Measuring Project Exposure Using Types Declared in a Java Project

Michael D. Feist
Department of Computing Science
University of Alberta
Edmonton, Canada
mdfeist@ualberta.ca

Ian Watts
Department of Computing Science
University of Alberta
Edmonton, Canada
watts1@ualberta.ca

Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Canada
abram.hindle@ualberta.ca

ABSTRACT

A Java project contains many different types which are defined by the language, the developers and included libraries. This paper examines how many of those types are being declared by developers in Java projects. A tool for computing the difference between Abstract Syntax Trees (AST) is used to compare revisions in GitHub repositories to find what libraries and types are contributed and changed by different authors. From these differences, the number of type declarations by an individual developer is calculated. A developer's exposure to different parts of a project is measured by how many out of the total number of types they have used. A comparison between developers is used to see if programmers specialize type usage in their contributions. We find that most projects use a large number of types and most developers only declared a small portion of the total types. We also propose the use of this metric to encourage developers to increase their exposure to more parts of a project.

CCS Concepts

•**Software and its engineering** → *Software organization and properties;*

Keywords

Software Engineering; Mining Software Repositories; Programming Languages; Abstract Syntax Trees

1. INTRODUCTION

In Java, a type can be a primitive type or an object reference. Primitive types are those that are built into the language such as an int or char. Object references point to instances of classes in memory which are defined in the project or through an included library. The types in a project represent the functionality available for a developer to work with. A large or complex project may have many different types, whereas a small project might only use a few. Different parts of a program are going to use different types and libraries. When writing or contributing to a project, the author's choice of which part of the program to edit will in turn affect what types they end up working with.

There is lots of information on how types are used as a language feature, but there is little on how types are used by developers. There are many factors that could affect how many types developers use in a project. In particular, collaborating with multiple developers could have quite an effect. Work is partitioned among developers which could limit exposure to some types. Developers typically have a role in the development process that is defined by the project, their collaborators and their experience level [10]. A programmer may choose to stay within their comfort areas and not use certain types or leave their use to other developers. As a developer's experience with a project increases they may be inclined to use more types. Some types fulfill a specific task, so they may only apply to a small part of a project.

The biggest factor of an author's type useage is likely their activity level and involvement in a project. There are many more people watching repositories than actually contributing to them. Increasing the activity level and encouraging aspiring developers would be beneficial to the overall health of the repository [10].

In this paper we study the types developers are using by examining each revision in a repository and finding the types which are being declared by the developer making the revision. We looked at Java repositories to answer the following research questions:

RQ1: Do developers declare a small subset of the total types in a project?

RQ2: How many developers in each project had large type coverage?

2. RELATED WORKS

The usage of language features in Java and the behaviour of developers has been studied before. Robert Dyer *et al.* [2] mined AST nodes to study the use of new Java language features over time. They found that all new Java features do get used, but not nearly as often as they could have been. New features varied greatly in popularity and adoption rate. Developers would also modify old code to use new features as they were released. The authors also found the most popular features and how teams adopted new features. They did not check to see how much a developer used a feature but instead

MSR16 May 14–15, 2016, Austin, TX, USA

DOI: 10.1145/1235

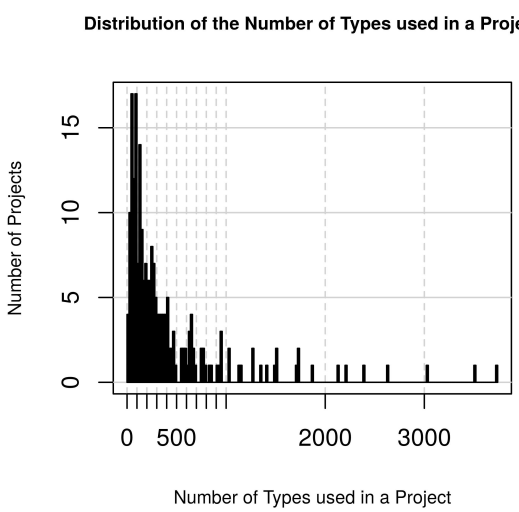

Figure 1: Shows how many different types are present in each project.

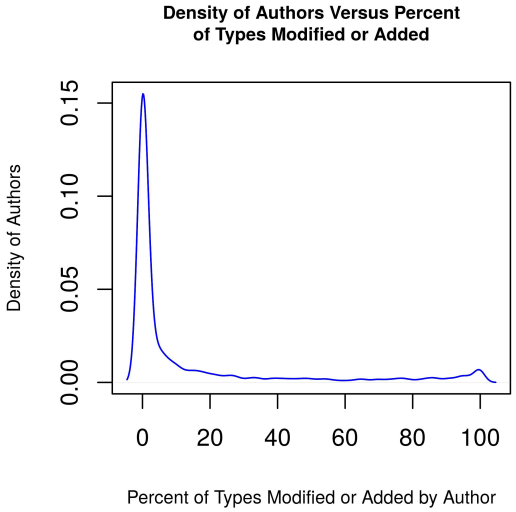

Figure 2: The distribution of authors by their coverage level.

only checked to see if they used it at all.

Grechanik *et al.* [4] examined the structure of Java programs mined from 2080 programs. This paper examined the breakdown of syntactic structures in open source repositories, however it does not consider per author statistics.

Meyerovich and Rabkin [7] surveyed developers and examined repositories to learn about language adoption and usage. Developers feel that certain language features are more important than others.

Parnin *et al.* [8] mined repositories to see how Java generics have been used in open source projects and found that generic usages were often introduced by a single developer in a project. Generics usage was also fairly narrow, with the primary use being collecting and traversing lists of objects.

Patrick Wagstrom *et al.* [10] explored the roles that developers take in a networked, social development environments like GitHub. These role were defined as their level of contribution as well as the types of issues they dealt with. Developers will take on multiple roles in a project and will sometimes fulfill the same role across different projects.

Lämmel *et al.* [6] used ASTs to examine the usage of APIs in Java projects. They find what APIs are popular as well as if they are used in a framework like manner.

These studies examined at the usage of language features, APIs and object oriented structures but did not look at the types used in Java projects.

3. METHODOLOGY

In this section, we define our measure of project exposure, type coverage, and list the additional tools used. Next, we explain the data collection and methodology used to mine the repositories.

3.1 Metrics

In order to measure the amount of different areas that a developer has contributed to, or covered, we will be using the number of types they have declared out of the total number present in the repository. Therefore, a developer's type coverage of a project is what percent of the total types did a developer add or change at least one time. This includes the Java static types, other Java objects in the repository and all objects added from libraries. If a developer has declared every type involved in the project, they have likely seen every part of it. Even if they are not editing every file, a developer with full type coverage would have knowledge of all types existing in the untouched file.

An Abstract Syntax Tree (AST) is a tree structure which represents a piece of code by breaking its syntactic constructs into a tree structure. Each node in the tree represents a syntactically valid chunk of code which can be contained in a parent node or broken up into child nodes. This captures the structure of the code in an abstract manner. When a change is made to a file, the difference will also appear in the AST. This means that the AST of revisions in code repositories can be compared to see what structures a developer is touching.

3.2 Data Set

Our data set consists of 216 GitHub repositories. To ensure that we only looked at Java repositories of reasonable size, we first queried BOA [1] for eligible repositories. We ran our query on the 2015 GitHub September dataset. The criteria for a repository to be accept was that it included at least 10 Java files, 3 different committers, 30 commits, and at least one commit from after 2014 [5]. 22475 repositories fit the criteria. This allowed us to narrow our search down to sizable Java projects that were recently active. Out of the possible repositories returned from BOA we randomly sampled the 216 repositories and pulled them from GitHub.

3.3 Tools

The ASTs were generated using Spoon [9]. Spoon is a tool for transforming and analysing Java source code. It breaks code up into a meta-model such as an AST. The model consists of three parts: structural elements, code elements and

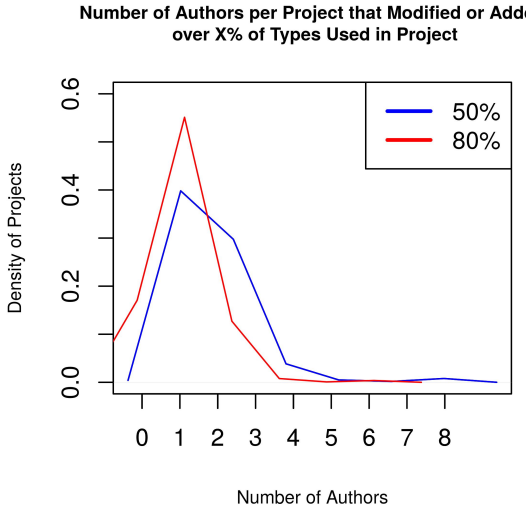

Figure 3: Shows how common developers with above X% coverage levels are in a project.

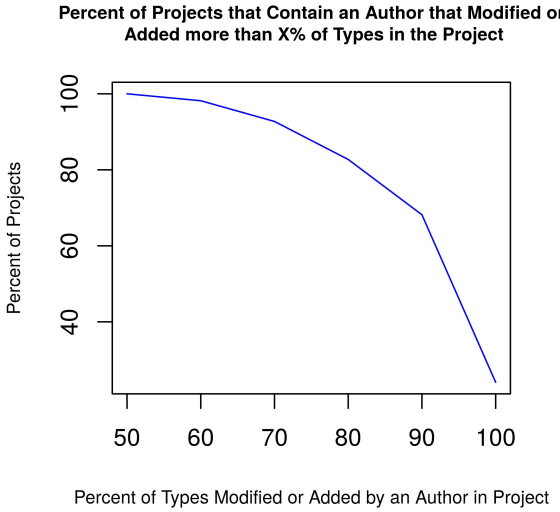

Figure 4: Shows that most projects have at least one developer with a high coverage level.

references to program elements. The Spoon model is convenient because it provides the structure for performing an AST diff while retaining the code elements for analysis afterwards. Spoon also preserves all type and library information, which is what we were interested in.

GumTree [3] is a library which is used to compute the difference between two ASTs generated by Spoon. Gumtree does not handle empty files by default, so in order to compare new files we had to modify Gumtree to consider a new file as an empty AST.

3.4 Approach

To determine what types an author declared in a project we looked at all revisions in the master branch in each of the 216 GitHub repositories. Using the Gumtree algorithm with Spoon, we were able to generate ASTs for each of the Java files differences between revisions. We only looked at additions and modifications since deletions do not necessarily show that an author has used the type they are deleting. We then counted the number of times a type is declared in the ASTs. By adding up the unique types declared by each author in a given project, we were able to determine the total number of types declared in each project. This then allowed us to calculate the percent of types an author has modified or added compared to the total number of types in a project. Computing the ASTs of every revision was very computationally expensive which limited the number of repositories which could be analysed.

4. ANALYSIS AND FINDINGS

We found that more than half of the projects used less than 250 different types but there were a few projects that used thousands of different types. The number of types were distributed in a power-law-like fashion.

As an example of a developer with high type coverage, we identified one developer with high type coverage who worked on a large project of over 4,000 total commits. We found that the developer had contributed to the project for over 5 years since the time of its inception. Working on the project from the start gave them the opportunity to create many of the project specific types and use them over their 5 years of development. Another developer in the same project with very low type coverage had more recent commits. According to the commit logs this developer appeared to be cleaning up code and fixing bugs.

4.1 RQ1: Do developers declare a small subset of the total types in a project?

We found that among the total of 3334 developers the majority cover less than 30% of the types in the project they contribute to. With a small group covering more than 80% of the types. Interestingly, there were many developers who did not make any type declarations and only made changes to other parts of Java files. There were also very few developers who had a type coverage level between 30% and 80%, suggesting that there were not many developers with only moderate involvement in a repository. They either stuck to a small portion of types or utilized nearly all of them. This is shown in Figure 2.

4.2 RQ2: How many developers in each project had large coverages?

We found that on average a project has about 7 developers. On average, these projects only had two developers who modified or added over 50% of the types in the project. Furthermore, on average only one developer modified or added over 80% of the types. This is shown if Figure 3.

Looking at Figure 4 we can see that approximately 70% of the projects have at least one developer who has modified or added over 90% of the types used in a project. We found that all projects have at least one developer that modified or added over 50% of the types. The highest number of developers on a project was 347 but the max number of developers with 50% or more type coverage was only 8.

This means no large projects had many developers who had covered all types.

4.3 Discussion

If developers were able to see their type coverage when contributing to a repository, it might make them more aware of the parts of the project that they had not touched. This could encourage them to fix bugs, add features, or make changes to new areas in order to increase their type coverage. This could also provide developers with lower activity incentive to contribute more to a project to distinguish themselves. When type coverage rate drops, it would also make the developer more aware of new types and changes in the repository that may have gone unnoticed without such a metric.

4.4 Threats to Validity

Threats to the validity of this project include threats of construct and external validity. Discriminant threats to construct validity include projects that had many files that were not Java source files such as HTML or Ruby files. These projects might not accurately represent an average Java project or behaviour of Java developers. There were also some authors who committed a large number of Java files all at once. This suggests that they may have uploaded a library and did not write the code themselves. Git settings allow for a user to set a name which is used in the revision info. This means that multiple users in the stats could be the same author under different names. This usually happens when the author uses multiple computers.

External threats to validity include sampling exclusively from the September GitHub dataset and the possibility that there could be large projects that only use a few types however uncommon they may be.

5. CONCLUSIONS AND FUTURE WORK

In this paper we investigated the number of types used in Java projects and the number of types covered by each developer. In the data provided for RQ1, we found that most developers declare a small percentage of the total number of types in a project. Developers declare a small subset of types and there are many types that go unused by developers in a project. For RQ2 we found that most repositories had at least a few developers with high type coverage. Having a large number of developers with high type coverage was very uncommon.

There are many more questions that can be answered by comparing differences in ASTs as well as looking at type coverage. AST diffs could be used to look at more structural differences between code revisions. The number of other language features used by the developers could be counted to see how much of the Java language they use and what they do not know.

The type coverage of a developer could be weighted against the number of bugs they commit, which could indicate that a developer has gone beyond their expertise level. ASTs could be used to find what structure the bugs were committed in to help the developer identify their weak areas. A study could also be done to see if developers react positively to knowing their type coverage rate in a repository.

6. REFERENCES

[1] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *35th International Conference on Software Engineering*, ICSE 2013, pages 422–431, May 2013.

[2] R. Dyer, H. Rajan, H. A. Nguyen, and T. N. Nguyen. Mining billions of ast nodes to study actual and potential usage of java language features. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 779–790, New York, NY, USA, 2014. ACM.

[3] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus. Fine-grained and Accurate Source Code Differencing. In *ASE 2014*, page 11 p., France, 2014.

[4] M. Grechanik, C. McMillan, L. DeFerrari, M. Comi, S. Crespi, D. Poshyvanyk, C. Fu, Q. Xie, and C. Ghezzi. An empirical investigation into a large-scale java open source code repository. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 11:1–11:10, New York, NY, USA, 2010. ACM.

[5] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 92–101, New York, NY, USA, 2014. ACM.

[6] R. Lämmel, E. Pek, and J. Starek. Large-scale, ast-based api-usage analysis of open-source java projects. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 1317–1324, New York, NY, USA, 2011. ACM.

[7] L. A. Meyerovich and A. S. Rabkin. Empirical analysis of programming language adoption. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '13, pages 1–18, New York, NY, USA, 2013. ACM.

[8] C. Parnin, C. Bird, and E. Murphy-Hill. Java generics adoption: How new features are introduced, championed, or ignored. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 3–12, New York, NY, USA, 2011. ACM.

[9] R. Pawlak, M. Monperrus, N. Petitprez, C. Noguera, and L. Seinturier. Spoon: A library for implementing analyses and transformations of java source code. *Software: Practice and Experience*, page na, 2015.

[10] P. Wagstrom, C. Jergensen, and A. Sarma. *Roles in a Networked Software Development Ecosystem: A Case Study in GitHub*, 2012.