

**ANALYZING THE IMPACT ON IDENTIFIABLE
DEFECTS IN A CODE INDEPENDENT OF
TYPESCRIPT**



G.M. Deshan Madurajith
(199343H)

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

June 2022

**ANALYZING THE IMPACT ON IDENTIFIABLE
DEFECTS IN A CODE INDEPENDENT OF
TYPESCRIPT**

G.M. Deshan Madurajith
(199343H)

Thesis/Dissertation submitted in partial fulfilment of the requirements for the degree
Master of Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

June 2022

DECLARATION OF THE CANDIDATE & SUPERVISOR

I declare that this is my own work and this thesis does not incorporate without acknowledgment of any material previously submitted for a Degree or Diploma in any other universities or institutes of higher learning and to the best of my knowledge and belief does not contain any material previously published or written by another person except where the acknowledgment is made in the text.

Also, I hereby grant to the University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic, or another medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name of Candidate: G.M. Deshan Madurajith

Signature:

Date:

The above candidate has carried out research for the Master' thesis under my supervision.

Name of the supervisor: Prof. Indika Perera

Signature:

Date:

Abstract

Javascript has become the most widespread programming language used to build software for many platforms, including web, mobile, backends, hardware, and desktop applications. It is a dynamically typed programming language that gives freedom to ignore types and build applications quickly. Still, Javascript is a poor language for identifying bugs at the compile time and maintaining large applications because of the dynamic behavior. The developers can use Typescript to add type syntaxes on top of JavaScript as a solution.

However, not enough empirical studies exhibit how Typescript impacts detecting defects in applications. We decided to follow an empirically quantified "what-if" style of experimentation. We collected merged javascript bug-fix pull requests from selected open-source projects. Then we selected candidate bugs by applying our predefined criteria such as discarded pull requests with more than five files, code-refactoring, and configuration changes. We manually added Typescript annotation to the buggy code base and checked whether Typescript could detect the defects at the compile type. We assessed 500 bug-fix pull requests over five projects and identified that using Typescript over Javascript could have prevented 22.7% of bugs.

According to our literature review, this is one of the few studies related to Typescript identifying bug impact. We believe this study will be significant evidence to consider using Typescript over Javascript in the future to reduce the significant number of bugs. This result will influence developers to adapt to the Typescript from Javascript. However, Typescript is not a silver bullet for Javascript because developers have to add extra code and complexity to the codebase. So, More research on Typescript is needed. In the future, we plan to explore the cognitive complexity of applying Typescript and the number of required lines to annotate. Furthermore, we plan to use a Typescript converter for annotating to increase the number of sample bugs to analyze.

Keywords: Typescript, Javascript, static typed, dynamic typed

ACKNOWLEDGEMENT

First, I am indebted to my thesis advisor, Prof. Indika Perera. He helped to finalize the scope and advised me on how the contents and research should be. I also thank all the lecturers who teach me because those subjects helped me a lot to choose this research. Finally, yet importantly, I want to mention the role of my parents in my journey. They add unconditional support to manage my career & studies in a free mindset.

TABLE OF CONTENTS

DECLARATION OF THE CANDIDATE & SUPERVISOR.....	ii
Abstract	iii
ACKNOWLEDGEMENT	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	vii
LIST OF TABLES	ix
1. INTRODUCTION.....	1
1.1 Motivation	2
1.2 Aim and objective	3
1.3 Scope	4
1.4 Structure of the thesis	5
2. LITERATURE REVIEW.....	6
2.1 Background	6
2.2 Bug related studies	6
2.2.1 BUGSJS: A benchmark and taxonomy of javascript bugs	6
2.2.2 To type or not to type: quantifying detectable bugs in Javascript.....	10
2.2.3 Qualitative methods in empirical studies of software engineering	13
2.2.4 Discovering bug patterns in JavaScript.....	14
2.2.5 To Type or Not to Type - A Systematic Comparison	21
2.3 Bug collections & classifications techniques	26
2.4 Quantifying and analyzing detectable bugs	29
3. METHODOLOGY	33
3.1 Selecting projects and configuration	33
3.2 Selecting candidate bugs	36

3.4 Typescript annotation	38
3.5 Taxonomize & analyzing	42
4. RESULTS & DISCUSSION	47
4.1 Challenges in identifying gugs	47
4.2 Typescript configuration impacts on results	48
4.3 Results of selected pull requests.....	50
4.3.1 ChartJS	50
4.3.2 ThreeJS	50
4.3.3 Create react app	51
4.3.4 Express	51
4.3.2 Lodash	52
4.5 Results and discussion.....	53
5. SUMMARY AND CONCLUSION.....	57
References	59

LIST OF FIGURES

Figure 1.1 Javascript vs Typescript trends. Adapted from [13]	3
Figure 1.2 Flow-bin vs Typescript trends. Adapted from [15]	4
Figure 2.1 Overview of the bug selection and inclusion process. Adapted from [19].	7
Figure 2.2 Bug-fixing commits inclusion criteria. Adapted from [19]	7
Figure 2.3 Taxonomy of bugs in the benchmark of Javascript programs of bugs. Adapted from [19]	9
Figure 2.4 Formular. Adapted from [6].....	10
Figure 2.5 TS detectable bug chart. Adapted from [6].....	11
Figure 2.6 Typescript annotation code. Adapted from [6]	12
Figure 2.7 Undetectable bugs from Flow and TypeScript. Adapted from [6]	12
Figure 2.8 Basic Change Type Extraction. Adapted from [17]	16
Figure 2.9 Protect with false check. Adapted from [17]	19
Figure 2.10 Protect with no value check.. Adapted from [17].....	19
Figure 2.11 Protect with type check. Adapted from [17].....	19
Figure 2.12 Incorrect Comparison – I. Adapted from [17]	19
Figure 2.13 Incorrect Comparison – II. Adapted From [17].....	19
Figure 2.14 Missing argument. Adapted from [17]	20
Figure 2.15 Incorrect API Config. Adapted from [17]	20
Figure 2.16 "this" not correctly bound. Adapted from [17].....	20
Figure 2.17 Try Caught. Adapted from [17]	20
Figure 2.18 Callback error. Adapted from [17]	21
Figure 2.19 Callback error exception. Adapted from [17].....	21
Figure 2.20 Block diagram of proposed methodology. Adapted from [33].....	30
Figure 2.21 Entropy calculator for summary and comment. Adapted from [33].....	30
Figure 2.22 Formula I. Adapted from [33].....	31
Figure 2.23 Formula II. Adapted from [33]	32
Figure 3.1 Typescript Default Configuration	36
Figure 3.2 Bug Collection Process	37
Figure 3.5 Code Conversion and validation methodology.....	38
Figure 3.6 After applying buggy js change	40
Figure 3.7 After applying typescript code annotation	40

Figure 3.8 Github Sample Pull Request	41
Figure 3.9 Conditional error	42
Figure 3.10 Heavy depend on utils.....	43
Figure 3.11 Not a bug.....	43
Figure 3.12 Unused	44
Figure 3.13 Global variable.....	44
Figure 3.14 Type error	45
Figure 3.15 Github labels	45
Figure 3.4 Final Results Sheet.....	46
Figure 4.1 Merged VS Total PR.....	47
Figure 4.2 Code without strick nullcChecks	48
Figure 4.3 Code with strict null checks.....	49
Figure 4.4 Default tsconfig.json	49
Figure 4.5 Chart JS.....	50
Figure 4.6 ThreeJS	50
Figure 4.7 Create React App	51
Figure 4.8 Express JS	52
Figure 4.9 Lodash.....	52
Figure 4.11 Typescript Bug – Stringify. Adapted from [37]	54
Figure 4.12 Typescript Bug Reproduce – Stringify	54
Figure 4.13 Comparison Chart	55
Figure 4.15 Typescript found vs not found bugs	56

LIST OF TABLES

Table 2.1 Chante Type Inspection. Adapted from [17].....	18
Table 2.2 Null hypotheses with their alternatives for both RQs. Adapted from [26]	22
Table 2.3 Properties for JavaScript (JS) and TypeScript (TS). Adapted from [26] ...	23
Table 2.4 Share of the primary programming language (PL). Adapted from [26]	23
Table 2.5 Code smells for JavaScript. Adapted from [26].....	24
Table 2.6 Cognitive complexity. Adapted from [26]	24
Table 2.7 Cognitive complexity. Adapted from [26]	25
Table 2.8 Bug resolution. Adapted from [26]	25
Table 2.9 Descriptive statistics on any type usage. Adapted from [26].....	25
Table 2.10 Summary of hypothesis testing results.....	26
Table 3.1 Selected projects.....	34
Table 4.1 Reviewed vs potential TS fixable PR.....	53
Table 4.2 Comparison Table	55