Violação de padrões de uso de APIs em sistemas configuráveis

Bruno Mecca[1], **Diogo Boaventura**[1], **Bruno B. P. Cafeo**[1], **Elder Cirilo**[2]

[1] Faculdade de Computação – Universidade Federal de Mato Grosso do Sul (UFMS)
Cidade Universitária – 79070-900 – Campo Grande – MS

[2]Departamento de Ciência da Computação – Universidade Federal de São João del-Rei (UFSJ)
Campus Tancredo de Almeida Neves (CTAN) – 36301-360 – São João del-Rei – MG

{bruno.mecca,diogo.b.fonseca}@aluno.ufms.br,
cafeo@facom.ufms.br, elder@ufsj.edu.br

Resumo. *API (Application Programming Interface) tem sido vastamente adotada no desenvolvimento de software. Com isso, é comum que funções de API sejam utilizadas em diferentes contextos, assim como também frequentemente aplicadas de maneira conjunta de modo que seus usos seguem algumas regras ou padrões de uso. No entanto, não se sabe se violações nos padrões de uso ocorrem em sistemas implementados com diretivas de pré-processamento. Dessa forma, o objetivo deste estudo é verificar se ocorrem violações de padrões de uso em sistemas configuráveis devido a diretivas de pré-processamento. Para isso, foram extraídos padrões de uso de quatro sistemas configuráveis de código aberto: totalizando 253.589 linhas de código analisadas e mais de 9.848 commits. Após a identificação dos padrões, foram analisadas possíveis violações de uso. Como principais contribuições, pode-se destacar um primeiro estudo que mostra que violações de padrões de uso de APIs podem ocorrer devido à diretivas de pré-processamento.*

1. Introdução

API (*Application Programming Interface*) é uma importante forma de reúso que tem sido vastamente adotada no desenvolvimento de software [Zhong et al. 2009]. É comum que uma função de API seja utilizada em diferentes contextos ao longo do ciclo de vida de software, independentemente se a API é pública (ex., *Java Development Kit* e *C Standard Library*) ou privada (funções internas do código). Algumas funções de API são frequentemente utilizadas de maneira conjunta e seus usos seguem algumas regras que o desenvolvedor aplica para a implementação de funcionalidades. A combinação de funções de API que geralmente são invocadas conjuntamente é chamada de padrão de uso [Zhong et al. 2009]. Exemplos ilustrativos de padrões de uso de APIs citados em diversos estudos da área são: *lock()* e *unlock()*, *open()* e *close()*, e *start()* e *stop()*.

Sistemas configuráveis, assim como a maioria dos sistemas atuais, também fazem uso de APIs. Sistemas configuráveis compartilham um núcleo comum e também possuem diferentes funcionalidades em suas configurações resultantes [Apel et al. 2013]. Quando consideramos sistemas configuráveis implementados em linguagem C, desenvolvedores comumente utilizam a técnica de pré-processamento para anotar a implementação correspondente à variabilidade (ou opção de configuração) devido à sua flexibilidade e expressividade [Apel and Kästner 2009]. O pré-processador identifica o código que deve ser compilado ou não baseado em diretivas de pré-processamento, tais como `#ifdef`. Desenvolvedores usam as diretivas de pré-processamento para envolver desde estruturas inteiras de código (ex., funções) até uma simples variável.

Vários estudos criticam o uso de pré-processamento devido aos seus efeitos negativos no entendimento de código, manutenibilidade e propensão a erros [Ernst et al. 2002, Medeiros et al. 2015]. Além disso, diversos trabalhos exploram a identificação de padrões de uso de APIs [Li and Zhou 2005, Wang and Godfrey 2013]. No entanto, não existem trabalhos que explorem padrões de uso no contexto de sistemas configuráveis implementados com pré-processadores. Mais especificamente, não se sabe, devido à complexidade inerente de códigos com diretivas de pré-processamento, se padrões de uso de APIs podem ser violados.

Este trabalho tem por objetivo verificar violações de padrões de uso de APIs (internas e externas) em sistemas configuráveis implementados com diretivas de pré-processamento. Para isso, foram extraídos padrões de uso utilizando uma adaptação das técnicas [Li and Zhou 2005]. Após a identificação dos padrões, buscou-se possíveis violações de uso devido às diretivas de pré-processamento. Quatro sistemas configuráveis de código aberto foram analisados, totalizando 253,589 linhas de código e mais de 9,848 commits. Como principais contribuições, pode-se destacar uma lista de padrões de uso de APIs para os sistemas analisados, bem como um primeiro estudo que mostra que violações de padrões de uso de APIs podem ocorrer devido ao uso de diretivas de pré-processamento.

2. Problema Motivacional

Nessa seção são apresentados os conceitos básicos (Seção 2.1) para o entendimento do problema abordado no estudo exploratório (Seção 2.2).

2.1. Fundamentação Teórica

O pré-processamento é usado em muitos sistemas para suportar a implementação de variabilidades. Desenvolvedores comumente utilizam diretivas de pré-processamento, tais como `#ifdef` e `#endif`, para anotar blocos de código-fonte como condicionais. O pré-processador identifica o código que deve ser compilado ou não com base em diretivas de pré-processamento que envolvem estruturas de código associando-as a uma variabilidade. Na Figura 1 observam-se fragmentos de código do sistema *Hexchat* cercados por diretivas de pré-processamento. As instruções `#ifdef` e `#endif` definem as fronteiras de parte da implementação da variabilidade e indicam ao pré-processador o que incluir no processo de compilação. Dessa forma, as instruções entre as linhas 03 e 04 só serão compiladas se a variabilidade `USE_PLUGIN` estiver definida. Por fim, o código da linha 09 só será compilado se a variabilidade `USE_DBUS` estiver definida.

```
01.    static void irc_init (session *sess){
02.       #ifdef USE_PLUGIN
03.          if (!arg_skip_plugins)
04.             plugin_auto_load (sess); /* autoload ~/.xchat *.so */
05.       #endif
06.
07.       #ifdef USE_DBUS
08.          plugin_add (sess, NULL, NULL, dbus_plugin_init, NULL, NULL, FALSE);
09.       #endif
10.
11.       snprintf (buf, sizeof (buf), "%s/%s", get_xdir_fs (), "startup.txt");
12.       load_perform_file (sess, buf);
13.    }
```

Figura 1. Violação de padrão de uso na presença de variabilidades.

2.2. Descrição do Problema

Ao longo da implementação de sistemas configuráveis, desenvolvedores comumente utilizam APIs para fazer uso de funcionalidades já implementadas internamente ou em bibliotecas externas. Muitas vezes restrições implícitas podem existir quando se faz uso de funções de APIs. Isto é, o desenvolvedor deve obedecer tais restrições, comumente chamadas de padrões de uso de APIs, para garantir a execução adequada das funcionalidades desejadas no sistema. Padrões de uso de APIs geralmente são caracterizadas por pares de chamadas de funções de APIs que são utilizadas de maneira conjunta [Li and Zhou 2005]. Na Figura 1 podemos notar as chamadas das funções *plugin_auto_load* e *plugin_add*. As funções de API *plugin_auto_load* e *plugin_add* são consideradas padrões de uso, pois comumente estão relacionadas no código-fonte para a implementação de uma ou mais funcionalidades.

Pré-processadores são populares por conta da flexibilidade e simplicidade de uso, além de estarem incorporados em muitas linguagens de programação (ex.: C, C++, Java Micro Edition). No entanto, vários estudos apontam para uma crescente complexidade e ofuscamento de código, o chamado *#ifdef hell* [Apel and Kästner 2009]. Tal situação pode levar à uma dificuldade de entendimento e altos custos de manutenção [Medeiros et al. 2015]. Nesse contexto, esse trabalho supõe que padrões de uso de APIs podem ser violados devido à complexidade inerente de código com diretivas de pré-processamento. Na Figura 1 podemos notar que as chamadas da função de API *plugin_auto_load* ocorre no contexto da variabilidade `USE_PLUGIN` e a chamada da função de API *plugin_add* ocorre no contexto da variabilidade `USE_DBUS`. Caso uma das variabilidades não seja selecionada para compor a configuração final do software, podemos ter um problema devido à violação de um padrão de uso. Ou seja, não haverá violação do padrão de uso apenas se uma configuração contiver ambos `USE_PLUGIN` e `USE_DBUS` na configuração final.

3. Estudo Exploratório

Esta seção descreve o estudo conduzido nesse artigo. Primeiramente é apresentado o projeto do estudo e, posteriormente, os resultados e discussões.

3.1. Projeto do Estudo

Para avaliar as violações de padrões de uso de APIs foram considerados quatro sistemas configuráveis de código aberto implementados em linguagem C com diretivas de pré-processamento. Os sistemas, disponíveis em repositórios Git, são de diferentes domínios, tamanho e idade. A seleção desse conjunto baseou-se em trabalhos anteriores que exploram sistemas configuráveis [Medeiros et al. 2017]. A Tabela 1 apresenta os sistemas configuráveis analisados, bem como o número de commits, a idade em anos, o número médio de linhas de código, o número médio de funções, o número médio de chamadas de funções ao longo dos anos e o número médio de variabilidades.

Para realizar o estudo, foram executados três principais atividades: (i) mineração e extração de dados, (ii) identificação de padrões de uso de APIs e (iii) análise de possíveis violações de padrões de uso. Cada uma das atividades é detalhada a seguir.

Mineração e extração de dados. Inicialmente analisou-se o repositório dos projetos selecionados com o propósito de extrair chamadas de funções de API (internas e externas). Para isso, implementou-se uma ferramenta que recupera cada *commit* armazenado no repositório. Posteriormente a ferramenta cria uma Árvore Sintática Abstrata (AST) de

Tabela 1. Conjunto de sistemas analisados

Sistema	# Commits	Idade	LOC	Funções	Chamadas de Funções	Variabilidades
Hexchat	3380	8	69988	1672,43	6101,33	17
Lighttpd	2568	13	53284	726,35	3568,23	31
Mpsolve	1674	5	46967	574,52	4092,55	10
OpenVPN	2226	11	83350	500,51	1240,03	17

cada arquivo em cada *commit* e preserva todas as informações de variabilidade, tendo em cada nó da AST uma macro associada à diretiva de pré-processamento. Dessa forma, identificaram-se as chamadas de funções e as variabilidades em que ocorreram tais chamadas.

Identificação de padrões de uso de APIs. Para identificar os pares de funções com a mesma frequência de chamadas, ou seja, padrões de uso não-ordenado de funções do tipo *push-pop*, aplicou-se o algoritmo APRIORI. O algoritmo prediz regras de associação entre itens genéricos baseado na ocorrência dos itens em uma base de ocorrências. Utilizou-se como base de ocorrência o histórico de chamadas das funções para cada commit analisado. Considerou-se como padrões relevantes aqueles com alto nível de confidência e baixo suporte, conforme apresentado na Figura 2.

Análise de possíveis violações de padrões de uso. Com os padrões de uso mais relevantes, verificou-se os locais das chamadas das funções dos padrões identificados. É importante analisar o local das chamadas, pois caso as funções de um padrão sejam chamadas em variabilidades diferentes, há a possibilidade de se gerar configurações do sistema em que o padrão seja violado. Dessa forma, para cada par, foram armazenadas as variabilidades em que ocorreram as chamadas. Caso ao menos uma variabilidade fosse diferente entre o par, considerou-se como uma possível violação do padrão de uso. Além disso, uma análise manual foi feita nas violações detectadas para evitar possíveis ocasionalidades de padrões e de violações.

3.2. Resultados

A análise de possíveis violações de padrões de uso resultou na descoberta de 105 possíveis cenários de violações nos sistemas configuráveis. A Tabela 2 apresenta para cada sistema configurável, o número de padrões onde pelo menos uma de suas funções ocorre no

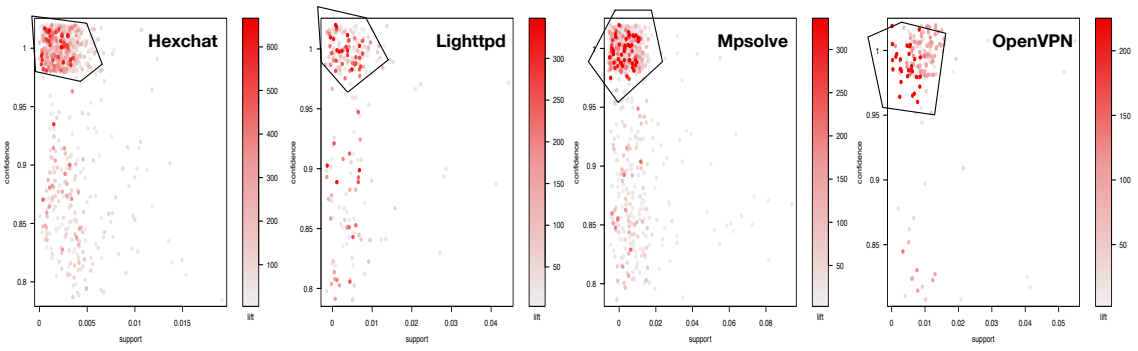

Figura 2. Valores de suporte e confianças aplicados para balancear a relevância dos padrões em cada sistema configurável.

contexto de variabilidades (ou seja, chamadas em código de variabilidade) e o número total de padrões identificados entre parentesis; o número variabilidades com padrões de uso de APIs e o número entre parentesis é o número de variabilidades de cada sistema; a porcentagem de padrões em variabilidades; e a porcentagem de variabilidades que possuem funções que fazem parte de padrões de uso.

Pelos dados apresentados, percebe-se que apenas um pequeno número de padrões de uso de API ocorrem no contexto de variabilidades (aprox. 7,3%). Ou seja, a maioria dos padrões de uso de APIs acontecem em código que não possui diretivas de pré-processamento associadas. Esse é uma observação interessante, pois a maioria dos padrões de uso não estão relacionados com variabilidades. Um contraponto é o sistema *OpenVPN* que possui 26,2% dos padrões de uso identificados ocorrendo no contexto de variabilidades. É interessante observar que o *OpenVPN* é um dos sistemas com o menor número variabilidades (Tabela 1). Dessa forma, acredita-se que existirá uma grande variação no número de padrões relacionados à variabilidades dependendo de diversos fatores do sistema e, por consequência, uma variação no possível número de violação de padrões devido ao uso de diretivas de pré-processamento. Uma análise mais extensa com vários sistemas de diferentes domínios pode ajudar a entender a relação entre número de variabilidades, trechos de código com compilação condicional e padrões de uso de APIs.

Um outro ponto a ser destacado é o alto número de variabilidades com chamadas à funções que fazem parte de algum padrão de uso de API com relação ao número total de variabilidades. No geral, padrões de uso ocorrem em 58,3% das variabilidades. Quanto maior o número de variabilidades com chamadas à funções participantes de padrões de API, maior a chance de violação de padrões. Por meio de uma análise manual e mais detalhada por alguns padrões, acredita-se que este resultado é correlacionado ao grau de espalhamento da implementação das variabilidades. Ou seja, quanto maior o espalhamento na implementação das variabilidades, maior a chance de envolver a chamada de alguma função participante de um padrão. Como consequência, acredita-se que haja uma maior chance de violação de padrões de API devido ao aumento no número de possíveis produtos gerados pelo sistema configurável e pelo maior ofuscamento de código devido ao espalhamento.

Tabela 2. Dados extraídos dos sistemas analisados.

Sistema	# Padrões em variabilidades	# Variabilidades com padrões	% de padrões em variabilidades	% de variabilidades com padrões
Hexchat	18 (509)	3 (17)	3,5	17,6
Lighttpd	41 (253)	24 (31)	16,2	77,4
Mpsolve	13 (551)	5 (10)	2,3	50,0
OpenVPN	33 (126)	15 (17)	26,2	88,2
Média	26,2 (331,2)	11,7 (18,7)	12,0	58,3

Complementarmente, em análise manual realizada sob cada possível cenário de violação identificado, observou-se que algumas violações se confirmam e permanecem no código por um longo número de commits. Por exemplo, observou-se uma possível violação no padrão de uso (*lua_islightuserdata* $\Rightarrow$ *lua_touserdata*) no commit `fe6b720`[1]. Esta mesma violação permaneceu no código até o commit `4184c38`[2], o que corresponde a um espaço de tempo de 120 meses. No geral, em média, as possíveis violações de

[1] github.com/lighttpd/lighttpd1.4/commit/fe6b720
[2] github.com/lighttpd/lighttpd1.4/commit/4184c38

padrões que ocorrem no contexto de variabilidades permaneceram no código por 454 commits.

O estudo revela um relevante problema no desenvolvimento de sistemas configuráveis ainda não explorado. A partir desse problema podemos ressaltar as seguintes implicações imediatas que podem ser alvo de estudos futuros:

- **Identificação de padrões de uso de APIs em sistemas configuráveis:** Vários trabalhos abordam a identificação de padrões de uso de APIs. No entanto, não existem trabalhos que explorem essa identificação em sistemas configuráveis implementados com diretivas de pré-processamento. Apesar de uma leve tendência nos dados encontrados, notam-se variações em alguns sistemas da tendência comum. Acredita-se que a diferença na codificação de sistemas configuráveis implique em diferenças nos padrões de APIs identificados. Ademais, uma identificação de padrões de APIs internas se faz necessário devido ao impacto de uma violação de um padrão em diversas configurações.
- **Detecção de violação de padrões de uso de APIs:** Inúmeras configurações de um sistema podem conter violações no uso de APIs. Com isso, surge a necessidade pelo desenvolvimento de novas técnicas para evitar violações no uso de APIs. No contexto de sistemas configuráveis, isso se torna um desafio haja vista a explosão no número de possíveis configurações a medida que o número de variabilidades aumenta com a evolução natural do sistema. É necessário explorar técnicas de análises estática e dinâmica para auxiliar na detecção de violação de padrões de uso de APIs já conhecidos de maneira escalável.
- **Análise das razões da violação e de eventuais correções de violações:** Pelos dados extraídos no estudo, notou-se uma grande quantidade de commits entre a violação de alguns padrões e sua eventual correção. Foge do escopo deste trabalho, mas é percebido que algumas das violações parecem ocorrer devido à existência das diretivas de pré-processamento. O desenvolvedor não consegue abstrair no momento da codificação quais as possíveis combinações de configurações e quais as possíveis violações que ocorreriam em cada combinação. Um outro ponto interessante para ser explorado é a análise das correções realizadas para evitar as violações. Tanto a análise da causa quanto a correção realizada são de suma importância para auxiliar o desenvolvedor na codificação e para a criação de ferramentas de apoio.

4. Ameaças à validade

Existem pelo menos três ameaças à validade dos nossos resultados. Primeiro, não pode-se afirmar que nossas conclusões são mantidas quando se consideram outras APIs externas, outros domínios de aplicações (por exemplo, sistemas que não são configuráveis) ou outras linguagens de programação. Para minimizar tais riscos argumenta-se que os sistemas utilizam diversas APIs externas, além de um número considerável de APIs internas. Os sistemas configuráveis são de diferentes domínios e de diferentes tamanhos. Por fim, utiliza-se a linguagem C com pré-processadores que é considerada a mais empregada para a implementação de variabilidades em sistemas industriais [Apel and Kästner 2009]. Segundo, a escolha dos valores de suporte e confiança foram subjetivas, já que comumente não existem valores recomendados para o contexto do estudo abordado neste trabalho. Para controlar essa ameaça foram testados vários valores de suporte e confiança para balancear cobertura e representatividade dos padrões encontrados. Terceiro, a extração de padrões de uso de APIs e a análise de

violações dos padrões foram feitas nos mesmos sistemas. Dessa forma, acredita-se que algumas violações de padrões podem ter sido removidas ao longo do desenvolvimento. Ou seja, o número de violações de padrões pode ser maior que o detectado nesse estudo. Para minimizar essa ameaça, foram analisados todos os commits ao longo da evolução e identificadas possíveis violações corrigidas ao longo da evolução dos sistemas.

5. Trabalhos relacionados

APIs e padrões de uso. Recentemente diferentes aspectos sobre uso de APIs ganharam atenção. Alguns estudos exploram os obstáculos encontrados no uso de APIs [Wang and Godfrey 2013]. Para isso a maioria dos trabalhos utiliza sites de Q&A (ex., Stack Overflow) como base para o entendimento de problemas encontrados com o uso de APIs. A maioria dos trabalhos, que está mais relacionada com este trabalho, foca na mineração de padrões de uso de APIs [Li and Zhou 2005, Borges and Valente 2015]. Tais trabalhos adotam diferentes categorias de padrões de uso, diferentes técnicas para inferência de padrões e diferentes formas de avaliar os padrões. As categorias que mais se destacam são: temporais, não-ordenadas e ordenadas. Diferentemente destes trabalhos, não focamos na melhoria do uso de APIs e nem na proposta de um nova técnica para detecção de padrões de uso de APIs. Neste trabalhos utilizamos uma variação do padrão não-ordenado *push-pop* para a detecção de padrões de uso. Além disso, nenhum dos trabalhos anteriores utiliza sistema configuráveis como foco principal do trabalho.

Sistemas configuráveis. Desenvolvedores devem considerar múltiplas configurações quando executam testes ou realizam análise estática em sistemas configuráveis. Como o espaço de configuração geralmente cresce exponencialmente com o número de variabilidades, várias abordagens são propostas para analisar problemas específicos em sistemas configuráveis [Tartler et al. 2012b, Thüm et al. 2014]. Além disso, diferentes técnicas são utilizadas para realizar a análise considerando variabilidades (do inglês, *variability-aware analysis*) como, por exemplo, *t-wise* [Perrouin et al. 2010], *statement-coverage* [Tartler et al. 2012a] e *one-disabled* [Abal et al. 2014]. Neste trabalho, não focamos na proposta de técnicas ou melhorias de abordagens que tratam a explosão combinatorial de configurações em sistemas configuráveis. Além disso, nenhum desses trabalhos abordam problemas nos quais variabilidades afetam os padrões de uso de APIs. Neste trabalho abordamos violações de padrões de uso nas inúmeras configurações possíveis de um sistema configurável.

6. Conclusões

Este artigo apresentou um estudo sobre violações de padrões de uso de APIs em sistemas configuráveis implementados com diretivas de pré-processamento. O estudo considerou a análise de quatros sistemas configuráveis de código aberto: totalizando 253.589 linhas de código analisadas e mais de 9.848 commits. Os resultados do estudo indicam que a suspeita inicial se confirma: a complexidade inerente de códigos com diretivas de pré-processamento pode levar os desenvolvedores a violar os padrões de uso de APIs, muito provavelmente de maneira inconsciente. Neste contexto, entende-se que é necessária uma identificação de padrões de uso de APIs no contexto de sistema, pois ferramentas de auxílio já existentes não estão sendo utilizadas ou não são suficientes para apoiar desenvolvedores de sistemas configuráveis. Um outro ponto importante é o desenvolvimento de abordagens que possibilitem a detecção de possíveis violações em sistemas configuráveis de maneira escalável. Por fim, é necessário um maior entendimento sobre as causas das eventuais inserções de violações (relacionadas ou não a

diretivas de pré-processamento), bem como as soluções dadas pelos desenvolvedores para corrigir eventuais violações.

Agradecimentos

O autor Bruno Mecca agradece o apoio do PIBIC/UFMS para a execução deste trabalho.

Referências

Abal, I., Brabrand, C., and Wasowski, A. (2014). 42 variability bugs in the linux kernel: A qualitative analysis. In *29th ACM/IEEE International Conference on Automated Software Engineering (ASE)*, pages 421–432.

Apel, S., Batory, D., Kästner, C., and Saake, G. (2013). *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company.

Apel, S. and Kästner, C. (2009). Virtual separation of concerns - a second chance for preprocessors. *Journal of Object Technology*, 8(6):59–78.

Borges, H. S. and Valente, M. T. (2015). Mining usage patterns for the android api. *PeerJ Computer Science*, 1:1–12.

Ernst, M. D., Badros, G. J., and Notkin, D. (2002). An empirical analysis of c preprocessor use. *IEEE Trans. Softw. Eng.*, 28(12):1146–1170.

Li, Z. and Zhou, Y. (2005). Pr-miner: Automatically extracting implicit programming rules and detecting violations in large software code. In *13th International Symposium on Foundations of Software Engineering (FSE)*, pages 306–315.

Medeiros, F., Kästner, C., Ribeiro, M., Nadi, S., and Gheyi, R. (2015). The Love/Hate Relationship with the C Preprocessor: An Interview Study. In *29th European Conference on Object-Oriented Programming (ECOOP 2015)*, pages 495–518.

Medeiros, F., Ribeiro, M., Gheyi, R., Apel, S., Kastner, C., Ferreira, B., Carvalho, L., and Fonseca, B. (2017). Discipline matters: Refactoring of preprocessor directives in the #ifdef hell. *Transactions on Software Engineering*, 44(5):453–469.

Perrouin, G., Sen, S., Klein, J., Baudry, B., and l. Traon, Y. (2010). Automated and scalable t-wise test case generation strategies for software product lines. In *3rd International Conference on Software Testing, Verification and Validation (ICST)*, pages 459–468.

Tartler, R., Lohmann, D., Dietrich, C., Egger, C., and Sincero, J. (2012a). Configuration coverage in the analysis of large-scale system software. *SIGOPS Oper. Syst. Rev.*, 45(3):10–14.

Tartler, R., Sincero, J., Dietrich, C., Schröder-Preikschat, W., and Lohmann, D. (2012b). Revealing and repairing configuration inconsistencies in large-scale system software. *International Journal on Software Tools for Technology Transfer*, 14(5):531–551.

Thüm, T., Apel, S., Kästner, C., Schaefer, I., and Saake, G. (2014). A classification and survey of analysis strategies for software product lines. *ACM Comput. Surv.*, 47(1):1–45.

Wang, W. and Godfrey, M. W. (2013). Detecting API usage obstacles: A study of ios and Android developer questions. In *10th Working Conference on Mining Software Repositories (MSR)*, pages 61–64.

Zhong, H., Xie, T., Zhang, L., Pei, J., and Mei, H. (2009). Mapo: Mining and recommending API usage patterns. In *23rd European Conference on Object-Oriented Programming (ECOOP)*, pages 318–343.