 1997. 1, 10

FENTON, N. E. **Software metrics: a practical and rigorous approach**. 2. ed. Boca Raton, USA: International Thomson, 1996. 15

FERREIRA, M. d. R. Dissertação (Mestrado em Informática), **Detecção de anomalias de código de relevância arquitetural em sistemas multilinguagem**. Rio de Janeiro: [s.n.], 2014. 13

FOWLER, M. **Patterns of enterprise application architecture**. Boston, USA: Addison-Wesley Longman, 2002. 2, 10, 11

GALL, H.; HAJEK, K.; JAZAYERI, M. Detection of logical coupling based on product release history. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 1998. **Proceedings...** Bethesda, USA: IEEE, 1998. p. 190–198. 3, 7, 14, 15

GUERRA, E. M.; SOUZA, J. T. D.; FERNANDES, C. T. A pattern language for metadata-based frameworks. In: CONFERENCE ON PATTERN LANGUAGES OF PROGRAMS, 2009. **Proceedings...** Illinois, USA: ACM, 2009. p. 3. 5

HAHSLER, M.; GRÜN, B.; HORNIK, K. Introduction to arules–mining association rules and frequent item sets. **SIGKDD Explorations**, v. 2, n. 4, 2007. 22

HASSAN, A. E. The road ahead for mining software repositories. In: FRONTIERS OF SOFTWARE MAINTENANCE, 2008. **Proceedings...** Beijing, China: IEEE, 2008. p. 48–57. 14

HITZ, M.; MONTAZERI, B. Measuring coupling and cohesion in object-oriented systems. Citeseer, 1995. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.409.4862&rep=rep1&type=pdf>. 15

JACOBSEN, E. E.; KRISTENSEN, B. B.; NOWACK, P. Architecture = abstractions over software. In: TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES AND SYSTEMS, 1999. **Proceedings...** Melbourne, AU: IEEE, 1999. p. 89–99. 10

JOHNSON, R. E. Frameworks=(components+patterns). **Communications of the ACM**, v. 40, n. 10, p. 39–42, 1997. 1

JOHNSON, R. E.; FOOTE, B. Designing reusable classes. **Journal of Object-Oriented Programming**, v. 1, n. 2, p. 22–35, 1988. 10

JONES, T. C. **Estimating software costs**. New York, USA: McGraw-Hill, 1998. 13

KAGDI, H.; COLLARD, M. L.; MALETIC, J. I. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. **Journal of Software Maintenance and Evolution: Research and Practice**, v. 19, n. 2, p. 77–131, 2007. 13

KARUS, S.; GALL, H. A study of language usage evolution in open source software. In: WORKING CONFERENCE ON MINING SOFTWARE REPOSITORIES, 2011. **Proceedings...** Honolulu, USA: ACM, 2011. p. 13–22. 4, 13

KONTOGIANNIS, K.; LINOS, P.; WONG, K. Comprehension and maintenance of large-scale multi-language software applications. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 2006. **Proceedings...** Philadelphia, USA: IEEE, 2006. p. 497–500. 2, 13

KRASNER, G. E.; POPE, S. T. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. **Journal of Object Oriented Programming**, v. 1, n. 3, p. 26–49, 1988. 2, 10

LI, B.; SUN, X.; LEUNG, H.; ZHANG, S. A survey of code-based change impact analysis techniques. **Software Testing, Verification and Reliability**, v. 23, n. 8, p. 613–646, 2013. 7

LINOS, P. K.; CHEN, Z.-h.; BERRIER, S.; O'ROURKE, B. A tool for understanding multi-language program dependencies. In: INTERNATIONAL WORKSHOP ON PROGRAM COMPREHENSION, 2003. **Proceedings...** Portland, USA: IEEE, 2003. p. 64–72. 13

MAFFORT, C.; VALENTE, M. T.; TERRA, R.; BIGONHA, M.; ANQUETIL, N.; HORA, A. Mining architectural violations from version history. **Empirical Software Engineering**, v. 21, n. 3, p. 854–895, 2016. 8

MAXCOM. **Linux.org.ru source code repository**. 2018. Disponível em: <https://github.com/maxcom/lorsource>. 54

MOSER, R.; PEDRYCZ, W.; SUCCI, G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2008. **Proceedings...** New York, USA: ACM, 2008. p. 181–190. 14

OFFEN, R. J.; JEFFERY, R. Establishing software measurement programs. **IEEE Software**, v. 14, n. 2, p. 45–53, 1997. 15

OLIVA, G. A.; SANTANA, F. W.; GEROSA, M. A.; SOUZA, C. R. D. Towards a classification of logical dependencies origins: a case study. In: INTERNATIONAL WORKSHOP ON PRINCIPLES OF SOFTWARE EVOLUTION, 12; ERCIM WORKSHOP ON SOFTWARE EVOLUTION, 7, 2011. **Proceedings...** New York, USA: ACM, 2011. p. 31–40. 16

OMAN, P.; HAGEMEISTER, J. Metrics for assessing a software system's maintainability. In: CONFERENCE ON SOFTWARE MAINTENANCE, 1992. **Proceedings...** Orlando, USA: IEEE, 1992. p. 337–344. 2

POSHYVANYK, D.; MARCUS, A.; FERENC, R.; GYIMÓTHY, T. Using information retrieval based coupling measures for impact analysis. **Empirical Software Engineering**, v. 14, n. 1, p. 5–32, 2009. 15

RISBIC. **Data broker source code repository**. 2016. Disponível em: <https://github.com/RISBIC/DataBroker>. 64

RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W.; EDDY, F.; LORENSEN, W. E. **Object-oriented modeling and design**. New Jersey, USA: Prentice-Hall Englewood Cliffs, 1991. 1

SHAN, T. C.; HUA, W. W. Taxonomy of java web application frameworks. In: IEEE INTERNATIONAL CONFERENCE ON E-BUSINESS ENGINEERING, 2006. **Proceedings...** Shanghai, China: IEEE, 2006. p. 378–385. 2, 11

STEVENS, W. P.; MYERS, G. J.; CONSTANTINE, L. L. Structured design. **IBM Systems Journal**, v. 13, n. 2, p. 115–139, 1974. 1, 2

THUMMALAPENTA, S.; XIE, T. Spotweb: detecting framework hotspots via mining open source repositories on the web. In: INTERNATIONAL WORKING CONFERENCE ON MINING SOFTWARE REPOSITORIES, 2008. **Proceedings...** L'Aquila, Italy: ACM, 2008. p. 109–112. 8

TURNER, M.; BUDGEN, D.; BRERETON, P. Turning software into a service. **Computer**, v. 36, n. 10, p. 38–44, 2003. 10

W3C. **Common Gateway Inteface Specification**. 2009. Disponível em: <https://www.w3.org/CGI/>. 11, 71

YACOUB, S. M.; AMMAR, H. H.; ROBINSON, T. Dynamic metrics for object oriented designs. In: INTERNATIONAL SOFTWARE METRICS SYMPOSIUM, 6, 1999. **Proceedings...** Boca Raton, USA: IEEE, 1999. p. 50–61. 15

YAU, S. S.; COLLOFELLO, J. S.; MACGREGOR, T. Ripple effect analysis of software maintenance. In: COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 2, 1978. **Proceedings...** Chicago, USA: IEEE, 1978. p. 60–65. 14

YING, A. T.; MURPHY, G. C.; NG, R.; CHU-CARROLL, M. C. Predicting source code changes by mining change history. **IEEE Transactions on Software Engineering**, v. 30, n. 9, p. 574–586, 2004. 16

ZEPEDA, J. S.; CHAPA, S. V. From desktop applications towards ajax web applications. In: INTERNATIONAL CONFERENCE ON ELECTRICAL AND ELECTRONICS ENGINEERING, 4, 2007. **Proceedings...** Mexico City, Mexico: IEEE, 2007. p. 193–196. 9

ZERO TURNAROUND. **Java Tools and Technologies Landscape Report**. 2016. Disponível em: <https://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-2016/>. 6, 32, 71

ZIMMERMANN, T.; ZELLER, A.; WEISSGERBER, P.; DIEHL, S. Mining version histories to guide software changes. **IEEE Transactions on Software Engineering**, v. 31, n. 6, p. 429–445, 2005. 3, 5, 7, 14, 15, 16, 17, 21