Testando Mecanismos de Refatoração Utilizando Programas Gerados Aleatoriamente*

Luiz Felipe Kraus[1], **Bruno Coelho**[1], **Samuel da Silva Feitosa**[1]

[1] Instituto Federal de Santa Catarina (IFSC)– Caçador – SC – Brasil

{luiz.k1999, bruno.sc11}@aluno.ifsc.edu.br, samuel.feitosa@ifsc.edu.br

Abstract. *With the advance in need, the use of random program generators for tool testing has been presented, since the specific test cases required from the programmer's imagination, where hardly all possibilities are tested. In this sense, this work aims to use the random code generator to test as the main refactoring tools in the Java language, to find possible refactoring bugs in IDEs such as Netbeans, Eclipse and IntelliJ.*

Resumo. *Com o avanço na computação, o uso de geradores de programas aleatórios para testes de ferramentas tem se mostrado promissor, uma vez que os casos de testes definidos manualmente dependem da imaginação do programador, onde dificilmente são testadas todas as possibilidades. Neste sentido, este trabalho tem como objetivo utilizar o gerador de códigos aleatórios para testar as principais ferramentas de refatoração da linguagem Java, para encontrar possíveis bugs de refatoração em IDEs como Netbeans, Eclipse e IntelliJ.*

1. Introdução

Atualmente, o Java é uma das linguagens mais utilizadas no mundo [Cass 2019]. Por se tratar de uma linguagem de propósito geral, fortemente tipada e orientada a objetos, tem se tornado cada vez mais utilizada em projetos de médio à grande porte. Devido aos avanços nas ferramentas de desenvolvimento, a introdução de mecanismos de refatoração automática tem auxiliado na produção de código mais legível, simples e com melhor desempenho, visto que a proposta é alterar o código-fonte sem modificar o seu comportamento.

Entretanto, a partir desta refatoração automática, o projeto pode introduzir erros, sejam de sintaxe ou semântica, os quais não existiam antes da aplicação da refatoração, com isso o objetivo do projeto é utilizar programas gerados de forma aleatória para testar os principais mecanismos de refatoração da linguagem Java, testando assim a confiabilidade das ferramentas. Sendo assim, é necessário garantir que as alterações de código realizadas de forma automática mantenham as propriedades definidas inicialmente. Esta necessidade vem ao encontro do que se propõe neste projeto, sendo a realização de testes desses mecanismos de refatoração a partir da execução de milhares de programas ou trechos de código gerados de forma automática e aleatória.

Para possibilitar o desenvolvimento deste projeto, se faz necessário cumprir diversas etapas:

*Este trabalho encontra-se em fase de desenvolvimento.

- Identificação das principais refatorações aplicadas e os mecanismos mais utilizados.
- Criação de um mecanismo de geração de código Java que permita testar as refatorações.
- Execução de testes dos mecanismos de refatoração utilizando os códigos gerados de forma aleatória.

Para viabilizar o avanço deste projeto, foi desenvolvido um gerador de códigos Java, que permite a geração de acesso a atributos, invocação de métodos, criação de objetos, conversor de tipos, além de algumas diretivas de controle de fluxo (condicionais e repetição). Para tal, foram utilizadas as ferramentas *Java Reflection*[1], que obtém informações sobre código Java; a biblioteca *JavaParser*[2], que manipula os dados de construtores sintáticos e exportar código real; e a biblioteca *JQWik*[3], que dispõe da geração para construções básicas e permite a execução dos testes com as propriedades definidas.

O restante deste texto está organizado da seguinte forma: a seção 2 apresenta os trabalhos relacionados. As seções 3 e 4 apresentam os mecanismos de refatoração e a geração de código aleatório, respectivamente. Na seção 5 são apresentados os resultados parciais. Finalmente, a seção 6 encerra este documento com os resultados esperados e contribuições.

2. Trabalhos Relacionados

A ferramenta de teste de refatoração JDolly [Soares et al. 2013] com o mecanismo do SafeRefactor [Soares et al. 2009] são instrumentos que propõem uma técnica para testar os motores de refatoração da linguagem Java. A diferença entre a técnica utilizada no projeto citado e o presente projeto é a possibilidade de trabalhar com construções inseridas em versões mais recentes da linguagem Java, como, por exemplo, expressões lambda, referências para métodos, etc.

O trabalho de Kraus, Coelho e Feitosa [Kraus et al. 2021] apresenta uma arquitetura para a geração de código aleatório em Java, com o intuito de aplicar testes baseados em propriedades para verificar inconsistências em APIs. No presente artigo, é proposto a utilização do mesmo gerador de códigos aleatórios para a realização de testes dos mecanismos de refatoração presentes nas principais IDEs de desenvolvimento Java. Além destes, existem também diversos outros métodos e mecanismos para geração de código, como, por exemplo, a ferramenta de teste Csmith [Yang et al. 2011], sendo um gerador de programas para a linguagem C, onde suporta praticamente toda a geração dos recursos da linguagem.

3. Mecanismos de Refatoração Automática

Mecanismos de refatoração automática permitem a mudança de código sem alteração de comportamento, ou seja, ao aplicar uma refatoração, o programa deve executar apresentando o mesmo resultado. Essas ferramentas são muito utilizadas em Java, pois permite aos programadores uma melhor utilização dos conceitos implementados pela linguagem.

[1] `https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/package-summary.html`

[2] `www.javaparser.org`

[3] `www.jqwik.com`

As principais IDEs (Ambiente de Desenvolvimento Integrado) da linguagem Java, como o IntelliJ, Eclipse e Netbeans possuem mecanismos de refatoração integrados. Estas ferramentas ajudam em problemas como extrair métodos, mover métodos, extrair classes, encapsular atributos, etc., além de sugerir modificações para que se utilizem os novos conceitos introduzidos na linguagem, para manter um código base atualizada.

4. Geração de Código Aleatório

O uso de testes com códigos gerados aleatoriamente na área de linguagens de programação é realizado desde meados da década de 60 [Sauder 1962]. Estes testes utilizam uma série de trechos de código como entrada para a checagem de diferentes condições que o sistema deve respeitar.

Como parte dos objetivos deste projeto, encontra-se em fase avançada de desenvolvimento um gerador de código aleatório Java. O trecho de código apresentado a seguir apresenta parte do método *genExpression*, o qual é responsável por gerar diferentes expressões de Java de forma recursiva.

```
@Provide
public Arbitrary<Expression> genExpression(Type t) {
    List<Arbitrary<Expression>> cand = new ArrayList<>();
    // Verifica se existem candidatos de tipos Primitivos
    if (t.isPrimitiveType()) {
        cand.add(Arbitraries.oneOf(mBase.genPrimitiveType(
        t.asPrimitiveType())));
    }
    ...
    //Verifica se existem candidatos de Methods
    if (!mCT.getCandidateMethods(t.asString()).isEmpty()) {
        cand.add(Arbitraries.oneOf(genMethodInvokation(t)));
    }
     return Arbitraries.oneOf(cand);
}
```

Figura 1. Código em Java: Geração de Código Aleatório

Como pode ser notado, o método *genExpression* recebe como parâmetro um tipo válido, que pode ser tanto de tipos primitivos como classes ou interfaces pré-definidas, resultando em possíveis expressões que devem ser do tipo solicitado. No trecho de código apresentado há duas condições: na primeira é verificado se o tipo a ser gerado é primitivo, o qual chama um método específico para proceder com a geração, adicionando o resultado em uma lista de expressões candidatas. De forma similar, na segunda condição é verificado se existe algum método na base de classes / interfaces existentes que possui tipo de retorno igual ao recebido por parâmetro, também adicionando as possíveis expressões candidatas na lista resultante.

5. Resultados Parciais

Para o desenvolvimento deste projeto, a partir das tecnologias escolhidas, foi definida a arquitetura apresentada na Figura 1.

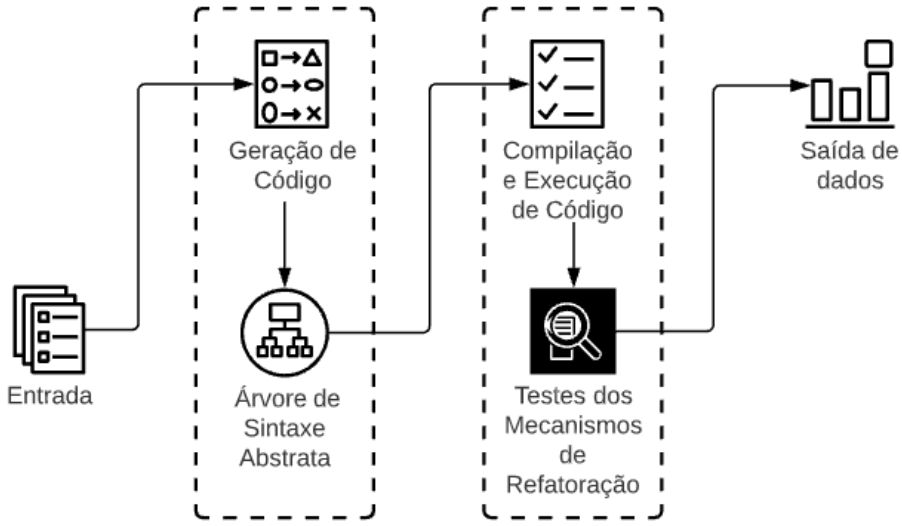

Figura 2. Arquitetura para implementação do projeto

A arquitetura proposta prevê a utilização de dois componentes principais: (1) o mecanismo gerador de código Java; e (2) a ferramenta que executa os códigos gerados antes e depois da refatoração. A execução é iniciada a partir da informação das entradas, que descreve as restrições que o gerador precisa respeitar para gerar testes válidos para a refatoração escolhida. Estas informações são repassadas para o módulo gerador (*JR-Generator*), que ao final, produz uma árvore de sintaxe abstrata contendo o caso de teste gerado, a qual será compilada para código Java, e inserida como entrada para a ferramenta de teste. Esta ferramenta, compila e executa o código gerado, coletando e armazenando seus possíveis resultados. Na sequência, é aplicada a refatoração escolhida no código original, repetindo o processo de compilação, execução e coleta de resultados. Ao final, os dados coletados são comparados para verificar possíveis inconsistências (erro de compilação ou execução) a partir dos códigos refatorados. Como saída, o sistema produz uma listagem contendo as estatísticas da execução.

A metodologia proposta para este projeto é dividida em cinco fases, conforme descrito a seguir.

- Fase 1: pesquisa a respeito do aparato ferramental para o desenvolvimento do projeto.
- Fase 2: definição de bibliotecas e *frameworks* que possibilitem a geração de código Java.
- Fase 3: criação do mecanismo de geração de código aleatório em Java.
- Fase 4: criação do instrumento que permita os testes de refatoração a partir dos códigos gerados.
- Fase 5: aplicar o instrumento de testes nas principais ferramentas de refatoração da linguagem Java.

Atualmente, o projeto encontra-se na Fase 3, possuindo o mecanismo de geração de código praticamente finalizado, dispondo da geração de tipos primitivos, acesso a atributos e métodos, construtores, *casts*, diretivas, etc. Na sequência serão realizadas

adaptações no mecanismo gerador, para permitir a entrada para o instrumento de teste, que deve ser implementado na sequência, possibilitando assim a execução dos testes dos mecanismos de refatoração.

6. Resultados Esperados e Contribuições

O escopo do projeto consiste no estudo dos mecanismos de refatoração automática da linguagem Java, e a realização de testes destas ferramentas a partir da execução de casos de teste gerados de forma aleatória. É sabido que através da utilização de casos de teste bem estruturados é possível detectar *bugs* em projetos de software de forma antecipada, evitando a liberação de ferramentas de desenvolvimento que possam introduzir problemas em projetos futuros.

Esta proposta pretende contribuir para o avanço desta área de pesquisa, fornecendo uma especificação formal (através de semântica operacional) e a implementação do mecanismo de geração de código utilizando a linguagem Java, além do instrumento de testes descrito neste artigo. Após o encerramento deste projeto, pretende-se ter uma ferramenta de software livre que permita agilizar o processo de testes e aumentar a confiabilidade das principais ferramentas de refatoração da linguagem Java.

Referências

Cass, S. (2019). As principais linguagens de programação de 2019. IEEE Spec.

Kraus, L. F., Schafaschek, B., and Feitosa, S. d. S. (2021). Desenvolvimento de um gerador de programas aleatórios em java. *Anais do Computer on the Beach*, 12(0):485–487.

Sauder, R. L. (1962). A general test data generator for cobol. In *Proceedings of the May 1-3, 1962, Spring Joint Computer Conference*, AIEE-IRE '62 (Spring), page 317–323, New York, NY, USA. Association for Computing Machinery.

Soares, G., Cavalcanti, D., Gheyi, R., Massoni, T., Serey, D., and Cornélio, M. (2009). Safe refactor: tool for checking refactoring safety.

Soares, G., Gheyi, R., and Massoni, T. (2013). Automated behavioral testing of refactoring engines. *IEEE Transactions on Software Engineering*, 39(2):147–162.

Yang, X., Chen, Y., Eide, E., and Regehr, J. (2011). Finding and understanding bugs in C compilers. *SIGPLAN Not.*, 46(6):283–294.