UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS

Aprimoramento das Funcionalidades de uma Ferramenta para Detecção de Coocorrências entre Padrões de Projeto e *Bad Smells*

Relatório Final do POC 1

MARCO ANTÔNIO SILVEIRA SOUZA ALVES

Orientadora: Mariza Andrade da Silva Bigonha
Coorientador: Bruno Luan de Sousa

Belo Horizonte
Fevereiro, 2022

1 Introdução

A Engenharia de Software surgiu com o propósito de fornecer técnicas e métodos que auxiliem desenvolvedores na criação de software bem estruturado e com qualidade. Dentre os diversos conceitos disponibilizados, dois que estam diretamente ligados à qualidade são: padrões de projeto e *bad smell*. Padrão de projeto consiste em um conjunto de soluções gerais para problemas recorrentes em contextos específicos [Gamma et al., 1994]. Estas soluções foram propostas como uma tentativa de alcançar maior organização no desenolvimento de software. Por outro lado, *bad smell* refere-se à sintomas presentes no código fonte de um sistema que indicam possíveis problemas de projeto e necessitam de refatoração [Fowler and Beck, 1999]. Apesar de não serem considerados erro, estes sintomas contribuem negativamente para a qualidade e organização do código.

Apesar de padrões de projeto e *bad smells* serem conceitos antagônicos, alguns pesquisadores têm investigado a coocorrência dessas estruturas durante a construção de um software [Cardoso and Figueiredo, 2015, Jaafar et al., 2013, Walter and Alkhaeir, 2016, Sousa et al., 2017]. Para realizar a identificação de coocorrências envolvendo estas duas estruturas, estudos têm implementado soluções, via *scripts*, como uma forma de realizar a dectecção de forma automática [Cardoso and Figueiredo, 2015, Jaafar et al., 2013, 2016, Walter and Alkhaeir, 2016]. Contudo, os *scripts* desenvolvidos não têm sido disponibilizadas de forma pública para que outros estudos possam utilizá-las. Como uma forma de mitigar este problema, Sousa et al. [2017] desenvolveram um protótipo de ferramenta, denominada `Design Pattern Smell`, para condução de sua pesquisa. Além disso, este protótipo foi disponibilizado de forma pública para uso externo [1].

Embora `Design Pattern Smell` seja uma boa proposta e tenha um potencial grande para auxiliar outros estudos na condução de pesquisas referentes à coocorrência entre padrões de projeto e bad smells, a ferramenta possui algumas limitações. A primeira delas é o fato da mesma não se encontrar disponível de forma *online*. O protótipo dessa ferramena foi desenvolvida na linguagem Java e necessita ser baixado e instalado de forma local pelo usuário para que possa ser utilizado. A segunda limitação está relacionada com o fato de `Design Pattern Smell` não contar com gráficos e visualizações interativas que facilitem a análise do usuário. A terceira limitação é a necessidade do uso de ferramentas externas para a geração das entradas de dados necessárias, sendo elas o conjunto de padrões de projeto existentes no código, bem como os *bad smells* presentes. Desta forma, este trabalho de conclusão de curso, POC I e II, propõe o aprimoramento de *Design Pattern Smells* visando tornar a ferramenta mais acessível e completa.

[1] http://llp.dcc.ufmg.br/Products/indexProducts.html

1.1 Objetivo Geral

O objetivo do projeto é aprimorar a ferramenta, `Design Pattern Smell`, que fornecerá aos engenheiros de software, desenvolvedores e pesquisadores um meio prático de identificar possíveis coocorrências entre padrões de projeto e *bad smells* nos seus projetos, facilitando o processo de manutenção da qualidade de código e de estudos a respeito da relação entre as duas estruturas.

Atualmente `Design Pattern Smell` é disponibilizada como uma aplicação *desktop* e espera como entrada uma especificação dos padrões de projeto e *bad smells* presentes no código. O objetivo final deste trabalho é disponibilizar esta ferramenta como uma aplicação *web* de livre acesso, com melhores visualizações dos resultados e que ela encapsule as lógicas necessárias para a identificação dos padrões de projeto presentes, bem como os trechos que se enquadrem como *bad smell*.

1.2 Objetivos Específicos

Os objetivos para o POC I são: (i) adaptação e disponibilização da ferramenta para um contexto de aplicação *web*; (ii) estudo e implementação de melhores visualizações para os dados gerados; e (iii) produção de um artigo científico detalhando a ferramenta, sua aplicabilidade, funcionamento e resultados alcançados.

No POC II pretende-se encapsular no código a lógica necessária para a identificação dos padrões de projeto presentes no projeto. Pretende-se também realizar o mesmo processo para identificação dos *bad smells*, eliminando a necessiade do usuário utilizar programas de terceiros para a geração das entradas de dados atualmente necessárias.

1.3 Contribuições

Destaca-se como principais contribuições deste trabalho do POC1:

- a construção da ferramenta `Design Pattern Smell` como uma aplicação *web*, ampliando assim o seu alcance e acessibilidade quando comparado com o seu protótipo. A versão atual da ferramenta apresenta os mesmos resultados produzidos pelo seu protótipo possibilitando sua exportação para um arquivo CSV.
- Inclusão de novas visualizações para alguns dos resultados que a ferramenta gera, com o objeto de ampliar as possibiliaddes de análise dos dados.

1.4 Organização do Relatório

Este relatório está organizado da seguinte maneira: Seção 2 descreve os principais conceitos relacionados com a ferramenta `Design Pattern Smell` que possibilitaram e motivaram o seu desenvolvimento. Seção 3 apresenta, suscintamente, as tecnologias disponíveis na literatura relacionadas à coocorrência entre padrões de projeto e *bad smells* e termina mostrando a diferença entre `Design Pattern Smell` e as ferramentas existentes sobre esse tema. Seção 4 descreve as etapas principais do processo de desenvolvimento de `Design Pattern Smell`. Seção 5 apresenta as visualizações implementadas, mostra o funcionamento da ferramenta e apresenta os resultados obtidos. Seção 6 conclui este relatório e apresenta algumas propostas de trabalhos futuros.

2 Referencial Teórico

`Design Pattern Smell` baseia-se nos conceitos de padrões de projeto e *bad smells*. Assim sendo, esta seção está organizada da seguinte forma: Seção 2.1 detalha o conceito de padrões de projeto. Seção 2.2 descreve o conceito de *bad smells* e a Seção 2.3 discute as principais funcionalidades atuais de `Design Pattern Smell`.

2.1 Padrões de Projeto

Padrões de projeto são um conjunto de soluções gerais para problemas recorrentes em contextos específicos de um projeto de software [Gamma et al., 1994]. O uso dessas soluções durante a construção de um software permite um desenvolvimento mais flexível, extensível e modular, facilitando o entendimento do software.

Gamma et al. [1994] construíram um catálogo formado por 23 padrões de projeto que ficou conhecido por GOF (*Gang of Four*). Apesar de vários trabalhos na literatura terem definido e estendido padrões de projeto existentes ao longo dos anos [Cooper, 2000, Grand, 2001, Rising, 1998, Marinescu, 2002], os mais conhecidos e utilizados pela comunidade acadêmica ainda são os propostos por Gamma et al. [1994].

Os padrões de projeto que compõem o catálogo GOF podem ser classificados em três diferentes categorias: criação, estrutural e comportamental. Alguns padrões, cuja análise é atualmente tratada pela `Design Pattern Smell`, são abordados a seguir.

2.1.1 Padrão de Projeto de Criação

O objetivo dos padrões de criação é abstrair o processo de instanciamento de objetos, de tal forma que o sistema não precise se preocupar com a composição, representação real ou como o objeto é criado. A seguir são mostrados três exemplos de padrões que fazem parte desse grupo.

- *Factory Method*: define uma interface para criação de um objeto, delegando a responsabilidade da instanciação para as subclasses.
- *Singleton*: garante que uma classe seja única, fornecendo um ponto de acesso global para a instância da classe.
- *Prototype*: especifica tipos de objetos a serem criados a partir de uma instância protótipo. A criação de tais objetos se dá a partir da clonagem do protótipo.

2.1.2 Padrão de Projeto Estrutural

Esta categoria tem como objetivo determinar a maneira como as classes e objetos são compostos. Esses padrões facilitam o projeto do sistema e identificam formas como as suas entidades se relacionam. Nessa categoria destacam-se os padrões a seguir.

- *Adapter*: converte a interface de uma classe em outra interface que seja esperada pelo cliente. Nesse processo, esse padrão permite que entidades com interfaces incompatíveis possam trabalhar em conjunto.
- *Bridge*: desacopla a abstração de uma entidade da sua implementação permitindo que ambas possam variar de forma independente e de acordo com as necessidades do sistema.
- *Decorator*: permite adicionar um comportamento a um objeto existente de forma dinâmica. É uma alternativa ao uso de herança para alcançar extensibilidade de funcionalidades.

2.1.3 Padrão de Projeto Comportamental

Padrões de projeto comportamentais atribuem responsabilidades à entidades. Esse grupo contém estruturas que facilitam a comunicação entre objetos. A seguir, destaca-se nessa categoria os seguintes padrões.

- *Chain of Responsibility*: fornece a mais de um objeto a oportunidade de tratar a solicitação e evitar que um receptor seja acoplado ao remetente de uma solicitação. Esse padrão une os receptores em uma corrente, passando uma solicitação ao longo da cadeia até que um objeto a trate.

- *Observer*: define uma dependência de um para muitos entre objetos de tal forma que, quando um objeto muda, todos os seus dependentes são notificados e atualizados automaticamente.

- *Template Method*: define o esqueleto de um algoritmo na superclasse e deixa que as sublcasses sobrescrevam etapas específicas do algoritmo de acordo com suas necessidades, mas sem alterar sua estrutura.

2.2 *Bad Smells*

Bad smells são sintomas ou características encontradas na estrutura de um software que indicam a existência de possíveis problemas e a necessidade de refatoração do código [Fowler and Beck, 1999]. Apesar de não serem considerados erros, a presença de trechos categorizados como *bad smell* aumentam a complexidade e diminuem a legibilidade do código, prejudicando a sua manutenção e evolução. Por conta disso, é importante saber identificar estas estruturas para que tal refatoração possa ser efetuada.

Fowler and Beck [1999] descrevem um conjunto de 22 *bad smells*, apresentando suas características principais de maneira a auxiliar sua identificação. Contudo, não é apresentado uma metodologia ou ferramenta para a realização dessa tarefa.

A seguir são definidos alguns dos *bad smells* descritos por Fowler and Beck [1999] que são objetos de investigação por parte de `Design Pattern Smell` quanto as suas coocorrências com os padrões de projeto listados na Seção 2.1.

- *Data Class*: consiste em uma classe que não possui funcionalidade, apenas dados. Classes que se enquadram nesse sintoma geralmente possuem apenas métodos *get* e *set*.

- *Feature Envy*: ocorre quando métodos de uma classe estão mais interessados em métodos de outras classes, utilizando-os em excesso. É um sinal que existem métodos que deveriam ser removidos da classe.

- *Large Class*: caracterizado por classes que realizam muitas tarefas e possuem muitas responsabilidades. São objetos que possuem muitas instâncias e centralizam a lógica do sistema.

- *Refused Bequest*: identificado por uma subclasse que não utiliza atribudos e funcionalidades que foram herdadas da superclasse. É um indício de que há algo errado com a estrutura de hierarquia das classes.

2.3 `Design Pattern Smell`

Nos últimos anos a coocorrência de padrões de projeto e *bad smells* tem sido investigada por alguns pesquisadores [Cardoso and Figueiredo, 2015, Jaafar et al., 2013, Walter and Alkhaeir, 2016, Sousa et al., 2017], mas o processo de identificação de tais coocorrências na maioria das vezes foi feito de maneira automatizada por meio de *scripts* que não foram disponibilizados. A exceção foi o trabalho de Sousa et al. [2017], o qual desenvolveu um protótipo de uma ferramenta, chamada `Design Pattern Smell`, que desempenha essa tarefa.

`Design Pattern Smell` realiza a identificação de estruturas ~~que~~ apresentando coocorrências entre padrões de projeto e *bad smells*. Para isso, a ferramenta recebe como entrada de dados:

- um ou mais arquivos XML contendo instâncias de padrões de projeto presentes no código e
- um arquivo CSV contendo artefatos[1] onde foram detectados ocorrências de *bad smell*.

Atualmente, tais informações precisam ser obtidas previamente ao uso de `Design Pattern Smell`, via ferramentas como *Design Pattern Detection* [Tsantalis et al., 2006] e *JDeodorant* [Tsantalis et al., 2008].

Após receber os dados, `Design Pattern Smell` realiza a análise e cruzamento das informações, identificando os artefatos onde exista a coocorência das estruturas. O formato dos dados nos arquivos XML devem seguir o formato de *Design Pattern Detection*, e o formato esperado do CSV está especificado no *site* da ferramenta[2].

Além de detectar as coocorências, o usuário também pode aplicar regras de associação [Agrawal et al., 1993, Brin et al., 1997] para analisar a intensidade de tais fenômenos. A ferramenta aplica tais regras automaticamente, o usuário precisa apenas definir o número de transações no sistema analisado e escolher quais das quatro regras – *support, confidence, lift, conviction* –, atualmente implementadas, ele deseja que sejam aplicadas. As transações se referem ao número de classes ou métodos do sistema, de acordo com a granularidade do *bad smell* analisado.

[1] Artefatos são classes ou métodos

[2] http://llp.dcc.ufmg.br/Products/indexProducts.html

Por fim, a ferramenta gera um relatório contendo o número de instâncias de padrões de projeto no sistema bem como a quantidade de informações a respeito dos artefatos que apresentaram coocorrências de padrões de projeto e *bad smell*. O resultado das regras de associação também está presente no relatório, que pode ser exportado para um arquivo CSV.

3 Trabalhos Relacionados

A literatura disponibiliza algumas ferramentas que são utilizadas por pesquisadores e desenvolvedores ao longo dos anos. Tais ferramentas possibilitam a detecção de padrões de projeto e *bad smells* em código-fonte. Dentre elas, destacam-se: *Design Pattern Detection* [Tsantalis et al., 2006] e *DPFinder* [Bernardi et al., 2013]. Essas duas ferramentas identificam instâncias de padrões de projeto pertencentes ao GOF em códigos Java, por meio de análises estáticas. Com relação a detecção de *bad smells* destacam-se *JDeodorant* [Tsantalis et al., 2008] e *JSqIRIT* [Vidal et al., 2016], onde a primeira usa uma abordagem AST para a identificação dos sintomas enquanto a segunda se baseia em métricas.

Mesmo com estudos recentes de coocorrências entre padrões de projeto e *bad smells* [Jaafar et al., 2013, Cardoso and Figueiredo, 2015, Jaafar et al., 2016, Walter and Alkhaeir, 2016] e com a existência de ferramentas que fazem a identificação de maneira individual de cada uma dessas estruturas, não havia um protótipo de ferramenta disponível até que Sousa et al. [2017] propuseram e desenvolveram `Design Pattern Smell`. Para detectar a presença de coocorrências, estudos realizavam este processo de forma manual ou implementavam *scripts* próprios que não eram disponibilizados de forma pública. Com o surgimento da *Design Pattern Smell*, tornou-se prático a detecção de coocorrências entre padrões de projeto e *bad smells*. Tal protótipo permite que coocorrências entre estas duas estruturas sejam identificadas por meio de dados de padrões de projeto e *bad smells*. Além disso, é possível identificar a intesidade das coocorrências nos sistemas, via uso de regras de associação.

`Design Pattern Smell` é uma ferramenta que veio para suprir a falta de um processo automatizado de detecção e metrificação dessa coocorrência, mas, ela ainda é só um protótipo. Este projeto de POC, POC 1 e POC 2, propõe o aperfeiçoamento dessa ferramenta e a sua disponibilização de maneira mais acessível.

4 Metodologia

A metodologia utilizada para o desenvolvimento desse trabalho é definida a seguir.

- **Entendimento do protótipo da ferramenta:** esta etapa visa entender o funcionamento atual da `Design Pattern Smell`, bem como suas entradas e saídas de

dados, e sua lógica interna para a realização da detecção de coocorrências. Para isso estudou-se o código que atualmente está disponível como *open source*[1].

- **Implementação da nova versão da ferramenta:** esta etapa visou a construção de uma nova versão `Design Pattern Smell` utilizando esse protótipo como base. A nova versão foi desenvolvida com foco para ser uma aplicação *web*, e para tal foram utilizadas tecnologias como *Golang* [2], *ReactJS* [3], *HTML*, [4] *CSS* [5] e *ChartJS* [6].
- **Teste da ferramenta:** uma vez implementada a nova versão, foram realizados testes na ferramenta. Para condução dos testes o sistema *WebMail* [7] na versão 0.7.10, foi utilizado como projeto base. Foram extraídas instâncias de padrões de projeto e bad smells usando `Design Pattern Detection` [Tsantalis et al., 2006] e `RAFTool` [Filó et al., 2014], respectivamente. Tais instâncias foram foram exportadas em arquivos *XML* e *CSV* e utilizados como entradas para a ferramenta durante a condução desta etapa.
- **Estudo de novas visualizações para os resultados:** esta etapa visou o estudo e o projeto de algumas visualizações que se encaixam nos dados gerados pela ferramenta, a fim de melhorar o seu entendimento e facilitar um trabalho de análise dos resultados.
- **Implementação das novas visualizações:** definidas as visualizações a serem utilizadas na `Design Pattern Smell`, esta etapa foi responsável pela implementação das visualizações dos resultados da ferramenta. Foram escolhidas duas visualizações adicionais inicialmente, um gráfico de pizza para a página de *instances* e um gráfico de barra para a página de *intersections*.
- ***Deploy* da ferramenta em um contexto *web*:** disponibiliza a ferramenta como uma aplicação *web*.

5 Desenvolvimento do Aprimoramento de `Design Pattern Smell` e Resultados

O desenvolvimento da ferramenta iniciou-se com o estudo e entendimento de seu protótipo desenvolvido em Java. Como o objetivo deste trabalho visa a criação de uma aplicação *web*, o *back-end* da nova versão da `Design Pattern Smell` foi desenvolvido na

[1] https://github.com/BrunoLSousa/DesignPatternSmell
[2] https://go.dev/
[3] https://pt-br.reactjs.org/
[4] https://html.spec.whatwg.org/multipage/
[5] https://www.w3.org/Style/CSS/#specs
[6] https://www.chartjs.org/
[7] https://osdn.net/projects/sfnet_jwebmail/

linguagem *Golang*. O *back-end* é o responsável por manter o estado da aplicação e realizar as agregações e cálculos das métricas. Foi escolhido *Golang* por conta de sua natureza, já que é uma linguagem criada para o desenvolvimento de aplicações *web* altamente escaláveis, resilientes e muito performáticas. *Golang* é uma linguagem de fácil utilização, *C-like* e, apesar de ter sido construída com o propósito de ser utilizada no context de aplicações *web*, se sai bem em qualquer cenário.

Para implementação do *front-end* foram utilizados *ReactJS*, *JavaScript*, *HTML* e *CSS*, uma vez que essas são as tecnologias atualmente dominantes quando se trata da construção de páginas *web* dinâmicas. *ReactJS* em comparação com *JavaScript* puro se apresenta de maneira muito mais amigável para o programador, e por conta disso esse *framework* foi escolhido para o desenvolvimento do *front-end*. O *front-end* é a parte onde o usuário tem acesso a uma interface interativa para o controle da aplicação. A comunicação do *back-end* e do *front-end* é feita via uma API REST.

5.1 Resultados Obtidos

Após a implementação do *back-end* foram realizados testes de corretude da execução da aplicação, usando nesse processo o sistema `WebMail 0.7.10` para validação das métricas e métodos. Após validar o funcionamento do *back-end*, foi realizada a implementação da interface gráfica da ferramenta utilizando como base as telas do protótipo.

As páginas principais da nova versão de `Design Pattern Smell`, bem como uma pequena descrição de cada uma delas são exibidas nas seções 5.2, 5.3, 5.4, 5.5 e 5.6, respectivamente.

5.2 Página Principal

A Figura 1 mostra a página principal da ferramenta. Nela aparecem os campos para definição do nome do software a ser analisado e do *bad smell* presente, bem como os locais de *upload* dos arquivos XML e CSV necessários. Ao clicar em *send*, o *front-end* envia os arquivos para o *back-end*, que os recebe, realiza o *parser* e calcula as coocorrências presentes.

5.3 Intâncias

A página de instâncias, exibida na Figura 2 apresenta duas visualizaços. A primeira é uma tabela com os valores absolutos e percentuais para cada uma das instâncias de

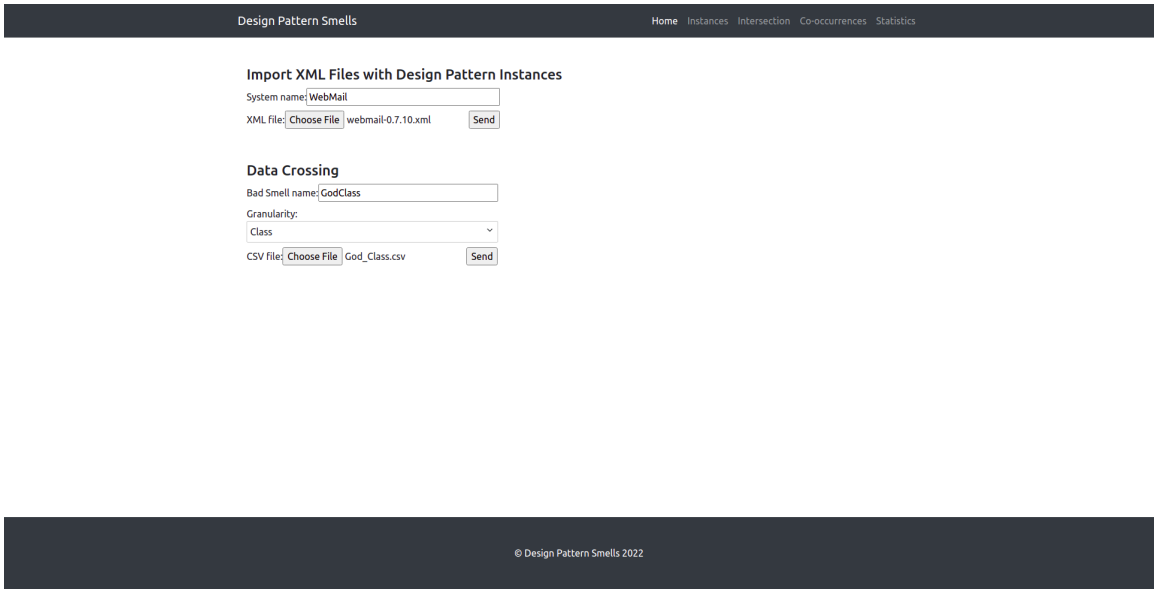

Figura 1 – Página principal da aplicação.

padrões de projeto detectadas no sistema analisado, e a segunda consiste em um gráfico de pizza contendo os padrões de projetos identificados em tal sistema e a sua quantidade em termos percentuais. Além disso, existe um botão no final da página, *"export"*, que possui como funcionalidade realizar o *download* dos dados da tabela em um arquivo *CSV*.

Figura 2 – Apresentação das instâncias de padrões de projeto presentes.

5.4 *Interseção*

Nesta página, mostrada na Figura 3, é apresentado ao usuário a inteserção entre os padrões de projeto e *bad smells* presentes no software. Por meio das visualizações existentes

nesta tela é possível visualizar a volumetria das instâncias afetadas de cada padrão de projeto, tanto na tabela quanto no gráfico de barras empilhadas.

Figura 3 – Apresentação das interseções entre as intâncias de padrões de projeto e o *bad smell* em questão.

5.5 *Coocorrências*

A página apresentada na Figura 4 mostra uma visualização em granularidade de nome e pacote referente a cada artefato presente em cada padrão de projeto que foi afetados pelo *bad smell* analisado. É possível filtrar quais padrões de projeto podem ser analisados nessa página, por meio do filtro definido no topo da página. Neste filtro existem opções de *checkbox* para cada padrão de projeto identificado no software. Ao selecionar uma destas opções e clicar no botão *"filter"* a tabela é atualização conforme as opções selecionadas.

5.6 *Estatísticas*

A Figura 5 exibe a página onde são computadas as estatísticas da ferramenta. Nesta página a ferramenta aplica regras de associação nas coocorrências identificadas no sistema analisado a fim de identificar o quão intensas elas são. As regras de associação são mensuradas por meio de quatro fórmulas: suporte, confiança, *lift* e convicção.

Para calcular estas fórmulas, a ferramenta exibe um campo na página de estátística onde o usuário deve digitar o número total classes ou métodos existentes no sistema analisado, de acordo com a granularidade especificada no momento de *upload* do arquivo

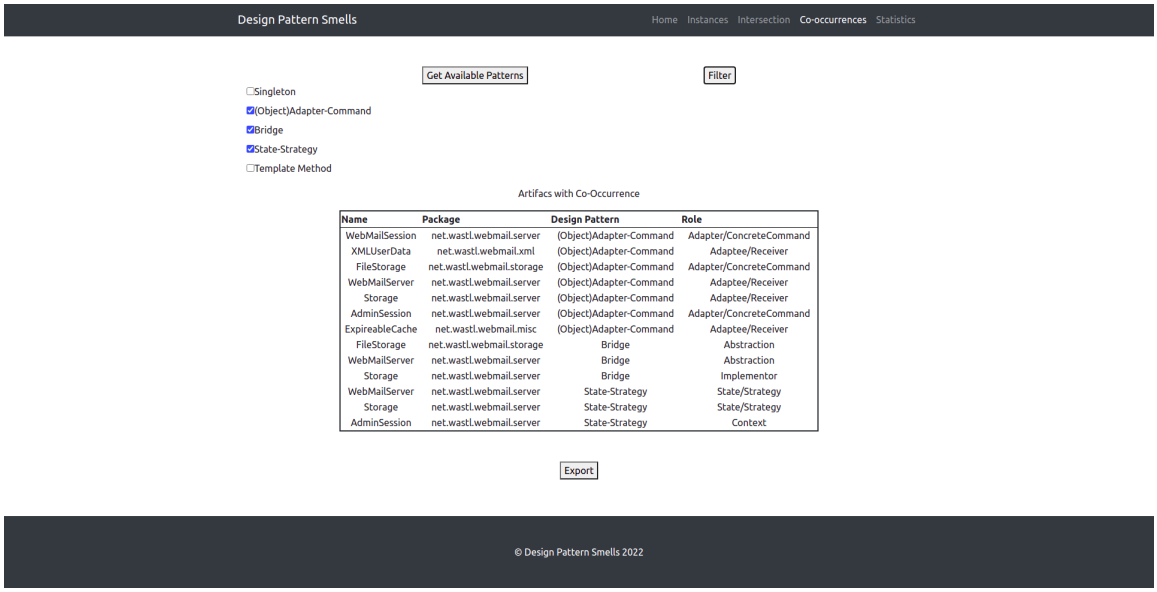

Figura 4 – Artefatos afetados de cada padrão de projeto.

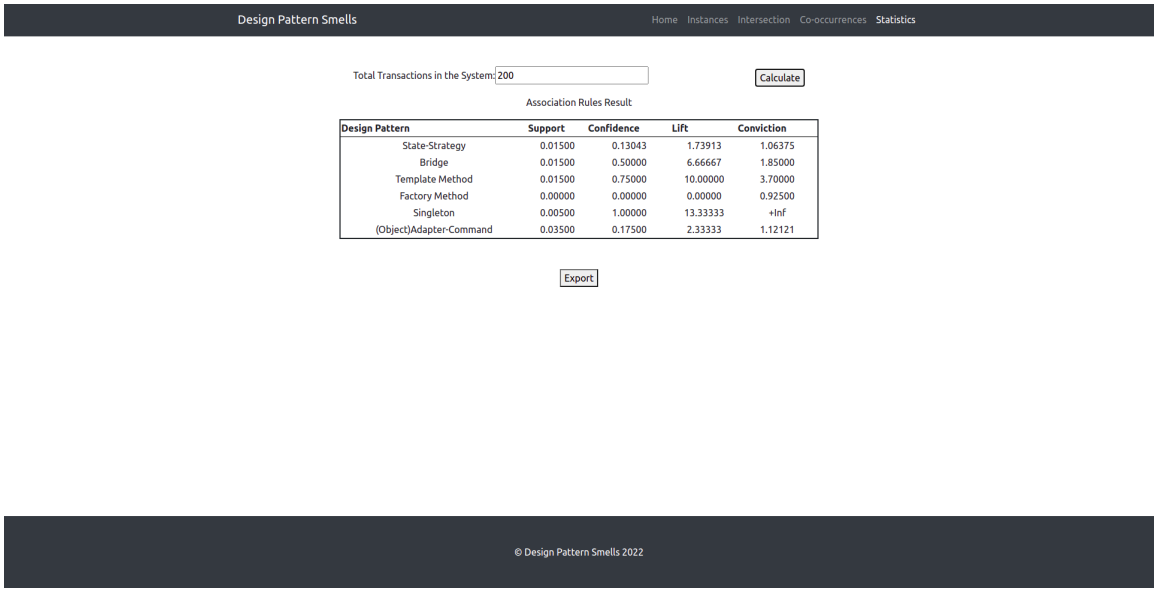

Figura 5 – Resultado das regras de associação.

CSV com as instâncias de *bad smell*. Ao digitar o valor e clicar no botão *"calculate"*, a ferramenta realiza o cálculo das quatro métricas disponíveis para cada padrão de projeto existente no sistema e exisbe o resultado em formato de tabela. Os resultados apresentados nesta tabela referem à intensidade da relação de coocorrência, representada pelas métricas de regras de associação, entre um dado padrão de projeto com o *bad smell*.

5.7 Considerações Finais

Esta seção apresentou os principais resultados obtidos a partir da condução deste trabalho de Fim de Curso, POC I. Além disso, foram apresentadas as principais telas da ferramenta na versão *web* e discutido como ocorre o funcionamento de cada uma delas.

6 Conclusão

Durante este trabalho conclusão de curso, POC I, foram realizados estudos acerca do protótipo de uma ferramenta, denominada `Design Pattern Smell`. Tal protótipo se apresentava como a única alternativa para desenvolvedores e pesquisadores que buscam identificar coocorrências de padrões de projeto e *bad smells* em um projeto de *software*. Esse estudo culminou em uma nova versão de *Design Pattern Smells*.

Este trabalho não só implementou uma versão mais robusta da ferramenta inicial como também a reimplementou no formato de aplicação *web* visando proporcionar uma maior acessibilidade para os usuários que antes precisavam lidar com uma aplicação *desktop* em Java. Além de mais acessível, a versão *web* de `Design Pattern Smell` também apresenta melhorias nas visualizações dos dados, o que facilita estudos e análises mais aprofundados dos resultados.

Como trabalhos futuros pretende-se: (i) introduzir melhorias no *front-end* que ainda é muito rústico, (ii) realizar estudos e implementações de novas visualizações para os resultados gerados, (iii) promover o *deploy* da ferramenta como uma aplicação *web* e (iv) disponibilizar para os usuários a internalização das lógicas de detecção de padrões de projeto e *bad smells* de um código.

Referências

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman, 1994.

M. Fowler and K. Beck. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.

Bruno Cardoso and Eduardo Figueiredo. Co-occurrence of design patterns and bad smells in software systems: An exploratory study. In Proc. of the Conference on Brazilian Symposium on Information Systems: Information Systems: A Computer Socio-Technical Perspective - Volume 1, pages 347–354. Brazilian Computer Society, 2015.

Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Foutse Khomh. Analysing anti-patterns static relationships with design patterns. ECEASST, 2013.

Bartosz Walter and Tarek Alkhaeir. The relationship between design patterns and code smells: An exploratory study. Information and Software Technology, pages 127–142, 2016.

Bruno Sousa, Mariza Bigonha, and Ferreira Kecia. Evaluating co-occurrence of gof design patterns with god class and long method bad smells. In Proceedings of the Brazilian Symposium on Information Systems, pages 1–8, 2017.

Fehmi Jaafar, Yann-Gael Gueheneuc, Sylvie Hamel, Foutse Khomh, and Mohammad Zulkernine. Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults. Empirical Software Engineering, pages 896–931, 2016.

James W. Cooper. Java Design Patterns – A Tutorial. Addison- Wesley, 2000.

Mark Grand. Java Enterprise Design Patterns - Patterns in Java, volume 3. Wiley, 2001.

Linda Rising. The patterns handbook: Techniques, strategies, and applications. Cambridge University Press, 1998. URL http://books.google.co.uk/books?id=HBAuixGMYWEC.

Floyd Marinescu. Ejb design patterns: Advanced patterns, processes, and idioms. 2002.

Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. IEEE transactions on software engineering, 32(11):896–909, 2006.

Nikolaos Tsantalis, Theodoros Chaikalis, and Alexander Chatzigeorgiou. Jdeodorant: Identification and removal of type-checking bad smells. In 2008 12th European conference on software maintenance and reengineering, pages 329–331. IEEE, 2008.

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD international conference on Management of data, pages 207–216, 1993.

Sergey Brin, Rajeev Motwani, Jeffrey D Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In Proceedings of the 1997 ACM SIGMOD international conference on Management of data, pages 255–264, 1997.

Mario Luca Bernardi, Marta Cimitile, and Giuseppe Antonio Di Lucca. A model-driven graph-matching approach for design pattern detection. In 2013 20th Working Conference on Reverse Engineering (WCRE), pages 172–181, 2013. doi: 10.1109/WCRE.2013.6671292.

Santiago A. Vidal, Claudia Marcos, and J. Andrés Díaz-Pace. An approach to prioritize code smells for refactoring. Automated Software Engg., 23(3):501–532, sep 2016. ISSN 0928-8910. doi: 10.1007/s10515-014-0175-x. URL https://doi.org/10.1007/s10515-014-0175-x.

T. G. S. Filó, M. A. S. Bigonha, and K. A. M. Ferreira. Raftool - filtering tool of methods, classes and packages with uncommon measurements of software metrics (in portuguese). In Proceedings of the X WAMPS 2014, pages 1–6, 2014.