Um Estudo em Larga Escala sobre Estabilidade de APIs

Laerte Xavier, Aline Brito, André Hora, Marco Tulio Valente

[1]ASERG, Departamento de Ciência da Computação (DCC)
Universidade Federal de Minas Gerais (UFMG), Brasil

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

Abstract. *APIs are constantly being evolved. As a consequence, their clients are compelled to update and, thus, benefit from the available improvements. However, some of these changes may break contracts previously established, resulting in compilation errors or behavioral changes. In this paper, questions related to API evolution and stability are studied, with the purpose of characterizing (i) the frequency of changes inserted, (ii) the behavior of these changes among time and (iii) the impact in client applications. Therefore, the top-100 GitHub most popular Java libraries are analyzed, and so their possible clients. As a result, insights are provided for the development of tools to support both library developers and clients in evolution and maintenance activities.*

Resumo. *APIs estão em constante evolução. Como consequência, seus clientes são compelidos a atualizarem-se e, assim, aproveitarem as melhorias disponibilizadas. Entretanto, algumas dessas mudanças podem quebrar contratos previamente estabelecidos, resultando em erros de compilação ou mudanças comportamentais. Neste artigo, estudam-se questões relativas a evolução e estabilidade de APIs, visando caracterizar (i) a frequência de mudanças inseridas, (ii) o comportamento dessas mudanças ao longo do tempo e (iii) o impacto em aplicações clientes. Dessa forma, foram analisadas as 100 bibliotecas Java mais populares do GitHub, bem como seus possíveis clientes. Como resultado, são apresentados insights de ferramentas para auxiliar ambos os clientes e os desenvolvedores de bibliotecas em suas atividades de evolução e manutenção.*

1. Introdução

Em desenvolvimento de *software*, mudança é uma constante. Estima-se que 80% do custo desse processo está relacionado às fases de manutenção e evolução. Nesse contexto, APIs (*Application Programming Interfaces*) também estão susceptíveis a atualizações e, como consequência, aplicações clientes são naturalmente compelidas a migrarem para novas versões. Compatibilidade torna-se, então, um desafio relevante na evolução de APIs, uma vez que mudanças introduzidas em uma nova versão podem quebrar contratos previamente assumidos com milhares de clientes.

Dessa forma, as mudanças em APIs podem ser classificadas em ***breaking changes:*** quebram compatibilidade com versões anteriores, alterando ou removendo elementos previamente disponíveis; e ***non-breaking changes:*** não quebram compatibilidade e, em geral, incluem adição de novos componentes ou extensão de funcionalidades [Dig and Johnson 2006]. Assim, diversos trabalhos utilizam essa classificação com o objetivo de analisar questões sobre a evolução e a estabilidade de

APIs [Raemaekers et al. 2012, McDonnell et al. 2013]. No entanto, nota-se que esses estudos são realizados apenas em pequena escala, ou seja, num contexto onde poucos sistemas são avaliados. Além disso, eles possuem uma importante restrição em suas validações, uma vez que não avaliam o impacto das *breaking changes* nos reais interessados, i.e., as aplicações clientes.

Neste trabalho, essas questões são analisadas em larga escala, relacionando os resultados obtidos ao possível impacto produzido em clientes reais. Tem-se como objetivo medir *(i)* a frequência de mudanças entre versões de bibliotecas, *(ii)* o comportamento dessas mudanças ao longo do tempo, e *(iii)* o impacto em aplicações clientes. Especificamente, são propostas três questões de pesquisa centrais:

QP #1. Qual a frequência de mudanças de bibliotecas?
QP #2. Como a frequência de *breaking changes* se comporta ao longo do tempo?
QP #3. Qual o impacto real das *breaking changes* em aplicações clientes?

Para responder a essas questões, são analisadas as 100 bibliotecas Java mais populares do GitHub, bem como potenciais clientes afetados por alterações em seus componentes. Assim, as principais contribuições deste trabalho são: *(i)* prover dados e análises para auxiliar clientes a escolherem bibliotecas baseados no critério de estabilidade, *(ii)* prover bases para o desenvolvimento de uma ferramenta que mitigue os riscos de migração entre versões de bibliotecas, e *(iii)* fornecer *insight* de ferramenta que alerte os desenvolvedores de bibliotecas acerca do impacto de possíveis mudanças pretendidas.

O restante deste artigo está organizado da seguinte forma: a Seção 2 aprofunda as definições de *breaking change* e *non-breaking change*, apresentando o catálogo de mudanças utilizado. Na Seção 3 é descrita a metodologia dos estudos realizados e na Seção 4 são apresentados os resultados obtidos. A Seção 5 discute os resultados, apresentando aplicações práticas. Na Seção 6 são apresentados os riscos à validade deste estudo e, por fim, as Seções 7 e 8 discutem trabalhos relacionados e as conclusões obtidas.

2. Catálogo de Mudanças em APIs

APIs são definidas como componentes de um sistema de *software* que podem ser reusados por aplicações clientes. Utilizam-se, portanto, de modificadores de visibilidade para expor interfaces ditas estáveis (e.g., em Java utiliza-se `public` ou `protected`). Durante seu ciclo de vida, entretanto, estão sujeitas a mudanças evolutivas, tais como adição, remoção, modificação ou depreciação de seus elementos. Essas mudanças foram catalogadas em estudos anteriores no contexto de *refactoring* [Dig and Johnson 2006] e são classificadas em *breaking change* e *non-breaking change*. Neste trabalho, utiliza-se tal classificação, aplicando o catálogo proposto sem as referências a refatoração.

Dessa forma, são consideradas ***breaking changes*** alterações abruptas que quebram contratos estabelecidos, sem notificação prévia por meio de mensagens de depreciação. São elas: remoção de elementos (tipos, atributos ou métodos); modificação da assinatura de métodos (nome, visibilidade e tipos de retorno, de exceção ou de parâmetro); e alterações em atributos (tipos e valores de inicialização). Por outro lado, consideram-se ***non-breaking changes*** aquelas mudanças inseridas em elementos depreciados, ou aquelas que adicionam novas funcionalidades, mas mantém compatibilidade com clientes. Em geral, representam adição de elementos ou funcionalidades às bibliotecas.

3. Metodologia

Seleção de Sistemas. A fim de selecionar os sistemas cujas APIs serão analisadas, utilizou-se uma base de dados desenvolvida em trabalhos anteriores [Brito et al. 2016]. Nessa base, foram classificados em *Library* e *Non-Library* 623 repositórios Java hospedados no GitHub, escolhidos e ordenados pelo número de estrelas. Desses, foram selecionados para análise no presente estudo os 100 sistemas mais populares com classificação de *Library*. Dessa forma, os sistemas analisados neste trabalho possuem, na mediana, 1.190,5 estrelas e 23,5 *releases*. Entre os melhores classificados, destacam-se: NOSTRA13/ANDROID-UNIVERSAL-IMAGE-LOADER com 9.793 estrelas e 28 *releases*; SQUARE/PICASSO com 6.948 estrelas e 20 *releases*; e LIBGDX/LIBGDX, com 6.839 estrelas e 29 *releases*.[1]

Extração de Métricas. Com o propósito de responder às questões de pesquisa propostas, foram coletadas as frequências de ocorrência das *breaking changes* e *non-breaking changes* descritas na Seção 2. Para tanto, foi implementado um *parser* baseado na biblioteca `JDT` do Eclipse que compara tais mudanças entre as versões analisadas. Sendo N a última *release* lançada do sistema, e 1 a primeira, na *QP #1* essas frequências foram comparadas entre as versões N e $N-1$ (finais). Já para a *QP #2*, estendeu-se a coleta para as versões 1 e 2 (iniciais) e $N/2$ e $N/2-1$ (intermediárias). Assim, observa-se que tais *releases* constituem momentos representativos do ciclo de desenvolvimento das bibliotecas (iniciais, intermediários e finais), mitigando o alto custo de análise de todo histórico de versões.

Por outro lado, para responder a *QP #3*, utilizou-se o `JAVALI`[2], uma ferramenta para comparar e categorizar APIs Java de acordo com o uso por aplicações clientes. Ela utiliza um *dataset* com 263.425 projetos e 16.386.193 arquivos Java hospedados no GitHub e cujas informações foram extraídas utilizando a linguagem e infraestrutura `Boa` [Dyer et al. 2013]. Os clientes das bibliotecas em estudo foram, então, identificados através da análise das declarações de *import* nos projetos disponíveis no `Boa`, buscando por referências aos tipos modificados. Dessa forma, foram analisadas as quantidades de projetos e arquivos possivelmente impactados pelos tipos que sofreram *breaking change* em pelo menos um de seus elementos nas versões finais das bibliotecas analisadas.

4. Resultados

QP #1: Qual a frequência de mudanças de bibliotecas?

A frequência de mudanças foi calculada para tipos, atributos e métodos entre as duas últimas versões das 100 bibliotecas analisadas. Desse total, 81 sistemas apresentaram pelo menos uma *breaking change*, somando 70.040 mudanças (23,3%). Por outro lado, 230.064 *non-breaking changes* foram observadas em 82 sistemas (76,7%). Com o objetivo de estudar a estabilidade dessas bibliotecas nas versões mais recentes, foram analisados os valores absolutos para ambos os tipos de mudanças e, em seguida, avaliados os valores percentuais daquelas mudanças que afetam aplicações clientes.

A distribuição dos valores absolutos registrados para cada elemento de API é apresentada na Figura 1. O primeiro quartil, a mediana e o terceiro quartil de *breaking changes* em tipos são iguais a 0, 1 e 25,5. Para mudanças em atributos, o primeiro quartil é 0, a

[1] A lista completa dos sistemas avaliados está disponível em: `https://goo.gl/VaQ8jI`

[2] `http://java.labsoft.dcc.ufmg.br/javali`

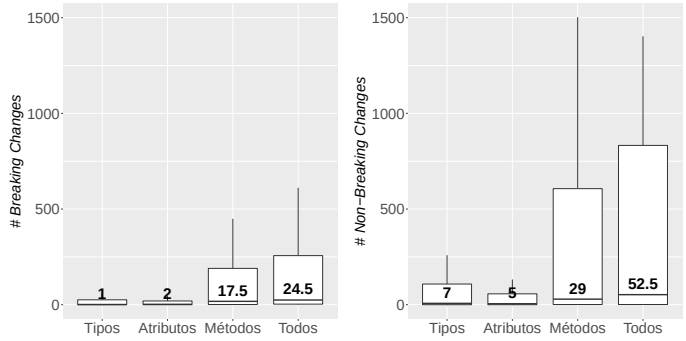

Figura 1. Total de mudanças na versão final dos sistemas analisados.

mediana, 2, e o terceiro quartil, 19,5. *Breaking changes* em métodos apresentam ainda primeiro quartil igual a 1, mediana igual a 17,5, e terceiro quartil 189,7. Por outro lado, a frequência de *non-breaking changes* em tipos tem primeiro quartil 0, mediana 7 e terceiro quartil 108. Ainda nesse contexto, atributos apresentam primeiro quartil, mediana e terceiro quartil iguais a 0, 5 e 56,7. Para métodos, observa-se que o primeiro e terceiro quartis são, respectivamente, 1 e 606,5, com mediana 29. Por fim, considerando todos os elementos de API, os valores obtidos são: 3, 24,5 e 256,5, para *breaking changes*; e 1, 52,5 e 832,5, para *non-breaking changes*.

Em termos percentuais, a Figura 2 apresenta a distribuição das taxas de ocorrência de *breaking changes* nos elementos de API das bibliotecas em estudo, em relação ao seu total de mudanças. Dessa forma, considerando todos os elementos analisados, observa-se uma elevada taxa de 24,6% de mudanças desse tipo, com destaque para atributos e métodos, com valores medianos correspondentes a 22,1% e 21,3%, respectivamente. Por fim, verifica-se que apenas 5,7% das mudanças em tipos são *breaking changes*.

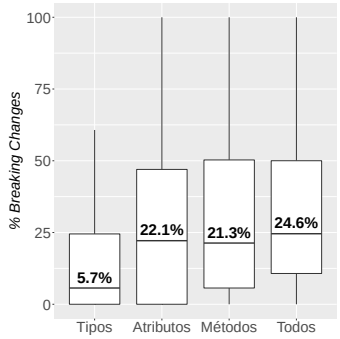

Figura 2. Taxa de *breaking changes* na versão final dos sistemas analisados.

Resumo: Em valores medianos, 24,6% das mudanças que ocorrem em APIs afetam aplicações clientes, isto é, são *breaking changes*. Ou seja, um quarto das interfaces consideradas estáveis podem afetar seus clientes, causando problemas de compatibilidade.

QP #2: Como a frequência de breaking changes se comporta ao longo do tempo?

A fim de analisar a estabilidade durante o ciclo de vida das 100 bibliotecas em estudo, foram calculadas as frequências de *breaking changes* entre as versões iniciais e inter-

mediárias, comparando-as com os valores finais obtidos na *QP #1*. Entre as versões iniciais, foram encontradas 76.047 mudanças desse tipo. Por outro lado, 73.053 e 70.040 foram registradas nas versões intermediárias e finais, respectivamente.

A Figura 3 apresenta a distribuição do percentual de *breaking changes* em cada elemento de API, em relação ao seu total de mudanças, nas versões iniciais, intermediárias e finais. Para mudanças em tipos, observa-se que os valores obtidos nas medianas são iguais a 13,6%, 18,6% e 5,7%. Já para atributos, a mediana entre versões da fase inicial é 29,2%, 33,3% da intermediária e 22,1% da final. Por fim, nota-se que as medianas para métodos em cada uma das fases analisadas são, respectivamente: 26,6%, 36,7% e 21,3%.

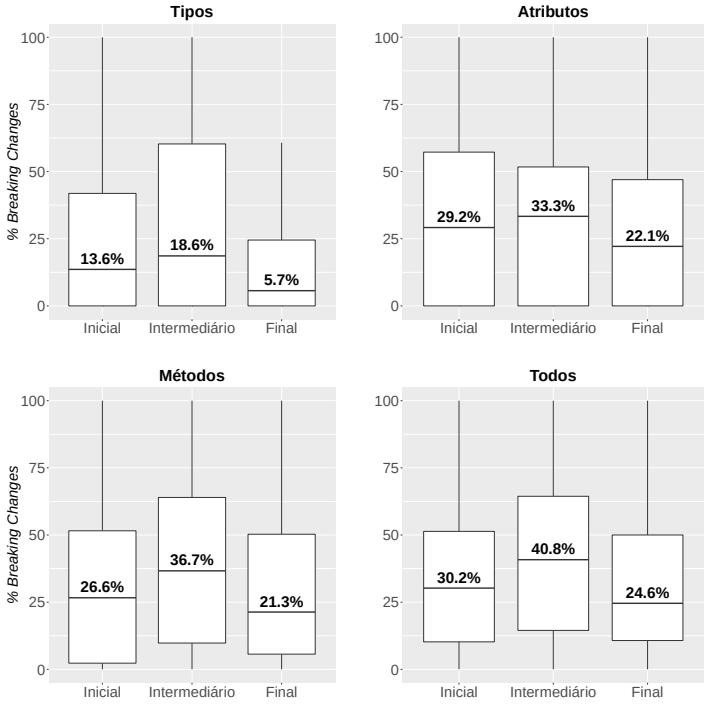

Figura 3. Taxa de *breaking changes* nas versões inicial, intermediária e final.

Considerando todos os elementos de API, observa-se que a taxa de *breaking changes* tende a aumentar entre as fases inicial e intermediária, com medianas iguais a 30,2% e 40,8%, respectivamente. Entretanto, observa-se também que, na segunda metade do seus tempos de vida, esses valores diminuem de maneira que a taxa de mudanças entre as versões intermediária e final possuem medianas iguais a 40,8% e 24,6%. Em ambos os casos, foram obtidos *p-values* $< 0{,}05$ no teste de Mann-Whitney. Isto é, existe uma diferença estatística na taxa de *breaking changes* entre as versões analisadas.

Resumo: Na primeira metade do tempo de vida das APIs, o percentual de *breaking changes* aumenta; na metade final, observa-se uma redução relativa desse tipo de mudança. Isso mostra que interfaces fornecidas por bibliotecas tendem a ficar mais estáveis.

QP #3: Qual o impacto real das breaking changes em aplicações clientes?

Com o propósito de mensurar o possível impacto das *breaking changes* em aplicações clientes, foram sumarizados os tipos que sofreram essas alterações nas versões finais das

bibliotecas estudadas. Em seguida, minerou-se, a partir dos 263 mil projetos e 16 milhões de arquivos do `JAVALI`, o número de clientes de tais tipos em termos de projetos e arquivos. Do total de 70.040 *breaking changes* registradas em todos os elementos de API, observa-se que tais mudanças estão relacionadas a 10.206 tipos. Desses, 4.024 possuem pelo menos uma aplicação cliente que foi, possivelmente, afetada.

Na Figura 4, apresenta-se a distribuição da quantidade de projetos e arquivos impactados por tipos que sofreram alguma *breaking change* e possuem pelo menos uma aplicação cliente. No primeiro quartil, verifica-se que 1 projeto e 3 arquivos são possivelmente afetados. Na mediana, verifica-se que esses valores são iguais a 4 e 11, respectivamente. Por fim, no terceiro quartil, observam-se os totais de 11 projetos e 37 arquivos.

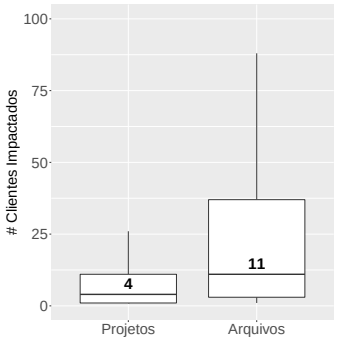

Figura 4. Projetos e arquivos impactados por *breaking changes* na versão final.

Apesar dos tipos que sofreram *breaking changes* nas APIs das bibliotecas estudadas não possuírem, no geral, quantidades elevadas de clientes, alguns casos merecem destaque. Por exemplo, o tipo `org.hibernate.Query` da biblioteca HIBERNATE/HIBERNATE-ORM, que é utilizado em 3.594 projetos e 20.897 arquivos clientes. Nesse sistema, entre as *releases* 4.2.23 e 5.2.0 foram inseridas um total de 26 *breaking changes*, algumas delas reportadas pelos usuários. Essas mudanças geraram a criação de uma *issue* de alta prioridade que acarretou na restauração do código legado menos de 30 dias após o lançamento da nova versão.[3]

Resumo: Em valores medianos, 4 projetos e 11 arquivos clientes são possivelmente impactados pelas *breaking changes* nas bibliotecas analisadas. O baixo impacto nos clientes fornece indícios de que os tipos alterados são internos, ou que os desenvolvedores são cautelosos quando introduzem *breaking changes* em certos elementos.

5. Aplicações Práticas

A partir dos resultados obtidos, observa-se que clientes devem tomar precauções antes de escolherem uma biblioteca, bem como antes de decidirem atualizar a versão daquelas que já utilizam. Por outro lado, seus desenvolvedores devem ser cautelosos antes de produzirem *breaking changes* em elementos muito utilizados. Dessa forma, evidenciam-se as seguintes aplicações práticas:

[3]Maiores detalhes sobre a *issue* podem ser encontrados em: `https://hibernate.atlassian.net/browse/HHH-10839`

Aplicação #1: A fim de auxiliar clientes a decidirem a respeito da utilização de uma determinada biblioteca, as métricas utilizadas neste trabalho podem ser estendidas para o histórico completo de versões, construindo-se uma curva que exponha o comportamento evolutivo de uma biblioteca em termos de *breaking changes*, e forneça uma visão geral da estabilidade do sistema.

Aplicação #2: Uma ferramenta de análise de impacto com o propósito de auxiliar o cliente a mensurar o custo de migração entre versões de bibliotecas. Para tanto, deve-se observar quais elementos sofreram *breaking changes* entre a versão atual do cliente e a que se pretende atualizar; em seguida, analisar o código do cliente a fim de observar quais dessas mudanças o impactam diretamente.

Aplicação #3: Uma ferramenta de análise de mudanças cujo objetivo seja alertar os desenvolvedores de biblioteca a respeito do impacto de possíveis *breaking changes*. Dessa forma, antes de fazer *commit* de alguma modificação que possivelmente afete pelo menos um cliente externo, um alerta seria produzido informando o impacto daquela mudança e sugerindo a utilização de mensagens de depreciação.

6. Ameaças à Validade

Validade Externa. Este estudo limitou-se à analise de 100 bibliotecas *open source* Java. Portanto, seus resultados não podem ser generalizados para outras linguagens ou para sistemas comerciais. Entretanto, foram selecionados sistemas relevantes, com alta popularidade, e classificados como *Library*, aumentando, assim, a relevância dos resultados.

Validade Interna. Dentre os aspectos que podem afetar os resultados apresentados, destaca-se o *parser* implementado para contabilizar as *breaking changes*. A fim de minimizar essa ameaça, utilizou-se a biblioteca `JDT` do Eclipse. Além disso, o teste de Mann-Whitney foi utilizado com o objetivo de assegurar a validade das análises.

Validade de Construção. Para cada um dos sistemas em estudo, foram analisadas seis versões, em três fases distintas: inicial, intermediária e final. Entretanto, observa-se que essa análise não caracteriza todo o seu desenvolvimento. Dessa forma, não se pode afirmar que os resultados apresentados refletem a evolução completa dos sistemas analisados.

7. Trabalhos Relacionados

Diversos trabalhos abordam questões sobre evolução de APIs, propondo ferramentas de apoio aos desenvolvedores [Hora and Valente 2015, Hora et al. 2014, Henkel and Diwan 2005], e fornecendo melhor entendimento acerca das características de mudanças [Dig and Johnson 2006]. Por outro lado, existe um esforço no sentido de caracterizar e mensurar a estabilidade de APIs [Raemaekers et al. 2012], apresentando-se evidências do impacto de *breaking changes* em aplicações clientes [Brito et al. 2016, McDonnell et al. 2013, Robbes et al. 2012].

No entanto, nota-se que esses estudos são realizados apenas em pequena escala, num contexto onde poucos sistemas são avaliados. Além disso, eles possuem uma importante restrição em suas validações, uma vez que não avaliam o impacto das *breaking changes* nas aplicações clientes. Este trabalho avança o entendimento dessas questões, a partir de um estudo em larga escala, e da análise do impacto nos clientes, i.e., 100 bibliotecas, 263 mil sistemas clientes e 16 milhões de arquivos clientes.

8. Conclusões

Neste estudo, caracterizou-se a estabilidade de APIs através da análise de ocorrência de *breaking changes* em 100 bibliotecas *open source* Java. Verificou-se que uma quantidade relevante de alterações correspondem a esse tipo de mudança (24,6%). Observou-se, ainda, que esses valores são maiores em versões anteriores (30,2% na inicial e 40,8% na intermediária), sugerindo uma possível estabilização com o passar do tempo. Por fim, constatou-se que o impacto dessas mudanças é relativamente baixo (4 projetos e 11 arquivos), fornecendo indícios de que os tipos usualmente alterados são internos, ou que os desenvolvedores são cautelosos quando introduzem *breaking changes* em elementos muito utilizados. A partir desses resultados, três aplicações práticas foram brevemente apresentadas com o objetivo de auxiliar tanto clientes como desenvolvedores de bibliotecas em suas atividades de evolução e manutenção.

Agradecimentos

Esta pesquisa é financiada pela FAPEMIG, e pelo CNPq.

Referências

Brito, G., Hora, A., Valente, M. T., and Robbes, R. (2016). Do developers deprecate APIs with replacement messages? A large-scale analysis on Java systems. In *23rd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 360–369.

Dig, D. and Johnson, R. (2006). How do APIs evolve? A story of refactoring. In *22nd International Conference on Software Maintenance (ICSM)*, pages 83–107.

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 (ICSE)*, pages 422–431.

Henkel, J. and Diwan, A. (2005). Catchup!: Capturing and replaying refactorings to support API evolution. In *27th International Conference on Software Engineering (ICSE)*, pages 274–283.

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 *29th International Conference on Software Maintenance (ICSM)*, pages 70–79.

Raemaekers, S., van Arie Deursen, and Visser, J. (2012). Measuring software library stability through historical version analysis. In *28th International Conference on Software Maintenance (ICSM)*, pages 378–387.

Robbes, R., Lungu, M., and Rothlisberger, D. (2012). How do developers react to API deprecation? The case of a smalltalk ecosystem. In *20th International Symposium on the Foundations of Software Engineering (FSE)*, pages 56:1–56:11.