Um Estudo em Larga-Escala sobre Características de APIs Populares

Caroline Lima[1], **Pedro Henrique de Moraes**[1], **Andre Hora**[1]

[1] Faculdade de Computação (FACOM)
Universidade Federal de Mato Grosso do Sul (UFMS)

carollimaxp@gmail.com, pedhmoraes@gmail.com, hora@facom.ufms.br

Abstract. *Software libraries are commonly used via APIs to improve productivity and decrease development costs. While some APIs are very popular, others face much lower usage rates. In this context, one important question emerges: are there particular characteristics that differentiate popular APIs from other APIs? In this paper, we assess several characteristics of popular APIs, at code and evolution level, aiming to reveal development aspects of globally used APIs. This analysis is performed in the context of 897 APIs provided by libraries such as JDK, Android, and JUnit. As a result, we find that popular APIs are better encapsulated, have more documentation, and have more updates. Finally, based on our findinds, we propose several implications to library designers.*

Resumo. *Bibliotecas de software são comumente utilizadas, através de suas APIs, para aumentar a produtividade e diminuir os custos de desenvolvimento. Enquanto algumas são muito populares, outras equivalentes apresentam taxas de uso relativamente menores. Nesse contexto, uma questão importante surge: existem características particulares que diferenciam as APIs populares das demais APIs? Neste artigo, analisa-se diversas características de APIs populares, em nível de código e de evolução, visando revelar aspectos de desenvolvimento de APIs utilizadas globalmente. Essa análise é realizada no contexto de 897 APIs, fornecidas por bibliotecas como JDK, Android e JUnit. Como resultado, detecta-se que as APIs populares possuem melhor encapsulamento, mais documentação e mais alterações que as demais APIs. Por fim, com bases nesses resultados, diversas implicações são sugeridas para projetistas de bibliotecas.*

1. Introdução

Hoje em dia, bibliotecas de software são frequentemente utilizadas através de suas APIs (*Application Programming Interfaces*), para apoiar a criação de sistemas, melhorar a produtividade e diminuir os custos de desenvolvimento [Moser and Nierstrasz 1996]. Enquanto algumas APIs são muito populares e utilizadas por milhares de sistemas clientes, outras equivalentes apresentam taxas de uso relativamente menores. Por exemplo, em uma análise considerando 4.532 sistemas, detectou-se que 97% desses sistemas utilizavam a biblioteca de teste JUnit, enquanto apenas 12% utilizavam o seu concorrente TestNG [Zerouali and Mens 2017]. Considerando 260 mil sistemas, a diferença entre as taxas de uso continuam altas: 20% usam o JUnit, enquanto 1% usam o TestNG [Dyer et al. 2013].

Nesse contexto, uma questão importante surge: *existem características particulares que diferenciam as APIs populares das demais APIs?* Por exemplo, elas se diferenciam com relação a documentação, complexidade, estabilidade? Responder essas

perguntas pode revelar aspectos de desenvolvimento presentes em APIs utilizadas globalmente, de forma a ajudar projetistas de APIs em suas atividades de manutenção de bibliotecas de software. Neste artigo, analisa-se diversas características presentes em APIs populares. Especificamente, investiga-se aspectos de código (*ie*, tamanho, complexidade e documentação) e de evolução (*ie*, estabilidade) dessas APIs. Logo, duas questões de pesquisa centrais são propostas: *QP1: Quais as características de código das APIs populares?* e *QP2: Quais as características evolucionárias das APIs populares?*

Para responder essas questões de pesquisa, estuda-se 897 APIs (fornecidas por bibliotecas como JDK, JUnit, Android e Facebook Fresco), que são utilizadas por 1.000 sistemas. Essas APIs são categorizadas em três grupos com relação aos seus níveis de popularidade e comparadas entre si. Logo, as principais contribuições desse trabalho são: (1) um estudo em larga-escala sobre as características de código e de evolução de APIs populares, (2) uma análise contrastando APIs populares das demais APIs e (3) um conjunto de lições aprendidas e implicações para projetistas de bibliotecas.

2. Popularidade de APIs

Bibliotecas fornecem interfaces para componentes de software criados para serem reutilizados por sistemas clientes [Reddy 2011]. Na linguagem Java, essas interfaces (também conhecidas como APIs) podem ser importadas por sistemas clientes. Exemplos de APIs comumente utilizadas em Java são `java.util.Map` e `java.util.HashMap`.

A Tabela 1 apresenta dentre todos os tipos APIs as mais frequente em Java, extraídas de 260K sistemas [Dyer et al. 2013]. Nota-se que uma alta taxa de uso está concentrada em poucas APIs e que essa taxa diminui rapidamente. Por exemplo, a API mais utilizada (`ArrayList`) é importada por 143.454 (54%) dos 260K sistemas, enquanto a API na 500ª posição (`AbstractTableModel`) é usada por apenas 3.977 ($\sim$2%).

Posição	API	Uso Absoluto	Uso Relativo (%)
1ª	`java.util.ArrayList`	143.454	54%
100ª	`android.graphics.Color`	16.382	6%
200ª	`org.junit.Assert`	10.857	4%
300ª	`android.location.Location`	6.786	3%
400ª	`android.view.SurfaceHolder`	5.149	$\sim$2%
500ª	`javax.swing.table.AbstractTableModel`	3.977	$\sim$2%

Tabela 1. APIs Java mais utilizadas nos 260K sistemas [Dyer et al. 2013].

Diversas pesquisas levam em consideração a popularidade de APIs para aprender com elas. Por exemplo, estudos mineram tendências de uso de APIs populares, com a finalidade de sugerirem se determinada API deve ser adotada ou evitada pelos clientes [Mileva et al. 2009]. Outros trabalhos avaliam a evolução de APIs populares para apoiar a migração de bibliotecas [Hora and Valente 2015]. No melhor do nosso conhecimento, entretanto, aspectos relacionados as APIs populares (*eg*, complexidade, tamanho, documentação e evolução) foram pouco explorado pela literatura.

3. Metodologia

3.1. Detectando APIs Populares

Esta pesquisa foca na análise de sistemas escritos na linguagem de programação Java (uma das mais populares na atualidade) e hospedados na plataforma GitHub (a mais utilizada hoje em dia para armazenar sistemas *open-source*). Para responder as questões

de pesquisa, primeiramente, precisa-se detectar um conjunto de APIs populares. Essa detecção é realizada em três fases: (1) coleção de um conjunto de sistemas relevantes; (2) detecção das APIs usadas por esses sistemas; e (3) categorização das APIs de acordo com sua popularidade, conforme descrito a seguir.

Etapa 1: Coleta de Sistemas Clientes. Primeiramente, realizou-se a coleta de um conjunto de sistemas Java; essa coleta foi necessária para descobrir quais APIs esses sistemas estavam utilizando. Especificamente, coletou-se os primeiros 1.000 sistemas com mais *stars*[1] para focar nos sistemas mais populares. Alguns desses sistemas são criados por grandes empresas, como Google, Facebook, Twitter e Microsoft. Os sistemas coletados pertencem a diversos domínios, tais como linguagens de programação (*eg*, Kotlin e Clojure), bibliotecas (*eg*, Facebook Fresco e Google Guava), frameworks (*eg*, Android e Spring). Esses sistemas apresentam na mediana 1.797,5 *stars* e 54 arquivos Java.

Etapa 2: Detecção das APIs Usadas pelos Sistemas Clientes. O próximo passo na metodologia é detectar as APIs utilizadas pelos 1.000 sistemas (em sua última *release*). Para isso, extraiu-se as APIs importadas, e, em seguida, para cada API, calculou-se o número de sistemas distintos que a utilizavam. No total, detectou-se 230.365 APIs distintas importadas por pelo menos um sistema. Para filtrar APIs locais, descartou-se aquelas utilizadas por menos de 10 sistemas (APIs com pelo menos 1% do total de sistemas). Essa filtragem resultou em 2.994 APIs com pelo menos 10 clientes. A Figura 1(a) apresenta a popularidade das APIs filtradas em relação ao números de sistemas clientes. O primeiro quartil é 13 clientes, a mediana é 20 e o terceiro quartil é 39. Também percebe-se uma grande quantidade de *outliers*, que representam APIs altamente populares.

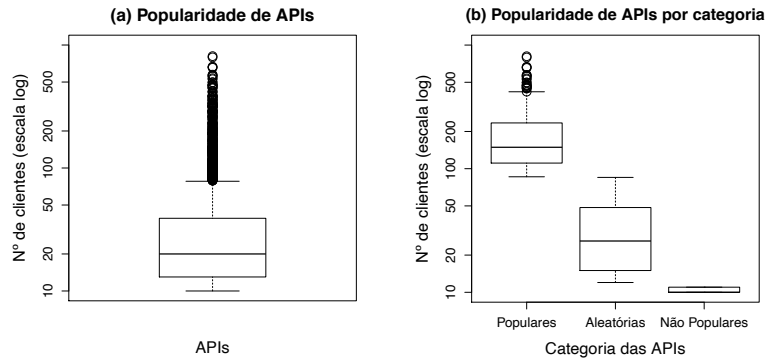

Figura 1. (a) Popularidade das APIs em relação ao número de clientes. (b) Popularidade das APIs em relação ao número de clientes por categoria.

Etapa 3: Categorização das APIs em Populares, Não Populares e Aleatórias. O último passo consiste em categorizar as APIs detectadas em *populares* e *não populares*. Nesse contexto, ordenou-se de forma decrescente as 2.994 APIs pelo número de clientes. Em seguida, seguindo o mesmo projeto experimental adotado para avaliar *apps* populares e não populares [Tian et al. 2015], classificou-se como *populares* os primeiros 10% (*ie*, as APIs com mais clientes) e como os *não populares* os últimos 10% (*ie*, as APIs menos clientes); cada categoria, portanto, inclui 299 APIs. Adicionalmente, selecionou-se um conjunto *aleatório* de 299 APIs para contrastar com as *populares* e as *não populares*. Dessa forma, foram consideradas três categorias (*populares*, *aleatórias* e *não populares*),

[1]Métrica que indica a popularidade de um projeto na plataforma GitHub.

totalizando **897** APIs distintas (299 x 3). A Figura 1(b) apresenta a popularidade das APIs considerando as três categorias. Na mediana, as APIs *populares* possuem 149 clientes, as *aleatórias* 26 clientes e as *não populares* 10 clientes. Observou-se também que as três categorias são estatisticamente diferentes entre si (*p-value* $<$ 0,01 para o teste de Mann-Whitney e *effect-size* 0,99 para o Cliff's Delta).

3.2. Mensurando Características das APIs

QP1 (Código das APIs). Para responder a Questão de Pesquisa 1, sobre as características de tamanho, complexidade e documentação das APIs estudadas, extraiu-se 5 métricas de código, com suporte da ferramenta Understand[2], conforme descrito na Tabela 2. Para o cálculo da métrica de complexidade considerou-se todos os métodos e para a métrica documentação considerou-se todas linhas com comentários.

QP2 (Evolução da API). Para responder a Questão de Pesquisa 2, sobre as características de evolução das APIs, utilizou-se os dados fornecidos pelo controle de versão git. Nesse contexto, foram extraídas métricas relacionadas ao nível de estabilidadedas APIs a partir do histórico de versões. Utilizou-se o comando `git log` as API e implementou-se uma ferramenta para calcular as métricas número de *commits* e *lifetime* (ver Tabela 2). *Lifetime* é o número de dias entre o commit em que a API foi inserida até atualmente. Na métrica *commits* contabilizou-se o número de commits que alteraram o arquivo da API.

RQ	Categoria	Métrica	Descrição
Código das APIs (QP1)	Tamanho	NPM	Número de métodos públicos.
		RPM	Número relativo de métodos públicos (razão entre NPM e o número total de métodos).
	Complexidade	CC	Média da complexidade ciclomática dos métodos (expressões e condicionais são consideradas).
	Documentação	NCL	Número de linhas com comentários.
		RCL	Número relativo de linhas com comentários (razão entre NCL e número total de linhas).
Evolução das APIs (QP2)	Estabilidade	Lifetime	Número de dias entre o primeiro e último *commit*.
		Commits	Número de *commits* que modificar a API.
		RLCM	Razão entre *lifetime* e *commits*.

Tabela 2. Métricas estudadas ao nível de API.

4. Resultados

QP1: Quais as características de código das APIs populares?

Tamanho. A Figura 2 compara o tamanho das APIs através do número de métodos públicos (NPM) e do número relativo de métodos públicos (RPM). Observa-se que as APIs *populares* fornecem mais métodos públicos (mediana 19) para os clientes do que as *aleatórias* (mediana 10) e as *não populares* (mediana 9) (*p-value*$<$0,01 e *effect-size* pequeno). Considerando o valor relativo, nota-se o oposto; a razão entre métodos públicos e o número total de métodos é menor para as APIs *populares* (mediana 81%) do que para as *aleatórias* (mediana 89%) e *não populares* (mediana 86%). Desse modo, as APIs *populares* incluem outros tipos de modificadores de visibilidade (*ie*, `private`, `package` e `protected`), indicando maior preocupação com encapsulamento.

[2]https://scitools.com

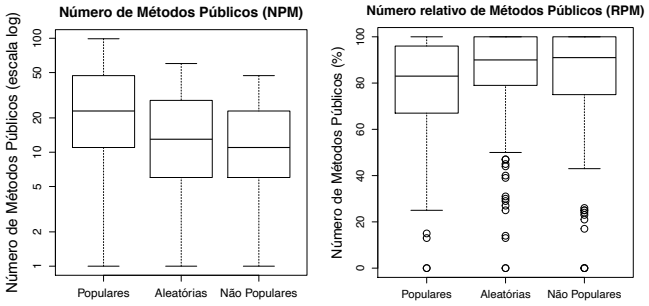

Figura 2. Tamanho das APIs.

Complexidade. A Figura 3 apresenta os resultados da métrica média da complexidade ciclomática (CC). Essa métrica conta cada estruturada (*eg*, `if`, `else` e `while`) e cada operador lógico (`&&` e `||`). O resultado é a razão entre a soma da complexidade ciclomática e o número de métodos da API. As *populares* são um pouco mais complexas (mediana 2) do que as demais, ambas com mediana 1 (*p-value*<0,01 e *effect-size* pequeno).

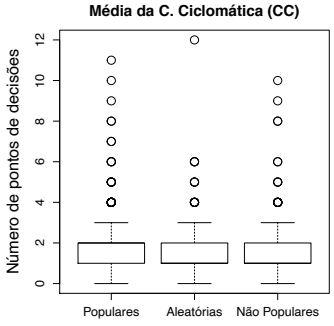

Figura 3. Complexidade das APIs.

Documentação. A Figura 4 avalia a documentação das APIs considerando duas métricas: número de linhas com comentários (NCL) e número relativo de linhas com comentários (RCL). As APIs *populares* possuem mais linhas com comentários (mediana 347) do que as *aleatórias* (mediana 140) e *não populares* (mediana 97) (*p-value*< 0,01, *effect-size* médio e grande). Relativamente, as APIs *populares* possuem mais comentários (mediana 60%) do que as *não populares* (mediana 52%) com *p-value*<0,01 e *effect-size* pequeno. Esse resultado mostra que os desenvolvedores de APIs *populares* possuem maior preocupação com documentação, o que pode facilitar a utilização dessas APIs.

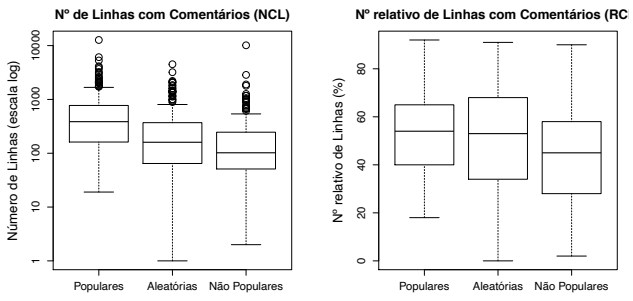

Figura 4. Documentação das APIs.

QP2: Quais as características evolucionárias das APIs populares?

Estabilidade. A Figura 5 apresenta a estabilidade das APIs em relação a três métricas: *lifetime* (em dias), número de *commits* e razão entre *lifetime* e *commits*. As APIs *populares* são mais antigas (mediana 3.593 dias) do que as demais (medianas 3.114 e 2.538 dias), sendo *p-value*<0,01 e *effect-size* médio. Considerando o número de *commits*, as APIs *populares* possuem mais alterações (mediana 14) do que as demais (medianas 8 e 7), (*p-value*<0,01 e *effect-size* pequeno). Para entender melhor esses resultados, a métrica razão entre *lifetime* e *commits* permite inferir que as APIs *populares* mudam mais frequentemente (mediana de 251 dias por *commit*) do que as demais APIs (mediana 277). Em suma, as APIs *populares* são mais antigas, possuem mais *commits* e mudam com mais frequência ao longo do tempo.

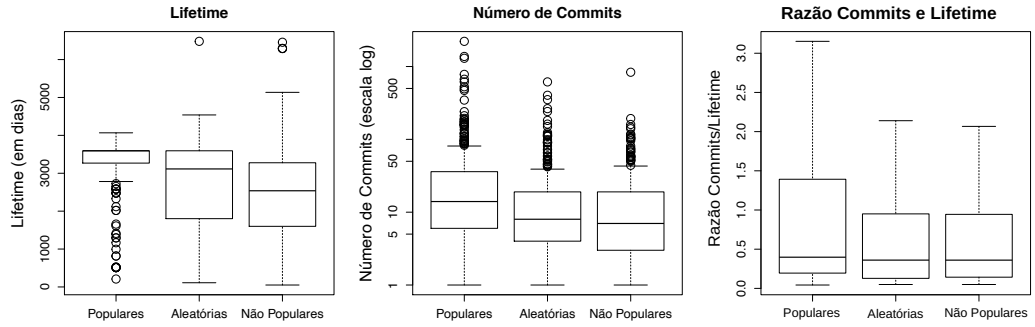

Figura 5. Estabilidade das APIs.

5. Discussão

APIs populares são maiores e mais encapsuladas. Em números absolutos, detectou-se que as APIs *populares* possuem mais métodos públicos. Em termos relativos, entretanto, observou-se que as APIs *populares* possuem proporcionalmente menos métodos públicos (ou seja, elas possuem mais métodos com outras visibilidades), indicando um melhor encapsulamento. De fato, projetistas de bibliotecas devem refletir cuidadosamente sobre a visibilidade das APIs para reduzir da exposição de elementos internos. Elementos internos que são desnecessariamente expostos aos clientes podem causar retrabalho e perda de qualidade na biblioteca [Hora et al. 2016].

APIs populares possuem mais documentação. Tanto em valores absolutos quanto em relativos, detectou-se que as APIs *populares* possuem mais comentários em seus códigos. Desse modo, as APIs *populares* podem facilitar o uso das suas funcionalidades. De fato, desenvolvedores de bibliotecas devem dar atenção especial a documentação; por exemplo, idealmente, todo elemento público deve possuir documentação.

APIs populares são mais antigas e mudam mais frequentemente. As APIs *populares* possuem mais tempo de vida e *commits*. Além disso, elas mudam com maior frequência ao longo do tempo, possivelmente para acomodar demandas dos seus clientes [Xavier et al. 2017, Brito et al. 2018]. Na prática, APIs não são elementos imutáveis, mas evoluem ao longo do tempo; mesmo as APIs populares, com milhares de clientes, podem ser alteradas. Desse modo, desenvolvedores e projetistas de APIs devem avaliar os impactos antes de alteração no código, visando reduzir possíveis quebras de contratos.

6. Ameaças à Validade

Categorização das APIs. Durante o processo de categorização, removeu-se algumas APIs, por exemplo, sem código fonte disponíveis, ou que não eram classes ou interfaces.

Generalização dos Resultados. Analisou-se 1K sistemas Java *open source* hospedados no GitHub. Logo, os resultados não podem ser diretamente generalizados para outros sistemas, especificamente para os implementados em outras linguagens e comerciais.

7. Trabalhos Relacionados

Diversos trabalhos investigam bibliotecas e APIs em relação a aspectos de uso e popularidade. Por exemplo, estudos focam verificar tendências de uso de APIs [Mileva et al. 2009], identificar qual versão atende as necessidades de um cliente [Kim and Notkin 2009] e melhorar documentação de forma automática [Treude and Robillard 2016]. Nota-se também que algumas bibliotecas são mais populares que outras com funcionalidades semelhantes [Zerouali and Mens 2017]. Estudos também avaliam a popularidade de APIs, a fim de sugerir migração de interfaces (*eg*, [Hora and Valente 2015]). No contexto do Android, investigou-se a correlação entre a popularidade dos aplicativos na *PlayStore* com a utilização de APIs menos propensas a erros [Bavota et al. 2015]. No GitHub, investigou-se as caracteristicas de projetos populares [Borges et al. 2016a, Borges et al. 2016b]. Além da aplicação na Engenharia de Software, popularidade é um tema recorrente em outras áreas de pesquisa, principalmente relacionadas à redes sociais. Por exemplo, estudos analisaram a popularidade de vídeos do YouTube [Ahmed et al. 2013, Figueiredo et al. 2014] e hashtags do Twitter [Lehmann et al. 2012].

8. Conclusão

Neste estudo foi apresentado uma análise inicial sobre algumas características de código e evolução encontradas em APIs globalmente utilizadas. Como resultado, detectou-se que APIs populares são maiores, mais encapsuladas, melhor documentadas e alteradas com maior frequência. Como trabalhos futuros, pretende-se: (i) analisar mais sistemas clientes, (ii) expandir a análise para outras categorias de APIs (*eg*, Android e JDK) e (iii) analisar outras métricas de código (*eg*, legibilidade), evolução e uso das APIs.

Agradecimentos: Esta pesquisa é financiada pelo CNPq e pela CAPES.

Referências

Ahmed, M., Spagna, S., Huici, F., and Niccolini, S. (2013). A peek into the future: Predicting the evolution of popularity in user generated content. In *International Conference on Web Search and Data Mining (WSDM)*.

Bavota, G., Linares-Vasquez, M., Bernal-Cardenas, C. E., Di Penta, M., Oliveto, R., and Poshyvanyk, D. (2015). The impact of API change and fault-proneness on the user ratings of android apps. *Transactions on Software Engineering*, 41(4).

Borges, H., Hora, A., and Valente, M. T. (2016a). Predicting the Popularity of GitHub Repositories. In *International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*.

Borges, H., Hora, A., and Valente, M. T. (2016b). Understanding the factors that impact the popularity of GitHub repositories. In *International Conference on Software Maintenance and Evolution (ICSME)*.

Brito, A., Xavier, L., Hora, A., and Valente, M. T. (2018). Why and how Java developers break APIs. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*.

Dyer, R., Nguyen, H. A., Rajan, H., and Nguyen, T. N. (2013). Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *International Conference on Software Engineering (ICSE)*.

Figueiredo, F., Almeida, J. M., Gonçalves, M. A., and Benevenuto, F. (2014). On the dynamics of social media popularity: A youtube case study. *Transactions on Internet Technology (TOIT)*, 14(4):24.

Hora, A. and Valente, M. T. (2015). apiwave: Keeping track of api popularity and migration. In *Inter. Conference on Software Maintenance and Evolution (ICSME)*.

Hora, A., Valente, M. T., Robbes, R., and Anquetil, N. (2016). When should internal interfaces be promoted to public? In *International Symposium on Foundations of Software Engineering (FSE)*.

Kim, M. and Notkin, D. (2009). Discovering and representing systematic code changes. In *International Conference on Software Engineering (ICSE)*.

Lehmann, J., Gonçalves, B., Ramasco, J. J., and Cattuto, C. (2012). Dynamical classes of collective attention in twitter. In *International Conference on World Wide Web (WWW)*.

Mileva, Y. M., Dallmeier, V., Burger, M., and Zeller, A. (2009). Mining trends of library usage. In *International Workshop on Principles on Software Evolution (IWPSE) and ERCIM Workshop on Software Evolution (EVOL)*.

Moser, S. and Nierstrasz, O. (1996). The effect of object-oriented frameworks on developer productivity. *Journal of Computer*, 29(9).

Reddy, M. (2011). *API Design for C++*. Morgan Kaufmann Publishers.

Tian, Y., Nagappan, M., Lo, D., and Hassan, A. E. (2015). What are the characteristics of high-rated apps? a case study on free Android applications. In *International Conference on Software Maintenance and Evolution (ICSME)*.

Treude, C. and Robillard, M. P. (2016). Augmenting API documentation with insights from Stack Overflow. In *International Conference on Software Engineering (ICSE)*.

Xavier, L., Brito, A., Hora, A., and Valente, M. T. (2017). Historical and impact analysis of API breaking changes: A large-scale study. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*.

Zerouali, A. and Mens, T. (2017). Analyzing the evolution of testing library usage in open source Java projects. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*.