Avaliação da Frequência de Mudanças em Dependências entre Variabilidades em Sistemas Configuráveis

Raiza A. Oliveira[1], **Bruno Mecca**[1], **Bruno B. P. Cafeo**[1], **André Hora**[1]

[1]Faculdade de Computação
Universidade Federal de Mato Grosso do Sul (UFMS)
Caixa Postal 43017-6221 – Campo Grande – MS – Brasil

{raiza.a.oliveira,bruno.mecca}@aluno.ufms.br,{cafeo,hora}@facom.ufms.br

Abstract. *Configurable systems enable mass customization by exploiting similarities between members of the program family. Approaches and tools to support source code maintenance in the presence of features dependencies are often proposed. However, few studies deal with the evolution of dependencies. This way, this work analyzes how often dependencies changes in terms of release and commit and what these changes are. The results show a considerable frequency of changes to each commit and many changes to the code that implements the dependencies.*

Resumo. *Sistemas configuráveis possibilitam a customização em massa, explorando semelhanças entre os membros da família de software. Abordagens e ferramentas para apoiar a manutenção de código na presença de dependências entre variabilidades são frequentemente propostas. No entanto, poucos trabalhos abordam sobre a evolução das dependências. Dessa forma, esse trabalho analisa a frequência de mudanças das dependências em termos de release e commit e quais são essas mudanças. Os resultados mostram uma frequência considerável de mudanças a cada commit e muitas mudanças no código que implementa as dependências.*

1. Introdução

Sistemas de software vêm se tornando configuráveis para atender demandas constantes em segmentos de mercado distintos [Clements and Northrop 2002]. Opções de configurações, também conhecidas como variabilidades [Apel et al. 2013], envolvem desde pequenos trechos de código até módulos completos. Sistemas que se utilizam de variabilidades para implementar tais funcionalidades são chamados de sistemas configuráveis [Apel et al. 2013].

Em nível de código-fonte, as variabilidades podem se relacionar por meio do compartilhamento de elementos de programa, como funções e variáveis. Por exemplo, uma variável definida em uma variabilidade, e usada em outra. Essas relações são chamadas de dependências entre variabilidades [Cafeo et al. 2016a]. Pode-se dizer que uma dependência entre variabilidades ocorre sempre que um ou mais elementos de programa dentro dos limites de uma variabilidade dependem de elementos externos a essa variabilidade.

Em um cenário de evolução de software, mudanças ocorrem para acomodar novas funcionalidades, corrigir erros, entre outros. Em sistemas configuráveis, as dependências dificultam a manutenção, pois o desenvolvedor precisa garantir que uma mudança não afetará a consistência das variabilidades já existentes. Para isso,

baseia-se nas dependências. O problema é que, no código-fonte, variabilidades geralmente não estão contidas em um único módulo, e sim espalhadas em fragmentos de código [Cafeo et al. 2016a]. Dessa forma, desenvolvedores devem analisar minuciosamente o código-fonte para realizar mudanças.

Diversos trabalhos discutem soluções de apoio à manutenção em sistemas configuráveis na presença de dependências entre variabilidades [Cafeo et al. 2016b, Ribeiro et al. 2014, Schröter et al. 2014]. No entanto, ainda faltam estudos que analisem a frequência com que mudanças em dependências ocorrem ao longo da evolução. Mais especificamente, não se sabe a frequência com que tais mudanças ocorrem. É importante saber essa frequência, para avaliar a quantidade de informação perdida ao se analisar *releases* ao invés de *commits*. Para tal, este trabalho busca responder as seguintes questões de pesquisa:

QP1: Mudanças em dependências devem ser analisadas em termos de *releases* ou *commits*?

QP2: Qual a categoria (incluída, excluída ou preservada) de mudanças na implementação de dependências entre variabilidades ocorre com mais frequência?

Para responder tais questões, realiza-se uma análise de todo o histórico de evolução de 6 sistemas configuráveis hospedados no GitHub[1], em termos de *releases* e *commits*. Dessa forma, as principais contribuições deste trabalho são prover: *(i)* dados sobre dependências entre variabilidades que revelam frequência e intensidade de mudanças na evolução de sistemas configuráveis para guiar trabalhos futuros, e *(ii)* bases para o desenvolvimento de ferramentas que apoiem a manutenção de sistemas configuráveis na presença de dependências entre variabilidades, e *(iii)* uma estratégia de identificação de dependências entre variabilidades e mudanças de granularidade fina em tais dependências baseada na ferramenta TypeChef[2].

O restante deste artigo está organizado conforme descrito a seguir. Na Seção 2 são apresentados os conceitos básicos abordados nesse artigo. Na Seção 3 é descrita a metodologia proposta. Na Seção 4 são apresentados os resultados obtidos. As implicações dos resultados são apresentadas na Seção 5. Na Seção 6 são apresentados os riscos à validade deste trabalho. Os trabalhos relacionados são apresentados na Seção 7. Por fim, na Seção 8 apresenta-se as conclusões deste trabalho e os possíveis desdobramentos dos resultados em outros estudos.

2. Fundamentação Teórica

Sistemas configuráveis são usados em diferentes domínios de aplicação, por proverem flexibilidade ao software. Por exemplo, um sistema operacional para dispositivo móvel que compartilha um conjunto comum de funcionalidades (ex. calendário), mas varia em outras (ex. resolução de tela) dependendo de vários fatores, tais como hardware utilizado ou preferências de usuário. As configurações do sistema são formadas por meio da combinação de um núcleo comum com funcionalidades específicas, chamadas de variabilidades [Apel et al. 2013].

Segundo Apel et al., sistemas configuráveis são comumente implementados por meio de compilação condicional [Apel et al. 2013][3]. Nessa abordagem o pré-processador

[1]https://github.com/

[2]https://github.com/ckaestne/TypeChef

[3]Está fora do escopo deste trabalho avaliar outras abordagens de implementação como, por exemplo,

identifica o código que deve ser compilado ou não com base em diretivas de pré-processamento (ex. #ifdef) que envolvem estruturas de código associando-as à uma variabilidade. Portanto, no nível de código-fonte, uma variabilidade é um conjunto de elementos de programa (i.e., variáveis ou funções) cercados por diretivas de pré-processamento [Cafeo et al. 2016a, Kästner et al. 2008].

Observa-se na Figura 1 um fragmento de código extraído do Hexchat[4], em que é possível observar exemplos de implementação de variabilidades. As instruções `#ifdef` e `#endif` definem as fronteiras de partes da implementação das variabilidades e indicam ao pré-processador o que incluir no processo de compilação. Dessa forma, as instruções entre as Linhas 2 e 15 (arquivo hexchat.c) só serão compiladas se a variabilidade `USE_LIBPROXY` estiver definida.

```
1  #ifdef !def(USE_LIBPROXY)
2  static void irc_init (session *sess){
3  #ifdef USE_PLUGIN
4      if (!arg_skip_plugins)
5        plugin_auto_load (sess);    /* autoload ~/.xchat *.so */
6  #endif
7  #ifdef USE_DBUS
8      plugin_add (sess, NULL, NULL, dbus_plugin_init, NULL, NULL,
9  FALSE);
10 #endif
11     snprintf (buf, sizeof (buf), "%s/%s", get_xdir_fs (), "startup.txt");
12     load_perform_file (sess, buf);
13 }
14 . . .
15 #endif
```

Arquivo hexchat.c

```
1  #ifdef !def(WIN32)
2  void gtkutil_file_req (const char *title, void *callback, void
3  *userdata,  char *filter, char *extensions, int flags){
4      struct file_req *freq;
5      GtkWidget *dialog;
6      GtkFileFilter *filefilter;
7      extern char *get_xdir_fs (void);
8      char *token;
9      . . .
10     freq->userdata = userdata;
11     freq->title = g_locale_from_utf8 (title, -1, 0, 0, 0);
12     . . .
13     thread_start (freq->th, win32_thread, freq);
14 . . .
15 #endif
```

Arquivo gtkutils.c

Figura 1. Fragmento de código contendo dependência entre variabilidades

Variabilidades comumente se relacionam para realizar tarefas específicas de um sistema configurável [Apel et al. 2013]. Neste trabalho, seguindo a definição de trabalhos da área [Cafeo et al. 2016a, Ribeiro et al. 2011, Rodrigues et al. 2016], considera-se que sempre que um ou mais elementos de programa definidos em uma variabilidade se relacionem com elementos de programa de outra variabilidade configura-se uma dependência entre variabilidades. Ou seja, uma dependência entre duas variabilidades pode ser composta por diversos relacionamentos entre elementos de programas de ambas as variabilidades.

Na Figura 1, pode-se observar uma dependência entre as variabilidades `USE_LIBPROXY` (arquivo hexchat.c) e `WIN32` (arquivo gtktils.c). A dependência é configurada por meio de apenas um par de elementos de programa (cada um pertencente a uma variabilidade). Mais especificamente, a função `get_xdir_fs` definida na variabilidade `WIN32` (Linha 7) é utilizada na variabilidade `USE_LIBPROXY` (Linha 11) configurando uma dependência.

3. Metodologia

Neste trabalho são analisados 6 sistemas configuráveis de código aberto implementados em linguagem C, abrangendo diferentes domínios de aplicação. Tal seleção, baseou-se em trabalhos anteriores [Cafeo et al. 2016b, Rodrigues et al. 2016]. Observa-se na Tabela 1 a lista dos sistemas analisados, bem como o número de *releases* e *commits*, idade do sistema, número total de linhas de código considerando todas as *releases*, a média de *commits* entre cada *release*, a média de dias entre cada *commit* e cada *release*.

abordagens composicionais.

[4]https://github.com/hexchat/hexchat

Tabela 1. Conjunto de sistemas analisados

Sistema	# *Releases*	# *Commits*	Idade (anos)	LOC (Total)	Média de *Commits* por *Release*	Intervalo entre *Commits* (dias)	Intervalo entre *Releases* (dias)
Curl	178	23202	18	13457614	130,3	0,29	38
GZip	8	439	25	98362	54,9	18,65	1023
Hexchat	17	3380	8	1329498	54,9	0,87	172
Lighttpd	56	2568	13	4704730	198,8	1,86	85
Libssh	33	3222	10	1225425	97,6	1,13	111
OpenVPN	76	2226	11	9904685	29,3	1,88	55

Para cada sistema, analisou-se todo o histórico disponível no repositório[5], totalizando mais de 30 milhões de linhas de código distribuídos em 368 *releases* e mais de 35 mil *commits*. O procedimento de coleta de dados foi dividido em duas fases. Primeiramente foram coletados os dados referentes às *releases* dos sistemas e, em seguida, coletou-se os dados referentes aos *commits*. Cada uma dessas fases possuem 4 atividades, sendo:

Minerar repositórios de software. A primeira atividade foi recuperar o repositório dos projetos selecionados para analisar as mudanças históricas nas dependências entre variabilidades. Para isso, implementou-se uma ferramenta que recupera cada *release* (ou *commit*) do projeto por meio da identificação de *tags* e *logs*.

Identificar dependências entre variabilidades. Após recuperar o código-fonte de cada versão do projeto, identificou-se as dependências entre variabilidades. Para tal, implementou-se uma ferramenta baseada no TypeChef [Kästner et al. 2011], que identifica dependências entre variabilidades a partir da Árvore Sintática Abstrata (AST) gerada pelo TypeChef, que contém a declaração e definição dos elementos de programa.

Detectar mudanças. Para detectar as mudanças nas dependências entre variabilidades, armazenou-se dados de duas *releases* (ou *commits*) subsequentes para compará-las. Tal comparação possibilitou identificar as dependências entre variabilidades incluídas, excluídas e preservadas. Dentre as dependências preservadas, buscou-se identificar quais sofreram algum tipo de alteração estrutural.

Comparar *releases* e *commits*. Após a extração de dados referentes aos *commits* e *releases* dos sistemas, realizou-se uma comparação para identificar em qual caso as alterações na implementação das dependências são mais frequentes. Na Figura 2 apresenta-se uma ilustração do método utilizado.

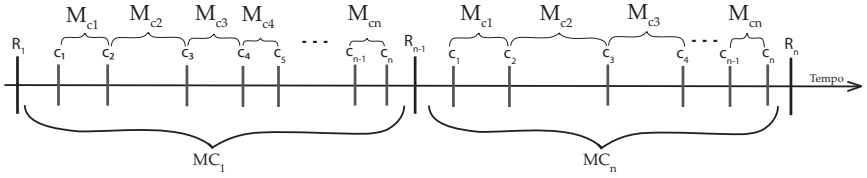

Figura 2. Em que R_i indica a *release*, c_i indica o *commit*, M_{ci} indica a quantidade de mudanças entre dois *commits* subsequentes, e MC_i indica $\sum_{i=1}^{n} M_{ci}$

Inicialmente contabilizou-se o total de mudanças em *commits* subsequentes (M_{ci}), em seguida calculou-se a quantidade de mudanças acumuladas em *commits* subsequentes (MC_i) de acordo com a fórmula: $\sum_{i=1}^{n} M_{ci}$, em que n é a quantidade de

[5]Considerou-se o repositório hospedado no GitHub até o dia 15 de junho de 2018.

commits entre duas *releases* subsequentes. Por fim, fez-se a comparação entre o resultado obtido (MC_i) e a quantidade de mudanças em *releases* (R_i) no mesmo período de tempo.

4. Resultados

Nessa seção são apresentados os resultados do estudo. Na Seção 4.1 são apresentadas as mudanças em dependências em termos de *commit* e em termos de *release*. Na Seção 4.2 são apresentados dados descritivos sobre as mudanças que ocorrem nas dependências.

4.1. Commit ou release?

Na Figura 3 pode-se observar a distribuição da quantidade de mudanças ao longo da evolução dos sistemas. Observa-se que a quantidade de mudanças em *commits* tende a aumentar mais rapidamente ao longo da vida do sistema do que em *releases*.

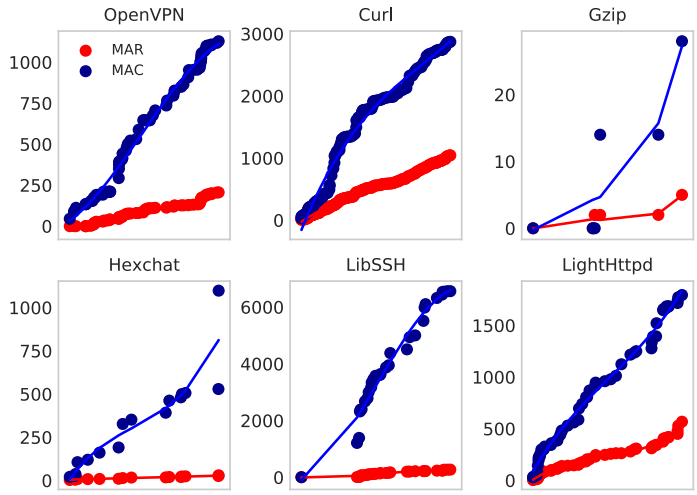

Figura 3. Mudanças acumuladas em *releases* (MAR) e em *commits* (MAC)

Os dados apresentados na Tabela 2 contemplam mudanças de uma forma geral, ou seja, estão contabilizadas inclusões de novas dependências, exclusões e dependências preservadas durante uma evolução, que sofreram algum tipo de alteração estrutural. Observa-se que as mudanças em *commits* são mais comuns que em *releases*, chegando a ser até 1604% maior no mesmo intervalo de tempo.

Tabela 2. Quantidade média de mudanças ao longo da evolução dos sistemas

	Curl	Gzip	Hexchat	LighHttpd	LibSSH	OpenVPN
Média de mudanças em commits	13,40	6,50	108,65	27,30	220,82	12,17
Média de mudanças em releases	10,32	5,17	6,38	8,76	14,07	5,25

Ao se considerar somente *releases* pode-se perder algumas mudanças. Por exemplo, ao comparar as *releases* 1.4.11 e 1.4.12 do sistema LigthHttpd[6] foram contabilizadas 11 mudanças em dependências. Entre essas *releases* há um total de 69 *commits*, que acumulam um total de 27 mudanças em dependências. Isso ocorre porquê uma mesma dependência pode sofrer diversas alterações em seus elementos, e quando considera-se apenas *releases* tais alterações podem ser ignoradas.

[6]https://github.com/lighttpd/lighttpd1.4

4.2. Tipos de mudança

Embora mudanças em dependências entre variabilidades ocorram em maior número em termos de *commits*, a distribuição dos tipos dessas mudanças ocorre de forma diferente em *releases* e *commits*. Na Figura 4 observa-se a taxa de cada tipo de mudança em termos de *commits* e *releases* ao longo da evolução dos sistemas. As exclusões e inclusões de novas dependências correspondem a maior parte das mudanças em termos de *releases*. No entanto, as mudanças em dependências preservadas, ou seja, que não são novas e nem excluídas durante uma evolução, mas que sofrem algum tipo de alteração estrutural é maior em termos de *commits*.

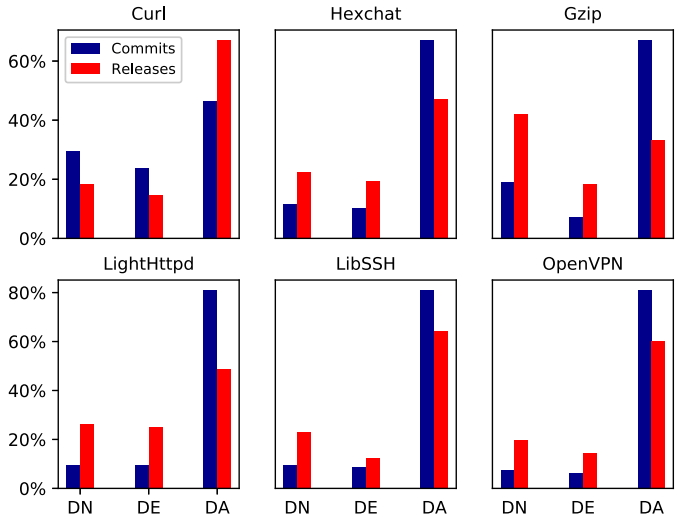

Figura 4. Taxa de dependências novas (DN), dependências excluídas (DE) e dependências preservadas alteradas (DA) nos sistemas analisados

Com exceção do sistema Curl[7], nos demais sistemas a taxa de dependências preservadas e alteradas é maior em *commits* do que em *releases*. Em média, cerca de 73% das mudanças em *commits* pertencem a essa classe, contra cerca de 55% em *releases*. A média de dependências novas e excluídas ao longo da evolução dos sistemas mostra-se menor em *commits*. Cerca de 15% das mudanças em *commits* são referentes a inclusão de novas dependências, contra aproximadamente 26% em *releases*. As mudanças que se referem a exclusão de dependências em *commits* é cerca de 11%, em *releases* essa quantidade é de aproximadamente 18% em média.

5. Implicações

A partir dos resultados obtidos, identificou-se que mudanças nas dependências são mais frequentes quando analisa-se *commits*. Ressalta-se que mudanças de granularidade grossa (inclusão e exclusão de dependências) correspondem a maior parte das mudanças nas dependências em termos de *releases*. Em termos de *commits*, as mudanças de granularidade fina correspondem a maior parte das mudanças. Esse resultado indica que analisar *commits* mostra-se uma melhor opção quando se busca identificar mudanças de granularidade fina.

[7]https://curl.haxx.se/

Dependências entre variabilidades tendem a ser preservadas ao longo da evolução dos sistemas. Dependências com mudanças constantes tendem a impactar mais o código, e portanto merecem mais atenção nas manutenções. Dessa forma, nota-se a necessidade de ferramentas para: *(i)* análise de impacto no código; *(ii)* predição de erros; e *(iii)* análise de degradação arquitetural que pode indicar se as alterações estão violando a arquitetura planejada, ou analisar como uma eventual degradação evoluiu.

6. Ameaças à Validade

Validade Interna. Neste trabalho foram exploradas as alterações nos elementos de programa (funções e variáveis) por meio de análise estática do histórico de evolução do código. Não foram consideradas outros elementos de programa que poderiam participar da implementação de dependências, bem como técnicas de análise dinâmica, casos de teste ou técnicas mais custosas, como fluxo de dados.

Validade Externa. Os sistemas configuráveis podem não ser representativos, tal ameaça foi reduzida avaliando sistemas configuráveis de diferentes domínios de aplicação, amplamente utilizados e avaliados em pesquisas anteriores [Medeiros et al. 2016, Rodrigues et al. 2016].

Validade de Construção. Neste estudo analisou-se sistemas configuráveis escritos em C que utilizam compilação condicional para implementar variabilidades. Tal mecanismo pode aumentar o número de dependências quando comparado a outras técnicas de implementação. Argumenta-se que esse é um dos mecanismos mais utilizados para implementar sistemas configuráveis [Ribeiro et al. 2014].

Validade de Conclusão. Os sistemas escolhidos vieram de diferentes domínios de aplicação. Existe o risco de que a variação devido a diferenças individuais seja maior do que a decorrente do tratamento. Tal variação ajuda a promover a validade externa do estudo, melhorando a capacidade de generalizar os resultados do experimento.

7. Trabalhos Relacionados

Estudos empíricos. Diversos trabalhos tentam compreender o fenômeno relacionado à dependências entre variabilidades em sistemas configuráveis por meio de estudos empíricos [Cafeo et al. 2016a, Ribeiro et al. 2011, Rodrigues et al. 2016]. Esses trabalhos analisam: a quantidade de dependências; métricas que caracterizam as dependências; e possíveis impactos negativos que tais dependências podem causar em atributos de manutenção. Apesar desses trabalhos confirmarem a importância das dependências na manutenção de sistemas configuráveis, nenhum deles exploram a frequência, ou se tais dependências são alteradas.

Apoio à manutenção. Nos últimos anos surgiram trabalhos propondo abordagens e ferramentas para o apoio à manutenção de código-fonte na presença de dependências entre variabilidades [Cafeo et al. 2016b, Schröter et al. 2014]. Tais trabalhos analisam como a implementação de dependências pode afetar a manutenção e propõem formas de apoiá-la. Tais trabalhos não focam especificamente em evolução. Este estudo busca oferecer informações à respeito da frequência com que dependências sofrem alterações ao longo da evolução do sistema em termos de *releases* e *commits*.

8. Conclusões

A análise realizada neste trabalho caracterizou a frequência com que dependências entre variabilidades sofrem alterações ao longo da evolução de sistemas configuráveis,

abordando somente *releases* e *commits*. Tal caracterização se deu por meio da análise histórica de 6 sistemas implementados em C com diretivas de pré-processamento. Verificou-se que dependências são mais propensas a mudanças em *commits* do que em *releases*. Em trabalhos futuros essa análise poderá considerar janelas de tempo independentes de *releases* e/ou *commits*.

Mudanças em dependências preservadas são predominantes em termos de *commits*. Em termos de *releases*, a maior parte das mudanças correspondem a inclusão e exclusão de dependências. Tais resultados apontam que estudos que envolvem a investigação de mudanças de granularidade fina em dependências entre variabilidades devem considerar a análise em termos de *commits*.

Agradecimentos: Este trabalho é parcialmente financiado pelo PIBIC/UFMS edital nº 61 e pelo CNPq (processo 133822/2017-6).

Referências

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

Cafeo, B. B. P., Cirilo, E., Garcia, A., Dantas, F., and Lee, J. (2016a). Feature dependencies as change propagators. *Information Software Technology*, 69:37–49.

Cafeo, B. B. P., Hunsen, C., Garcia, A., Apel, S., and Lee, J. (2016b). Segregating feature interfaces to support software product line maintenance. In *15th International Conference on Modularity (ICM)*, pages 1–12.

Clements, P. and Northrop, L. (2002). *Software product lines*. Addison-Wesley.

Kästner, C., Apel, S., and Kuhlemann, M. (2008). Granularity in software product lines. In *30th International Conference on Software Engineering (ICSE)*, pages 311–320.

Kästner, C., Giarrusso, P. G., Rendel, T., Erdweg, S., Ostermann, K., and Berger, T. (2011). Variability-aware parsing in the presence of lexical macros and conditional compilation. In *International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, pages 805–824.

Medeiros, F., Kästner, C., Ribeiro, M., Gheyi, R., and Apel, S. (2016). A comparison of 10 sampling algorithms for configurable systems. In *38th International Conference on Software Engineering (ICSE)*, pages 643–654.

Ribeiro, M., Borba, P., and Kästner, C. (2014). Feature maintenance with emergent interfaces. In *36th International Conference on Software Engineering (ICSE)*, pages 989–1000.

Ribeiro, M., Queiroz, F., Borba, P., Tolêdo, T., Brabrand, C., and Soares, S. (2011). On the impact of feature dependencies when maintaining preprocessor-based software product lines. In *10th International Conference on Generative Programming and Component Engineering (GPCE)*, pages 23–32.

Rodrigues, I., Ribeiro, M., Medeiros, F., Borba, P., Fonseca, B., and Gheyi, R. (2016). Assessing fine-grained feature dependencies. *Information Software Technology*, 78:27–52.

Schröter, R., Siegmund, N., Thüm, T., and Saake, G. (2014). Feature-context interfaces: Tailored programming interfaces for software product lines. In *18th International Software Product Line Conference (SPLC)*, pages 102–111.