Aluno(a): Hugo Rodrigues Torquato
Orientador(a): Heinrich Da Solidade Santos

Curso: MBA em Engenharia de Software

Analisador de software C++ baseados em programação orientado a objetos

Introdução

No contexto de programação orientada a objetos, as abstrações são fundamentais para organização e atribuição de responsabilidades. A compreensão da lógica é simples quando assuntos comumente conhecidos e/ou com poucos níveis de abstrações. Entretanto, o código se torna mais complexo e, por consequência, demanda mais esforço e tempo para compreendê-lo quando se trata de uma base de código extensa, desenvolvida durante vários anos, por vários profissionais e com foco em um conhecimento específico. Em uma ótica empresarial, este cenário implica em maiores investimentos para que novos engenheiros sejam incorporados à equipe por causa da complexidade, e adiciona um nível de abstração extra nas tomadas de decisão relacionados ao desenvolvimento do software por não ter uma visão geral do todo. Sob essa perspectiva, visualizar a organização de uma base de código clara e intuitivamente agrega valor, principalmente, na rotina das empresas desenvolvedoras de softwares.

A visualização de informações de um software auxilia o gerenciamento da base de código, o que, de forma secundária, reflete na maior mantenabilidade do produto. O livro Clean Code ilustra essa dinâmica ao comparar produtividade pelo tempo de existência do software: ao passo que a extensão e complexidade do código aumentam, especialmente sem planejamento, as chances de se transformar em um emaranhado de soluções improvisadas também aumentam. Nesse cenário, o autor argumenta que a produtividade para desenvolver novas funcionalidades e na manutenção do código declina drasticamente a médio e longo prazo, tornando-se quase impraticável e economicamente inviável.

O ponto de partida em projetos dessa natureza é a análise de código fonte, que é tema de estudos, a bastante tempo, para diversas áreas na computação: Otimização de compilador, como abordado por [1], que expõe benefícios ao realizar análises de hierarquia de classes como forma de resolver ambiguidades em chamadas de funções virtuais; Análises de qualidade de software, como abordado por [2], em alguns dos problemas frequentes na programação orientada a objetos; E até mesmo avaliar o quão eficiente os modelos inteligência artificial baseados em linguagens (LLMs) estão quando comparado programação funcional com orientada a objetos, abordado por [3], e que ressalta o quando precisa ser melhorado quando avariada especificidades ou em larga escala.

Das diversas aplicabilidades, soluções e produtos foram propostos com foco em prover uma melhor experiência durante e após o desenvolvimento de um software. As

ferramentas propostas podem ser introduzidas em diversos momentos do desenvolvimento de software, seja na etapa de codificação ou validação do que foi criado. [4] aborda em detalhe as benefícios gerais desse tipo de produto mas como forma de exemplificar temos: O SonarQube, uma plataforma de análise código com foco em qualidade e cobertura de testes; O *Clang Static Analyzer* uma ferramenta de análise nativa do LLVM aplicada ao C e C++; e *AddressSanitizer* como ferramenta para identificar erros de memória em tempo de execução.

Frente ao exposto, ferramentas de visualização e organização de uma base de código são bastante requisitadas em projetos de software. Por consequência, é tema de pesquisas na academia e também no desenvolvimento de novos produtos e soluções no âmbito privado. O conteúdo mais abordado no referencial teórico sobre o tema, ressalta uma investigação que é feita em nível de compiladores e nos ambientes de desenvolvimento integrado (IDEs), como forma de auxiliar o programador na escrita e otimização da compilação de código. Entretanto, o mapeamento estruturado e exposição organizada do código para discussões e tomada de decisão não vem sendo amplamente abordado e/ou com relevantes estudos e produtos para cobrir essa necessidade.

Objetivo

Este trabalho tem como objetivo desenvolver uma API capaz de realizar uma análise estática em códigos baseados em programação orientada a objetos, extrair informação sobre a organização das classes e exportar os resultados de forma estruturada.

Metodologia ou Material e Métodos

O presente trabalho tem como base a análise de código que, pela literatura, pode ser feito tanto por uma abordagem estática ou dinâmica. Um comparativo é apresentado por [4], que ressalta os aspectos positivos e os reveses de cada estratégia. O autor conclui enfatizando o potencial positivo alcançado quando utilizadas em conjunto. No contexto do projeto de pesquisa proposto, a análise estática revela-se como opção depender somente do mais adequado o código fonte, sem a necessidade de comprá-lo. O que permite uma expansão para demais linguagens de forma mais concisa e avalia a presente intenção do desenvolvedor por não considerar nenhuma otimização feita pelo compilador, o que é benéfico para compreensão da real arquitetura do software projetada para o sistema.

A análise estática proposta tem um fluxo de desenvolvimento bastante alinhado com o desenvolvimento de um compilador, com foco principal até a parte da análise do código. O processo começa com um mapeamento dos caracteres de um código para, posteriormente,

organização das informações em forma de tokens. Esse grupamento é utilizado como entrada para um processo chamado *parsing*, que vai agrupar os tokens em expressões maiores viabilizando a construção das árvores de sintaxe. Estas sim são utilizadas para realizar o processo de análise do código de acordo com a especificidade de cada linguagem. As informações necessárias para o presente trabalho vão ser coletadas nessa etapa e tangem, principalmente, a árvore de sintaxe. Mas existem outras etapas em um projeto de compilador tão importantes quanto as apresentadas, [5] apresenta uma visão detalhada de todo o processo explicando todo o processo de criação de um interpretador. Sendo elas o processo de otimização do código fonte para posterior criação de um novo código, em linguagem mais próxima da máquina, para ser compilada de acordo com a escolha do ambiente em que o software vai rodar e como o código será executado.

Os livros [5], [6], e [7] vão ser utilizados como fontes principais no tema compiladores. Explorando as etapas que serão utilizadas no presente trabalho em detalhe, temos o mapeamento dos caracteres de um código como a primeira delas. Nesse momento, a performance é crucial, sendo o momento de maior carga por precisar passar por todos os caracteres de uma determinada entrada. Dentre as opções de definições que devem ser observadas temos palavras reservadas, números e estruturas de repetição. Todas essas opções têm de ser categorizadas para obter o agrupamento dos caracteres nos chamados tokens, um abstração de tipo de informações úteis para o interpretador. Em termos simples, é o menor grupo de caracteres que representem algo, sendo tratadas as ambiguidades. A identificação dos grupos é feita, dentre outras maneiras, por expressões regulares que identificam os padrões definidos na gramática léxica da linguagem de programação utilizada no código analisado.

Parser

análise do código

Implementação utilizando visitors (adição de tipos e de operações)

estrutura da API

Frente ao projeto proposto, é fundamental associar a implementação de testes junto ao desenvolvimento do projeto de forma a garantir sua qualidade e confiabilidade. Atrelado com a proposta do projeto de pesquisa é interessante desenvolver dois tipos de testes: unitários e de integração. Os testes unitários vão ser utilizados para validar cada componente da implementação de maneira isolado e explorando as particularidades da implementação. O framework escolhido para os testes unitários foi o google tests [Gtests], amplamente utilizado na indústria e com fácil acesso a informação, muito do que será implementado segue o padrão do *test-driven development* do livro [8], que aborda os conceitos com exemplos utilizando o google testes com C++. Apesar de fundamentais no desenvolvimento, os testes unitários não conseguem cobrir a completude do algoritmo,

nesse contexto os testes de integração são importantes por, de uma visão geral, avaliar se o software está comportando da maneira esperada. Este tipo de teste consiste em prover entradas já conhecidas para a aplicação e avaliar o retorno obtido com o esperado para aquela determinada entrada. No caso desse projeto, fornecer um código fonte com arquitetura de classes já conhecida e avaliar a resposta .json criara como resultado da aplicação. Essa combinação de testes explora a granularidade dos testes unitários com a abrangência dos testes de integração para garantir um software de melhor qualidade.

repositório git (pipeline?)

Resultados Esperados

É planejado obter, com a conclusão da pesquisa, uma API capaz de realizar uma análise estática em códigos C + + e retornar um conjunto de dados estruturados em formato .json.

Nesse primeiro momento, a análise estática está focada na hierarquia entre as classes, de forma a estruturar seu relacionamento em um grafo capaz de agregar as informações e relações entre as classes.

Também será abordado uma discussão sobre a arquitetura escolhida para o desenvolvimento do software, de forma a explorar os benefícios dos padrões escolhidos frente às necessidades do presente projeto.

Cronograma de Atividades

Atividades planejadas	**Mês**									
	03	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**
Entrega do Projeto de Pesquisa	x									
Organização do repositório GIT		x								
Implementação esqueleto API		x	x							
Implementação do Scan			x	x						
Implementação do Parser			x	x	x					
Implementação da análise				x	x	x				

Planejamento da Arquitetura de software.		x	x							
Estudo do referencial teórico	x	x	x	x	x					
Implementação dos testes		x	x	x	x					
Escrita do TCC					x	x	x			
Preparar apresentação								x		

Referências

Nas Referências deve ser elencado todas as obras utilizadas para a elaboração do Projeto de Pesquisa e normatizadas conforme o que se pede no "Manual de Instruções e Normas para Trabalhos de Conclusão de Curso".

Atenção: antes de enviar o arquivo para o Sistema de TCCs, remova todas as instruções originais que estão abaixo do conteúdo dos tópicos.