Higor Amario de Souza, 5619750 – 20 de maio de 2013.

MAC 5779 – Engenharia de Software Experimental

Projeto de Pesquisa (Doutorado): Experimento controlado.

Tema: Depuração automatizada em programas contendo múltiplos defeitos.

Problema e contextualização

As atividades de teste e depuração de programas são responsáveis por uma parcela significativa do processo de desenvolvimento de programas. O esforço para detectar (teste), localizar e corrigir (depuração) defeitos consome entre 50% e 80% do desenvolvimento e manutenção (Collofello and Woodfield, 1989). A depuração de programas é geralmente realizada de forma manual pelos desenvolvedores (Jones et al., 2007).

Diferentes estudos propõem técnicas para automatizar a tarefa de depuração de programas, a maioria delas baseada em informações de cobertura de código (Naish et al., 2010; Wong et al., 2010; Mariani et al., 2011). A partir da cobertura obtida dos testes executados, tais técnicas indicam comandos mais suspeitos de conter defeitos. No entanto, os experimentos para avaliar a eficácia de localização das técnicas em geral são realizados em programas contendo um único defeito por versão. A eficácia de localização é medida pela quantidade de código que precisa ser verificado até que o defeito seja localizado. Nos casos em que a avaliação é feita em programas contendo múltiplos defeitos, o desempenho de localização é inferior ao desempenho em programas com um defeito por versão (Naish et al., 2010; Wong et al., 2010).

Na prática, ao procurar defeitos em programas reais, a quantidade de defeitos em um programa é desconhecida (Jones et al., 2007). Portanto, é necessário que as técnicas de localização de defeitos consigam indicar comandos suspeitos em programas com múltiplos defeitos de forma eficaz para que sejam adotadas em ambientes reais de desenvolvimento.

Objetivo geral

Melhorar a eficácia de localização de defeitos em programas contendo múltiplos defeitos por meio da avaliação da técnica de depuração a ser proposta em comparação com o atual estado da prática. A eficácia é medida pela quantidade de defeitos que são localizados e a eficiência é medida pela quantidade de código do programa que precisa ser verificada até atingir o código defeituoso.

Objetivos específicos

- Avaliar a eficácia e a eficiência de localização para o uso prático da técnica realizando experimentos com desenvolvedores.

- Avaliar a eficácia e a eficiência de localização em programas reais (contendo milhares de linhas de código e utilizados de modo prático).

Questões de pesquisa

A técnica proposta é mais eficaz e mais eficiente para a localização de defeitos em programas contendo múltiplos defeitos quando comparada ao atual estado da prática?

A técnica proposta é eficaz e eficiente para ser usada por desenvolvedores para localizar defeitos em programas reais?

Hipótese

A técnica proposta é mais eficaz e mais eificente para a localização de defeitos em programas contendo múltiplos defeitos em relação ao atual estado da prática.

Experimento

O objetivo do experimento controlado será avaliar a **eficácia** e a **eficiência** da técnica proposta Multiple Faults (**MF**) em comparação com o uso tradicional de depuração (**T**). Utilizaremos o IDE Eclipse para o experimento. Será desenvolvido um plugin da técnica MF para a avaliação de seu uso. O uso tradicional também será realizado no Eclipse, em que o desenvolvedor poderá utilizar o depurador simbólico para procurar pelo defeito.

A seleção dos participantes deverá contar com desenvolvedores **universitários** e da **indústria** e será feita por **conveniência**. É esperada a participação de **30 pessoas**. Ao aceitar participar do experimento, os participantes responderão a uma **prova** de conhecimentos sobre a linguagem Java, assim como um **questionário** sobre informações pessoais como idade, tempo de experiência em programação, tempo de experiência com depuração, etc. A prova será baseada em questões de certificações na linguagem, contendo diferentes níveis de conhecimento. Os participantes deverão ter conhecimento prévio do Eclipse.

De acordo com o resultado das respostas, os participantes serão classificados como iniciantes ou experientes. Os participantes serão separados em **dois grupos**, controle e experimental, usando a seleção por **casamento**.

O grupo de controle usará o Eclipse para buscar pelos defeitos sem o plugin. Esse grupo receberá um treinamento sobre o uso do depurador do Eclipse e uma breve explicação sobre o JUnit. O grupo experimental receberá um treinamento sobre o uso do plugin MF. O tempo previsto para ambos **treinamentos** será de 30 minutos, incluindo um **exercício prático**.

Serão utilizados **dois programas contendo dois defeitos** cada. Os programas são de domínios diferentes (Ant e Commons-Math). Os mesmos defeitos serão utilizados pelos dois grupos. O grupo de controle receberá também as informações sobre o resultado da execução dos testes JUnit. O tempo para a localização de cada defeito será de **30 minutos**.

A eficácia será medida por meio da localização ou não dos defeitos dentro do tempo previsto.
A eficiência será medida pela quantidade de cliques realizados nas listas geradas pelas técnicas até a localização do defeito. Para medir a quantidade de cliques do grupo de controle será necessário criar um plugin para o Eclipse que faça essa medição. O plugin da técnica MF também contará o número de cliques. Ao localizar o defeito, o desenvolvedor clica em um botão no Desktop para encerrar a busca por aquele defeito.

O tempo total máximo para a realização do experimentos é de **3h30m** por participante, incluindo a prova (1h), realizada em uma data anterior ao experimento, o treinamento (30m) e o experimento para localização dos quatro defeitos (2h).

Entre as ameaças à validade interna esperadas estão a **seleção**, que tentou-se minimizar usando a seleção por casamento e a prova de conhecimentos. As ameaças de **competição** e **desmoralização** devem ser consideradas e seus efeitos são difíceis de mensurar porque pessoas de um mesmo grupo podem comportar-se de uma das duas formas. Como não estamos aplicando um pré-teste, não há

como medir tais ameaças. Por outro lado, a ausência de um pré-teste minimiza as ameaças de contaminação, seleção-testagem, seleção-maturação e seleção-abandono.

Entre as ameaças externas estão a **generalização da amostragem**, já que a amostra não é representativa de população como um todo.

O tempo exigido para localizar os defeitos pode ser muito longo ou curto, e apenas a execução de um piloto pode indicar se o tempo está adequado.

A análise será feita com uso do teste de hipótese de Wilcoxon rank-sum não-pareado para a quantidade de cliques supondo que a distribuição da amostra não será normal. Os dados coletados serão tempo de localização e quantidade de cliques, duas medidas de razão. O teste de Anderson-Darling será aplicado para avaliar a normalidade dos dados.

Resultados esperados

Espera-se que a técnica resultante deste trabalho possa ser usada de forma prática na indústria de software. O uso da técnica deve proporcionar redução do tempo de desenvolvimento de programas e um aumento na qualidade dos programas.
A técnica deve ser adaptável a diferentes linguagens de programação e a programas com características diversas.

Referências

Collofello JS, Woodfield S. Evaluating the effectiveness of reliability-assurance techniques. Journal of Systems and Software 9 (3), 191–195. 1989.

Jones, J. A.; Bowring, J. F.; Harrold, M. J. Debugging in parallel. In: Proceedings of the 2007 International Symposium on Software Testing and Analysis. New York, NY, USA: ACM, 2007. (ISSTA ’07), p. 16–26.

Mariani, L.; Pastore, F.; Pezze, M. Dynamic analysis for diagnosing integration faults. Software Engineering, IEEE Transactions on, v. 37, n. 4, p. 486 –508. 2011.

Naish, L.; Lee, H. J.; Ramamohanarao, K. Statements versus predicates in spectral bug localization. In: Software Engineering Conference (APSEC), 2010 17th Asia Pacific. [s.n.], 2010. p. 375 –384.

Wainer, J. Experimento em sistemas colaborativos. Sistemas Colaborativos, Cap. 24, p.405-432, 2011.

Wong, W. E.; Debroy, V.; Choi, B. A family of code coverage-based heuristics for effective fault localization. Journal of Systems and Software, v. 83, n. 2, p. 188–208, 2010.