Ciência de Dados e I.A.
Escola de Matemática Aplicada
Fundação Getúlio Vargas

Engenharia de Requisitos

TCC

Reverse Engineering of Source Code to Sequence Diagram Using Abstract Syntax Tree

Aluno: Isabela Yabe
Orientador: Rafael de Pinho André
Escola de Matemática Aplicada, FGV/EMAp
Rio de Janeiro - RJ.

Rio de Janeiro, 2025

1 Revisão literária

Artigo revisado Fauzi, Hendradjaya e Sunindyo (2016):

A revisão tem o objetivo de compreender o estado da arte das abordagens de engenharia reversa que partem de código-fonte e produzem artefatos de alto nível, como diagramas UML. Para garantir uma análise sistemática e comparável entre diferentes propostas, foram definidas perguntas de pesquisa (*Research Questions — RQs*) que orientam a coleta e síntese dos dados extraídos dos estudos selecionados.

- **RQ1.** Em quais linguagens e domínios as abordagens que partem de código-fonte foram aplicadas?
- **RQ2.** Quais modelos/artefatos de alto nível são gerados?
- **RQ3.** Qual aspecto é privilegiado (estático, dinâmico, híbrido) e com qual objetivo (compreensão, redocumentação, migração, qualidade)?
- **RQ4.** Quais técnicas e transformações viabilizam a passagem do código para o modelo de alto nível?
- **RQ5.** Quais ferramentas/frameworks são utilizados?
- **RQ6.** Como as abordagens são validadas e com que qualidade prática?

2 RQ1. Em quais linguagens e domínios as abordagens que partem de código-fonte foram aplicadas?

A abordagem REVUML foi aplicada ao domínio de **sistemas orientados a objetos desenvolvidos em Java**. O método realiza a engenharia reversa diretamente sobre o código-fonte Java, utilizando a estrutura *Abstract Syntax Tree* (AST) para gerar representações comportamentais de alto nível no formato de diagramas de sequência UML.

A ferramenta proposta, denominada REVUML, foi implementada e validada por meio de experimentos que utilizaram 126 casos de teste distintos em Java, abrangendo múltiplos conceitos da orientação a objetos, como herança, polimorfismo, classes internas, métodos estáticos, expressões *lambda* e abstração.

3 RQ2. Quais modelos/artefatos de alto nível são gerados?

O trabalho tem como objetivo gerar **diagramas de sequência UML** a partir de código-fonte Java, constituindo um modelo comportamental de alto nível obtido por meio de engenharia reversa. O diagrama é produzido automaticamente após a análise da *Abstract Syntax Tree* (AST), que representa a estrutura sintática do código.

O diagrama gerado descreve a **interação entre objetos e chamadas de métodos**, incluindo construções comportamentais como laços, condicionais e retornos.

A ferramenta REVUML implementa todo o fluxo de engenharia reversa, desde a extração do código e construção da AST até a geração do diagrama final, organizando os resultados visuais de forma automatizada

4 RQ3. Qual aspecto é privilegiado (estático, dinâmico, híbrido) e com qual objetivo (compreensão, redocumentação, migração, qualidade)?

O trabalho de Fauzi, Hendradjaya e Sunindyo (2016) privilegia o **aspecto estático** do código-fonte, pois toda a geração do diagrama de sequência ocorre a partir da *Abstract Syntax Tree* (AST) obtida por análise léxica e sintática do código Java, sem execução do sistema. Trata-se, portanto, de uma abordagem de **análise estática voltada à reconstrução de comportamento dinâmico**, cujo objetivo principal é a **compreensão e redocumentação do sistema**.

5 RQ4. Quais técnicas e transformações viabilizam a passagem do código para o modelo de alto nível?

REVUML baseia-se na aplicação do modelo conceitual **Extract–Abstract–Present (EAP)**, em que o código-fonte é extraído, abstraído e finalmente apresentado como um diagrama UML. O processo utiliza a estrutura **Abstract Syntax Tree (AST)** representar a hierarquia e as relações sintáticas do código Java.

Durante a etapa de *extração*, o código é analisado lexical e sintaticamente por meio da API **JavaParser**, que gera a árvore sintática abstrata em memória.

Na fase de *abstração*, os nós da AST são percorridos e analisados para identificar elementos relevantes , como classes, métodos, chamadas, laços, condicionais, herança e polimorfismo. Essa análise é realizada por meio de um **algoritmo de travessia DFS pós-ordem (Depth-First Search Post-Order)**, que garante a ordenação correta das instruções conforme a sequência real do código-fonte.

Durante essa travessia, são aplicadas operações de registro de variáveis, rastreamento de chamadas de métodos, criação de objetos e identificação de fragmentos combinados (*alt*, *loop*) para representar estruturas condicionais e iterativas no diagrama.

Por fim, na etapa de *apresentação*, os dados abstraídos são convertidos em representações visuais por meio da biblioteca **PlantUML**, que gera automaticamente as imagens do diagrama de sequência.

6 RQ5. Quais ferramentas/frameworks são utilizados?

A implementação da abordagem proposta por Fauzi, Hendradjaya e Sunindyo (2016) foi realizada com o auxílio de duas bibliotecas principais: o **JavaParser API** e o **PlantUML API**.

O **JavaParser** é responsável pela análise léxica e sintática do código-fonte Java e pela construção da *Abstract Syntax Tree (AST)*, que serve de base para as transformações descritas na RQ4.

Já o **PlantUML** é empregado na etapa final para converter os dados abstraídos em representações visuais de diagramas de sequência UML, nos formatos PNG ou SVG.

A integração entre essas ferramentas constitui um diferencial da REVUML, pois permite uma conversão direta do código-fonte em diagramas UML, sem necessidade de modelos intermediários em XML, tornando o processo mais ágil e automatizado.

6.1 Polimorfismo

O polimorfismo é tratado durante a fase de *abstração*, quando a ferramenta percorre a *Abstract Syntax Tree* (AST). Nessa etapa, o algoritmo realiza o registro e o rastreamento de variáveis para identificar possíveis mudanças de tipo dinâmico, uma vez que, em Java, o tipo de um objeto pode variar conforme as atribuições realizadas.

A REVUML detecta e mantém o vínculo entre variáveis, métodos e objetos, de modo a refletir corretamente, no diagrama de sequência, o tipo efetivo do objeto que executa cada método, e não apenas o tipo declarado. Por exemplo, quando uma variável declarada como `Animal` recebe uma instância de `Cat`, o sistema reconhece que a chamada deve ser associada ao método da subclasse.

Durante essa análise, a ferramenta gera representações explícitas tanto das subclasses quanto das superclasses, exibindo no diagrama as relações de herança correspondentes. Essa representação simultânea preserva a semântica hierárquica e facilita a compreensão da estrutura de chamadas entre objetos relacionados por herança.

De forma resumida, a abordagem lida com o polimorfismo por meio de três mecanismos complementares: (i) rastreamento de variáveis na AST para detectar alterações de tipo em atribuições; (ii) representação conjunta de subclasses e superclasses para manter a coerência estrutural; e (iii) resolução estática das chamadas herdadas, exibindo corretamente o método original da superclasse (por exemplo, `Animal.setName()` chamado por `Cat`).

7 RQ6. Como as abordagens são validadas e com que qualidade prática?

A abordagem proposta foi validada por meio de um **estudo experimental** que avaliou a ferramenta **REVUML** em **126 casos de teste**. O experimento teve como

objetivo verificar a precisão da engenharia reversa de código Java em diagramas de sequência UML, considerando diferentes estruturas da orientação a objetos.

Os resultados mostraram que a REVUML é capaz de gerar diagramas corretos e consistentes, representando com precisão chamadas recursivas, ciclos de execução e interações entre classes e subclasses. Além disso, o algoritmo demonstrou robustez na detecção de dependências complexas e recursivas, mantendo a ordem lógica das instruções conforme o fluxo real do código.

De modo geral, os autores concluem que a abordagem apresenta **boa aplicabilidade prática**, uma vez que a geração automatizada dos diagramas reduz significativamente o esforço manual de documentação e facilita a compreensão do comportamento dinâmico do software, especialmente em sistemas legados sem documentação atualizada.

Autores / Referência	**Linguagem / Domínio**	**Modelo Gerado**	**Aspecto**	**Técnica / Transformação**	**Ferramenta / Framework**	**Validação / Estudo de Caso**
Fauzi, Hendradjaya e Sunindyo (2016)	Java; sistemas orientados a objetos	UML Sequência (comportamental)	Estático; Compreensão/Redocumentação	**Código → AST (JavaParser) → DFS pós-ordem → PlantUML (Seq)**	REVUML; JavaParser; PlantUML	126 casos de teste (8 categorias: herança, polimorfismo, laços, recursão, métodos estáticos); geração correta e consistente de diagramas

Tabela 1: Resumo das abordagens

Referências

FAUZI, E.; HENDRADJAYA, B.; SUNINDYO, W. D. Reverse Engineering of Source Code to Sequence Diagram Using Abstract Syntax Tree. *In:* 2016 International Conference on Data and Software Engineering (ICoDSE). Denpasar, Indonesia: IEEE, 2016. p. 1–6.