Lessons Learned from Large-Scale Refactoring

Hyrum K. Wright
Google, LLC
Pittsburgh, Pennsylvania, USA
hwright@google.com

***Abstract*—Google maintains a large multi-language codebase containing hundreds of millions of lines code across many different projects. Over the past several years, our team has developed processes which enable us to update that codebase efficiently at scale. In this talk, I discuss some of the lessons which we have learned and existing open problems in the space.**

***Index Terms*—refactoring, software maintenance**

Over the last two decades Google has evolved a strategy for refactoring and maintaining our large corpus of source code. This corpus consists of hundreds of millions of lines of code, primarily in Java, C++, Python, Go and assorted configuration languages [1]. Standard localized refactoring techniques, such as IDE-based refactoring tools, do not scale to this size, as refactoring targets may include tens of thousands of references.

Our previous work developed refactoring tools which operate at scale across the abstract syntax trees of the entire repository [2], or by using examples to generate changes [3]. These tools enable the raw generation of changes at scale, but this exposes additional problems, such as how to atomically test, review and commit a change touching tens of thousands of files in a ever-evolving repository [4]. The answers to these problems influence the design of new systems, and how we approach the maintenance of existing ones.

Over the last several years, engineers at Google have undertaken the largest known refactoring efforts in any organization. These projects have included:

- Changing core APIs used by hundreds of thousands of client projects.
- Migrating 500,000 variable types from a custom container to one from the C++ standard library.
- Changing the standard dictionary type to a more efficient one.

Our refactoring techniques make it cost efficient to perform these, and other tasks, some as mundane as changing the import sort order in every file throughout our 200M-line Java codebase.

In this talk, I discuss the motivation behind taking on such large refactoring tasks, such as:

- Adopting new libraries
- Rolling out updated infrastructure, such as compilers
- Adapting our codebase to new language standards
- Fixing common bug-prone patterns at scale
- Address general ecosystem changes (i.e. "bitrot")

I also explore the challenges involved with performing these large refactoring projects, and what we are doing to address them. These challenges and open problems include:

- Organizational and cultural concerns around large-scale migrations
- Tooling support for refactoring at scale
- Design considerations for large refactorable systems
- Programming language support for large-scale refactoring
- Reducing expertise and resources needed to perform large refactorings

Over the last few years, our approach has increasingly been one of "sustainability": over the expected lifespan of our codebase, we need to be capable of reacting to any likely change, be it hardware, software, programming language, new paradigm, or changing business requirement. Without operationalizing the ability to make broad changes across the organization, that goal would rapidly become impossible, as our legacy code overwhelms our ability to develop new products and features. With the tools and techniques that we are pioneering, we believe we are capable of changing anything and everything with a reasonable investment of tools and human expertise.

Refactoring has become an important consideration as our engineers design new systems and update our old ones. We can now make technical decisions knowing that they are mutable. We hope to use this talk to share what we've learned and encourage the research community to explore the many problems which aren't yet tractable.

References

[1] R. Potvin and J. Levenburg, "Why Google stores billions of lines of code in a single repository," *Communications of the ACM*, 2016.

[2] H. K. Wright, D. Jasper, M. Klimek, C. Carruth, and Z. Wan, "Large-scale automated refactoring using ClangMR," in *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, ser. ICSM '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 548–551. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2013.93

[3] L. Wasserman, "Scalable, example-based refactorings with refaster," in *Workshop on Refactoring Tools*, 2013. [Online]. Available: http://dx.doi.org/10.1145/2541348.2541355

[4] T. Winters, "Non-atomic refactoring and software sustainability," in *Proceedings of the 2nd International Workshop on API Usage and Evolution*, ser. WAPI '18. New York, NY, USA: ACM, 2018, pp. 2–5. [Online]. Available: http://doi.acm.org/10.1145/3194793.3194794

2576-3148/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSME.2019.00058