Test Case Selector: Uma Ferramenta para Seleção de Testes

Juliana Mafra, Breno Miranda, Juliano Iyoda, Augusto Sampaio

[1]Centro de Informática
Universidade Federal de Pernambuco
Cidade Universitária - CEP 50740-540 - Recife - PE - Brasil

{jndm,bafm,jmi,acas}@cin.ufpe.br

Abstract. *This paper describes a test selection tool called Test Case Selector (TCS). The tool has been developed as part of the BTC project: a cooperation project between Centro de Informática and Motorola. Regression test selection at the BTC is not a trivial task. The test architects select few hundred tests from a suite containing thousands of tests. The TCS assigns points to each test based on 4 criteria: number of times the test was previously chosen; number of faults per execution; number of new faults uncovered by the test; and the test complexity. The user assigns weights to each criteria and the TCS calculates the weighted mean in order to produce a ranking of the most relevant tests (the higher the score, the higher the rank). The application of the selection criteria using the TCS is illustrated by a case study.*

Resumo. *Este artigo descreve a ferramenta* Test Case Selector *(TCS), desenvolvida para auxiliar o usuário a selecionar testes. O* Test Case Selector *foi desenvolvido como parte do projeto BTC de cooperação entre o Centro de Informática da UFPE e a Motorola. A seleção manual de testes de regressão no BTC não é uma tarefa fácil. Em geral, uma suíte possui milhares de testes e, em um ciclo de regressão, apenas algumas centenas podem ser executados. O processo de seleção do TCS atribui pontos a cada teste de acordo com 4 critérios de seleção: número de execuções, taxa de falhas por execução, número de defeitos novos encontrados e complexidade do teste. O usuário atribui pesos a cada critério e o TCS calcula uma média ponderada para cada teste. Os testes que mais pontuaram são mais relevantes para o ciclo de regressão. Tanto o critério de seleção quanto sua implementação pelo TCS são ilustrados em um estudo de caso.*

1. Introdução

A atividade de teste de software é fundamental para o desenvolvimento de sistemas com qualidade. Entretanto, esta é uma das atividades mais custosas dentro do ciclo de vida de desenvolvimento de software. O custo pode chegar até a 90% do custo total [Beizer 1990]. Em particular, testes de regressão têm como objetivo garantir que, após a integração de novas funcionalidades, novos defeitos não foram inseridos em componentes já testados, assegurando, portanto, a estabilidade do software. Executar novamente todos os casos de testes é em geral economicamente inviável. É necessário portanto selecionar, dentro do universo dos testes, um subconjunto daqueles que são mais adequados para assegurar a qualidade do software. Além disso, dependendo do estágio de desenvolvimento do software, os *critérios* para a seleção adquirem importância (*peso*) diferente. Por exemplo,

no primeiro ciclo de testes de regressão, selecionam-se testes que cubram pelo menos os requisitos funcionais mais básicos das novas funcionalidades. Já no último ciclo de regressão, quando a maturidade do software é maior, é aconselhável selecionar testes que exercitem as funcionalidades do sistema de forma mais elaborada.

Este artigo descreve uma ferramenta que auxilia no processo de seleção de testes de regressão: o *Test Case Selector*. Esta ferramenta produz um *ranking* de relevância dos testes baseado em 4 critérios de seleção. Cada critério atribui uma pontuação a cada teste e o usuário atribui um peso a cada critério. A pontuação final de cada teste é calculada por uma média ponderada dos pontos.

Este trabalho faz parte do projeto Brazil Test Center (BTC), uma cooperação entre a Motorola e o Centro de Informática da Universidade Federal de Pernambuco (CIn/UFPE). O BTC executa testes caixa-preta em celulares, ou seja, os testes são criados a partir da especificação do software e não há acesso ao código-fonte. A maioria dos testes são executados manualmente. A quantidade de testes disponíveis para se testar determinado componente de um software pode ser da ordem de milhares, enquanto os recursos disponíveis (tempo e testadores) suportam um ciclo com apenas algumas centenas de casos de teste. Fazer a seleção dos testes manualmente requer vasto conhecimento da suíte e bastante experiência com testes de celulares.

A maioria dos trabalhos existentes na literatura que propõem técnicas e ferramentas para seleção de testes de regressão utilizam a estratégia caixa-branca e, consequentemente, não se aplicam ao contexto do BTC [Rothermel and Harrold 1996, Elbaum et al. 2001, Harrold and Chittimalli 2008, Bahsoon and Mansour 2001, Harrold and Rothermel 1997]. Alguns trabalhos na área de depuração de código permitem selecionar testes caixa-preta através de análises de predição de defeitos [Hassan and Holt 2005, Kim et al. 2007, Schröter et al. 2006]. Apesar dos critérios propostos nesta área utilizarem a estratégia caixa-preta, não foi possível implementar a maioria deles pois a base de dados dos testes no BTC não contém as mesmas informações requeridas pelas análises de predição. Os arquitetos de teste do BTC então propuseram 4 critérios para compatíveis com a base de dados existente e baseado nas suas experiências em seleção manual de testes. Este critérios foram implementados no *Test Case Selector*.

As principais contribuições deste artigo são: a definição de 4 critérios para seleção de testes caixa-preta baseados em informações históricas e julgamento de especialistas; a implementação de uma ferramenta para mecanizar a seleção e um estudo de caso ilustrando o uso da ferramenta com suítes reais do BTC.

As próximas seções estão organizadas como segue. A Seção 2 introduz os critérios de seleção adotados e o cálculo da pontuação de cada teste. A Seção 3 explica detalhes da implementação da ferramenta. A Seção 4 descreve o estudo de caso realizado. A Seção 5 apresenta trabalhos relacionados e a Seção 6 conclui e descreve trabalhos futuros.

2. Seleção de Testes

De acordo com [Myers 1979], um bom caso de teste é aquele que tem alta probabilidade de detectar falhas. Partindo deste princípio, o *Test Case Selector* faz uso de alguns critérios de seleção com o objetivo de identificar quais testes estão mais propensos a esta

tarefa. Cada caso de teste é pontuado de acordo com os critérios adotados pelo *Test Case Selector* (TCS) e, aqueles que obtiverem as maiores pontuações, serão considerados os mais relevantes para serem testados naquela fase do ciclo de vida do *software*.

2.1. Critérios de Seleção

Os critérios de priorização utilizados pelo TCS bem como os atributos utilizados para o cálculo da relevância de cada teste serão detalhados abaixo.

- **Número total de execuções:** representa o número de vezes que determinado teste foi executado em ciclos anteriores. Se um teste é aplicável a um determinado produto e foi executado poucas vezes, maior será a sua pontuação. Ou seja, quanto *menos* um teste foi executado, *maior* sua pontuação. O cálculo deste critério é realizado da seguinte forma:

$$crit_1 = \frac{1}{execucoesCT}$$

onde $execucoesCT$ é o número de execuções do caso de teste. Se um teste executou zero vezes, a equação acima torna-se indefinida. Neste caso, este teste é arbitrariamente colocado em primeiro do *ranking*. Esta decisão preserva o princípio de valorizar testes pouco executados.
Exemplo. A Tabela 1 mostra 5 testes fictícios e o número de execuções de cada.

Tabela 1. Total de Execuções

Teste	No. Execuções
A	2358
B	83
C	0
D	1250
E	741

Tabela 2. Resultado do critério 1.

Rank	Teste	Pontos	Normalização
1^o	C	0,012049	45,18
2^o	B	0,012048	45,17
3^o	E	0,001350	5,06
4^o	D	0,000800	3,00
5^o	A	0,000424	1,59

O resultado da aplicação deste critério pode ser visto na coluna "Pontos" da Tabela 2. Observe que o teste C tem zero execuções (Tabela 1). A pontuação recebida por tal teste é a mínima necessária para ele ser primeiro do *ranking*. A coluna "Normalização" apresenta o percentual de pontos de cada teste em relação ao total de pontos de todos os testes. Esta normalização será aplicada a todos os critérios para que possamos fazer comparações entre critérios posteriormente.

- **Taxa de Falhas por Execução:** Este critério calcula a proporção de falhas sobre o número total de execuções de determinado caso de teste. Este critério valoriza testes que foram mais eficazes no passado.
O cálculo deste critério é realizado da seguinte forma:

$$crit_2 = \frac{falhasCT}{execucoesCT}$$

onde $falhasCT$ é o número de falhas que o caso de teste detectou e $execucoesCT$ é o número de execuções do caso de teste. Se $execucoesCT$ for zero, o teste recebe relevância zero.

Exemplo. Para os mesmos testes do exemplo anterior, assumimos alguns valores para o número de falhas encontrados por eles (veja a Tabela 3). Se ordenarmos a Tabela 3 de acordo com a pontuação obtida por cada teste, teríamos o seguinte *ranking* (do mais relevante ao menos relevante): 1^o E, 2^o D, 3^o A, 4^o B e 5^o C.

Tabela 3. Critério 2: Falhas por execução.

Teste	No. Execuções	No. de Falhas	Pontos	Normalização
A	2358	65	2,75	10,09
B	83	2	2,40	8,82
C	0	0	0	0
D	1250	120	9,60	35,14
E	741	93	12,55	45,94

- **Número de defeitos únicos encontrados:** Um mesmo caso de teste pode ter falhado inúmeras vezes, porém revelando o mesmo defeito repetidamente: isso pode acontecer se o caso de teste foi reusado em diferentes produtos que apresentavam a mesma falha; ou se o caso de teste foi utilizado mais de uma vez no mesmo produto num curto espaço de tempo e a falha não havia sido corrigida ainda. Ou seja, é possível um teste falhar 3 vezes, mas nas 3 falhas ele revela o mesmo defeito. Neste caso, apenas uma única *Change Request* (CR) é aberta e atribuída a este teste. O critério anterior foca no número de vezes que o teste falhou sobre o número total de execuções, independente das repetições. O critério 3 leva em consideração os defeitos encontrados por cada teste sem contar as repetições.

$$crit_3 = CRs$$

onde CRs é o número de CRs abertas pelo caso de teste.
Exemplo. A Tabela 4 mostra os testes dos exemplos anteriores e a quantidade de CRs abertas por cada. Observe que, neste critério, o número de CRs já define a pontuação do teste. A ordem por relevância para este critério é: 1^o E, 2^o A, 3^o D, 4^o B e 5^o C.

Tabela 4. Critério 3: Número de CRs abertas.

Teste	CRs	Pontos	Normalização
A	18	18	35,29
B	2	2	3,92
C	0	0	0,00
D	11	11	21,57
E	20	20	39,22

- **Regression Level:** cada caso de teste no BTC está classificado de acordo com um código chamado *Regression Level*, que pode variar de 1 a 5. Quanto maior o valor do *Regression Level*, mais complexo será o caso de teste e um maior número de interações com outros componentes estará presente. Um caso de teste que possui

Regression Level igual a 1, por exemplo, tem o objetivo de validar os requisitos funcionais mais básicos de determinada funcionalidade, sem a preocupação de interagir com outros componentes do *software*. Um teste com *Regression Level* 5 é complexo, elaborado e testa várias interações entre componentes.
Neste critério, o usuário deve atribuir pontos a cada *Regression Level*. Por exemplo, dependendo da fase do desenvolvimento, testes com *Regression Level* 1 e 2 são mais relevantes que testes com *Regression Level* 3, 4 e 5. Neste caso, testes com *Regression Level* 1 e 2 pontuam mais que testes com *Regression Level* 3, 4, e 5. Para um teste com *Regression Level* n, o cálculo deste critério é realizado da seguinte forma:

$$crit_4 = pontosRegLevel_n$$

onde $pontosRegLevel_n$ são os pontos previamente atribuídos a um teste cujo *Regression Level* é n.
Exemplo. A Tabela 5 exibe uma pontuação fictícia para cada *Regression Level* (*Reg Level*). Para este cenário, testes com níveis 1 e 2 serão mais valorizados que testes com níveis 3, 4 e 5.

Tabela 5. Pontuações.

Reg Level	Pontos
1	60
2	50
3	10
4	15
5	20

Tabela 6. Resultado por *Reg Level*.

Teste	*Reg Level*	Pontos	Normalização
A	4	15	7,69
B	2	50	25,64
C	5	20	10,26
D	2	50	25,64
E	1	60	30,77

Para cada um dos 5 testes (de A a E) usados nos exemplos, atribuímos de forma aleatória valores para seus *Regression Levels*. A Tabela 6 mostra os níveis de *Regression Level* de cada teste e quantos pontos cada teste fez. Para o critério de *Regression Level*, o *ranking* de relevância é: 1^o E, 2^o B, 3^o D, 4^o C e 5^o A. A ordem entre B e D é atribuída de forma aleatória, já que ambos obtiveram a mesma pontuação.

2.2. Cálculo da Média Ponderada

A cada critério poderá ser atribuído um peso diferente. Desse modo, quanto mais importante um critério for para a seleção, maior deverá ser o seu peso correspondente. Após a definição dos pesos, a ferramenta irá calcular a média ponderada de todos os critérios e exibir os casos de teste em ordem decrescente de relevância.

A relevância final de um teste é calculada através da média ponderada dos pontos normalizados.

$$relevancia = \frac{\sum_{i=1}^{4}(crit_i \cdot pesoCrit_i)}{\sum_{j=1}^{4} pesoCrit_j}$$

onde $crit_i$ é a pontuação normalizada no critério i e $pesoCrit_j$ é o peso do critério j.

Exemplo. A Tabela 7 mostra uma atribuição fictícia de pesos para cada critério. Neste exemplo, os critérios 4 e 1 são os mais relevantes.

Tabela 7. Pesos por critério.

Critério	Peso
1 - Número total de execuções	5
2 - Taxa de Falhas por Execução	2
3 - Número de defeitos únicos encontrados	1
4 - *Regression Level*	8

Tabela 8. Resultado final

Teste	Média ponderada
A	7,81
B	28,29
C	19,25
D	19,50
E	25,16

A Tabela 8 exibe o resultado final da média ponderada para cada um dos testes. A ordem de relevância final é: 1^o B, 2^o E, 3^o D, 4^o C e 5^o A. Como o maior peso foi dado ao critério *Regression Level*, é esperado que o teste mais relevante neste critério (teste E) tenha ficado no alto do *ranking* final também. O teste E obteve as seguintes pontuações (normalizadas):

$$crit_1 = 5,06, \quad crit_2 = 45,94, \quad crit_3 = 39,22, \quad crit_4 = 30,77.$$

O cálculo final de relevância para o teste E é:

$$relevancia = \frac{(5,06 \cdot 5)+(45,94 \cdot 2)+(39,22 \cdot 1)+(30,77 \cdot 8)}{5+2+1+8} = 25,16.$$

3. A Ferramenta

O *Test Case Selector* foi implementado em Java [Arnold et al. 2005]. Esta escolha se deu principalmente pela sua portabilidade e facilidade de futuras integrações com outros sistemas desenvolvidos no BTC. O desenvolvimento foi conduzido dentro do Ambiente de Desenvolvimento Integrado *Eclipse* [Holzner 2004]. O acesso a dados do *Test Case Selector* é implementado através de consultas SQL executadas em uma base de dados local. O Sistema Gerenciador de Banco de Dados utilizado é o *Microsoft SQL Server 2005* [Rankins et al. 2007].

A Figura 1 exibe a tela principal do TCS. Nesta primeira guia, chamada de *Add Tests*, o usuário deve especificar um componente (campo *Groups*) e o código das *features*[1] dos casos de teste que deseja selecionar (números 54368 e 63187 na Figura 1). Em seguida, o sistema irá exibir o nome, a descrição e o *Regression Level* dos casos de teste. Os demais campos somente serão preenchidos após o cálculo de relevância (feita na guia *Apply Weights*). O usuário sempre pode eliminar um teste da seleção manualmente (veja o botão *Delete* na parte de baixo da tela).

A Figura 2 exibe a segunda guia, chamada *Search Terms*, onde o usuário poderá filtrar casos de teste de acordo com a ocorrência de determinadas palavras-chave. No exemplo da Figura 2, o usuário digitou a palavra-chave "flip". O botão na parte de baixo chamado *Invert Selection* tanto pode selecionar os testes contendo a palavra-chave "flip", quanto selecionar os testes que não contém a palavra "flip".

[1] *Feature* é um termo usado para denominar uma funcionalidade coesa e identificável [Turner et al. 1998]. No caso de celulares, por exemplo, *Messaging* é a *feature* associada ao envio e recebimento de mensagens.

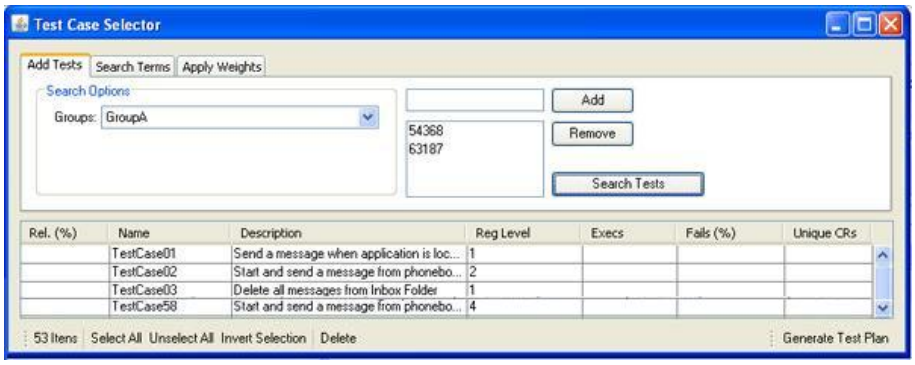

Figura 1. Tela Principal - Casos de Teste Iniciais

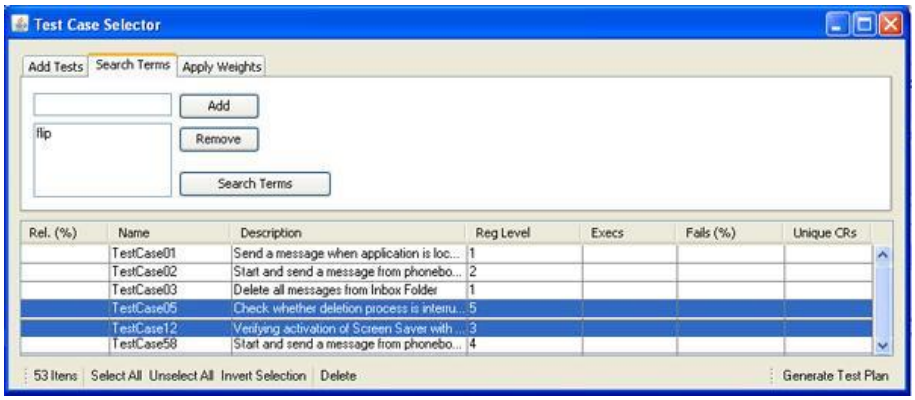

Figura 2. Busca de Casos de Teste por Palavras-chave

A Figura 3 exibe a terceira guia, chamada *Apply Weights*, onde o usuário poderá aplicar pesos aos critérios de seleção e também os pontos de cada *Regression Level*. Após a configuração dos pesos, a ferramenta irá exibir os casos de teste em ordem decrescente de relevância. No exemplo da Figura 3, os 3 testes mais relevantes são *TestCase26*, *TestCase09* e *TestCase02*. Para gerar um plano de teste real, os casos de testes selecionados pela ferramenta poderão ser convertidos no formato do Excel (dentro do padrão adotado pelo BTC) de forma que a planilha resultante seja compatível com outras ferramentas do BTC.

4. Estudo de Caso

Para realizar o estudo de caso, foi simulada a criação de um plano de testes para o primeiro ciclo de regressão, chamado de *Rolling Regression C1* (RR C1). Um componente do software foi escolhido arbitrariamente junto com uma quantidade de testes a serem executados no ciclo: 450 testes. Em seguida, os seguintes passos foram aplicados:

1. Adicionar todas as suítes de testes disponíveis para validar as funcionalidades (*features*) já integradas ao software até a data de início do ciclo. Para a nossa simulação, 33 suítes foram adicionadas resultando em 862 casos de teste. O tempo de carga da suíte foi de aproximadamente 5 minutos.[2]

[2]Utilizamos um computador Intel(R) Pentium(R) Dual, 1,98GB de RAM, 2GHz.

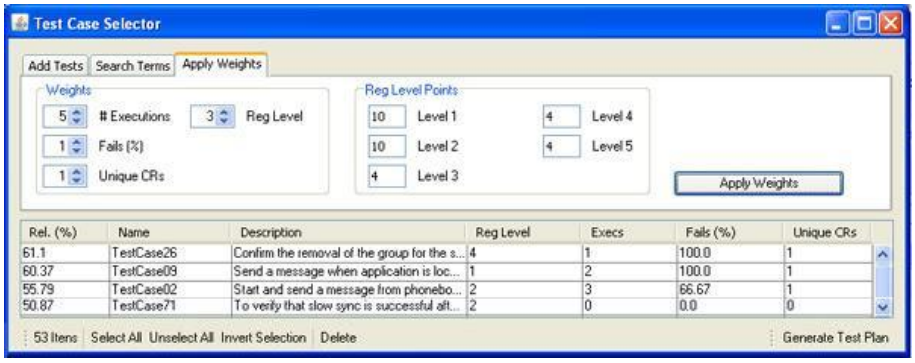

Figura 3. Aplicação de Pesos aos Critérios

2. Remover casos de testes não aplicáveis ao produto utilizando o método de busca por palavras-chave. Nesta operação, 17 palavras-chave foram adicionadas resultando em 582 casos de teste.
3. Atribuir pesos para classificar os casos de testes de acordo com a relevância dos mesmos. Para o ciclo sendo simulado, atribuímos os pesos 2, 5, 4 e 3 para os critérios 1, 2, 3 e 4, respectivamente. Os níveis 1, 2, 3, 4 e 5 de *Regression Level* foram associados à 10, 8, 6, 1 e 1 pontos, respectivamente.
4. Remover os testes considerados menos relevantes até atingir a quantidade estipulada para este ciclo. Na nossa simulação, removemos os 132 testes menos relevantes até alcançar a quantidade estipulada de 450 casos de teste.
5. Exportar os casos de testes selecionados para uma planilha no formato padrão do BTC.

A Figura 4 exibe os casos de teste considerados mais relevantes pelo TCS após a realização dos passos acima especificados. O cálculo de relevância leva poucos segundos. Portanto, um tempo desprezível em comparação com o tempo de carga inicial dos testes (5 minutos).

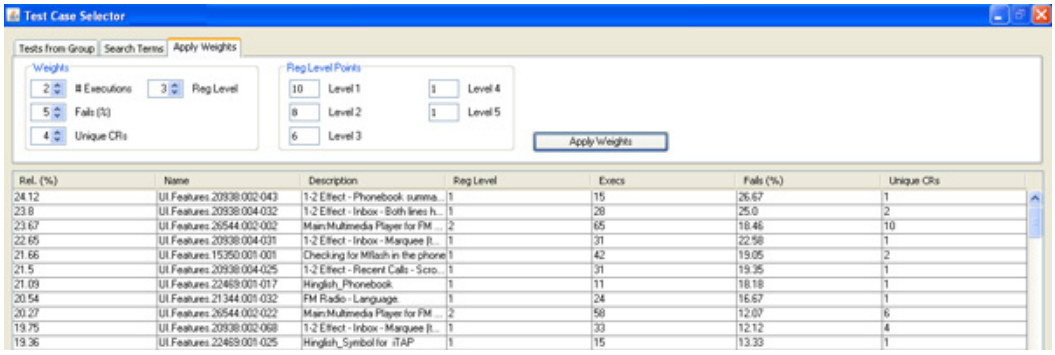

Figura 4. Casos de teste considerados mais relevantes pelo TCS.

5. Trabalhos Relacionados

[Harrold and Chittimalli 2008] propõem uma abordagem para seleção de testes de regressão que, ao invés de utilizar modelos ou código fonte, utiliza os requisitos do

sistema. A associação dos requisitos com os casos de teste é feita através de um fator de criticidade inserido em uma matriz de cobertura. Uma ferramenta, parcialmente manual, utiliza a matriz de cobertura para selecionar e ordenar os casos de teste. [Bahsoon and Mansour 2001] apresentam alguns resultados iniciais de um estudo empírico de técnicas de seleção de testes de regressão. O estudo examinou alguns dos custos e dos benefícios das técnicas existentes. Os resultados destacaram várias diferenças entre as técnicas e apresentaram vantagens e desvantagens essencias que podem ser consideradas na escolha de uma técnica para aplicação prática. [Harrold and Rothermel 1997] implementam um algoritmo seguro de seleção de casos de testes utilizando a técnica de re-teste de código modificado. O algoritmo gera grafos de fluxo de controle para as versões originais e modificadas do software e compara os dois grafos para identificar trechos modificados nas duas versões. [Hassan and Holt 2005] apresentam uma abordagem que destaca os dez subsistemas (*The Top Ten List*) mais susceptíveis a conter defeitos, auxiliando os gerentes a determinar em quais subsistemas focar ao se selecionar testes. No trabalho proposto por [Kim et al. 2007], foi desenvolvido um algoritmo de predição que ajuda a identificar as entidades do software mais propensas a conter defeitos utilizando um mecanismo de *cache* para armazenar esta lista de entidades. Em [Schröter et al. 2006], foram extraídos bugs do Eclipse e de versões de bancos de dados com o intuito de mapear falhas em componentes do Eclipse. De acordo com os dados resultantes dessa análise, pode-se concluir que a propensão a falhas é uma combinação de múltiplos fatores. Algumas possíveis fontes de defeitos foram identificadas: desenvolvedores, complexidade do código e domínio do problema.

Os trabalhos de seleção são, na maioria, específicos para testes de caixa-branca. Alguns trabalhos de predição de defeitos podem ser usados como ferramentas de seleção de testes caixa-preta. Entretanto, dos trabalhos estudados, poucos critérios propostos na predição de defeitos são mecanizáveis no contexto do BTC.

6. Conclusão

Este trabalho propõe um método e implementa uma ferramenta para apoiar a seleção de testes regressão de modo que esta seleção ordende casos de testes baseados em dados históricos. Com este propósito, quatro critérios foram definidos para serem utilizados no processo de seleção de testes. Estes critérios foram definidos pelos arquitetos do BTC, que se basearam em informações históricas e nas suas experiências em seleção manual de testes.

Para validar o método proposto, uma ferramenta foi desenvolvida na linguagem Java a fim de verificar como os critérios se comportam na prática. Esta ferramenta produz um *ranking* de relevância dos testes através de uma média ponderada calculada com base em pesos atribuídos pelo usuário aos quatro critérios de seleção. A aplicação da ferramenta foi ilustrada através de um estudo de caso realizado no BTC.

Alguns trabalhos futuros são importantes para aprimorar e consolidar este trabalho: implementar novos critérios; realizar experimentos que possibilitem a comparação do método proposto com o método de seleção manual; e a configuração automática dos pesos dos critérios de acordo com o perfil dos usuários e a fase do ciclo de testes.

Em particular, a comparação entre os métodos manual e automático é feita através da aplicação da seleção de testes nas mesmas suítes utilizando ambas as técnicas.

Avaliaremos tanto a interseção dos testes selecionados quanto a interseção dos testes que acham defeitos presentes nos 2 subconjuntos selecionados.

References

Arnold, K., Gosling, J., and Holmes, D. (2005). *The Java Programming Language*. Prentice Hall PTR, fourth edition.

Bahsoon, R. and Mansour, N. (2001). Methods and metrics for selective regression. *ACS/IEEE International Conference on Computer Systems and Applications*, pages 463–465.

Beizer, B. (1990). *Software Testing Techniques*. International Thomson Computer Press, second edition.

Elbaum, S., G.Malishevsky, A., and Rothermel, G. (2001). Prioritizing test cases for regression testing. *Proceedings of the International Symposium on Software Testing and Analysis*, pages 102–112.

Harrold, M. J. and Chittimalli, P. K. (2008). Regression test selection on system requirements. *Proceedings of the 1st India Software Engineering Conference*, pages 19–22. Hyderabad, India.

Harrold, M. J. and Rothermel, G. (1997). A safe, efficient regression test selection. *ACM Transactions on Software Engineering and Methodology*, 6(2):173–210.

Hassan, A. E. and Holt, R. C. (2005). The top ten list: Dynamic fault prediction. *Proceedings of International Conference on Software Maintenance*, pages 263–272. Budapest, Hungary.

Holzner, S. (2004). *Eclipse*. O'Reilly Media, Inc, first edition.

Kim, S., Zimmermann, T., Whitehead, E. J., and Zeller, A. (2007). Predicting faults from cached history. *Proceedings of the 29th International Conference on Software Engineering*, pages 489–498.

Myers, G. J. (1979). *The Art of Software Testing*. John Wiley & Sons inc, New Jersey.

Rankins, R., Bertucci, P., Gallelli, C., and Silverstein, A. T. (2007). *Microsoft SQL Server 2005 Unleashed*. Sams, first edition.

Rothermel, G. and Harrold, M. J. (1996). Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22:529–551.

Schröter, A., Zimmermann, T., Premraj, R., and Zeller, A. (2006). Where do bugs come from? *A Challenge for Empirical Software Engineering; ACM SIGSOFT Software Engineering Notes*, 31:1–2.

Turner, C. R., Wolf, A. L., Fuggetta, A., and Lavazza, L. (1998). Feature engineering. *Proceedings of the 9th International Workshop on Software Specification and Design*, page 192.