Estudo de Ferramentas de Análise de Código

Engenharia Gramatical

Afonso Ferreira | PG52669
Fernando Alves | PG54470
Joana Pereira | PG53895

11 de março de 2024

1 Introdução

Neste trabalho prático, irá ser explorado o tema da *Refatorização* no contexto do desenvolvimento de software.

A refatorização é uma prática crucial para melhorar a estrutura, legibilidade e capacidade de manutenção do código-fonte, sem modificar seu comportamento. Essa abordagem torna-se essencial ao enfrentar a constante evolução de projetos, facilitando a incorporação de novos requisitos e a correção de problemas.

Este trabalho tem como objetivo aprofundar o conhecimento sobre ferramentas de refatoração, compreender a sua importância no contexto do desenvolvimento de software, determinar as situações em que a sua aplicação é adequada e pertinente, e analisar e comparar as diferentes funcionalidades disponíveis em diversas ferramentas.

Para isso, pesquisamos por ferramentas de refatorização, recolhendo o máximo de informação bibliográfica disponível acerca do funcionamento das ferramentas. O grupo irá focar-se em ferramentas específicas para as linguagens Python, Java e C.

2 Fundamentos da Refatorização

2.1 O que é?

Refatorizar é uma abordagem estruturada para melhorar o design do software. Envolve a reorganização cuidadosa do código existente para torná-lo mais eficiente, compreensível e adaptável. Ao contrário da simples correção de *bugs*, a refatorização visa melhorar significativamente o código sem introduzir novas funcionalidades. Com a refatorização, podemos transformar um design fraco e mal estruturado num código bem estruturado, limpo e com design simples. Assim os principais objetivos da refatorização é tornar o código:

- Claro e compreensível;
- Com ausência de duplicação;
- Curto, conciso e simples;

- Aprovado em todos os testes;
- Com baixo custo e fácil de manutenção.

2.2 Onde utilizar refatorização

A prática da refatorização surge como resposta à identificação de *code smells*, que são indicadores de potenciais problemas no design ou implementação do software. Esses indícios apontam áreas específicas do código que podem beneficiar da refatorização, de modo a melhorar a qualidade e legibilidade desse código. Alguns exemplo de *code smells*:

- **Bloaters** (Elementos que tornam o código com grandes proporções e difícil de gerenciar)
  - → Long Method - métodos extensos levando a complexidade do código e difícil leitura;
  - → Large Class - classes com muita informação e difícil de gerir;
  - → Primitive Obsession - excesso de dependência de tipos;
  - → Long Parameter List - métodos ou funções com um número excessivo de parâmetros;
  - → Data Clumps - Grupos de variáveis idênticos em diferentes partes do código
- **Object-Orientation Abusers** (Práticas incorretas de programação orientada a objetos)
  - → Alternative Classes with Different Interfaces - classes idênticas funcionalmente mas com diferentes nomes;
  - → Refused Bequest - mal uso de herança ;
  - → Switch Statements - excesso do use de *switch* ou *if* ;
  - → Temporary Field - uso de campos temporários.
- **Change Preventers** (Padrões de código que dificultam mudanças)
  - → Divergent Change - mudanças numa dada classe que obrigam a mudanças em muitos métodos;
  - → Parallel Inheritance Hierarchies - quando uma subclasse de uma classe A é conectada a uma subclasses de uma classe B, tornando o código mais complexo e difícil de manter;
  - → Shotgun Surgery - para implementar uma mudança são necessárias fazer muitas alterações em diferentes classes (one-to-many).
- **Dispensables** (Elementos desnecessários)
  - → Comments - comentários desnecessários (*The best comment is a good name for a method or class*);
  - → Duplicate Code - dois fragmentos do código muito semelhante;
  - → Data Class - classes que contêm apenas campos e métodos básicos;
  - → Dead Code - código que já não é utilizado;
  - → Lazy Class - classes pouco usadas e que não justifica a sua existência;
  - → Speculative Generality - código criado para implentações futuras mas que não está a ser usado;
- **Couplers** (Dependências fortes e complexas entre classes)
  - → Feature Envy - quando uma classe utiliza excessivamente métodos de outra classe;

→ Inappropriate Intimacy - quando duas classes estão fortemente vinculadas entre si;
→ Incomplete Library Class - classes de biblioteca incompletas;
→ Message Chains - quando uma classe A, para obter dados de uma classe D, precisa acessar classes intermédias;
→ Middle Man - existência de classes principalmente responsáveis apenas pela delegação.

2.3 Quando utilizar refatorização

- **Regra dos Três**

  → A primeira vez que uma tarefa é realizada, é suficiente concluí-la.
  → Na segunda vez que uma tarefa similar é realizada, pode-se sentir um certo desconforto com a necessidade de repetir, mas ainda assim, a ação é realizada.
  → Na terceira vez que uma tarefa é realizada, é o momento de começar a refatorizar.

- **Ao adicionar uma funcionalidade**

  → Refatorizar facilita a compreensão do código de outros desenvolvedores. Quando se lida com código de outra pessoa, tentar refatorizá-lo primeiro pode ser benéfico. Código limpo é mais fácil de compreender. Isso não só beneficiará o indivíduo que realiza a refatorização, como também melhorará a experiência de quem usar o código posteriormente.
  → Refatorizar torna mais fácil a inclusão de novas funcionalidades. É mais fácil fazer alterações num código limpo.

- **Ao corrigir um *bug***

  → *Bugs* no código têm comportamento semelhante aos problemas da vida real: eles tendem a esconder-se nos lugares mais obscuros e complexos do código. Limpar o código pode tornar os erros mais evidentes.
  → A refatorização proativa é valorizada, pois reduz a necessidade de tarefas de refatorização adicionais no futuro.

- **Durante uma revisão do código**

  → A revisão de código pode ser a última oportunidade de organizar o código antes de torná-lo disponível ao público.
  → É recomendável realizar essas revisões juntamente com o autor do código. Deste modo, problemas simples podem ser corrigidos rapidamente e o tempo necessário para corrigir problemas mais complexos pode ser estimado.

2.4 Técnicas de refatorização

Entre as diversas técnicas de refatorização, destacam-se seis categorias gerais, cada uma com objetivos específicos para tornar o código mais eficiente e compreensível. Vamos explorar de seguida essas categorias:

- **Composing Methods** (Compondo Métodos):

  Simplifica métodos excessivamente longos e remove duplicações de código para facilitar futuras melhorias.

- **Moving Features between Objects** (Movendo Recursos entre Objetos):

  Facilita a transferência de funcionalidades entre classes, criando novas classes e ocultando detalhes de implementação do acesso ao público.

- **Organizing Data** (Organizando Dados):

  Ajuda na manipulação de dados, substituindo tipos primitivos por funcionalidades de classe mais complexas, simplificando associações de classes para torná-las mais flexiveis e reutilizáveis.

- **Simplifying Conditional Expressions** (Simplificando Expressões Condicionais):

  Lida com a crescente complexidade das expressões condicionais, oferecendo técnicas para consolidar, decompor ou substituir essas expressões.

- **Simplifying Method Calls** (Simplificando Chamadas de Métodos):

  Simplifica chamadas de métodos, tornando as interfaces de interação entre classes mais simples e compreensíveis.

- **Dealing with Generalization** (Lidando com Generalizações):

  Trata de questões relacionadas à herança, criando novas classes, movendo campos e métodos entre classes pai e filha, e substituindo herança por delegação e vice-versa.

Ao compreender os fundamentos da refatorização, é possível implementar práticas mais robustas de desenvolvimento de software, resultando em sistemas mais eficientes e fáceis de manter. A próxima etapa deste trabalho prático será a análise das ferramentas disponíveis para auxiliar neste processo, focando nas linguagens Python, Java e C.

3 Ferramentas

Foram selecionados três *refactors* desenvolvidos pela JetBrains, para três linguagens de programação distintas - Python, C e Java.

3.1 Python - PyCharm

O PyCharm é uma ferramenta de desenvolvimento integrado (IDE) altamente conceituada para Python, oferecendo uma vasta seleção de funcionalidades de refatorização, como renomear, extrair, mover e *inline* elementos de código, com o objetivo de otimizar a estrutura do código, torná-lo mais legível e facilitar a manutenção. Essas operações de refatorização podem ser realizadas através de uma interface amigável, permitindo que os desenvolvedores visualizem e apliquem as mudanças de forma eficiente e segura.

3.2 C - CLion

O CLion é um (IDE) para C (e C++) desenvolvido pela JetBrains, que oferece uma ampla gama de funcionalidades de refatorização para ajudar os desenvolvedores a manter a qualidade do código, melhorar a legibilidade e facilitar a manutenção. Através do CLion, os desenvolvedores podem realizar refatorizações como exclusão segura, cópia/movimentação de elementos, extração de métodos, introdução de variáveis, renomeação, inlining, e alteração de assinatura, entre outras, de forma eficiente e segura. Além disso, o CLion permite a visualização prévia das alterações antes de aplicá-las, oferecendo uma ferramenta poderosa para gerenciar conflitos que possam surgir durante o processo de refatorização. Essas funcionalidades, juntamente com a capacidade de gerar código automaticamente e implementar, substituir ou gerar definições de funções, tornam o CLion uma ferramenta valiosa para desenvolvedores que trabalham com C (e C++).

3.3 JAVA - IntelliJ IDEA

O IntelliJ IDEA é um IDE para Java, conhecido por oferecer uma vasta gama de funcionalidades de refatorização como parte do seu conjunto completo de recursos. Através da refatorização Java do IntelliJ IDEA, os desenvolvedores podem simplificar, tornar o código mais legível e facilitar a manutenção de forma segura e eficaz, com automatização e intervenção manual mínima.

4 Comparação Ferramentas

Decidiu-se comparar as ferramentas anteriores, desenvolvidas pela JetBrains, com outras desenvolvidas pelos concorrentes.

4.1 PyCharm vs Rope / Sourcery

Começando pela refatorização em Python, iremos ver as diferenças entre o *refactor* integrado no IDE PyCharm e outras ferramentas, como os *refactors* Rope e Sourcery.

Rope é uma biblioteca Python open-source para refatorizar código. Em termos de funcionalidades, no que toca à refatorização, é semelhante ao PyCharm, sendo possível renomear, extrair métodos, introduzir variáveis, entre outras. Diverge, contudo, no âmbito. PyCharm, é capaz de muito mais para além de refatorização, visto que é um IDE completo, ao invés que o Rope está dependente da utilização de um IDE. Para alguns isto pode ser visto como um benefício, para outros, uma limitação.

Os benefícios de utilizar Rope é por este ser eficiente, leve, open-source e personalizável às necessidades de cada um. Além disso, Rope é gratuito. PyCharm tem uma versão gratuita, mas também uma versão paga. No entanto, PyCharm vai mais além, fornecendo uma gama mais ampla de operações de refatorização seguras e inteligentes, juntamente com uma análise

de código contextual e feedback visual para facilitar o processo de refatorização em projetos de maior escala.

Sourcery é um assistente inteligente de análise de código e refatorização. O maior ponto forte deste assistente está na automatização da refatorização. Oferece uma vasta gama de sugestões, incluindo a renomeação de variáveis, simplificação de código e melhoramento de estrutura de código. Existe uma versão gratuita apenas para repositórios públicos, sendo necessário pagar para poder utilizar livremente a ferramenta. Claro que para cada sugestão pode ser necessária uma revisão e aceitação por parte do programador. Além disso, a análise é feita por Inteligência Artificial, que pode nem sempre ser perfeita.

4.2 CLion vs Visual Assist

Para a linguagem de programação C, foi selecionada a ferramenta Visual Assist para comparar com a integrada no IDE CLion. Visual Assist é um plugin para qualquer Microsoft IDE (e.g. Visual Studio Code) com bastantes funcionalidades, entre elas, refatorização. Além da refatorização, é capaz de analisar e completar código e verificar erros. É mais acessível, visto que não é difícil aprender a utilizar, enquanto que o CLion é mais complexo. Mas com complexidade, surgem benefícios. Entre eles, um maior leque de refatorizações, bem como uma melhor experiência de programação, em geral. A única desvantagem do CLion é a necessidade de pagar para usufruir deste IDE.

4.3 Intellij IDEA vs JDT Refactoring

Por último, na linguagem de programação Java, decidimos comparar *refactors* integrados nos IDEs IntelliJ IDEA e Eclipse. Ambos têm as funcionalidades básicas de refatorização. No entanto, no IDE da JetBrains existem funcionalidades mais avançadas, de reestruturação do código (e.g. Introduce Local Variable, Introduce Parameter Object, Convert Anonymous Class, Safe Delete). Tendo isto em conta, chegou-se à conclusão que, para refatorizações básicas em Java, o IDE Eclipse com a sua ferramenta integrada de refatorização é um bom ponto de partida. E, como é gratuito, bastante acessível. No entanto, para refatorizações mais avançadas, facilidade de uso e suporte para outras linguagens de programação, o IntelliJ IDEA é a melhor escolha, ainda que tenha custos associados.

Abaixo temos uma tabela que compara brevemente as várias ferramentas que até aqui foram analisadas, demonstrando as funcionalidades principais de cada uma.

Tabela 1 – Comparação de Ferramentas de refatorização

Ferramenta	**Linguagem**	**Funcionalidades Principais**	**Integração**
PyCharm	Python	Renomeação, extração de métodos, introdução de Variáveis, Safe Delete	JetBrains PyCharm
Rope	Python	Renomeação, extração de métodos, reorganização de código	Integração com IDEs (VSCode, PyCharm)
Sourcery	Python	Análise de código, refatorização automática	-
CLion	C/C++	Renomeação, extração de código, inline, changing function signatures	JetBrains CLion
Visual Assist	C/C#/C++	Renomeação, extração de código, mover funções, sugestões de melhorias	IDEs da Microsoft
IntelliJ IDEA	Java/Kotlin	Refatorização automática, sugestões de melhoria	JetBrains IntelliJ IDEA
JDT Refactoring	Java	Renomeação, extração, inline, introdução de variável	Eclipse

5 Conclusão

Com este trabalho prático foi possível perceber a importância da refatorização no desenvolvimento de software, entender os seus fundamentos, identificando onde e como aplicá-la. Foram ainda analisadas ferramentas específicas para as linguagens Python, C e Java, fornecendo uma comparação entre as soluções da JetBrains e algumas alternativas dos concorrentes.

Ao analisar as diversas ferramentas e as suas funcionalidades, foi possível concluir que a escolha de uma em detrimento de outra não é tão evidente como se pensava, dependendo das necessidades específicas do projeto e das preferências do desenvolvedor. Em projetos mais complexos, a utilização do PyCharm, CLion ou IntelliJ IDEA pareceu ser mais apropriada, enquanto que para projetos mais simples, a escolha por ferramentas como Rope, Visual Assist ou JDT Refactoring pode ser mais adequada.

Em suma, a refatorização é uma prática essencial para garantir a qualidade, legibilidade e capacidade de manutenção do código ao longo do tempo. A escolha de ferramentas de refatorização adequadas podem impactar significativamente a produtividade do programador e a qualidade do software. Através da análise das ferramentas disponíveis, da compreensão das suas funcionalidades e da avaliação das necessidades dos projetos, é possível selecionar a solução ideal para otimizar o processo de refatorização dos mesmos.

Referências

[1] Clion. https://www.jetbrains.com/help/clion/refactoring-source-code.html.

[2] Code smells. https://code-smells.com/couplers/middle-man.

[3] Documentation sourcery. https://docs.sourcery.ai.

[4] Eclipse. https://archive.eclipse.org/eclipse/downloads/drops/R-2.1-200303272130/whats-new-jdt-refactor.html.

[5] Intellij idea. https://www.jetbrains.com/help/idea/refactoring-source-code.html.

[6] Java refactoring tools. https://www.developer.com/java/best-java-refactoring-tools/.

[7] Pycharm. https://www.jetbrains.com/help/pycharm/refactoring-source-code.html.

[8] Python refactoring tools. https://www.developer.com/languages/python/best-python-refactoring-tools/.

[9] Refactoring. https://refactoring.com/.

[10] Refactoring for c/c++ and c#: Let the machine do the work. https://www.codeproject.com/Articles/15768/Refactoring-for-C-C-and-C-Let-the-Machine-do-the-W.

[11] Refactoring freecodecamp. https://www.freecodecamp.org/news/best-practices-for-refactoring-code/.

[12] Refactoring guru. https://refactoring.guru/.

[13] Refactoring techtarget. https://www.techtarget.com/searchapparchitecture/definition/refactoring.

[14] Rope. https://rope.readthedocs.io/en/latest/overview.html#refactorings.

[15] Sourcery. https://sourcery.ai/https://sourcery.ai/.

[16] Types of refactoring. https://stepsize.com/blog/the-ultimate-engineers-guide-to-refactoring.

[17] Visual assist. https://www.wholetomato.com/s.

[18] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Object Technology Series. Addison-Wesley Professional, 2nd edition, 2018.