# JAVALI: Uma Ferramenta para Análise de Popularidade de APIs Java

**Aline Brito, André Hora, Marco Tulio Valente**

[1]ASERG Group – Departamento de Ciência da Computação (DCC)
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – Minas Gerais – Brasil.

alinebrito@ufmg.br, {hora,mtov}@dcc.ufmg.br

***Abstract.*** *Everyday, libraries are updated, created or removed, and so their APIs. Such changes may impact stakeholders such as APIs providers and clients. In this context, it is important for APIs providers to know about strategic information like APIs change impact on clients and popularity over competitors. On the client side, it is interesting to compare APIs in order to select the most recommended in the market. To address these challenges, we propose JAVALI, a tool to analyze the popularity of Java APIs. Our database is composed of around 260K Java projects and 131M APIs. Our Web interface provides features to view and to export the results. We also report usage examples of JAVALI to solve real world API issues.*

***Resumo.*** *Todos os dias, bibliotecas são atualizadas, criadas ou removidas, assim como suas APIs. Essas alterações podem afetar stakeholders, como provedores de APIs e clientes. Nesse contexto, é importante para os provedores conhecer informações estratégicas, como o impacto das suas atualizações e a sua popularidade sobre os concorrentes. No lado do cliente, é interessante comparar APIs relacionadas ou concorrentes, visando detectar as mais recomendadas no mercado. Para enfrentar esses desafios, esse artigo apresenta JAVALI, uma ferramenta para analisar a popularidade de APIs Java. Utiliza-se um dataset com cerca de 260 mil projetos Java e 131 milhões de APIs. A interface Web possui recursos para visualizar/exportar os resultados. Apresenta-se também exemplos de uso da ferramenta JAVALI para resolver problemas do mundo real.*

***Vídeo da ferramenta.*** `https://youtu.be/N_yWxmEXx9o`

## 1. Introdução

A popularidade de um *software* é uma informação valiosa para compreender o quão bem (ou mal) está a sua evolução. Uma solução para medir a popularidade é através da apreciação do mesmo, verificando métricas em plataformas sociais de armazenamento de código, como por exemplo, o número de estrelas e observadores no GitHub [Borges et al. 2015]. Outra solução é medir o seu uso efetivo, verificando quantos são os seus clientes. Embora seja mais fácil saber o quanto um projeto é popular pelo seu número de estrelas, saber a real frequência da sua utilização é mais desafiador, uma vez que deve-se conhecer todos os sistemas clientes possíveis.

Muitos desses sistemas são bibliotecas[1], que constantemente são atualizadas, criadas e removidas, implicando em alterações nas suas APIs (*Application Programming Interfaces*) [McDonnell et al. 2013, Hora et al. 2014]. Nesse contexto, avaliar o uso de APIs é importante tanto para seus provedores como para seus clientes, uma vez que fornece informações não disponibilizadas por métricas de apreciação. Por exemplo, para o provedor da API android.view.View, comumente utilizada para criar *widgets* em aplicativos Android, saber quantos são os seus clientes e assim o público afetado por uma atualização é uma atividade crítica. Já os clientes, normalmente estão interessados em saber se outros sistemas estão compartilhando a sua decisão de usar uma determinada API, já que existem APIs semelhantes no mercado (por exemplo, para testes unitários existem duas bibliotecas famosas, org.junit e org.testng).

Com o objetivo de auxiliar tanto provedores quanto clientes de APIs nesses desafios, esse trabalho apresenta JAVALI (JAVA *Libraries and Interfaces*), uma ferramenta para analisar a popularidade de APIs Java de acordo com o uso por sistemas clientes, disponível em: `http://java.labsoft.dcc.ufmg.br/javali`.

Para disponibilizar informações de uso de APIs, a ferramenta se vale de um *dataset* com aproximadamente 260 mil projetos Java GitHub e 131 milhões de APIs. Para caracterizar o uso das APIs, a ferramenta concentra-se em dois níveis de granularidade: bibliotecas (por exemplo, quantos clientes estão usando JUnit?) e interfaces (por exemplo, quantos clientes estão usando org.junit.Test?).

O restante desse artigo está organizado conforme descrito a seguir. A Seção 2 descreve as principais funcionalidades, a arquitetura e o *dataset* utilizado pela ferramenta proposta. Na Seção 3, são apresentados exemplos de uso, onde são analisadas algumas interfaces e bibliotecas populares em Java. Finalmente, a Seção 4 discute trabalhos relacionados e a Seção 5 conclui o trabalho.

## 2. JAVALI: JAVA *Libraries and Interfaces*

Através da ferramenta JAVALI pode-se descobrir as interfaces mais utilizadas em Java, e as interfaces mais populares de uma dada biblioteca, assim como comparar duas ou mais bibliotecas/interfaces. A seguir descreve-se as principais funcionalidades disponibilizadas, as características da arquitetura e do *dataset* usado por JAVALI.

### 2.1. Principais Funcionalidades

JAVALI possui quatro telas principais, onde os resultados das consultas são exibidos em gráficos de barras ou tabelas:

***Top Interfaces*** e ***Top Interfaces Charts***. Nessas telas visualiza-se o *ranking* das interfaces mais usadas. O tamanho *ranking* é dinâmico, sendo incrementado pelos usuários.

***Customized Rankings*** e ***Customized Rankings Charts***. Nessas telas pode-se descobrir a quantidade de clientes e as interfaces mais usadas, para uma ou mais bibliotecas. Por exemplo, pode-se verificar qual biblioteca para injeção de dependências é mais popular, javax.inject ou com.google.inject, e quais interfaces de com.google.inject são mais usadas.

A Figura 1 apresenta a página inicial da ferramenta JAVALI, onde visualiza-se as cinco interfaces mais populares e informações sobre o *dataset*.

[1]Nesse trabalho, o termo "biblioteca" é utilizado para designar tanto *frameworks* quanto bibliotecas.

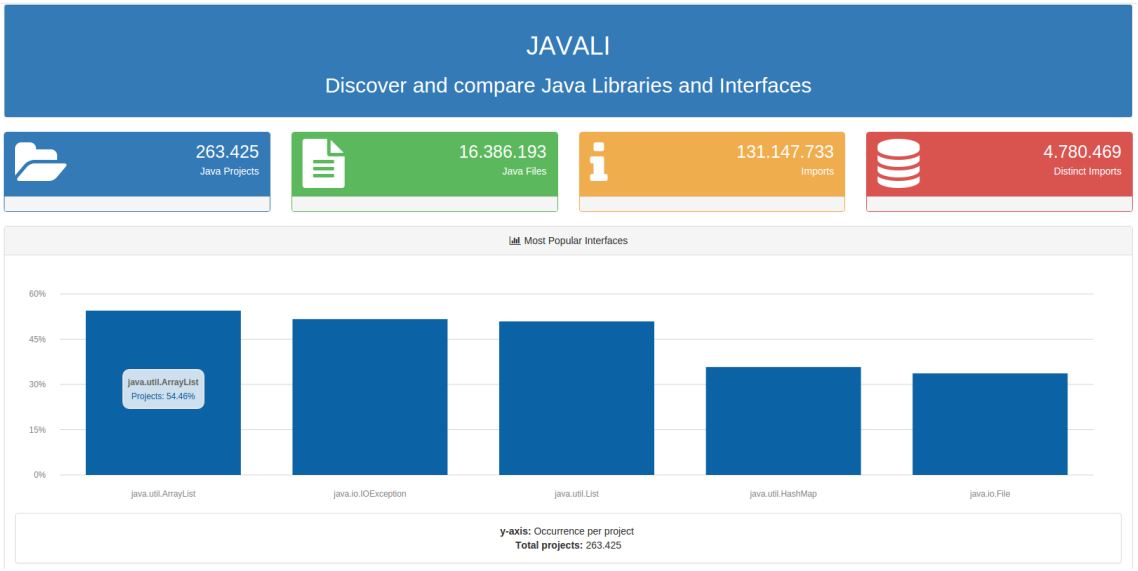



**Figura 1. JAVALI – Página Inicial**

## 2.2. Arquitetura

A Figura 2 apresenta a arquitetura de mais alto nível da ferramenta, que inclui três módulos principais: Mineração de Dados, Processamento e *Front-end*.

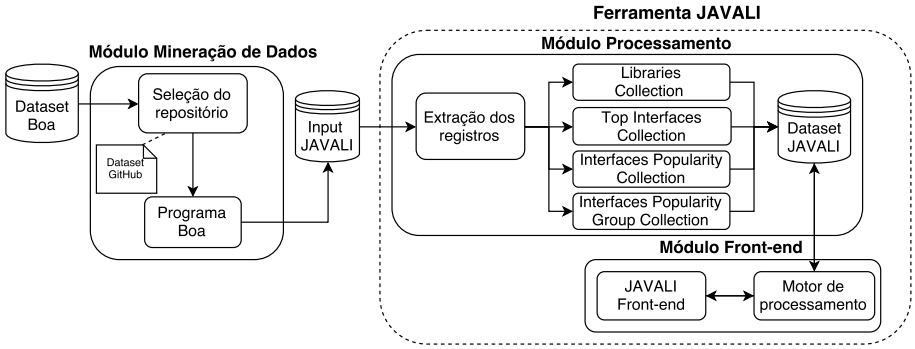



**Figura 2. JAVALI – Arquitetura da Ferramenta**

**Mineração de Dados.** Esse módulo é responsável pela obtenção dos dados da ferramenta JAVALI, minerados através da infraestrutura Boa [Dyer et al. 2013]. A Seção 2.3 apresenta mais detalhes sobre o mesmo.

**Processamento.** Esse módulo fornece os metadados de uso das APIs. Em **Extração dos Registros** os dados são extraídos em formato JSON, sendo organizados em quatro coleções, visando diminuir o custo computacional das consultas. Calcula-se o uso de interfaces por projeto. Assim, evita-se que interfaces muito usadas por poucos clientes sejam contabilizadas indevidamente.

***Front-end*.** Esse módulo é responsável por interpretar as solicitações dos usuários e apresentar os resultados nas telas. Envia-se para o **Motor de Processamento** os parâmetros da consulta, onde as opções são interpretadas e a resposta retornada em formato JSON para a interface *Web*. Em **JAVALI Front-end**, os resultados são verificados, formatados

e apresentados para os usuários. Utilizou-se os *frameworks* Angular, Bootstrap e Node.js para desenvolvimento das páginas *Web* e comunicação com o banco de dados MongoDB.

### 2.3. *Dataset*

JAVALI provê informações a partir de um *dataset* composto por 263.425 projetos Java GitHub, 16.386.193 arquivos Java e 131.147.733 de APIs/bibliotecas. Para extrair esses dados utilizou-se a infraestrutura Boa, uma ferramenta de mineração de *software* em larga escala [Dyer et al. 2013]. A mesma possui uma linguagem própria, caracterizada por uma programação declarativa e tipos específicos de domínio para acessar os metadados dos projetos. Além de uma DSL (*Domain Specific Language*) para mineração dos repositórios de *software*, o ambiente Boa inclui uma interface *Web* para execução dos programas. Utilizou-se o dataset GitHub dessa infraestrutura, composto por 554.864 projetos Java (referente a Setembro/2015).

Para minerar os dados, criou-se um *script* na linguagem Boa para extrair os metadados de todos os projetos que possuem pelo menos um arquivo Java e que importam no mínimo uma interface/biblioteca. Após a execução desse *script*, obteve-se 263.425 projetos válidos. Os metadados extraídos foram então utilizados como entrada para a base de dados atual da ferramenta JAVALI. Adicionalmente, JAVALI aceita entradas de dados de outras fontes, desde que o formato especificado pela ferramenta seja respeitado.

## 3. Exemplos de Uso

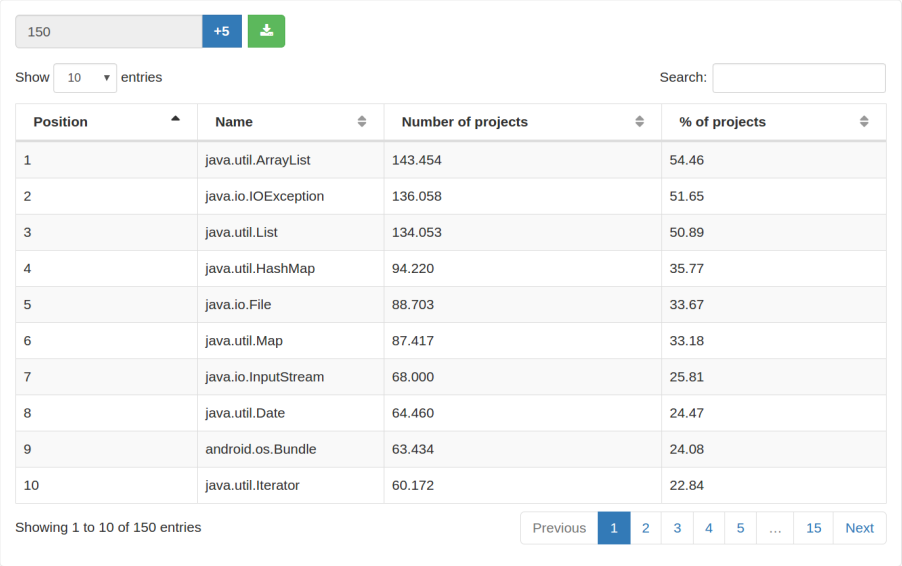
Essa seção apresenta alguns exemplos de uso da ferramenta proposta. Analisa-se no primeiro exemplo quais são as interfaces Java mais populares. No segundo exemplo, investiga-se quais interfaces estão atraindo mais clientes para as bibliotecas Android, Java, e JUnit, por serem amplamente utilizadas e bem consolidadas. No terceiro exemplo de uso, compara-se a popularidade de algumas bibliotecas e interfaces semelhantes.

### 3.1. Interfaces mais Utilizadas na Linguagem Java

O primeiro exemplo concentra-se na tela **Top Interfaces** do JAVALI, onde investiga-se quais são as interfaces mais usadas na linguagem Java. A Figura 3 apresenta a interface *Web* do JAVALI após essa consulta.

Esse *ranking* apresenta as interfaces mais usadas por sistemas do mundo real. Outros tipos de métricas, como por exemplo o número de estrelas, não fornecem esse tipo de informação, pois refletem apenas o apreço dos usuários pelo sistema [Hora and Valente 2015, Mileva et al. 2010]. Além disso, as estrelas informam a popularidade de bibliotecas, enquanto o *ranking* em JAVALI possui uma granularidade menor, onde pode-se descobrir o real uso em nível de interface, ou seja, pode-se visualizar as interfaces mais populares e quais são as suas respectivas bibliotecas.

Dentre as cinco APIs Java mais populares, apresentadas na Figura 1, java.util.ArrayList (55%), java.util.List (51%) e java.util.HashMap (36%) permitem criar estruturas de dados extremamente comuns. Já as APIs java.io.IOException (52%) e java.io.File (34%) são usadas em sistemas que demandam leitura e escrita em disco. As três interfaces Java mais populares são as únicas usadas em mais de 50% dos 263 mil projetos.



150 +5

Show 10 entries Search:

| Position | Name | Number of projects | % of projects |
|---|---|---|---|
| 1 | java.util.ArrayList | 143.454 | 54.46 |
| 2 | java.io.IOException | 136.058 | 51.65 |
| 3 | java.util.List | 134.053 | 50.89 |
| 4 | java.util.HashMap | 94.220 | 35.77 |
| 5 | java.io.File | 88.703 | 33.67 |
| 6 | java.util.Map | 87.417 | 33.18 |
| 7 | java.io.InputStream | 68.000 | 25.81 |
| 8 | java.util.Date | 64.460 | 24.47 |
| 9 | android.os.Bundle | 63.434 | 24.08 |
| 10 | java.util.Iterator | 60.172 | 22.84 |

Showing 1 to 10 of 150 entries Previous 1 2 3 4 5 ... 15 Next

**Figura 3. JAVALI – Tela Top Interfaces**

## 3.2. Interfaces Mais Importantes de uma Biblioteca

Nesse exemplo, analisa-se quais interfaces estão atraindo mais clientes para uma dada biblioteca. Para tanto, a tela **Customized Rankings** do JAVALI foi utilizada para consultar a popularidade das interfaces das bibliotecas JUnit, Java, e Android.

A Figura 4 apresenta a consulta realizada no JAVALI para descobrir as interfaces mais populares da biblioteca JUnit, utilizada para testes unitários. A coluna "% of projects" considera todos os projetos, assim pode-se visualizar a popularidade das interfaces dentre todos os possíveis clientes. Realizou-se esse tipo de consulta para as três bibliotecas citadas anteriormente. A biblioteca JUnit possui uma única interface muito popular, org.junit.Test (19%). Considerando o ecossistema da biblioteca, ou seja, apenas os 54.217 projetos que usam JUnit, esse valor corresponde a 92% dos seus clientes. A diferença entre ela e a próxima na lista, org.junit.Before (10%), é de 22.620 projetos, cerca de 41% dos clientes JUnit. A interface org.junit.Assert.assertEquals é a terceira mais usada na categoria.

As interfaces Java que mais atraem clientes são java.util.ArrayList (55%), java.io.IOException (52%), e java.util.List (51%), sendo as únicas usadas por mais de 50% dos clientes. As interfaces android.os.Bundle (24%) e android.app.Activity (22%) são as mais populares da biblioteca Android. Ainda nessa categoria, cerca de 18% dos clientes usam as interfaces android.view.View e android.content.Context, sendo que o pacote android.content possui duas bibliotecas entre as cinco mais usadas, android.content.Intent (16%) e android.content.Context (18%). A Figura 5 apresenta a tela **Customized Rankings Charts** em JAVALI, com o gráfico de barras das cinco interfaces Android mais populares.

Esse tipo de informação é relevante para os provedores da biblioteca, já que permite descobrir quais interfaces são mais populares dentre os seus clientes, assim como, a quantidade de clientes que serão afetados, por exemplo, pela atualização de uma determinada interface. No lado do cliente, a popularidade das interfaces numa dada biblioteca permite revelar quais interfaces são mais usadas pelos desenvolvedores.

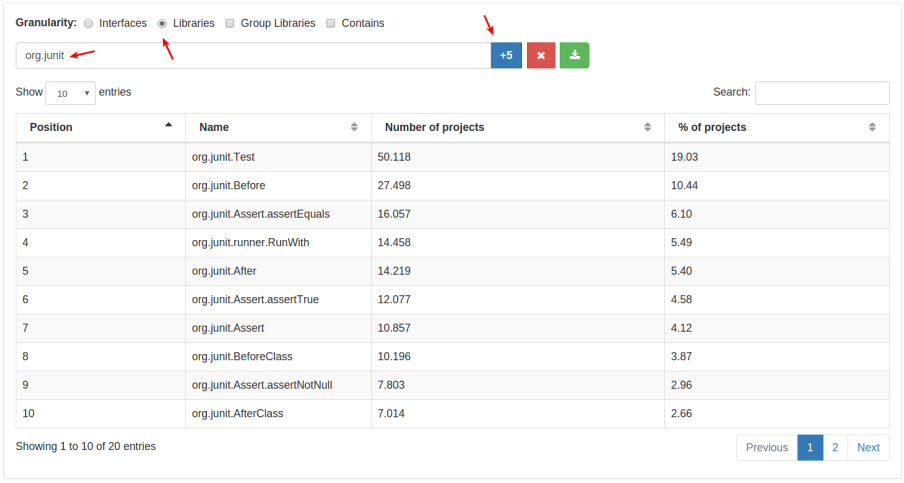



**Figura 4. JAVALI – Interfaces JUnit mais Usadas**

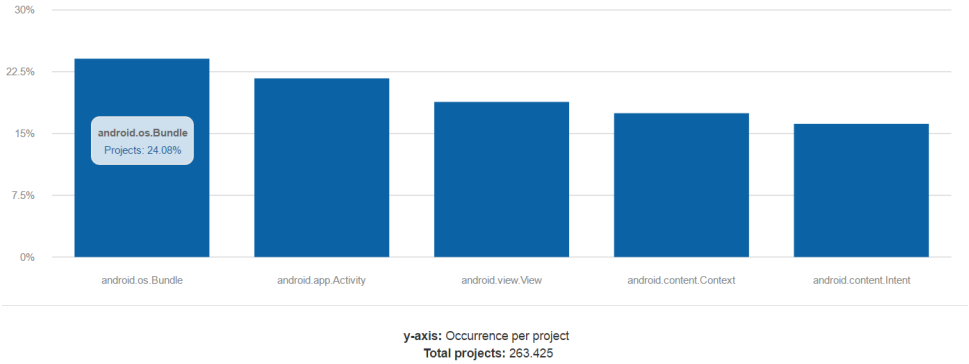



**Figura 5. JAVALI – Interfaces Android mais Usadas**

## 3.3. Comparando Bibliotecas e Interfaces

Utiliza-se nesse exemplo, a tela **Customized Rankings** do JAVALI, para comparar a popularidade de algumas interfaces e bibliotecas. Mais especificamente, compara-se bibliotecas utilizadas para realizar leituras e escritas em disco, para injetar dependências em código-fonte, para especificar testes unitários, e interfaces usadas para instanciar listas.

**Escrita/Leitura em Disco.** Compara-se três bibliotecas, normalmente utilizadas para realizar leitura e escrita em disco: org.apache.commons.io, com.google.common.io e java.io. A Figura 6 apresenta a consulta realizada em JAVALI para comparar as bibliotecas citadas[1]. A biblioteca mais popular é java.io (174.996 projetos - 66.43%), seguida por org.apache.commons.io (11.569 projetos - 4.39%). A terceira posição do *ranking* é ocupada pela biblioteca com.google.common.io (2.998 projetos - 1.14%).

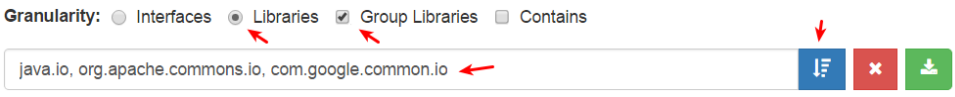



**Figura 6. JAVALI – Comparar Bibliotecas para Leitura/Escrita em Disco**

[1]Utilizou-se esse tipo de consulta para comparar as demais bibliotecas.

**Injeção de Dependências.** Para injetar dependências em código-fonte, existem duas bibliotecas famosas: com.google.inject e javax.inject.Verifica-se que javax.inject é a mais popular, usada em 6.092 projetos, enquanto com.google.inject é usada em 3.808 projetos.

**Testes Unitários.** Compara-se duas bibliotecas no contexto de testes unitários: org.junit e org.testng. A biblioteca JUnit é a mais popular, pois 54.217 projetos (20.58%) utilizam pelo menos uma interface JUnit, enquanto TestNG é usada em 3.588 projetos (1.36%). A diferença entre elas é de 50.629 projetos.

**Instanciar Listas.** Para instanciar listas, duas interfaces são comumente utilizadas: java.util.List e com.google.common.collect.Lists. A interface nativa de Java java.util.List é a mais popular, sendo usada em 134.053 projetos (50.89%), enquanto com.google.common.collect.Lists é usada por 5.120 sistemas clientes (1.94%). A Figura 7 apresenta a consulta realizada em JAVALI para compará-las.

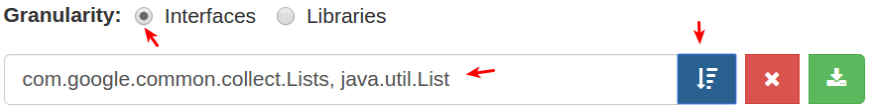



**Figura 7. JAVALI – Comparar Interfaces para Instanciar Listas**

A comparação da popularidade de bibliotecas e interfaces semelhantes permite aos usuários descobrirem aquelas que são mais adotadas pelo mercado. Por outro lado, os provedores podem descobrir a sua popularidade em relação aos concorrentes e o impacto de suas atualizações. Por exemplo, os provedores da biblioteca TestNG podem verificar em JAVALI que sua concorrente JUnit é mais popular, e que provavelmente a sua biblioteca precisa de intervenções para adequar-se ao mercado e atrair mais clientes.

## 4. Trabalhos Relacionados

Existem trabalhos que relatam a constante evolução de APIs [Hora and Valente 2015, Mileva et al. 2010]. Adicionalmente eles mencionam a necessidade de ferramentas que permitam ao cliente conhecer as APIs/bibliotecas mais adotadas no mercado, bem como os provedores conhecerem seu público alvo e abrangência das suas atualizações.

Na literatura, pode-se citar a ferramenta *apiwave*, com recursos que permitem acompanhar a popularidade de APIs e a migração dos clientes para bibliotecas concorrentes [Hora and Valente 2015]. As principais diferenças entre JAVALI e *apiwave* são: (i) JAVALI possui funcionalidades para comparar o uso de interfaces, bibliotecas, e as interfaces mais importantes por biblioteca; (ii) JAVALI disponibiliza dados sobre o uso dos pacotes das bibliotecas, ou seja, além da análise da biblioteca disponibilizada pelas ferramentas (por exemplo, a biblioteca org.springframework), pode-se explorar níveis mais internos (por exemplo, org.springframework.beans e org.springframework.stereotype); (iii) JAVALI possui a funcionalidade "*Contains*" para buscar interfaces/bibliotecas que possuem determinado pacote (por exemplo, interfaces da biblioteca org.eclipse que possuem o pacote ".internal."). Além disso, JAVALI utiliza um *dataset* composto por 263.425 projetos, enquanto *apiwave* utiliza apenas mil projetos. Para obter esse número bastante alto de projetos, JAVALI se beneficiou da infraestrutura Boa [Dyer et al. 2013].

Assim como JAVALI, a linguagem e infraestrutura Boa permite descobrir a popularidade de interfaces/bibliotecas Java [Dyer et al. 2013]. Porém é necessário o conheci-

mento prévio de uma DSL para criar o *script*, consequentemente minerar as informações de uso de interfaces e bibliotecas não é trivial. Por outro lado, JAVALI fornece uma plataforma onde torna-se simples explorar a popularidade de interfaces/bibliotecas, já que os dados são previamente processados, e a interface *Web* viabiliza a interação com o usuário.

Outro estudo no contexto de popularidade de APIs também serviu de inspiração para este trabalho [Mileva et al. 2010]. Porém, a pesquisa não propõe uma ferramenta que permita ao usuário minerar as informações de popularidade conforme a sua necessidade.

## 5. Considerações Finais

Este trabalho apresentou JAVALI, uma ferramenta para análise de popularidade de interfaces/bibliotecas Java. Verificou-se através de exemplos de uso quais são as interfaces Java mais populares e quais interfaces estão atraindo mais clientes para uma dada biblioteca. Além disso, comparou-se a popularidade de algumas interfaces/bibliotecas semelhantes. A partir desses exemplos procurou-se mostrar indícios de viabilidade da ferramenta para ajudar provedores e clientes a conhecerem o quão popular é uma interface/biblioteca.

Trabalhos futuros incluem uma nova versão da ferramenta, com informações de uso de bibliotecas e interfaces para outras linguagens. Todos os dados utilizados nessa pesquisa são públicos (programa Boa e entrada de dados[1], código-fonte da ferramenta JAVALI[2]) e portanto podem subsidiar futuras pesquisas nessa área.

**Agradecimentos:** Esta pesquisa é financiada pela FAPEMIG e pelo CNPq.

## Referências

Borges, H., Valente, M. T., Hora, A., and Coelho, J. (2015). On the popularity of GitHub applications: A preliminary note. *CoRR*, abs/1507.00604.

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 *35th International Conference on Software Engineering*, pages 422–431.

Hora, A., Etien, A., Anquetil, N., Ducasse, S., and Valente, M. T. (2014). APIEvolutionMiner: Keeping API evolution under control. In *IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), Tool Demonstration Track*, pages 420–424.

Hora, A. and Valente, M. T. (2015). apiwave: Keeping track of API popularity and migration. In *31st IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 321–323.

McDonnell, T., Ray, B., and Kim, M. (2013). An empirical study of API stability and adoption in the android ecosystem. In *International Conference on Software Maintenance*, pages 70–79.

Mileva, Y. M., Dallmeier, V., and Zeller, A. (2010). Mining API popularity. In *International Academic and Industrial Conference on Testing - Practice and Research Techniques*, pages 173–180.

[1] `http://boa.cs.iastate.edu/boa/index.php?q=boa/job/public/35521`
[2] `https://github.com/alinebrito/JAVALI`