Mining Concepts from Code using Community Detection in Co-occurrence Graphs

Toni Mattis
Software Architecture Group
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
toni.mattis@hpi.uni-potsdam.de

ABSTRACT

Software modularity is a quality that determines how fluently individual parts (modules) of a system can vary and be understood if taken by themselves. However, modularity tends to degrade during program evolution – old concepts may get entangled with code introduced into their modules, while new concepts can be scattered over many existing modules.

In this work, we propose to infer high-level *concepts* and *relations* between them independently from the current module decomposition by exploiting the *vocabulary* used by programmers. Our approach uses an extensible graph-based vocabulary representation in which we detect latent communities representing our concepts. Inferred concepts can be used to support program comprehension, track architectural drift over time, and provide recommendations for related code or refactorings.

CCS CONCEPTS

• **Software and its engineering** → **Abstraction, modeling and modularity**; Software libraries and repositories; • **Mathematics of computing** → *Random graphs*;

KEYWORDS

modularity, topic models, graphs

ACM Reference Format:
Toni Mattis. 2018. Mining Concepts from Code using Community Detection in Co-occurrence Graphs. In *Proceedings of 2nd International Conference on the Art, Science, and Engineering of Programming (<Programming'18> Companion).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3191697.3213797

1 MOTIVATION

Software modularity is a quality that determines how fluently individual parts (modules) can vary and be understood if taken by themselves. Modularity tends to degrade during program evolution mainly due to unforeseeable requirements not anticipated in the architecture, and due to limitations of the programming language itself. Old concepts may get *entangled* with code introduced into their modules, while new concepts can be scattered over many existing modules. On the one hand, programmers require increasingly more time and attention to understand and modify concepts. On the other hand, modifying programs under incomplete knowledge can reinforce architectural drift.

In the context of this work, we propose to infer high-level *concepts* and *relations* between them independently from the current module decomposition using the *vocabulary* built by programmers. Inferred concepts can be used to support program comprehension, track architectural drift over time, and provide recommendations for related code or refactorings.

We design a graph-based vocabulary representation based on lexical structure, that can be flexibly extended using run-time data and version history. In this graph, we then probabilistically infer latent communities that serve as our concepts. Within a programming environment, concept membership and distribution can be manipulated and queried through reflection interfaces and thus be used by tool builders.

Preliminary assessment of the new model shows that it outperforms the *LDA topic model* in terms of concept coherence, expressiveness, and the data sources it can utilize.

We are confident such a model can not only help with reverse engineering, but also support modularity during forward engineering by giving programmers immediate feedback on the alignment of their vocabulary with the current module decomposition.

2 BACKGROUND AND RELATED WORK

According to the distributional hypothesis, identifier names referring to the same concept are more frequently co-located than unrelated names. We also assume that they tend to share more control flows at run-time, and are more frequently edited at the same time.

Topic models, such as Latent Dirichlet Allocation (LDA)[2], are used to condense a large number of natural-language *documents* into histograms of words (*bag-of-words model*) and represent each histogram as proportions of fewer shared *topics*. Each topic constitutes a set of semantically related words. Topic models are frequently used to infer concepts from programs[1, 3, 5]. However, the bag-of-words model requires programs to be chunked into artificial documents, disregard the graph-like structure of code, do not support program-specific dependencies other than co-occurrence, and have poorer performance on shorter documents appearing frequently in source code.

Random Graph Models describe a stochastic process to generate graphs. They can explain a so called *community structure* by assigning nodes to latent communities and making edges between nodes

<*Programming'18*> *Companion, April 9–12, 2018, Nice, France*

ACM ISBN 978-1-4503-5513-1/18/04…$15.00
https://doi.org/10.1145/3191697.3213797

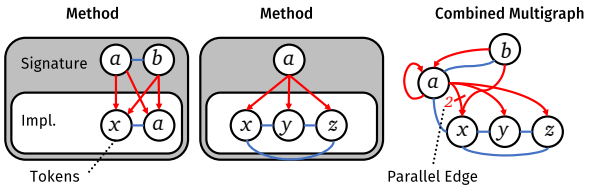

Figure 1: Lexical tokens in two methods being linked by undirected co-occurrence edges, blue, and directed abstraction/implementation edges, red. The graph jointly generated by both methods is depicted on the right.

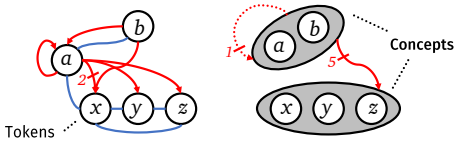

Figure 2: Community structure induced on the graph, directed edges are summarized to inter-concept relations

of the same community more likely than between groups. Conversely, if a graph has been *observed*, we can infer an assignment of nodes to communities that would have *most likely* generated the observed graph.

3 GRAPH-BASED SEMANTIC MODEL FOR CODE

Our approach mixes ideas from topic modeling with random graphs. In our graph model, we regard nodes as the *names* chosen by programmers. We introduce two types of edges: undirected edges indicating simple co-occurrence of two names (e.g., they are used in the same statement), and directed edges, indicating that one name is being defined in terms of another name (e.g, a method calling another method), see Figure 1. If run-time information is available, calling relations can be added as directed edges, and if version history is available, names modified within the same version/commit are connected by undirected edges. The same pair of nodes can be connected multiple times (multi-graph).

We designed a community mining algorithm that can deal with both directed and undirected edges to infer the most likely decomposition of the resulting graph into communities. Simultaneously, directed edges between nodes are translated to high-level relations between communities, see Figure 2. Their interpretation is that one (abstract) concept is using another (more concrete) concept for its implementation.

4 EVALUATION

Comparing latent concepts inferred from the graph model in four open-source projects from two different languages (Python and Smalltalk) to those inferred using the common topic model LDA shows a consistently higher intra-topic *coherence*[4], even when used without run-time data and version history. Qualitatively, our model infers links between concepts, which LDA is incapable of, and distinguishes highly entangled concepts more reliably, especially when run-time data or history is added. However, due to a vast body of topic model research, modern implementations of LDA are faster than the graph-based model by an order of magnitude and require less memory.

5 NEXT STEPS

We aim at measuring and visualizing how concepts drift during the evolution of source code in large open-source projects, plan to quantitatively evaluate the model when used as recommender system, and most importantly would like to integrate concepts into *programming tools* to make them visible, navigable, and modifiable.

REFERENCES

[1] David Binkley, Daniel Heinz, Dawn Lawrie, and Justin Overfelt. 2014. Understanding LDA in Source Code Analysis. In *Proceedings of the 22nd International Conference on Program Comprehension (ICPC 2014)*. ACM, Hyderabad, India, 26–36.

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3 (March 2003), 993–1022.

[3] Erik Linstead, Paul Rigor, Sushil Bajracharya, Cristina Lopes, and Pierre Baldi. 2007. Mining Concepts from Code with Probabilistic Topic Models. In *Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering (ASE '07)*. ACM, Atlanta, GA, USA, 461–464.

[4] David Mimno, Hanna M. Wallach, Edmund Talley, Miriam Leenders, and Andrew McCallum. 2011. Optimizing Semantic Coherence in Topic Models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '11)*. Association for Computational Linguistics, Edinburgh, United Kingdom, 262–272. http://dl.acm.org/citation.cfm?id=2145432.2145462

[5] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. 2011. Modeling the Evolution of Topics in Source Code Histories. In *Proceedings of the 8th Working Conference on Mining Software Repositories* (2011) *(MSR '11)*. ACM, 173–182.