Avaliação de Code Smells e Padrões de Projeto em Aplicações com Angular e React

Anderson Hilario Junior[1], **Paulo Roberto Farah**[2,3]

[1]SENIOR Sistemas – Blumenau, SC

[2]UDESC - Universidade do Estado de Santa Catarina – Ibirama, SC

[3]UFPR - Universidade Federal do Paraná – Curitiba, PR

anderson.junior@senior.com.br, paulo.farah@udesc.br

Abstract. *Angular and React web development technologies are very popular today. This paper analyzed fifteen applications developed with Angular and React and analyzed the presence of code smells and design patterns in these programs. React projects had less code smells and more patterns than Angular projects. The results suggest that tools may be developed to assist in identifying and reducing code smells and applying more standards for these technologies.*

Resumo. *As tecnologias para desenvolvimento web Angular e React são muito populares atualmente. Esse trabalho analisou quinze aplicações desenvolvidas com Angular e quinze com React e analisou a presença de code smells e padrões de projeto nesses programas. Os projetos em React apresentaram menos code smells e mais padrões do que os em Angular. Os resultados sugerem que ferramentas sejam desenvolvidas para auxiliar na identificação e redução de code smells e na aplicação de mais padrões para essas tecnologias.*

1. Introdução

O crescimento da demanda por aplicações *web* tem tornado suas interfaces de usuário mais complexas para serem desenvolvidas. Em consequência disso, houve um aumento na utilização de boas práticas e convenções com o objetivo de diminuir a complexidade, aumentar a produtividade e reutilizar código. Elas auxiliam no desenvolvimento e dão espaço para abordagens serem aplicadas na prática, tais como a orientação a objeto e o desenvolvimento baseado em componentes [CARVALHO 2016].

Uma das formas de aplicação prática da orientação a componentes se dá por meio dos *frameworks*. Eles são voltados à reutilização de código, implementam a arquitetura do software, seus comportamentos e interações internas e externas. Servem como modelos para novos sistemas, removem problemas estruturais e fornecem desde componentes a subsistemas inteiros, com alto grau de personalização e adaptação para problemas específicos.

Um dos *frameworks* mais populares atualmente é o *Angular*, um *framework* de software livre que tem como objetivo simplificar tanto o desenvolvimento quanto o teste de aplicações web e utiliza arquiteturas que facilitam a sua integração com os projetos [Google 2019]. Atualmente, mais de 309 mil páginas web utilizam o *Angular* [Similartech 2019a]. Além disso, possui mais de 534 mil repositórios no *Github* [GitHub 2019a] e mais de 34 mil usuários no *Reddit* [Reddit 2019a].

Outra tecnologia que tem se tornado muito popular é denominada *React*, mantido pelo *Facebook* [Facebook 2019]. É uma biblioteca que tem como objetivo oferecer desempenho, simplicidade e escalabilidade às aplicações web. É utilizada na criação de interfaces para usuários e manipula os elementos com regras próprias, tais como a criação de componentes reutilizáveis. O *React* é utilizado em mais de 777 mil páginas *web* [Similartech 2019b], conta com mais de 960 mil repositórios no *Github* [GitHub 2019b] e mais de 130 mil usuários no *Reddit* [Reddit 2019b].

O surgimento de paradigmas como a orientação a objetos proporcionou que conceitos sejam criado, tais como *code smells* e padrões de projetos. Um *code smell* é uma indicação superficial de um problema mais profundo de um sistema [Fowler 1999]. *Code smells* podem indicar decisões errôneas de desenvolvimento ou de design e podem afetar a manutenção de um *software*. Os *smells* de um código são revelados por meio de métricas e comparações com outras partes de um sistema.

Segundo [Gamma et al. 1995], um padrão de projeto abstrai características de um projeto comum para torná-lo reutilizável. Cada padrão foca em resolver um problema e é aplicado em uma situação específica e podem ser divididos em três categorias, padrões de criação, estruturais e comportamentais, cada qual com suas características e objetivos.

Apesar dos *code smells* e dos padrões de projeto serem práticas consolidadas para melhorar a estrutura dos códigos dos programas, não foram encontradas outras pesquisas que analisem o uso delas em aplicações com tecnologias atuais como *Angular* e *React*.

Assim, o presente trabalho tem como objetivo avaliar *code smells* e padrões de projeto em programas que utilizem as tecnologias Angular ou React no desenvolvimento de interfaces de usuário para ambiente *web*.

Esse artigo está organizado em cinco seções, das quais essa introdução é a primeira. A próxima seção descreve a fundamentação teórica. Em seguida, é apresentada a metodologia adotada. A seção subsequente contém os resultados obtidos e por fim, são apresentados a conclusão e os trabalhos futuros.

2. Padrões de Projeto JavaScript

Nessa seção, são descritos os padrões de projeto para *frameworks JavaScript*. Os padrões de projeto de *framework* utilizam as principais características das tecnologias para desenvolver formas otimizadas de desenvolvimento. Na seção abaixo serão descritos os principais padrões de projeto para *Angular* e *React*.

2.1. Angular

Os padrões de *Angular* utilizam a estrutura de componentes do *framework* para modularizar o *software*. Os padrões de projeto podem ser divididos em três grupos: Padrões de Navegação, de Estabilidade, de Desempenho.

2.1.1. Padrões de Navegação

Os padrões de navegação são usados para organizar eventos relacionados a usabilidade do sistema pelos usuários. Exemplos de padrões têm-se *model-view-controller* e o *Redux*. O *model-view-controller* ou MVC faz a separação do código em três partes: o *model*, o

controller e a *view*. O *model* armazena os dados da aplicação de acordo com os comandos enviado pelo *controller*. O *controller* recebe ações ou entradas do usuário, como o clique de um botão ou envio de um formulário, e atualiza o *model* com base nas novas informações. O *controller* atualiza e muda qual tela ou *layout* está sendo usado a cada momento. Por último, a *view* são as telas, *layouts* e outros elementos visuais que o usuário tem acesso. A *view* é atualizada cada vez que o *model* sofre alguma mudança.

O padrão *Redux* permite a manipulação de estados de um sistema de forma segura e isolada. Para armazenar os estados é criada uma interface e para manipulá-la usa-se ações. Essas ações são eventos acionados e interceptados por uma classe *Reducer*. Apenas nessa classe acontecem as mudanças de estados.

2.1.2. Padrões de Estabilidade

Os padrões de estabilidade focam na capacidade de um sistema em se manter operando quando há alguma falha ou erro interno. Exemplos de padrões de estabilidade são: *Timeout, Circuit Breaker, Factory, Memento, Prototype* e *Reusable Pool*. O *Timeout* é usado quando o *back-end* de uma aplicação deixa de responder o restante. Ele lida com esse problema de resposta fazendo com que o sistema espere um tempo predeterminado. Dessa forma, se o tempo for configurado para um segundo, a aplicação esperará uma resposta dentro desse intervalo e caso não a receba o software continua a execução normalmente.

O *Memento* é utilizado para criar uma cópia de um elemento e guardá-la. Esse padrão permite que mudanças sejam desfeitas de forma segura. Antes de executar a alteração, um clone do objeto é feito e, caso o processo seja cancelado, os dados do objeto são retornados com base no clone que foi criado.

2.1.3. Padrões de Desempenho

Os padrões de desempenho têm como objetivo definir arquiteturas e práticas que afetam características como tempo de resposta, taxa de serviço e outras métricas relacionadas ao desempenho das aplicações. Dentre eles há: *AJAX Overkill, Unbound result sets, Proxy, Filters and Pipes, Loops, Change Detection, Immutability* e *Prototype*. O padrão *Loop Count* é usado para carregar apenas os elementos que sofreram alteração dentro de uma página. Dependendo da quantidade de elementos numa lista ou tabela o carregamento pode se tornar custoso e lento. O *loop count* utiliza a função *trackBy* do Angular para acionar uma função que localiza o elemento alterado e recarrega apenas ele, não sobrecarregando a página.

A criação de objetos é comum e em grandes quantidades podem sobrecarregar uma aplicação *web*. O *Prototype* é responsável por clonar um modelo de objeto configurado quando necessário.

2.2. React

Os padrões de *React* utilizam a estrutura de componentes da biblioteca para granular o *software*, ou seja, dividi-lo em partes menores. O foco, no entanto, é criar componentes independentes e reutilizáveis no decorrer da aplicação. Alguns dos padrões de projeto são: *Container and Presentational, High-Order Components* e *Function as a Child*.

O padrão *container and presentional* tem como objetivo separar a parte lógica, manipulação de dados, eventos, da parte de apresentação, criação e manipulação de elementos HTML. A separação das duas partes faz com que elas sejam reutilizáveis e independentes.

O padrão *High-Order Components* (HoCs) consiste em funções que recebem um componente e retornam uma versão melhorada do componente. Por exemplo, caso vários componentes necessitem ter uma mesma classe, pode-se usar um HoCs que recebe o componente e agrega a classe a ele. Dessa forma um mesmo componente pode ser reutilizado.

Este padrão de projeto utiliza a inversão de componentes filhos e pais. Em vez de passar um componente filho, é definida uma função fixa como filha que receberá parâmetros de seus pais.

3. Metodologia

O objetivo principal dessa avaliação é analisar a qualidade de códigos-fonte de repositórios públicos do *GitHub* que utilizam as tecnologias *Angular* ou *React* e verificar se os mesmos contém *code smells* e se utilizam padrões de projeto. Para isso, esse trabalho foi guiado pelas seguintes questões de pesquisa: *Quais os principais padrões de projeto utilizados nos sistemas web elencados?* Essa pergunta de pesquisa tem como objetivo avaliar a presença de padrões de projetos nos softwares. *São identificados code smells nos sistemas web levantados?* Essa pergunta de pesquisa visa a identificar *code smells* nas aplicações.

Para avaliar as tecnologias foram analisadas trinta aplicações de *software* livre que as utilizem, sendo que quinze manipulam *Angular* e quinze *React*, descritas no apêndice. Os *softwares* foram elencados de acordo com suas estrelas e *commits* no *Github*, ambas métricas apresentando o mesmo peso de seleção. Os códigos dos sistemas foram avaliados por completo, a procura de *code smells* e padrões de projeto.

Para a tabulação dos resultados foi utilizado o método de "Análise de Conteúdo" para elencar as principais características dos softwares. Este método distingue a frequência na qual um aspecto está presente entre os objetos de pesquisa [Huang et al. 2018]. A principal diferença deste método para outros é o foco no dado bruto, ou seja, a quantificação dos aspectos dos *softwares*.

A identificação dos *code smells* foi baseada no modelo proposto em [Palomba et al. 2013, Palomba et al. 2015, Tufano et al. 2017], na análise dos códigos fonte de trinta sistemas de código aberto obtidos de repositórios *Github*. Os padrões de projeto foram avaliados com base nos mesmo trinta sistemas e visou identificar os padrões de cada *framework*. A pesquisa se baseou nos resultados encontrados dessas avaliações.

4. Resultados

Essa seção apresenta os resultados da análise de *code smells* e padrões de projeto identificados em repositórios de *software* livre desenvolvidos com *Angular* ou *React*.

4.1. Code Smells

A seguir são demonstrados os resultados encontrados ao analisar os *code smells* nos sistemas. A Tabela 1 mostra os *code smells* identificados nos softwares desenvolvidos com *Angular* ou *React* e o total dos dois.

Tabela 1. Code smells encontrados nas aplicações Angular, React e total

	LongMethod	Abstract Class	Class One Method	Few Methods	Low Cohesion Only	Data Class	Field Public
Angular	5	5	1	3	1	3	0
React	4	0	0	0	0	0	1
Total	9	5	1	3	1	3	1

Os dados demonstram que a maior parte (18) dos *code smells* foram encontrados nos sistemas *Angular*, principalmente o *Long Method* (5), equivalente a $78,3\%$. Esse *code smell* é caracterizado quando um método apresenta muitas linhas de código e que pode ser feita uma maior granulação. Em todos os casos, o *Long Method* foi identificado no atributo *template* dos componentes, gerando um extenso código HTML. A boa prática de programação, segundo [Google 2019], é que seja utilizado o atributo *template Url* para referenciar outro arquivo.

Nas aplicações com *React* houve menos (5) *code smells* identificados se comparado aos projetos que utilizam *Angular*, equivalente a $21,7\%$. Porém, o mais encontrado foi também o *Long Method*. Todas as aparições foram no método render do componente, sendo que o intuito do *React* é a componentização e reutilização de código. Como houveram métodos render extensos, eles engessam a estrutura orientada a componentes. Segundo as boas práticas adotadas pela comunidade [Facebook 2019], para evitar problemas com renderização extensa, deve-se utilizar o padrão de projeto *High-Order Componentes* e agrupá-los de acordo com a necessidade.

Outro *code smell* encontrado foi o *Abstract Class*. Ele pode ser identificado quando há métodos implementados, mas que não são utilizados na aplicação. Nos sistemas avaliados foram encontrados *constructors* vazios e interfaces de ciclo de vida de componentes implementadas, mas não utilizadas. Por exemplo, uma classe implementar o *OnInit* e deixar o método obrigatório *ngOnInit* vazio. Segundo as convenções utilizadas pela comunidade do Angular, os construtores são utilizados para gerar a injeção de dependência e as interfaces de ciclo de vida são usadas quando necessárias.

O *code smell DataClass* apresenta uma definição parecida ao *AbstractClass*. Seu conceito se baseia em quando uma classe recebe dados ou implementa métodos, mas não os processa ou utiliza. Nos sistemas foram encontrados *imports* e injeções de dependência realizadas, mas não utilizadas. As importações e injeções de dependência devem ser feitas apenas quando utilizadas.

4.2. Padrões de Projeto

A Figura 1 mostra os padrões de projeto encontrados nos programas analisados. Comparando com a quantidade de *code smells*, houveram menos padrões de projeto encontrados. A diferença numérica de padrões de projeto entre os *frameworks* foi pequena e ambos utilizaram apenas dois padrões. O padrão de projeto mais presente nos sistemas Angular analisados foi o *Timeout* (2). Ele é utilizado para que a aplicação não pare de funcionar caso haja algum problema de conexão ao *back-end*. Um tempo em milissegundos é determinado e caso o *back-end* não responda a uma requisição dentro do intervalo, o sistema continua a execução sem que comprometa a aplicação. Esse padrão de projeto traz benefícios pois o front-end Angular se torna independente de um servidor ou API.

O outro padrão de projeto encontrado nos sistemas *Angular* foi o *LoopCount* (1).

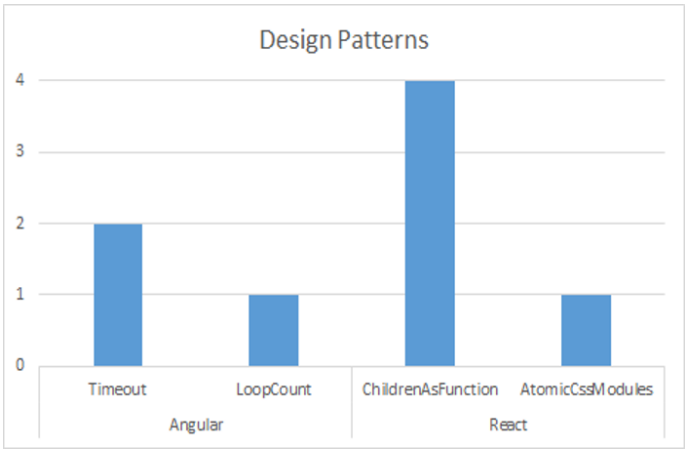

Figura 1. Padrões de projeto encontrados nas aplicações com Angular e React

O foco desse padrão é a otimização no carregamento de listas. Ele utiliza uma função auxiliar no *ngFor* para localizar os elementos da lista que sofreram alteração e recarregar apenas eles, sem que precise carregar todos os elementos. *O loopCount* traz benefícios pois a utilização de listagem, seja em tabelas ou outros componentes, é algo comum em cenários web, podendo haver infinitas listas e elementos em tela.

O padrão de projeto mais utilizado nas aplicações *React* foi o *FunctionAsChild* (4). Nesse padrão um componente filho genérico é criado e "lacunas" são deixadas propositalmente para que um componente pai possa fazer uso do componente filho e moldá-lo à sua maneira. Esse padrão traz o benefício de aumentar a reutilização de componentes filhos, pois cada pai acessa o mesmo filho.

O outro padrão de projeto presente nas aplicações *React* foi o *Atomic CSS Modules* (1). Ele componetiza a criação de estilos CSS por meio de classes atômicas de apenas uma linha de código e depois funde esses estilos em módulos que serão utilizados na aplicação. Esse padrão se torna trabalhoso inicialmente pois as principais classes atômicas devem ser feitas. Porém a junção delas em módulos e componentes torna a reutilização e manutenção mais fácil.

Dessa forma, pode-se observar uma grande quantidade de *code smells* nos projetos analisados e poucos padrões de projeto encontrados. Contudo, esses resultados sugerem que há uma correlação entre uma menor quantidade de *code smells* e os projetos com *React*, do que com Angular, sendo que foram encontrados 18 instâncias de *code smells* nos projetos em Angular e apenas 5 nos em *React*. Além disso, sugere-se que há a necessidade de que novas ferramentas sejam propostas para auxiliar os desenvolvedores a remover *code smells* e a aplicar mais os padrões de projeto em aplicações web. Uma tabela com os *code smells* e padrões de projeto encontrados em cada repositórios está disponível na url: `https://drive.google.com/file/d/1_tbb4oJOXA_OBsbEypJpdWspYvfVYxv_/view`.

5. Conclusão e Trabalhos Futuros

As aplicações web evoluíram em passo acelerado e vários termos foram desenvolvidos para o assunto. Atualmente ambas tecnologias centrais tratadas, Angular e React, são amplamente usadas, porém cada uma lida com os problemas de forma diferente.

Os *code smells* e padrões de projeto foram métricas utilizadas durante o trabalho. Os *code smells* são representações de uma má implementação e podem comprometer a manutenção de um sistema. Os padrões de projeto, por sua vez, são técnicas para otimizar a utilização, refatoração e legibilidade do código. Para avaliar a presença de *code smells* e padrões de projeto foram levantados trinta sistemas, quinze Angular e quinze *React*.

Foram encontrados mais *code smells* nas aplicações Angular, com a predominância do *LongMethod* e mais padrões de projeto em *React*. Os resultados apresentaram melhor qualidade dos códigos desenvolvidos em *React*, com 21,7% dos *code smells* e 62,5% dos padrões de projeto encontrados. Contudo, a quantidade de *code smells* é significativa e de padrões de projeto é baixa, sugerindo que trabalhos futuros possam explorar maneiras de auxiliar os desenvolvedores a identificar e corrigir um maior número de *code smells* e a aplicar mais padrões de projeto.

Referências

CARVALHO, T. (2016). *Orientação a Objetos: Aprenda seus conceitos e suas aplicabilidades de forma efetiva*. Casa do Código.

Facebook (2019). React a javascript library for building user interfaces. `https://reactjs.org/`. Accesso em: 09/08/2019.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

GitHub (2019a). Angular: Angular is an open source web application platform. `https://github.com/search?utf8=\%E2\%9C\%93&q=angular&type`. Accesso em: 09/08/2019.

GitHub (2019b). React: React is an open source javascript library used for designing user interfaces. `https://github.com/search?utf8=\%E2\%9C\%93&q=angular&type`. Accesso em: 09/08/2019.

Google (2019). Angular. `https://angular.io/`. Accesso em: 09/08/2019.

Huang, X., Zhang, H., Zhou, X., Babar, M. A., and Yang, S. (2018). Synthesizing qualitative research in software engineering: A critical review. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 1207–1218, New York, NY, USA. ACM.

Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., and Poshyvanyk, D. (2013). Detecting bad smells in source code using change history information. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 268–278.

Palomba, F., Bavota, G., Penta, M. D., Oliveto, R., Poshyvanyk, D., and De Lucia, A. (2015). Mining version histories for detecting code smells. *IEEE Transactions on Software Engineering*, 41(5):462–489.

Reddit (2019a). Angular (2+). `https://www.reddit.com/r/Angular2/`. Accesso em: 09/08/2019.

Reddit (2019b). ReactJS. https://www.reddit.com/r/reactjs/. Accesso em: 09/08/2019.

Similartech (2019a). Angular js market share and web usage statistics. https://www.similartech.com/technologies/angular-js. Accesso em: 09/08/2019.

Similartech (2019b). React js market share and web usage statistics. https://www.similartech.com/technologies/react-js. Accesso em: 09/08/2019.

Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Penta, M. D., De Lucia, A., and Poshyvanyk, D. (2017). When and why your code starts to smell bad (and whether the smells go away). *IEEE Transactions on Software Engineering*, 43(11):1063–1088.

Apêndice

A seguir, são listadas as URLs das aplicações com *Angular* e *React* utilizadas.

Angular

1. *angular-calendar*: https://github.com/mattlewis92/angular-calendar;
2. *angular-electron*: https://github.com/maximegris/angular-electron;
3. *angular-hmr*: https://github.com/gdi2290/angular-hmr;
4. *angular-realworld-example-app*: https://github.com/gothinkster/angular-realworld-example-app;
5. *angular-webpack-starter*: https://github.com/qdouble/angular-webpack-starter;
6. *angular2-notifications*: https://github.com/flauc/angular2-notifications;
7. *angularspree*: https://github.com/aviabird/angularspree;
8. *angulartics2*: https://github.com/angulartics/angulartics2;
9. *echoes-player*: https://github.com/orizens/echoes-player;
10. *flex-layout*: https://github.com/angular/flex-layout;
11. *nebular*: https://github.com/akveo/nebular;
12. *ngx-admin*: https://github.com/akveo/ngx-admin;
13. *platform*: https://github.com/ngrx/platform;
14. *SB-Admin-BS4-Angular-6*: https://github.com/start-angular/SB-Admin-BS4-Angular-6;
15. *yatrum*: https://github.com/aviabird/yatrum;

React

1. *react-color*: Replaces a BluetoothAdapter. isEnabled() call with ”true”;
2. *react-datasheet*: https://github.com/nadbm/react-datasheet;
3. *react-flight*: https://github.com/jondot/react-flight;
4. *react-fns*: https://github.com/jaredpalmer/react-fns;
5. *react-move*: https://github.com/react-tools/react-move;
6. *react-native-nw-react-calculator*: https://github.com/benoitvallon/react-native-nw-react-calculator;
7. *react-overdrive*: https://github.com/berzniz/react-overdrive;
8. *downshift*: https://github.com/paypal/downshift;
9. *formik*: https://github.com/jaredpalmer/formik;
10. *git-point*: https://github.com/gitpoint/git-point;
11. *hackernews-react-graphql*: https://github.com/clintonwoo/hackernews-react-graphql;
12. *imgsible*: https://github.com/BinaryMuse/imgsible;
13. *isomorphic500*: https://github.com/gpbl/isomorphic500;
14. *joplin*: https://github.com/laurent22/joplin;
15. *perseus*: https://github.com/Khan/perseus;