Descoberta de modularidade em códigos de projetos monolíticos em Java utilizando redes complexas

Marcos C. Brito Jr e Leando A. Silva

[1]Engenharia Elétrica e Computação (PPGEEC) – Universidade Presbiteriana Mackenzie
São Paulo – SP – Brazil

72307358@mackenzista.com.br, leandroaugusto.silva@mackenzie.br

Abstract. *Monolithic projects can bring great difficulties to software developers when maintenance or project scope expansion is necessary. Modularizing projects to have more defined classes and smaller modules to work with can bring benefits such as development time and productivity for the team. This work proposes the use of complex networks through the NetworkX library in Python using the greedy modularity algorithm for the static analysis of Java code with the purpose of discovering modules from the analysis of dependencies between classes, indicating the best way to find the clusters to be taken as code modules.*

Resumo. *Projetos monolíticos podem trazer aos desenvolvedores de sistemas grandes dificuldades quando necessário a manutenção ou ampliação de escopo de um projeto. Modularizar projetos para ter classes mais definidas e módulos menores para trabalhar podem trazer benefícios como o tempo de desenvolvimento do projeto e produtividade para a equipe. Este trabalho propõe o uso de redes complexas utilizando a biblioteca NetworkX em Python utilizando o algoritmo de modularidade gulosa, para a análise estática de um código escrito em Java com a finalidade de descobrir módulos a partir de análise de dependências entre classes, indicando a melhor forma de encontrar os agrupamentos a serem tomados como módulo de código.*

1. Introdução

Nos últimos anos, a Arquitetura Monolítica tem sido uma escolha popular entre os desenvolvedores de sistemas. Essa abordagem envolve combinar diversos componentes em um único sistema, como autorização, lógica de negócios e módulo de notificação, utilizando uma única plataforma e implantação unificada [Gos and Zabierowski 2020]. Embora essa arquitetura seja viável, é importante considerar que a manutenção e implementação de novas funcionalidades no código podem se tornar problemáticas ao longo do tempo, especialmente quando há alta rotatividade de desenvolvedores durante o projeto. À medida que o código-fonte cresce, a manutenção se torna mais complexa e as responsabilidades dos diferentes módulos se confundem, resultando em um crescimento desorganizado do projeto [Ponce et al. 2019].

A abordagem de redes complexas é uma técnica que pode ser utilizada para auxiliar na reescrita de projetos. Essas redes são estruturas compostas por elementos interconectados, como indivíduos em uma rede social ou palavras em um texto. Ao analisar essas redes, é possível identificar propriedades emergentes, como comunidades e a importância dos elementos [Van Steen 2010]. No caso de projetos de código, as redes complexas

podem ser aplicadas para descobrir grupos de classes correlacionadas [Newman 2003]. Algoritmos de detecção de comunidade, como os baseados na otimização de modularidade, podem ser utilizados para identificar esses grupos e auxiliar na reescrita dos projetos [Zhang et al. 2018].

Este trabalho utiliza redes complexas e a biblioteca *"NetworkX"* em Python para detectar automaticamente módulos em códigos monolíticos, utilizando o algoritmo de modularidade gulosa para auxiliar na busca de comunidades como módulos a serem desmembrados. Ele analisa as dependências do código fonte em projetos Java, visualizando as conexões por meio de grafos. Isso facilita a identificação de grupos de classes semelhantes e suas dependências comuns, permitindo a identificação de pontos de quebra de design para criação de módulos menores.

2. Trabalhos Relacionados

Dois importantes trabalhos na área e com grande relação a proposta deste artigo são de Šubelj e Bajec (2012) e Mzalami, Cito e Leitner (2017). Os trabalhos trazem propostas semelhantes com diferentes abordagens, sendo muito úteis para o entendimento de modularização de códigos através de redes complexas e análise de código-fonte de projetos de forma estática.

O estudo conduzido por Šubelj e Bajec (2012) analisou redes de sistemas construídas a partir de código fonte Java usando redes complexas. Foram abordadas as propriedades macroscópicas e microscópicas da rede, identificando as classes de sistemas mais influentes e vulneráveis. O estudo utilizou algoritmos de detecção de módulos e comunidades para revelar estruturas modulares dos projetos, facilitando a refatoração de pacotes Java. A descoberta de módulos é feita de forma automática, entretanto não se tem a proposta de modularização em vista a transformar uma arquitetura monolítica.

O projeto apresentado por Mzalami, Cito e Leitner (2017) tem a proposta de analisar os códigos-fonte de projetos de forma estática e gerar um grafo de conexão entre as classes para propor possíveis módulos a se tornarem microsserviços. Na proposta, apenas estratégias sobre como abordar o resultado dos grafos gerados são apresentados. A ideia é gerar trechos de código candidatos a módulos, a partir de projetos monolíticos com visualização em grafos.

Como mencionado anteriormente na introdução, o objetivo deste estudo é desenvolver um método de detecção automática de módulos para a transformação de projetos monolíticos, o que é uma das principais características distintivas em relação aos projetos mencionados anteriormente. A literatura citada acima está relacionada à organização de códigos monolíticos, porém, não aborda a descoberta automática dos módulos. Na próxima seção, apresentaremos a proposta completa deste trabalho.

3. Método Proposto

3.1. Fluxo do Processo

Para uma compreensão detalhada da proposta apresentada neste artigo, a figura 1 ilustra o fluxo de todo o processo funcional do trabalho, que encontra-se dividido em três seções: a primeira mostra as ações pelas quais o usuário é responsável, a segunda descreve as

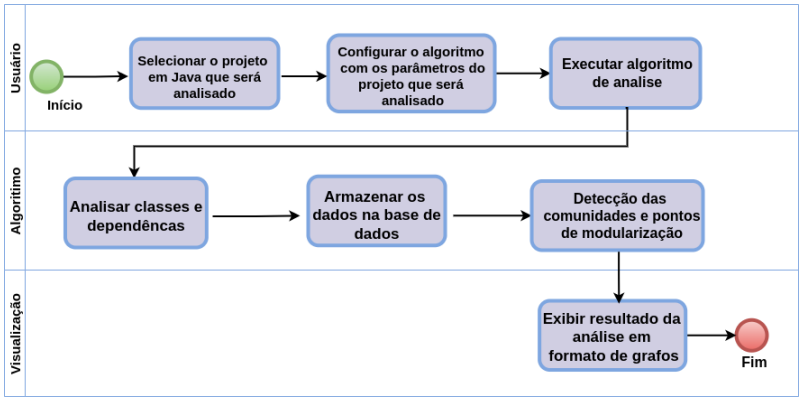

Figura 1. Fluxo do processo

ações executadas pelo sistema, identificadas na camada de ”Algoritmo”, e a terceira seção, denominada ”Visualização”, exibe os resultados na etapa final.

O processo começa com o usuário selecionando o projeto a ser analisado. Em seguida, é necessário informar ao algoritmo qual é o pacote Java usado no início de cada classe, pois é por meio desse nome de pacote que o algoritmo será capaz de mapear as classes interconectadas. Depois de ajustar os parâmetros, é possível iniciar o processo de análise. O algoritmo começa a analisar as classes, buscando por todos os arquivos `.java` e examinando as dependências de cada um, armazenando essas informações em um banco de dados relacional. Na etapa seguinte, as comunidades são identificadas e os pontos em que os módulos podem ser divididos são determinados. Essas informações geram os grafos de visualização das comunidades e os grafos de cada comunidade separadamente, listando as classes que devem permanecer juntas. Para realizar a detecção das comunidades, são usados algoritmos de redes complexas.

3.2. Redes Complexas

As redes complexas têm aplicações em diversas áreas do conhecimento e na computação podem ser utilizadas para analisar códigos, identificar vulnerabilidades e propor mudanças nos projetos com base nos resultados encontrados [Pan et al. 2011]. A detecção de comunidades pode indicar áreas em que um código complexo pode ser dividido em módulos menores. Além disso, é possível realizar uma análise estática do código fonte, diretamente no código criado pelos desenvolvedores.

3.3. Modularidade Gulosa

O algoritmo de modularidade gulosa é um método de detecção de comunidades em redes complexas. Ele é usado para identificar grupos ou módulos dentro de uma rede que possuem alta densidade de conexões internas e baixas conexões com outros grupos. Esse algoritmos será utilizado na detecção das classes nos projetos analisados.

3.4. Macro Arquitetura do Projeto

A solução proposta para encontrar dependências e pontos de modularização consiste em uma macro arquitetura ilustrado na figura 2. Ela apresenta a sequência de ações da perspectiva do usuário que realizará a análise até o armazenamento dos resultados no banco de dados e, por fim, a visualização do resultado por meio de grafos.

O processo inicia com usuário fornecendo informações de parametrização, como o caminho do projeto Java e a configuração do pacote das classes a serem analisadas. O algoritmo realiza a busca e leitura das classes, coletando informações para formar o grafo de dependências. Esses dados são armazenados em um banco de dados. Na etapa seguinte, o algoritmo analisa os dados armazenados para identificar possíveis pontos de modularização, que também são armazenados no banco de dados. Em seguida, a detecção de comunidades é realizada usando a biblioteca NetworkX e o algoritmo de modularidade gulosa. Os grafos resultantes mostram as relações entre as classes e os possíveis módulos por meio das comunidades detectadas. Finalmente, os resultados são visualizados por meio dos grafos gerados.

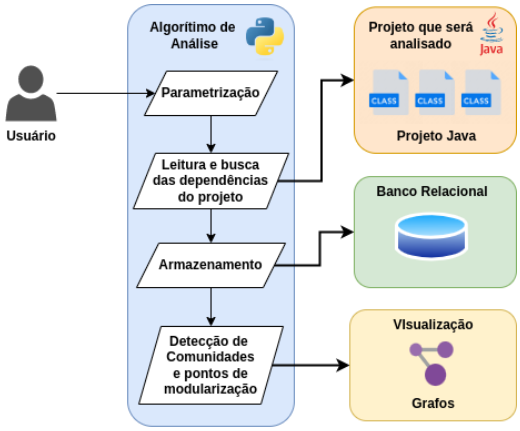

Figura 2. Arquitetura Macro do Projeto

4. Resultados Experimentais

Foram realizados alguns experimentos para validar a eficiência do algoritmo em encontrar dependências de classe em projetos e armazená-las em um banco de dados de grafo. Esses experimentos consistiram na análise de um código Java complexo que passou por crescimento desorientado e que possui módulos misturando muitas responsabilidades. Esse seria um cenário onde a análise ajudaria a organizar o projeto em módulos menores ou separar o projeto para que não tenham mais responsabilidades do que o necessário para um código coeso.

A figura 3 mostra o resultado da análise de um código Java chamado ”Consulta Kerberos”. Esse código é um módulo de conexão ao banco de dados *Hadoop*, utilizando autenticação *Kerberos*. Porém, existem muitos módulos não relacionado ao objetivo do projeto, com muitas regras de negócio espalhadas e realizando atividades distintas, ou seja, um projeto que cresceu de forma desordenada. As cores iguais são as classes que fazem sentidos ficarem juntas e as ligações em vermelho tracejado são os pontos onde os grupos precisam ser separados. A figura 4 ilustra de forma ampliada as classes a serem separadas. Todas as comunidades serão mostradas separadamente na análise, por fins de limitação não foram apresentadas.

Ainda como resultado, o método proposto fornece um mapeamento entre as classes do projeto e como devem ser separadas. Este resultados está ilustrado na tabela 2. Além disso, informações adicionais relevantes são calculadas e apresentadas para enriquecer a análise dos resultados, como ilustrado na tabela 1. Esses dados podem contribuir

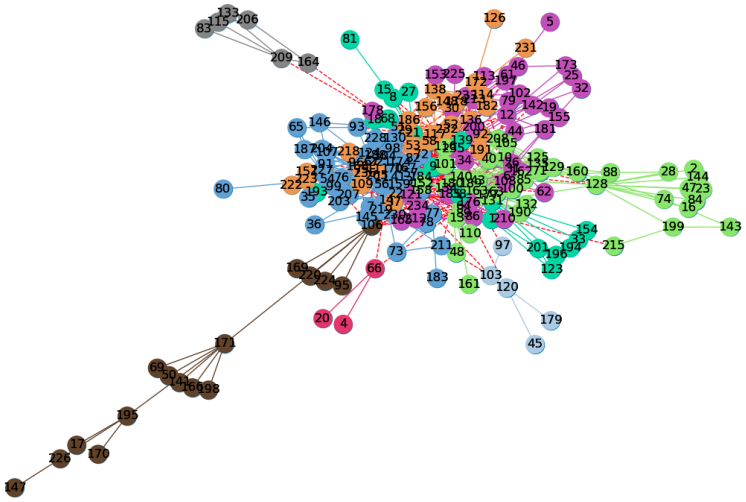

Figura 3. Comunidades detectadas

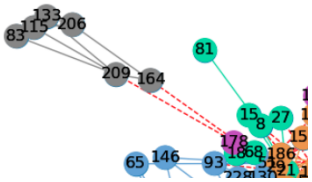

Figura 4. Conexões que deverão ser quebradas.

para uma melhor compreensão da complexidade do projeto em análise e o grau de sua complexidade para modularização.

5. Conclusões e Trabalhos Futuros

O artigo descreve uma abordagem automatizada para modularizar códigos monolíticos em projetos Java. Essa abordagem é baseada na detecção de comunidades em redes complexas usando a biblioteca *NetworkX* em Python e o algoritmo de modularidade gulosa. Ao utilizar essa técnica, o código pode ser dividido em módulos menores e mais gerenciáveis, o que pode melhorar significativamente sua manutenção e aumento de produtividade dos times de desenvolvimento.

Análise	Resultado
Medida de Modularidade:	0.07615483221665743
Número de nós do grafo:	211
Número de enlaces do grafo:	679
Densidade do grafo:	0.0306477093206951
Grau médio:	6.436018957345971
Grau máximo:	75
Coeficiente de aglomeração do grafo:	0.2672128326145455
Tamanho médio dos caminhos do grafo:	3.249785601444369
Sequência do grau:	[1, 2, 15, 5, 11, 11, 7, 15, 10, ...]
Classes da comunidade:	[1, 57, 3, 34, 55, 94, 103, 161, ...]

Tabela 1. Medida de modularidade e análises quantitativas

Classe	Quebrar	Classe
poc.consulta.ws.controller.testecontroller	–>	consulta.core.util.helper.messagehelper
poc.consulta.ws.controller.testecontroller	–>	consulta.core.model.enums.tipoconsumidor
consulta.core.util.helper.messagehelper	–>	consulta.ws.controller.consulentecontroller

Tabela 2. Ligações entre classes que serão quebradas.

Os resultados parciais obtidos por meio de detecção de comunidades em redes complexas e da separação de módulos de forma visual por meio de grafos, trazem importantes benefícios para o processo de refatoração do projeto. Além de fornecer uma clara representação das relações entre as classes e a organização dos módulos, essa abordagem permite que os desenvolvedores identifiquem quais classes poderão permanecer juntas e quais os pontos de separação com os outros módulos de forma automatizada.

Em trabalhos futuros, para aprimorar a compreensão do grafo gerado e o resultado da modularização, é importante explorar análises quantitativas e analisar outros projetos para validar a eficácia do algoritmo para obter dados de comparação. Acompanhar a evolução de um projeto após a reescrita utilizando os resultados obtidos permitirá comparar o estado anterior e posterior, proporcionando um contexto real de validação. Além disso, realizar uma avaliação abrangente das ameaças à validade, considerando diferentes tipos de ameaças, contribuirá para melhorar o resultado final da análise e incorporar melhorias relevantes.

Referências

Gos, K. and Zabierowski, W. (2020). The comparison of microservice and monolithic architecture. In *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, pages 150–153. IEEE.

Mazlami, G., Cito, J., and Leitner, P. (2017). Extraction of microservices from monolithic software architectures. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 524–531. IEEE.

Newman, M. E. J. (2003). The structure and function of complex networks. *SIAM Review*, 45(2):167–256.

Pan, W., Li, B., Ma, Y., and Liu, J. (2011). Multi-granularity evolution analysis of software using complex network theory. *Journal of Systems Science and Complexity*, 24(6):1068–1082.

Ponce, F., Márquez, G., and Astudillo, H. (2019). Migrating from monolithic architecture to microservices: A rapid review. In *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–7.

Šubelj, L. and Bajec, M. (2012). Software systems through complex networks science: Review, analysis and applications. In *Proceedings of the First International Workshop on Software Mining*, pages 9–16.

Van Steen, M. (2010). Graph theory and complex networks. *An introduction*, 144.

Zhang, X., Ma, Z., Zhang, Z., Sun, Q., and Yan, J. (2018). A review of community detection algorithms based on modularity optimization. In *Journal of Physics: Conference Series*, volume 1069, page 012123. IOP Publishing.