Resolução de Dependências Circulares em Inclusão de Código em Análise Estática de Código

Miguel Falé, Ibéria Medeiros, Nuno Neves

LaSIGE, Faculdade de Ciências da Universidade de Lisboa, Portugal
miguelfale94@gmail.com, imedeiros@di.fc.ul.pt, nuno@di.fc.ul.pt

Abstract. Construir aplicações web seguras tornou-se um fator crucial hoje em dia, contudo, a sua obtenção depende tanto dos conhecimentos de quem as elabora, bem como da correta utilização das linguagens de programação. O uso incorreto de funcionalidades das linguagens origina inconsistências na execução do código, tais como *dependências circulares* criadas pela inclusão de código de um ficheiro noutro recursivamente. As ferramentas de análise estática de código são sensíveis às dependências circulares, provocando deteções inválidas e quebra de execução. Este artigo apresenta o problema das dependências circulares e uma solução para a sua resolução através de análise estática de código. A solução foi implementada na ferramenta WAP, que identifica e resolve dependências circulares na análise de aplicações web e sem quebrar a sua execução.

Keywords: dependências circulares, análise estática de código, vulnerabilidades, aplicações web, segurança de software

1 Introdução

Hoje em dia as aplicações web desempenham um papel fundamental no acesso a uma miríade de serviços. Construir aplicações seguras tornou-se assim crucial, contudo, a sua obtenção depende tanto dos conhecimentos de quem as cria sobre código seguro, bem como da correta utilização das linguagens de programação.

A linguagem de programação PHP é a mais utilizada no desenvolvimento de aplicações e componentes web, tais como plugins de sistemas de gestão de conteúdos (CMS) [3]. A reutilização de código através da inclusão de um ficheiro noutro (ex., código de funções que desempenham tarefas comuns) é prática usual na criação de aplicações PHP. No entanto, o uso incorreto das diretivas de inclusão de ficheiros (ex., *include* e *include_once*) pode originar *dependências circulares* criadas pela inclusão recursiva de código de um ficheiro em outro e que resultarão em inconsistências de execução do código. Na construção de aplicações web é recorrente a incorporação de componentes desenvolvidos por terceiros [2]. Contudo, caso estes contenham bugs de implementação, tais como dependências circulares, estes poderão originar comportamentos erróneos nas aplicações e até deixar estas vulneráveis.

As ferramentas de análise estática de código (AEC) são utilizadas para detetar erros nos programas, incluindo vulnerabilidades de segurança. Através da

análise de comprometimento, as ferramentas analisam o código das aplicações, rastreando as entradas dos utilizadores (*inputs*) e verificando se estas atingem funções da linguagem suscetíveis de serem exploradas por código malicioso (ex., as funções *mysql_query* e *echo* do PHP). Durante a análise, estas ferramentas têm em consideração a inclusão de ficheiros nas aplicações, portanto, elas são sensíveis às dependências circulares que podem originar deteções inválidas e comportamentos erróneos.

O artigo apresenta o problema das dependências circulares na deteção de vulnerabilidades por análise estática de código e propõe uma abordagem para resolução do mesmo através da análise estática. A abordagem proposta permite, durante a análise do código de uma aplicação, a identificação de pontos no código com dependências circulares e dissolução destes, conseguindo desta forma evitar quebrar a análise estática. Para a correta identificação de tais pontos, um estudo do comportamento de quatro ferramentas de AEC e do PHP Zend engine foi realizado usando diferentes casos de dependências circulares. A abordagem foi implementada na ferramenta WAP [7] e testada com um conjunto de aplicações web e plugins do WordPress.

As contribuições deste trabalho são: 1) a definição de um conjunto de casos de uso onde ocorre dependências circulares e um estudo comportamental de ferramentas de AEC e do PHP Zend engine nestes casos de uso; 2) um algoritmo de identificação de caminhos de inclusão de ficheiros com dependências circulares; 3) um algoritmo para resolução de dependências circulares; 4) a implementação destes algoritmos na ferramenta WAP e a avaliação destes.

2 Trabalho Relacionado

A AEC pode considerar código fonte, binário, ou intermédio, e pode ser utilizada para vários fins, como a deteção de vulnerabilidades e o estudo da complexidade do código [14][5][13][12]. Esta secção foca-se no uso da AEC para a deteção de vulnerabilidades em aplicações web desenvolvidas em PHP.

A *análise de comprometimento* (*taint analysis*) é a técnica mais implementada pelas ferramentas de AEC. O código da aplicação é representado por uma *árvore sintática abstrata* (AST), para posteriormente ser navegada na descoberta de bugs. Esta técnica sinaliza todo o *input* que chega dos *pontos de entrada* de um ambiente externo como *comprometido* (*tainted*). Os *inputs* são rastreados pelos vários possíveis caminhos onde eles ou dependências deles são usados, e é detetado se algum deles é parâmetro de funções sensíveis, ou seja, suscetíveis de serem exploradas por maliciosos. Se tal acontecer, uma vulnerabilidade foi descoberta. A análise do estado das variáveis rastreadas pode alternar de comprometido para *não comprometido*, dependendo dos locais no código por onde passam os *inputs* [7] (ex., após o processamento de uma função de sanitização).

As ferramentas Pixy [4], phpSAFE [9], RIPS [1] e WAP [7] realizam análise de comprometimento na deteção de vulnerabilidades de validação de input. Pixy e phpSAFE detetam duas classes de vulnerabilidades - *injeção de SQL* (SQLI) e *cross-site scripting* (XSS) [5][9]. RIPS e WAP, para além destas classes, lidam

com outras, como a de *inclusão de ficheiros remotos* (RFI). RIPS e Pixy não suportam análise de código objeto, ao invés da phpSAFE e WAP. WAP é a única que oferece uma metodologia modular para adicionar novas classes [8]. WAP e phpSAFE lidam com funções específicas do CMS WordPress [9][7].

3 Dependências Circulares na Análise Estática

A maioria das linguagens de programação suportam a funcionalidade de *inclusão de ficheiro*, que consiste na indicação de que o conteúdo de um dado ficheiro será usado no fluxo de execução atual. Esta funcionalidade é útil na reutilização de código em diversas partes da aplicação. Desta forma, é possível a criação de programas modulares, em que um ficheiro pode armazenar, por exemplo, funções definidas pelo utilizador para serem empregues em vários módulos da aplicação.

A linguagem PHP contem esta funcionalidade através de dois tipos de diretivas – *require/include* e *require_once/include_once*. A primeira copia o conteúdo de um ficheiro alvo para o fluxo de execução atual, no momento em que a diretiva é chamada [10]. A segunda vai guardando a indicação dos ficheiros que já foram incluídos, garantindo assim a sua inclusão uma única vez [11].

Certas aplicações contêm dependências circulares, advindas do uso destas diretivas e que usualmente assentam em dois casos: 1) um determinado ficheiro é o seu próprio antecessor e sucessor, i.e., $\boldsymbol{a}$ inclui $\boldsymbol{b}$, $\boldsymbol{b}$ inclui $\boldsymbol{a}$; 2) um ficheiro inclui um outro ficheiro que já foi previamente incluído no caminho de execução, i.e., $\boldsymbol{a}$ inclui $\boldsymbol{b}$, $\boldsymbol{b}$ inclui $\boldsymbol{c}$, e $\boldsymbol{c}$ inclui $\boldsymbol{a}$. Em ambos casos podemos apelidar o ficheiro $\boldsymbol{a}$ de *causador de ciclo*, pois é ele quem gera a dependência circular. Nas ferramentas de AEC a não contemplação de tais casos durante a análise pode ter diversas consequências, desde a geração de falsos positivos e negativos, até a inconsistências na sua execução. Analisando tais cenários, podemos identificar quatro casos-tipo relacionados com dependências circulares na AEC, baseados nas diretivas de inclusão e no conteúdo do causador do ciclo (funções ou código), e indicar uma possível forma de resolução. Para causadores de ciclo com conteúdo misto (funções e código), este é tratado pelos casos-tipo 2 e 4.

1. *Diretiva include_once/require_once e causador do ciclo contém funções.* É imprescindível obter todas as funções do ficheiro incluído sem que, para tal, se desencadeie um ciclo infinito. Este caso é resolúvel, pois as funções possuem escopo global e as suas instruções são adquiridas pelo parser da ferramenta de AEC, logo são tratadas como redundantes.
2. *Diretiva include_once/require_once e causador do ciclo contém código.* É necessário ter em conta os vários pontos de entrada de um programa PHP, os quais originarão caminhos de execução diferentes, e que a inclusão de um dado ficheiro está presente em vários caminhos. Este facto é visível nas ferramentas de AEC que conseguem analisar projetos (i.e., ficheiros contidos em subdiretorias), logo é incorreto assumir a inclusão do código de um ficheiro alvo apenas uma vez [11]. Assim sendo, a gestão de ficheiros incluídos deve ser feita relativamente a cada ponto de entrada possível no projeto.

3. *Diretiva include/require e causador do ciclo contém apenas funções.* Segundo a documentação do PHP [10], esta diretiva pode incluir um mesmo ficheiro múltiplas vezes e não é possível redefinir uma função previamente existente. No entanto, na presença de redefinição de funções, [10] não esclarece o procedimento a adotar, pois não indica se deve ser apresentado um erro, ou se declarações duplicadas devem ser simplesmente ignoradas.
4. *Diretiva include/require e causador do ciclo contém código.* Este é o caso mais ambíguo, pois [10] não especifica o comportamento de ciclo infinito. Para além do já especificado em 3., as ferramentas de AEC devem ter um mecanismo que limite o problema do ciclo infinito [6].

Tendo em conta a ambiguidade dos últimos dois casos-tipo, é importante perceber o que acontece na prática, tanto ao nível do PHP Zend engine como ao nível das ferramentas de AEC. Para tal, definimos um conjunto de casos de uso baseado nas Fig. 1 e Fig. 2 e testamos com quatro ferramentas de AEC – Pixy [4], phpSAFE [9], RIPS [1] e WAP [7] – e o PHP Zend engine (versão 5.6.12). Os casos de uso implicam a existência de um ciclo ao longo da execução de três ficheiros, partindo do ficheiro ***a***, com inclusão de código ou funções. Este formato foi escolhido com o objetivo de entender as seguintes características: i) *Resolução de ciclos com código*: entender se uma ferramenta trata, ou não, as diretivas *include/include_once* da mesma forma. ii) *Ordem das instruções*: compreender se a inclusão do ficheiro ***a***, durante a execução de ***c*** (Fig. 1), resulta, ou não, numa segunda inclusão do ficheiro ***a***. iii) *Comportamento na definição de funções*: entender se as ferramentas de AEC têm o mesmo comportamento que o PHP Zend na definição de funções.

As Tabela 1 e Tabela 2 apresentam o resultados dos testes extraídos das Fig. 1 e Fig. 2, respetivamente. Na Tabela 1 podemos observar que a ferramenta Pixy manifesta o mesmo comportamento que o PHP Zend com *include_once*, mas difere com *include*. A redução do problema passa por tratar os dois tipos de diretivas da mesma forma. As ferramentas phpSAFE e RIPS não consideram o ficheiro de entrada ***a*** na verificação de repetidos, revelando um comportamento diferente do PHP Zend. Em *include_once*, o ficheiro é visitado de novo. No *include*, RIPS trata como *include_once*, mas phpSAFE não aplica qualquer prevenção, deixando o ciclo infinito consumir recursos e provocar um *crash*. WAP apresenta um comportamento erróneo para ambos os casos. Por fim, o próprio

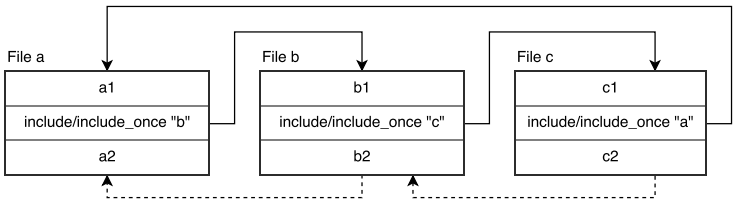

Fig. 1. Teste base com inclusão de ficheiros com código.

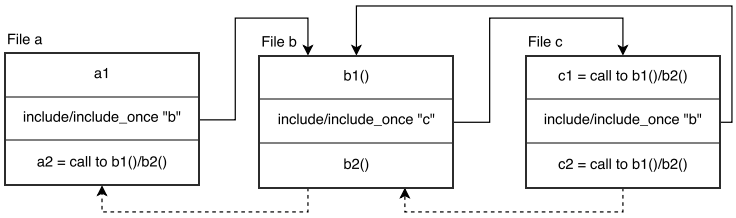

Fig. 2. Teste com funções. Ficheiro ***b*** contém a definição das funções ***b1()*** e ***b2()***.

PHP Zend não impede o ciclo infinito provocado por *include*, provocando um *crash*. A Tabela 2 mostra os mesmos tipos de problemas na execução dos testes.

4 Resolução de Ciclos em Dependências Circulares

À primeira vista, uma solução para o problema de dependências circulares consistiria em guardar informação sobre os ficheiros incluídos e impedir a repetida inclusão destes. Porém, uma aplicação pode ter vários pontos de partida, i.e., ficheiros que não são incluídos por nenhum outro, que a partir destes são originados diferentes caminhos de execução. Estes caminhos podem incluir um mesmo ficheiro, logo ao impedir a inclusão de um ficheiro repetido resulta numa análise incompleta e/ou errónea da aplicação. Por exemplo, sejam ***z*** e ***f*** ficheiros de ponto de partida e ***a*** um ficheiro a incluir usando a diretiva *include_once*. Se ***a*** é incluído uma vez no contexto do ficheiro ***z***, isso não justifica que ***a*** deva ser ignorado no contexto de ***f***. Claramente, é necessário analisar cada caminho iniciado em cada ponto de partida, e verificar em cada caminho a existência de ciclos. Denominamos de *raízes* os pontos de partida e é a partir delas que a verificação de ficheiros repetidos deve ser realizada.

	include_once	**include**
Pixy	*a1,b1,c1,a,c2,b2,a2* - A inclusão do ficheiro ***a***, durante a execução do ficheiro ***c***, é ignorada pela ferramenta	*a1,b1,c1,a,c2,b2,a2* - Igual a include_once
phpSAFE	*a1,b1,c1,a,a1,a2,c2,b2,a2* - A ferramenta resolve os alvos mas não segue a propagação de vulnerabilidades entre ficheiros - A inclusão do ficheiro ***a***, durante a execução do ficheiro ***c***, provoca uma revisita ao ficheiro ***a***, mas impede a revisita ao ficheiro ***b***	*a1,b1,c1,a, recursão infinita* - A ferramenta resolve os alvos mas não segue a propagação de vulnerabilidades entre ficheiros - A ferramenta não quebra o ciclo infinito iniciado na inclusão de ***a***, durante a execução do ficheiro ***c***
RIPS	*a1,b1,c1,a,a1,a2,c2,b2,a2* - A inclusão do ficheiro ***a***, durante a execução do ficheiro ***c***, provoca uma revisita ao ficheiro ***a***, mas impede a revisita ao ficheiro ***b***. - A ferramenta nem sempre propaga vulnerabilidades para o ficheiro alvo	*a1,b1,c1,a,a1,a2,c2,b2,a2* - Igual a include_once
WAP	*a1,b1,c1,a, análise vazia* - A análise é interrompida quando tenta revisitar o ficheiro ***a***	*a1,b1,c1,a, análise vazia* - Igual a include_once
Zend	*a1,b1,c1,a,c2,b2,a2* - A inclusão do ficheiro ***a***, durante a execução do ficheiro ***c***, é ignorada pelo interpretador.	*a1,b1,c1,a, recursão infinita* - A inclusão do ficheiro ***a***, durante a execução do ficheiro ***c***, desencadeia um ciclo infinito

Tabela 1. Comportamento das ferramentas e PHP Zend com os casos de uso extraídos da Fig. 1.

	include_once	include
Pixy	*a1,**b**, c1->b2,**b**, c2->b2, a2->b2* - A inclusão do ficheiro ***b***, durante a execução do ficheiro ***c***, é ignorada pela ferramenta	*a1,**b**, c1->b2,**b**, c2->b2, a2->b2* - Igual a include_once
phpSAFE	*a1,**b**, c1->b2,**b**, c2->b2, a2->b2* - A inclusão do ficheiro ***b***, durante a execução do ficheiro ***c***, é ignorada pela ferramenta	*a1,**b**, c1->b2,**b**, recursão infinita* - A inclusão do ficheiro ***b***, durante a execução do ficheiro ***c***, desencadeia um ciclo infinito
RIPS	*a1,**b**, c1->b2,**b**, c2->b2, a2->b2* - A inclusão do ficheiro ***b***, durante a execução do ficheiro ***c***, é ignorada pela ferramenta	*a1,**b**, c1->b2,**b**, c2->b2, a2->b2* - Igual a include_once
WAP	*a1,**b**, c1->b2, NullPointerException*	*a1,**b**, c1->b2, NullPointerException*
Zend	*a1,**b**, c1->b2,**b**, c2->b2, a2->b2* - A inclusão do ficheiro ***b***, durante a execução do ficheiro ***c***, é ignorada pelo interpretador	*a1,**b**, c1->b2,**b**, recursão infinita* - O interpretador desencadeia ciclo infinito advindo da redeclaração de função com ou sem guarda.

Tabela 2. Comportamento das ferramentas e PHP Zend com os casos de uso extraídos da Fig. 2.

Propomos uma abordagem que, durante a AEC de um projeto, identifique os pontos no código (ciclos) com dependências circulares e a dissolução destes. Para tal, primeiramente é necessário apurar as raízes e os caminhos iniciados nestas. A abordagem define dois algoritmos para a resolução de ciclos.

4.1 Identificação de Ciclos

Algoritmo que identifica os ciclos em dependências circulares. Para o efeito, o algoritmo executa três passos, nomeadamente, a extração das raízes do projeto, a extração dos caminhos de inclusão de ficheiros, e a descoberta de possíveis causadores de ciclos nestes caminhos. Estes passos são apresentados no Algoritmo 1.

Algoritmo 1 Deteção de ciclos dado um conjunto de ficheiros e respetivas dependências.

```
1: for each file ∈ directory do
2:     create AST
3:     create an empty ST
4:     for each node_ast ∈ AST do
5:         create an node_st with information from node_ast
6:         if node_ast is an include then
7:             add ancestor and sucessor information to node_st
8:         insert node_st in ST
9:     insert ST in MST
10: for each st ∈ MST do
11:     if st ∌ ancestors then
12:         add file information from st to roots_list
13: for each root ∈ roots_list do
14:     build file inclusion path
15:     identify cycles in path
```

O Algoritmo 1, para cada ficheiro PHP contido na diretoria base de um projeto (*input*), extrai informação sobre os ficheiros que ele inclui. Para cada ficheiro, o algoritmo obtém a AST e a tabela simbólica (ST) a partir dela (linhas 1 a 8). A ST é uma representação da AST, onde cada nó contem informação sobre cada elemento da AST (ex., linha, ficheiro, nome da variável). De cada nó da AST referente a uma diretiva de inclusão de ficheiro (ex., *include*) é extraída informação sobre o ficheiro a incluir (linhas 6 e 7), nomeadamente do seu ficheiro ancestral e do sucessor do ficheiro em processamento. As linhas 10 a 12 extraem

as raízes, i.e., ficheiros que não são incluídos, e as adicionam à lista de raízes. Por fim (linhas 13 a 15), o algoritmo para cada raiz constrói o caminho de inclusão de ficheiros, usando a informação dos ancestrais e descendentes, e sinaliza neste os causadores dos ciclos. Um causador de ciclo é um ficheiro em que um dos seus ancestrais é também seu sucessor.

4.2 Resolução de Ciclos

O algoritmo para a resolução de ciclos assenta na discussão realizada na Secção 3, i.e., nos quatro casos-tipo e nos testes realizados com os casos de uso.

Para os caminhos extraídos, o algoritmo na presença de causadores de ciclos, para as diretivas *include_once/require_once* e ficheiro a incluir somente com funções, inclui uma só vez o ficheiro. Por seu turno, para as diretivas *include/require* e inclusão de ficheiros que contêm somente funções, o algoritmo trata de igual forma às diretivas anteriores. Para os ficheiros a incluir que contêm instruções, o algoritmo desdobra o ciclo, verificando qual o código do ficheiro que já foi incluído e analisado em prévias inclusões para somente incluir aquele que ainda não foi analisado.

5 Avaliação Experimental

Os algoritmos apresentados na Secção 4 foram implementados na ferramenta WAP. Uma avaliação experimental foi realizada usando a WAP na análise de plugins de WordPress e aplicações web, e teve como objetivo testar os referidos algoritmos na identificação e resolução de dependências circulares.

A Tabela 3 apresenta os resultados da avaliação, que envolveu 159 plugins de diversas categorias e 15 aplicações web (colunas 1 e 2). 4 plugins continham 14 ciclos em dependências circulares (colunas 3 e 4). A WAP corretamente detetou e resolveu os ciclos, não ocorrendo assim execução errónea da ferramenta e denotando que ambos os algoritmos são eficazes. WAP detetou 153 vulnerabilidades de diferentes classes de vulnerabilidades (últimas 3 colunas).

Categoria	**Apps**	**Apps com ciclos**	**Ciclos**	**SQLI**	**XSS**	**RFI**
Plugins Beta	5	0	0	0	0	0
Plugins Destaque	6	1	2	0	0	0
Plugins Populares	59	0	0	0	7	0
Plugins Aleatórios	89	3	12	32	40	17
Aplicações web	15	0	0	33	20	4
Total	**174**	**4**	**14**	**65**	**67**	**21**

Tabela 3. Identificação e resolução de dependências circulares usando WAP.

6 Conclusão

O artigo apresentou o problema das dependências circulares nas ferramentas de análise estática de código (AEC), advindo da inclusão recursiva de código de um ficheiro noutro. Um conjunto de casos de uso com este problema foi definido e testado em quatro ferramentas de AEC e o PHP Zend engine, verificando-se inconsistências de execução. Dois algoritmos para deteção e resolução de tais

dependências foram desenvolvidos e implementados na ferramenta WAP. Uma avaliação usando WAP foi realizada com aplicações web e plugins, onde foram detetadas e resolvidas dependências circulares.

7 Agradecimentos

Este trabalho foi parcialmente suportado pela EC através do projeto FP7-607109 (SEGRID) e pelos fundos nacionais através da Fundação para a Ciência e a Tecnologia (FCT) com referência para UID/CEC/00408/2013 (LaSIGE).

Bibliografia

1. Dahse, J.: RIPS - A Static Source Code Analyser for Vulnerabilities in PHP Scripts. http://www.nds.rub.de/media/nds/attachments/files/2010/09/rips-paper.pdf (2010), accessed: 2016-11-1
2. Hoglung, G., McGraw, G.: Exploiting software: The Achilles' Heel of CyberDefense (June 2004)
3. Imperva: Web application attack report #6 (November 2015)
4. Jovanovic, N., Kruegel, C., Kirda, E.: Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper). In: Proceedings of the IEEE Symposium on Security and Privacy (2006), http://dx.doi.org/10.1109/SP.2006.29
5. Jovanovic, N., Kruegel, C., Kirda, E.: Precise Alias Analysis for Static Detection of Web Application Vulnerabilities. In: Proceedings of the Workshop on Programming Languages and Analysis for Security (2006), http://doi.acm.org/10.1145/1134744.1134751
6. Landi, W.: Undecidability of Static Analysis. ACM Letters on Programming Languages and Systems (Dec 1992), http://doi.acm.org/10.1145/161494.161501
7. Medeiros, I., Neves, N., Correia, M.: Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining. IEEE Transactions on Reliability (March 2016)
8. Medeiros, I., Neves, N., Correia, M.: Equipping WAP with WEAPONS to Detect Vulnerabilities: Practical Experience Report. In: Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (June 2016)
9. Nunes, P., Fonseca, J., Vieira, M.: phpSAFE: A Security Analysis Tool for OOP Web Application Plugins. In: Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (2015), http://dx.doi.org/10.1109/DSN.2015.16
10. PHP.net: PHP: include - Manual. https://secure.php.net/manual/en/function.include.php (2017), accessed: 2017-6-1
11. PHP.net: PHP: include_once - Manual. https://secure.php.net/manual/en/function.include-once.php (2017), accessed: 2017-6-1
12. L. de Poel, N.: Automated Security Review of PHP Web Applications with Static Code Analysis. Master's thesis, University of Groningen (May 2010)
13. Prause, C.R., Reiners, R., Dencheva, S.: Empirical Study of Tool Support in Highly Distributed Research Projects. In: Proceedings of the IEEE International Conference on Global Software Engineering (Aug 2010)
14. Wichmann, B., Canning, A., Marsh, D., Clutterbuck, D., Winsborrow, L., Ward, N.: Industrial Perspective on Static Analysis. Software Engineering Journal (March 1995), http://digital-library.theiet.org/content/journals/10.1049/sej.1995.0010