1.03.99 – Ciência da Computação.

MELHORIA DA QUALIDADE DE SOFTWARE ATRAVÉS DA ELIMINAÇÃO DA COMPLEXIDADE DESNECESSÁRIA EM CÓDIGO FONTE.

Nathan Manera Magalhães[1], Heleno Campos de Souza Junior[2], Marco Antônio Pereira Araújo[1, 2]
1. Instituto Federal do Sudeste de Minas Gerais
2. Universidade Federal de Juiz de Fora

Resumo:

Programadores iniciantes podem escolher priorizar o correto funcionamento de um código fonte sem focar em sua qualidade, podendo torná-los difíceis de serem mantidos ou testados. Ainda, pode ocorrer um fenômeno denominado complexidade estrutural desnecessária, onde o software apresenta uma complexidade ciclomática que pode ser reduzida sem que seu comportamento externo mude. Uma abordagem e uma ferramenta foram desenvolvidas para tratar deste problema em trabalhos anteriores. A abordagem é capaz de identificar a presença de complexidade ciclomática desnecessária e exibir para o desenvolvedor uma sugestão de reestruturação de seu código fonte, através de um grafo de fluxo de controle. O presente trabalho tem como objetivo a automatização do processo de refatoração do código fonte provido pela ferramenta, para dar suporte à eliminação da complexidade ciclomática desnecessária, e ajudar o desenvolvedor a despender menor esforço na melhora da qualidade de seu código fonte.

Palavras-chave: Complexidade Ciclomática; Teste de Software; Grafo de Fluxo de Controle;

Apoio financeiro: CNPQ, IF Sudeste MG.

Trabalho selecionado para a JNIC pela instituição: IF Sudeste MG

Introdução:

Desenvolvedores de software, sobretudo estudantes iniciantes de programação no meio acadêmico, podem optar por desenvolver códigos fonte se preocupando somente com que os mesmos sejam funcionais, deixando a qualidade de lado. Segundo Lehman [1], conforme um software evolui, seu código fonte tende a tornar-se cada vez mais complexo e, de acordo com Yu e Zhou [2], um código fonte com alta complexidade é de difícil compreensão, podendo ocasionar custos elevados para sua manutenção.

Uma forma de se medir o quanto complexo é um código fonte, é através da métrica complexidade ciclomática, proposta por McCabe [3]. Essa métrica se baseia na estrutura do fluxo de um código fonte para calcular a quantidade de fluxos de execução (caminhos únicos) presentes. A análise da estrutura do seu fluxo pode ser feita sobre Grafos de Fluxo de Controle (GFC), proposto por Allen [4]. Utilizam-se vértices para representar conjuntos de instruções de um código fonte, e arestas para representar a passagem de controle entre essas instruções. Um exemplo disso é a instrução condicional, que possui duas arestas representando os dois caminhos possíveis para esse tipo de instrução: a condição ser verdadeira ou ser falsa.

Conforme aumenta a complexidade ciclomática de um software em desenvolvimento, e o desenvolvedor não se preocupa em manter sua qualidade, pode ocorrer um fenômeno chamado de complexidade ciclomática desnecessária. Nesse fenômeno, um código fonte possui complexidade ciclomática que pode ser reduzida sem haver alteração no comportamento externo do mesmo. Esse fenômeno foi observado por Campos Junior et al. [5] no ambiente acadêmico de ensino de programação, onde se manifestou em 16% de 482 tarefas de programação analisadas. Os autores também desenvolveram uma ferramenta capaz de identificar complexidade ciclomática desnecessária e, exibir através do uso de GFCs, uma sugestão de reestruturação do código fonte analisado. Com base nessa sugestão, o desenvolvedor pode corrigir manualmente seu código fonte.

A interpretação dos GFCs pelo usuário e a correção manual do código fonte, são limitações que podem ser eliminadas para que o desenvolvedor despenda menos esforço ao melhorar a qualidade de seu software. Dessa forma, o presente trabalho tem como objetivo a adição de uma nova abordagem para automatizar a refatoração do código fonte, no intuito de eliminar a necessidade de se fazer a sua correção manual e também a de se interpretar os GFCs.

Metodologia:

O processo da abordagem desenvolvida neste trabalho permite com que a ferramenta proponha ao desenvolvedor uma

sugestão de refatoração do código fonte com base na comparação feita entre os dois GFCs, e assim diminuir o esforço do desenvolvedor em ter que refatorar o código fonte.

Este processo é iniciado com a execução da abordagem de identificação da complexidade desnecessária gerando dois GFCs, um representando o código fonte original e um refatorado, eliminando a complexidade desnecessária. São feitas listagens de todos os vértices do GFC original e do GFC refatorado a fim de que sejam determinados quais vértices foram excluídos do GFC original para formar o GFC refatorado. Os vértices identificados são inseridos em uma lista identificada como arranjo. Um laço de repetição é iniciado com a leitura do primeiro vértice do arranjo de vértices excluídos, cuja instrução condicional é identificada como condição atual.

Ao iniciar esse laço, é verificado primeiro se a condição atual possui uma expressão composta (com operadores lógicos de conjunção (&&) ou de disjunção (||)), e caso haja, ser decomposta para o formato de expressões simples (binária ou unária).

Em seguida, é procurado no GFC refatorado, um vértice representando uma condição que quando falsa, possua uma aresta que o liga diretamente ao vértice que representa a condição atual. Caso este vértice não seja encontrado, considera-se então que a condição atual é inalcançável, sendo esta eliminada do texto do código fonte junto com a sua estrutura interna. Com isso, o laço atual é encerrado para que se inicie um novo ao ler o próximo vértice do arranjo.

Caso o vértice procurado anteriormente tenha sido encontrado, então a condição atual é considerada uma condição redundante, ou seja, pode ser removida sem que o comportamento externo do programa mude. O laço atual então continua, e esse mesmo vértice é buscado na lista do GFC original para que seja feita uma comparação entre ele e o seu equivalente do GFC refatorado. Se o vértice da lista dos originais não possui um *else* em sua estrutura e o da lista dos refatorados possui um *else*, faz-se então a refatoração do texto do código fonte, excluindo o trecho correspondente à condição atual e adicionando em seu lugar um comando *else***.** Caso contrário, deve-se apenas excluir a condição atual. Quando a condição atual é considerada como redundante, a sua estrutura interna permanece preservada.

Por fim, o laço atual é encerrado para que se inicie um novo, caso ainda reste algum vértice condicional a ser lido do arranjo. O processo é encerrado quando todos os vértices do arranjo são lidos e assim, a sugestão de refatoração do código fonte lido é retornada à tela do usuário.

Resultados e Discussão:

A abordagem apresentada é capaz de exibir ao usuário uma sugestão de refatoração de seu código fonte para que a complexidade ciclomática desnecessária seja eliminada. A funcionalidade foi testada informalmente pelos seus desenvolvedores com exemplos de programas acadêmicos. Ainda, um estudo experimental executado anteriormente para avaliar a utlização da abordagem de identificação da complexidade ciclimática desnecessária foi reproduzido pelo desenvolvedor. Foi constatado o correto funcionamento da funcionalidade. Nesse estudo, um exemplo de programa desenvolvido, cuja complexidade ciclomática inicial era de 19, obteve redução para 10 e teve o código fonte corretamente refatorado.

Conclusões:

É esperado que com o uso da abordagem, o desenvolvedor tenha que despender menor esforço para melhorar a qualidade de seu software, podendo assim copiar e colar a sugestão provida.

Em trabalhos futuros, pretende-se evoluir a abordagem no sentido de automatizar a substituição do código fonte original pelo código fonte refatorado, integrar a ferramenta desenvolvida com Ambientes de Desenvolvimento Integrado (IDEs) através de *plug-ins* e realizar estudos de caso com usuários profissionais e softwares de maior porte, visando avaliar os benefícios práticos providos pelo uso da abordagem proposta.

Referências bibliográficas

[1] LEHMAN, M. M. “Programs, Life Cycles and Laws of Software Evolution”, **Proceedings of the IEEE**, vol. 68, no. 9, p.1060–1076, Set. 1980.

[2] YU, S. e ZHOU, S. 2010. “A Survey on Metric of Software Complexity”. In: IEEE INTERNATIONAL CONFERENCE ON INFORMATION MANAGEMENT AND ENGINEERING. 2nd, 2010, Chengdu. **Proceedings of the 2nd IEEE International Conference on Information Management and Engineering.** IEEE, p. 352-356, 2010.

[3] MCCABE, T. J. “A complexity measure”. In: **IEEE Trans. Software Eng**. Vol. SE-2, N. 4, p. 308-320, 1976.

[4] ALLEN, F. E. “Control flow analysis”, **Proceedings of a symposium on**

Compiler optimization, Urbana-Champaign, Illinois, p. 1-19, 1970.

[5] CAMPOS JUNIOR, H. S.; MARTINS FILHO, L. R. V.; ARAÚJO, M. A. P. “An Approach for Detecting Unnecessary Cyclomatic Complexity on Source Code”. **IEEE Latin America Transactions**, vol. 14, no. 8, 2016.