A comparison of the test to production code ratio of dynamically and statically typed languages

Christian Paling
University of Applied Sciences Regensburg
Regensburg, Germany
christian.paling@googlemail.com

***Abstract*—There are many arguments for and against static or dynamic typing, a common one being that when writing a software system with a statically typed system the programmer needs to employ less testing than when using a dynamically typed language. While the theoretical argument is fairly easy to follow the question is whether actual software systems do reflect this mindset. This paper presents a study that analyses 20 software projects and compares the test to production code ratio of both types of typing. In the results, it can be observed that projects using dynamically typed languages do have a higher test to production code ratio than their counterparts.**

***Index Terms*—Type Systems, Programming Languages**

I. Introduction

The question of whether dynamic typing is better or worse than static typing is a long and ongoing discussion in the software industry. Typical arguments in favor of statically typed languages (see [1]) are for example:

- Easier debugging of programming errors
- Higher productivity in large teams of programmers through the negotiation of interfaces
- Eliminating certain security breaches by default

Another common argument is that by compiling a statically typed program successfully the programmer can be sure that no incorrectly typed data is being passed through the system. On the other hand, a programmer using a dynamically typed language will need to write software tests to prove that his software is working as correctly as a statically typed program. Therefore software systems using dynamically typed languages will need more tests than programs implemented with statically typed languages. [1]

This paper analyses whether this argument is not only valid in theory but can be observed in reality. The test to production code ratios of open-source repositories of both types of languages are being calculated and compared to examine the previously stated theory.

The results show that software systems using dynamically typed languages do have a higher test to production code ratio by a significant margin.

The structure of this paper is organized as follows. Section 2 describes the methodology used for this study. Section 3 presents the results of the analysis which is followed by Section 4 that discusses these results. Afterwards possible threats to validity are pointed out in Section 5 and related work is presented in Section 6. Lastly, Section 7 concludes this paper by discussing possible future work.

II. Methodology

In order to have the results as unbiased as possible, we selected three different languages for both dynamically and statically typed languages. On the dynamically typed side Ruby, Python, and Clojure, and as statically typed languages Java, Haskell, and Go.

Afterwards we chose ten fairly popular open-source repositories from Github for both types of languages, for Ruby and Go four repositories each, for the rest of the languages three repositories. The chosen projects are employed in different domains like web applications, programming libraries, or command-line tools.

For each project, the lines of test code (TLOC), as well as the line of production code (PLOC), were analysed using a self-written tool[1] and compared by calculating the following ratio:

$$TLOC/PLOC = RATIO \tag{1}$$

The tool identifies tests by recognizing common patterns of test files for each language, e.g. in Go a test has to end with *_test.go*, and removes comments and newlines from each file to count the number of lines of actual code.

By comparing the ratio of the dynamically typed languages with the statically typed languages it is possible to get an idea of the number of tests that are being written.

III. Detailed Results

In table I the results for the dynamically typed languages are outlined in detail. It is important to note that all results were acquired on 03 November 2019. Therefore the numbers might vary at a different point in time. It can be seen that all projects have a ratio between about *0.7 - 2.4* and average at about *1.2*. This means that for every 10 lines of production code 12 lines of test code are written.

When comparing these results to the numbers of statically typed languages in table II clear differences are apparent. Every project except one has a ratio lower than *1.0*. With an average ratio of *0.4*, three times the amount of test code was written using dynamically typed languages compared to statically typed projects.

[1]https://github.com/bakku/dynamic-vs-static-typing-paper/tree/master/test_ratio_analyzer

TABLE I
DYNAMICALLY TYPED LANGUAGES

Language	Project	TLOC	PLOC	Ratio
Ruby	Mastodon	20084	28868	0.6957
	Gitlab	363082	224295	1.6188
	Sinatra	11819	5006	2.3610
	Sidekiq	4541	4222	1.0756
Python	Flask	5606	6799	0.8245
	Sentry	86115	142347	0.6050
	YAPF	9648	5756	1.6762
Clojure	Metabase	37618	45753	0.8222
	Compojure	628	555	1.1315
	Riemann	7610	10015	0.7560
Average Ratio	1.1567			

TABLE II
STATICALLY TYPED LANGUAGES

Language	Project	TLOC	PLOC	Ratio
Golang	Gin	5480	3831	1.4304
	Gogs	733	36122	0.0203
	Terraform	126286	1003481	0.1259
	Moby (Docker)	84152	951399	0.0885
Java	Jenkins	55738	102093	0.5460
	Guava	221843	288438	0.7691
	Spark	3093	7064	0.4379
Haskell	Pandoc	9512	53922	0.1764
	Haxl	1052	2639	0.3986
	Semantic	2189	28321	0.0773
Average Ratio	0.4070			

IV. DISCUSSIONS

While the basic result of the study may not be surprising, the big difference between the two types of typing provides interesting insights. The test to production code ratio is utilized in other works and studies as well. While it is not possible to define a correct and definite number for this metric across all projects and languages, it is commonly regarded to be at least *1:1* i.e. for each line of production code there should be one line of test code. [2] With this number in mind, nearly every project using statically typed languages that were analysed in this study would be undertested. Especially for a heavily used project in the software industry like the Moby (Docker) technology, this implies that important tools of a company's infrastructure are based on undertested software.

V. THREATS TO VALIDITY

The validity of this study rests on the diverse selection of languages and projects for its analysis. However, there are some potential threats to its validity. Firstly, the decision whether a certain piece of code should be counted as production or test code is not always easy. Some projects ship small libraries that are exclusively used in the test cases but do not expose patterns of test code and were therefore counted as production code in this study. Furthermore, the test to production code ratio might not be sufficient to come to conclusions concerning the actual amount of testing that the project employs. Different types of testing will result in different amounts of test coverage. While unit tests produce a lot of code but do not provide a high level of test coverage, integration tests cover larger parts of code.

VI. RELATED WORK

Dynamic and static typing have been compared in numerous works before, however, no works were found that focused as specifically on the comparison of testing as this study does. Generally, the results of the studies performed on dynamic vs static typing vary.

Stefan Hanenberg performed an experiment that evaluated the development time and quality of a programming task performed by two distinct groups. One group was using a dynamically typed language while the other group employed static typing. As a result, static typing had a negative impact on the development time of the task and did not show any significant difference in the quality of the solution that was implemented. [3]

Another study, performed by Ray et al. analysed 729 projects and tried to find various effects of language choices on the quality of software. Among the categories with which they labeled programming languages was dynamic and static typing. While not significant, the researchers found that projects using statically typed languages were less prone to have defects than dynamically typed languages. [4]

VII. CONCLUSION AND FUTURE WORK

In this paper, a study was presented which analysed multiple open-source repositories and compared the ratio of test to production code. The study was able to show that projects using dynamically typed languages had a significantly higher ratio of test to production code than projects written with statically typed languages. An interesting future experiment could verify and analyse the results of this study on a deeper level. By combining additional metrics like test coverage, it would be more clear whether projects using static typing do tend to be more undertested than projects with dynamic typing. It could also be studied whether this difference in the number of tests reflects on the amount of bugs that each system had. Finally, a survey could shed some light for the reasons of the considerably smaller amount of tests in projects with static typing. Why do developers using statically typed languages test less? Might they be relying too much on their type system?

REFERENCES

[1] L. Cardelli, "Type systems," in *Handbook of Computer Science and Engineering*, 2nd ed. CRC Press, 2004, ch. 97.

[2] A. van Deursen, L. Moonen, A. van den Bergh, and G. Kok, "Refactoring test code," in *Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001)*, 2001.

[3] S. Hanenberg, "An experiment about static and dynamic type systems: Doubts about the positive impact of static type systems on development time," in *ACM Sigplan Notices*, vol. 45, no. 10. ACM, 2010, pp. 22–35.

[4] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in github," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 155–165.