

# UC Irvine

## UC Irvine Electronic Theses and Dissertations

### Title

Bug Report Quality Prediction and the Impact of Including Videos on the Bug Reporting Process

### Permalink

<https://escholarship.org/uc/item/9tn4t5qs>

### Author

Paikari, Elahe

### Publication Date

2023

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA,  
IRVINE

Bug Report Quality Prediction and  
the Impact of Including Videos on the Bug Reporting Process

DISSERTATION

submitted in partial satisfaction of the requirements  
for the degree of

DOCTOR OF PHILOSOPHY

in Software Engineering

by

Elahe Paikari

Dissertation Committee:  
Professor André van der Hoek, Chair  
Assistant Professor Iftekhar Ahmed  
Assistant Professor Joshua Garcia

2023



## DEDICATION

To my parents,

Sima and Mahmoud,

who always inspire, support, and love me.

To my partner and best friend,

Ali,

who believes in me and encourages me to grow into a more capable person.

To my siblings,

Arash, Azadeh, and Elham,

who are my amazing role models.

To my nephews,

Raybod, Adrian, and Soren,

who introduced me to a new kind of love, a precious one that never ends.

To my dog,

Milo,

who adds just the right amount of chaos and joy to my life.



# TABLE OF CONTENTS

|  | Page |
|--|------|
| LIST OF FIGURES  | vi   |
| LIST OF TABLES   | ix   |
| ACKNOWLEDGEMENTS   | xi   |
| VITA   | xii  |
| ABSTRACT OF THE DISSERTATION   | xiv  |
| 1 CHAPTER 1: INTRODUCTION  | 1    |
| 1.1 Dissertation Structure   | 10   |
| 2 CHAPTER 2: BACKGROUND  | 12   |
| 2.1 Terminology  | 12   |
| 2.1.1 Bug  | 12   |
| 2.1.2 Bug Report   | 12   |
| 2.1.3 Bug Tracking System  | 13   |
| 2.1.4 Bug Report Fields  | 13   |
| 2.1.5 Discussions  | 19   |
| 2.1.6 Time to Resolve  | 22   |
| 2.1.7 Machine Learning   | 22   |
| 2.1.8 Text Preprocessing   | 26   |
| 2.1.9 Bag of Words   | 26   |
| 2.1.10 TF-IDF  | 27   |
| 2.1.11 Statistical Tests   | 27   |
| 2.2 Quality of Bug Report  | 27   |
| 2.2.1 Attributes of High-Quality Bug Reports   | 28   |
| 2.2.2 Missing Information and Clarification  | 29   |
| 2.2.3 Techniques for Predicting the Quality of Bug Reports                               | 30   |
| 2.2.4 Bug Resolution Time  | 33   |
| 2.2.5 Triaging   | 34   |
| 2.2.6 Video Submissions in Bug Reports   | 39   |
| 3 CHAPTER 3: Bug Report Quality Prediction: Actionable versus Non-Actionable Bug Reports | 42   |
| 3.1. Actionable and Non-actionable Bug Report  | 43   |
| 3.2. Methodology   | 51   |
| 3.1.1 Data Collection  | 53   |
| 3.1.2 Manual Labeling  | 54   |
| 3.1.3 Text Mining  | 56   |
| 3.1.4 Machine Learning Classifiers   | 57   |
| 3.1.5 Cross-Project Prediction   | 58   |
| 3.1.6 Survey   | 59   |

|       |   |     |
|-------|---|-----|
| 3.2   | Results and Discussion  | 60  |
| 3.2.1 | Predicting Actionable versus Non-Actionable   | 60  |
| 3.2.2 | Additional Features   | 65  |
| 3.2.3 | Cross-Project Prediction  | 76  |
| 3.2.4 | Survey Results  | 78  |
| 3.3   | Discussion  | 83  |
| 3.4   | Threats to Validity   | 86  |
| 3.5   | Conclusion  | 88  |
| 4     | CHAPTER 4: An Analysis of Video Submissions in Bug Reporting                              | 90  |
| 4.1   | Methodology   | 92  |
| 4.1.1 | Data Collection   | 93  |
| 4.1.2 | Identifying Videos  | 94  |
| 4.1.3 | Collecting Bug Report Details   | 94  |
| 4.1.4 | Statistical Analysis  | 97  |
| 4.1.5 | Mozilla-Specific Data Collection  | 98  |
| 4.2   | Results   | 100 |
| 4.2.1 | Prevalence  | 100 |
| 4.2.2 | Potential Benefits to Reporters   | 103 |
| 4.2.3 | A Deeper Dive into Mozilla  | 135 |
| 4.3   | Discussion  | 148 |
| 4.4   | Threats to Validity   | 151 |
| 4.5   | Conclusions   | 152 |
| 5     | CHAPTER 5: A Content-Based Analysis of Video Submissions in Bug Reporting                 | 154 |
| 5.1   | Methodology   | 155 |
| 5.1.1 | Data Collection   | 156 |
| 5.1.2 | Manual Labeling   | 156 |
| 5.1.3 | Statistical Analysis  | 161 |
| 5.1.4 | Survey  | 162 |
| 5.2   | Results   | 163 |
| 5.2.1 | Video Characteristics   | 163 |
| 5.2.2 | Impact on Videos Appearing to Be Perceived as Helpful by Developers                       | 167 |
| 5.2.3 | Impact on Bug Report Resolution Process   | 178 |
| 5.2.4 | Impact of video being perceived as helpful by developers on Bug Report Resolution Process | 197 |
| 5.2.5 | Survey  | 200 |
| 5.3   | Discussion  | 202 |
| 5.4   | Threats to Validity   | 209 |
| 5.5   | Conclusions   | 209 |
| 6     | CHAPTER 6: CONCLUSION   | 212 |

|     |  |     |
|-----|--|-----|
| 6.1 | Study 1 – Predicting Actionable versus Non-Actionable Bug Reports        | 213 |
| 6.2 | Study 2 – An Analysis of Video Submissions in Bug Reporting              | 213 |
| 6.3 | Study 3 – A Content-Based Analysis of Video Submissions in Bug Reporting | 215 |
| 6.4 | Contributions  | 216 |
| 6.5 | Future work  | 217 |
|     | Bibliography   | 220 |

## LIST OF FIGURES

|  | Page |
|--|------|
| Figure 1. Example of an actionable bug report (cropped).   | 3    |
| Figure 2. Example of a non-actionable bug report (cropped).  | 5    |
| Figure 3. Mozilla custom bug entry form.   | 14   |
| Figure 4. The life cycle of a bug report in Bugzilla.  | 17   |
| Figure 5. Comparison between the number of comments and back-and-forth in Mozilla bug id 1790809 (cropped).                                | 21   |
| Figure 6. Example of an actionable bug report, bug id 544833 (cropped).  | 44   |
| Figure 7. Comments from bug id 544833 (cropped).   | 45   |
| Figure 8. Example of a non-actionable bug report, bug id 1497399 (cropped).  | 46   |
| Figure 9. Comments from bug id 1497399 (cropped).  | 47   |
| Figure 10. Example of a more difficult bug report to label, bug id 1265173 (cropped).  | 48   |
| Figure 11. Comments from bug id 1265173 (cropped).   | 49   |
| Figure 12. Example of a more difficult bug report to categorize, bug id 621660 (cropped).  | 50   |
| Figure 13. Comments from bug id 621660 (cropped).  | 51   |
| Figure 14. Process of classifying the bug reports.   | 52   |
| Figure 15. Percentage of bug reports which have attachments.   | 68   |
| Figure 16. Average readability score of bug descriptions.  | 69   |
| Figure 17. Percentage of bug reporters who are developers.   | 70   |
| Figure 18. Average length of bug descriptions.   | 71   |
| Figure 19. Average length of bug descriptions submitted by developers and end-users.   | 72   |
| Figure 20. Average experience of bug reporters.  | 73   |
| Figure 21. Percentage of Actionable Bug Reports over 18 Years.   | 80   |
| Figure 22. Percentage of Bug Reports that Contained “Steps to Reproduce:”, “Actual Results:”, and “Expected Results:”.                     | 83   |
| Figure 23. Overall process of studying the role of videos in bug reporting.  | 92   |
| Figure 24. Additional steps to study the role of videos in Mozilla bug reports.  | 99   |
| Figure 25. Percentage of bug reports with video attachments.   | 101  |
| Figure 26. Difference between percentage of bug reports that were submitted initially and percentage submitted later.                      | 102  |
| Figure 27. Comparing the average days to resolve bug reports with and without video attachments.   | 105  |
| Figure 28. Difference between average number of days to resolve bug reports with videos attached initially and with videos attached later. | 106  |
| Figure 29. Difference between percentage of bug reports with and without video attachments that were successfully resolved with a patch.   | 116  |

|   |     |
|---|-----|
| Figure 30. Difference between average number of back-and-forth in bug reports with and without video attachments.   | 125 |
| Figure 31. Difference between average number of back-and-forth for bug reports with videos submitted initially and those submitted later.   | 126 |
| Figure 32. Categorization of videos. Note that “reporter” is listed twice so to distinguish reporters as developers versus end-users for videos that were initially submitted versus later submitted. | 136 |
| Figure 33. Example of video submitted later because of developer’s request, bug id 1517183 (cropped).   | 137 |
| Figure 34. Example of video submitted later voluntarily by bug reporter, bug id 1636769 (cropped).  | 138 |
| Figure 35. Comments from bug id 1636769 (cropped).  | 139 |
| Figure 36. Who attached videos later in the process.  | 140 |
| Figure 37. Percentage of developers and end-users who submitted video attachments later in the resolution of bug reports.   | 141 |
| Figure 38. Percentage of bug reporters who are developers or end-users.   | 141 |
| Figure 39. Comparing the average days to resolve of UI bug reports with and without video attachments.  | 143 |
| Figure 40. Percentage of UI bug reports with and without video attachments that were successfully resolved with a patch.  | 144 |
| Figure 41. Average number of back-and-forth in UI bug reports with and without video attachments.   | 145 |
| Figure 42. Severity of bug reports with and without video attachments.  | 146 |
| Figure 43. The overall process of studying the role of different characteristics of videos on videos appearing to be perceived as helpful by developers and bug report resolution process.            | 156 |
| Figure 44. Percentage of bug reports with videos that appeared to be perceived as helpful and not helpful.  | 168 |
| Figure 43. Percentage of bug reports with videos appeared to be perceived as helpful with and without annotation.   | 169 |
| Figure 46. Percentage of bug reports with videos appeared to be perceived as helpful with different lengths.  | 171 |
| Figure 47. Percentage of bug reports with videos appeared to be perceived as helpful with and without showing steps to reproduce.   | 172 |
| Figure 48. Percentage of bug reports with videos perceived as helpful with and without showing actual results.  | 174 |
| Figure 49. Comparing the average number of days to resolve bug reports with videos with and without different characteristics.  | 181 |
| Figure 50. Comparing the percentage of bug reports with videos with different characteristics being resolved with a patch.  | 187 |

|   |     |
|---|-----|
| Figure 51. Comparing the average number of back-and-forth in bug reports with videos with different video characteristics.                            | 193 |
| Figure 52. The average number of days to resolve bug reports with video that appeared to be perceived as helpful versus not helpful.                  | 198 |
| Figure 53. Percentage of bug reports with videos appeared to be perceived as helpful versus not helpful that were successfully resolved with a patch. | 199 |
| Figure 54. The average number of back-and-forth in bug reports with videos perceived as helpful versus not helpful.                                   | 200 |

## LIST OF TABLES

|  | Page |
|--|------|
| Table 1. Performance of the classifiers.   | 60   |
| Table 2. Results per class (hyper-parameter optimization).   | 62   |
| Table 3. Accuracy of the best performing model and Cuezilla's model.   | 63   |
| Table 4. Feature importance (sorted by relative importance).   | 64   |
| Table 5. Performance of the SVN + tuning model augmented with additional features, top six and bottom six.                           | 66   |
| Table 6. Average number of back-and-forth exchanges over 18 years.   | 74   |
| Table 7. Performance results of cross-project training (trained on Mozilla).   | 77   |
| Table 8. Portability with alternative label mappings.  | 78   |
| Table 9. Results of the survey.  | 79   |
| Table 10. Experimental Dataset.  | 93   |
| Table 11. Excerpt of some relevant bug report data.  | 95   |
| Table 12. Generalized Linear Model predicting the impact on average number of days to resolve bug reports.                           | 108  |
| Table 13. Generalized Linear Model predicting the impact on average number of days to resolve Android bug reports.                   | 109  |
| Table 14. Generalized Linear Model predicting the impact on average number of days to resolve IntelliJ bug reports.                  | 111  |
| Table 15. Generalized Linear Model predicting the impact on average number of days to resolve LibreOffice bug reports.               | 112  |
| Table 16. Generalized Linear Model predicting the impact on average number of days to resolve Minecraft bug reports.                 | 113  |
| Table 17. Generalized Linear Model predicting the impact on average number of days to resolve Mozilla bug reports.                   | 114  |
| Table 18. Generalized Linear Model predicting percentage of bug reports with and without videos being resolved as FIXED.             | 117  |
| Table 19. Generalized Linear Model predicting percentage of Android bug reports with and without videos being resolved as FIXED.     | 119  |
| Table 20. Generalized Linear Model predicting percentage of IntelliJ bug reports with and without videos being resolved as FIXED.    | 120  |
| Table 21. Generalized Linear Model predicting percentage of LibreOffice bug reports with and without videos being resolved as FIXED. | 121  |
| Table 22. Generalized Linear Model predicting percentage of Minecraft bug reports with and without videos being resolved as FIXED.   | 122  |
| Table 23. Generalized Linear Model predicting percentage of Mozilla bug reports with and without videos being resolved as FIXED.     | 123  |

|  |     |
|--|-----|
| Table 24. Generalized Linear Model predicting the impact on average back-and-forth by the number of videos included in a bug report.   | 128 |
| Table 25. Generalized Linear Model predicting the impact on average back-and-forth by the number of videos included in a Android bug report.   | 129 |
| Table 26. Generalized Linear Model predicting the impact on average back-and-forth by the number of videos included in an IntelliJ bug report.                                       | 130 |
| Table 27. Generalized Linear Model predicting the impact on average back-and-forth by the number of videos included in a LibreOffice bug report.                                     | 131 |
| Table 28. Generalized Linear Model predicting the impact on average back-and-forth by the number of videos included in a Minecraft bug report.                                       | 133 |
| Table 29. Generalized Linear Model predicting the impact on average back-and-forth by the number of videos included in a Mozilla bug report.   | 134 |
| Table 30. Count and percentage of the videos that contained different bug details.   | 164 |
| Table 31. Count and percentage of the videos with different type of sounds.  | 165 |
| Table 32. Count and percentage of the videos with different lengths.   | 166 |
| Table 33. Generalized Linear Model predicting the percentage of videos appeared to be perceived as helpful by developers by different video characteristics included in the video.   | 177 |
| Table 34. Generalized Linear Model predicting the impact on the average number of days to resolve bug reports by different video characteristics included in the video.              | 184 |
| Table 35. Generalized Linear Model predicting the impact on the percentage of bug reports being resolved as FIXED by different video characteristics included in the attached video. | 190 |
| Table 36. Generalized Linear Model predicting the impact on the average number of back-and-forth by different video characteristics included in the video.                           | 196 |
| Table 37. Observations derived from the analyses of video characteristics.   | 205 |



## **ACKNOWLEDGEMENTS**

I would like to thank my advisor, Professor André van der Hoek, for his great supervision, insightful comments and continuous support during my PhD studies and research. His guidance made all of this possible.

I thank Professor Iftekhar Ahmed. His insightful and helpful feedback throughout my PhD studies shaped the ideas and experiments in this thesis for the better.

I also thank my committee member Professor Joshua Garcia for his valuable comments which pushed me to think harder than before and forced me to reflect on my research more deeply.

## **VITA**

### **Elahe Paikari**

#### **EDUCATION**

|           |   |
|-----------|---|
| 2017-2023 | Ph.D. in Software Engineering<br>University of California – Irvine, CA, USA |
| 2012-2014 | M.S. in Software Engineering<br>University of Calgary, AB, Canada           |
| 2008-2012 | B.S. in Software Engineering<br>Azad University, Tehran, Iran               |

#### **PROFESSIONAL EXPERIENCE**

|             |   |
|-------------|---|
| Summer 2022 | AI Software Engineer, MIT-IBM Watson AI Lab       |
| 2015-2017   | Software Developer, Intelliview Technologies Inc. |
| 2012–2015   | Software Developer, University of Calgary         |

#### **REFERRED PUBLICATIONS**

L. Martie, J. Rosenberg, V. Demers, G. Zhang, O. Bhardwaj, J. Henning, A. Prasad, M. Stallone, J. Lee, L. Yip, D. Adesina, E. Paikari, O. Resendiz, S. Shaw, D. Cox, “Rapid Development of Compositional AI”, 45th International Conference on Software Engineering: New Ideas and Emerging Results Track (ICSE-NIER), 2023.

E. Paikari, I. Ahmed and A. Hoek, “Does Submitting Videos alongside Bug Reports Benefit Reporters? An Empirical Study”, Submitted to The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2023.

E. Paikari, J. Choi, S. Kim, S. Baek, M. Kim, S. Lee, C. Han, Y. Kim, K. Ahn, C. Cheong, and A. Hoek, “A Chatbot for Conflict Detection and Resolution”, 1<sup>st</sup> International Workshop on Bots in Software Engineering (BotSE’19), 2019, pp. 1-4.

E. Paikari and A. Hoek, “A Framework for Understanding Chatbots and their Future”, Proceedings of 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE’18), 2018, pp. 1-4.

E. Paikari, M. Moshirpour, R. Alhaji, and B. H. Far, "Data Integration and Clustering for Real Time Crash Prediction", 15th IEEE International Conference on Information Reuse and Integration (IRI), 2014, pp. 1-8.

E. Paikari and B. H. Far, "Analysis, Design and Implementation of an Agent Based System for Simulating Connected Vehicles", 26th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2014, pp. 1-6.

E. Paikari, S. Tahmasseby, and B. H. Far, "A Simulation-based Benefit Analysis of Deploying Connected Vehicles Using Dedicated Short-Range Communication", IEEE Intelligent Transportation Systems Society on Intelligent Vehicles (IV) Symposium, 2014, pp. 1-6.

E. Paikari, L. Kattan, S. Tahmasseby, and B. H. Far, "Modelling and simulation of advisory speed and re-routing strategies in connected vehicles systems for crash risk and travel time reduction", 26th Annual IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), 2013, pp. 1-4.

# **ABSTRACT OF THE DISSERTATION**

Bug Report Quality Prediction and  
the Impact of Including Videos on the Bug Reporting Process

by

Elahe Paikari

Doctor of Philosophy in Software Engineering

University of California, Irvine, 2023

Professor André van der Hoek, Chair

Many newly-submitted bug reports are not actionable: they do not have sufficient and clear information for developers to start the process of understanding and fixing them. The presence of such non-actionable bug reports leads to: (1) a waste of developers' time as they have to come up with the right questions to ask bug reporters in order to understand the essence of these bug reports, (2) slower overall progress, and (3) negative effects on the overall quality of software. Therefore, effective bug reporting is crucial, especially in projects where a large number of bug reports are submitted on a regular basis and a significant portion of the overall software development lifecycle is spent addressing these reports.

To address this issue, the dissertation first looks at understanding the overall quality of bug reports, contributing the implementation of a model that classifies bug reports as actionable or non-actionable. Second, through an empirical study of 2,814,599 bug reports across five systems, the dissertation answers the question of whether the inclusion of videos in bug reports may lead to tangible potential incentives for the reporters (i.e., reduced time to resolution, leading to an actual fix, or reduced number of

back-and-forth). Third, the dissertation examines whether certain characteristics (e.g., the presence of voice over, clear highlighting with the mouse, a video contains steps to reproduce) might differentiate videos that have a positive impact on the bug resolution process from those that do not, by studying how developers react to videos with different characteristics and whether these different characteristics have potentially observable benefits.

The main contributions of this dissertation include: (1) a machine learning model that with a 92% accuracy (F-measure of 0.91) separates actionable from non-actionable bug reports, which is significantly higher than the best results to date, (2) new findings concluding that the inclusion of videos in bug reports does not always translate to tangible benefits for reporters: bug resolution time is barely impacted, the percentage of bug reports being successfully resolved with a patch is lower for bug reports with videos, and back-and-forth is higher for bug reports with videos, and (3) new findings about videos with different characteristics: bug reports with videos that are less than 30 seconds or contain actual results appear to have observable benefits for bug reporters. The findings can serve as a basis for future research and developing tools that assist reporters in improving their bug reports before final submission, either by helping reporters to turn their non-actionable bug reports into actionable ones or by supporting them in attaching helpful videos exhibiting the characteristics found to be beneficial to both developers and bug reporters.

# 1 CHAPTER 1: INTRODUCTION

A bug report refers to a report filed manually by an end-user or a tester of software about a fault or defect [1]. Bug reports are essential in software development because they allow users to inform developers of the problems encountered while using a software. Bug reports typically contain a detailed description of a software failure and ideally hint at the location of the fault in the code (in the form of patches or stack traces) [2].

Bug reports vary in their quality of content; they often provide inadequate or incorrect information [3], [4]. A possible reason could be that reporters have different levels of experience or knowledge about the underlying software that has a bug and what they need to tell the assigned developer about the bug in order to resolve it [5], [6]. It could also be that the information, which is most useful for a developer resolving a bug report (e.g., steps to reproduce, actual results, and test cases) is often the most difficult information for reporters to provide [7], [8], [9]. Some researchers identified that inadequate tool support is one of the main reasons for inaccurate bug reports [7], [10].

With the rapid development of software (i.e., especially open source software), vast amounts of bug reports have been produced [11] and research shows that the lack of important information in bug reports is a main reason for low quality bug reports [12]. The presence of such low quality bug reports lead to developers having to spend more time and effort understanding bug descriptions and seek significant clarification or important additional information [13], [14]. Breu et al. [15] explored the questions asked during the resolution of bug reports and found that a significant proportion of these were related to missing or inaccurate information. Yet, when the developers ask reporters to clarify certain aspects of their reports, they do not always receive a response [15]. This in turn leads to non-reproduced bugs [7], unfixed bugs [16], and additional bug triage effort [7], [15]–[17].

A good bug report is a key ingredient to effective bug triaging and fixing [18]. As a result, understanding what makes for good bug reports has been the subject of extensive study. For instance, a number of researchers have studied the key elements that contribute to the quality of the bug reports being submitted [14], [19] by surveying developers working on different systems (e.g., Apache, Eclipse, and Mozilla). Their results show that developers

considered information such as code examples or an explanation of the difference between the expected and observed behaviors (e.g., [20], [21]) as important (while reporters only add stack traces and/or test cases to the bug reports).

Figure 1 shows extracts from Mozilla bug id 537507<sup>1</sup>, which is an example of a good (or in terminology of this dissertation actionable) bug report. The description of the report contains enough details and clear information about what happened, what the user expected to happen, and how to reproduce it (Figure 1 (part 1)). More importantly, the developer could understand and submit a patch that fixes the bug without asking for any significant additional information or clarifications (Figure 1 (part 2)).

---

<sup>1</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=537507](https://bugzilla.mozilla.org/show_bug.cgi?id=537507)

**Closed** Bug 1570935 Opened 4 years ago Closed 4 years ago

### 4.64 - 9.58% tart (linux64-shippable, windows10-64-shippable, windows7-32-shippable) regression on push 70057a7a799382a0ad8e1a3938a4f482ff686d05 (Wed July 31 2019)

- Categories (Firefox :: Messaging System, defect, P1)
- Tracking (regression bug RESOLVED as FIXED)
- People (Reporter: alexandru, Assigned: andreio)
- References (Regression)
- Details (Keywords: perf, regression, talos-regression)
- Attachments (1 file)

Bottom ↓ Tags Timeline ▾

**Alexandru Ionescu (needinfo me) [alexandru]** Reporter  
Description • 4 years ago

Talos has detected a Firefox performance regression from push:  
<https://hg.mozilla.org/integration/autoland/pushloghtml?changeset=70057a7a799382a0ad8e1a3938a4f482ff686d05>

As author of one of the patches included in that push, we need your help to address this regression.

Regressions:

- 10% tart linux64-shippable opt e10s stylo 2.03 -> 2.22
- 6% tart windows7-32-shippable opt e10s stylo 2.61 -> 2.77
- 6% tart windows7-32-shippable opt e10s stylo 2.61 -> 2.77
- 5% tart windows10-64-shippable opt e10s stylo 2.64 -> 2.76

You can find links to graphs and comparison views for each of the above tests at: <https://treeherder.mozilla.org/perf.html#/alerts?id=22250>

On the page above you can see an alert for each affected platform as well as a link to a graph showing the history of scores for this test. There is also a link to a treeherder page showing the Talos jobs in a pushlog format.

To learn more about the regressing test(s), please see: [https://wiki.mozilla.org/Performance\\_sheriffing/Talos/Tests](https://wiki.mozilla.org/Performance_sheriffing/Talos/Tests)

For information on reproducing and debugging the regression, either on try or locally, see:  
[https://wiki.mozilla.org/Performance\\_sheriffing/Talos/Running](https://wiki.mozilla.org/Performance_sheriffing/Talos/Running)

\*\*\* Please let us know your plans within 3 business days, or the offending patch(es) will be backed out! \*\*\*

Our wiki page outlines the common responses and expectations: [https://wiki.mozilla.org/Performance\\_sheriffing/Talos/RegressionBugsHandling](https://wiki.mozilla.org/Performance_sheriffing/Talos/RegressionBugsHandling)

Flags: needinfo?(andrei.br92)

1

**Andrei Oprea [andreio]** Assignee  
Comment 1 • 4 years ago

Attached file [Bug 1570935 - Remove clip-path property for causing performance regression](#) — Details

**Andrei Oprea [andreio]** Assignee  
Updated • 4 years ago

Assignee: nobody → andrei.br92  
Flags: needinfo?(andrei.br92)

**Pulsebot**  
Comment 2 • 4 years ago

Pushed by [aoprea@mozilla.com](mailto:aoprea@mozilla.com):  
<https://hg.mozilla.org/integration/autoland/rev/14c3f00eb334>  
 Remove clip-path property for causing performance regression r=k88hudson

**Dorel Luca [dluca]**  
Comment 3 • 4 years ago

**bugherder**  
<https://hg.mozilla.org/mozilla-central/rev/14c3f00eb334>

Status: NEW → RESOLVED  
 Closed: 4 years ago  
[status-firefox70](#): --- → [fixed](#)  
 Resolution: --- → [FIXED](#)  
 Target Milestone: --- → [Firefox 70](#)

2

Figure 1. Example of an actionable bug report (cropped).



On the other hand, Mozilla bug id 966595<sup>2</sup>, shown in Figure 2 (part 1) is an example of a bug report that is not good (non-actionable in the terminology of this dissertation). Even though its description seems to have all the information needed to resolve the bug, it lacks in providing what the developer needs to feel comfortable proceeding. As Figure 2 (part 2) shows, right after the bug report was submitted the developer had to ask the reporter to provide "some more information", because otherwise they are unable to reproduce the problem and process the report. Then, after a few months, in light of no response, the developer chose to close the bug report as INVALID.

---

<sup>2</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=966595](https://bugzilla.mozilla.org/show_bug.cgi?id=966595)

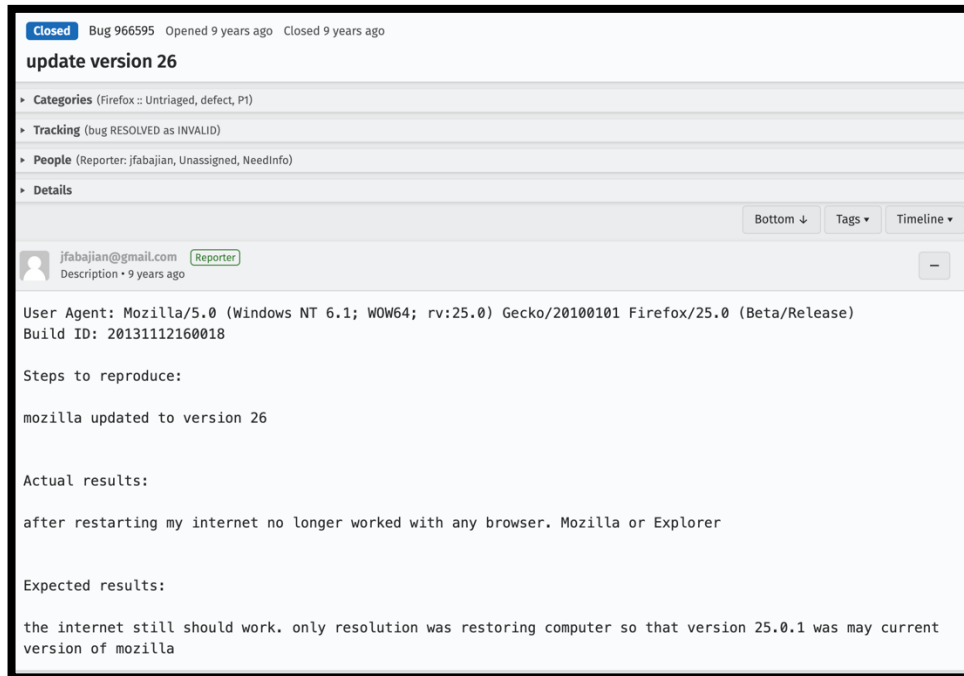


Figure 2. Example of a non-actionable bug report (cropped).

To address the problems surrounding non-actionable bug reports, many researchers have proposed potential solutions. Some studies focus on assisting in processing newly submitted bug reports by predicting, for instance, their severity or priority (e.g., [22]–[24]), whether a bug report can be resolved within a given time [25], or whether a new bug report represents a duplicate (e.g., [26]–[28]). Other research aims to assist in improving aspects of new bug reports prior to submission, for instance by predicting the values of missing data fields such as the operating system or product version [29], by identifying weak titles (e.g., [30], [31]), by flagging weak descriptions [32], by identifying whether a description contains observed behavior, expected behavior, and steps to reproduce [33], [34], or by giving an overall sense of bug report quality to the reporter together with generic suggestions for improvements [32], [35].

Some studies suggest that reporters should be encouraged to submit relevant videos as part of their bug reports to convey additional context for understanding bugs [36], [89]. Such research emphasizes that videos can visually communicate bugs and help developers comprehend any events that may have led to the manifestation of the bug [39]–[41]. In game applications, for instance, developers attempt to learn more about the problems that gamers encounter by studying gameplay videos, as they often offer important context surrounding a bug [42]. In the past few years, using videos to report bugs is also supported by many bug reporting services for mobile applications [37], [38], [43], [44], [45]. As a result, approaches have been introduced for automatically analyzing videos into replayable scenarios in order to assist developers in effectively processing and making sense of video content (e.g., [46], [47]).

This dissertation complements this continuously expanding body of bug report research by addressing the following three research questions:

**RQ1. Is it possible to predict the overall quality of bug reports in terms of whether they are actionable or non-actionable with sufficiently high accuracy?**

As repeatedly observed in prior research, the overall quality of bug reports is one of the aspects that should be improved in order to help bug reporters during the bug triaging and resolution. There is an opportunity for tools that, before final submission, could flag bug reports of low quality and guide reporters toward improving the quality of their bug reports. In order to design such tools, a classifier is needed to distinguish bug reports that are actionable and can be submitted as is from those that are non-actionable and thus need work. Unfortunately, the performance of bug report quality predictors to date is insufficient to be practically useful for such tools. Among them, Cuezilla [35] and the approach of Schuegerl et al. [32] provide the best results, with the former achieving a 50% accuracy in classifying bug reports as good, neutral, or bad, and the latter being fairly close with an accuracy of 44% in classifying reports on a scale of 1 (very high quality) to 5 (very low quality).

Therefore, this research question focuses on revisiting the question of predicting the overall quality of bug reports, though with the objective of just a binary classification: actionable versus non-actionable. To do so, 1,423 bug reports from all of the Mozilla Firefox projects

(e.g., Firefox, Firefox Build System, Firefox for FireTV) were manually classified as actionable or non-actionable, using characteristics of high-quality and low-quality bug reports from previous works as a guide (e.g., [48], [49]). Then, supervised learning was conducted using four different machine learning classifiers, namely Naive Bayes, Support Vector Machine, Decision Tree, and Random Forest with a variety of different input configurations (e.g., solely the text from the bug report, the text from the bug report with whether or not it was submitted by an end-user or developer, the text from the bug report and whether or not it has an attachment), and the one that performed best was selected.

To examine whether the best-performing model is portable (i.e., performing well on one dataset does not necessarily mean performing well on another), cross-project prediction was conducted. That is, the model with the best results from the training on bug reports from Mozilla Firefox projects was applied on an existing dataset of bug reports from Apache and Eclipse.

A short survey was conducted among Mozilla developers to complement the results of the best-performing classifier. The main goal of the survey was to understand developers' perspectives on the quality of newly submitted bug reports today as compared to the quality of newly submitted bug reports in prior years, as well as what might have caused a change in the quality being submitted according to the developers.

Together, the classifier and survey shed light on concerns for future tools that assist reporters in turning their non-actionable bug report to an actionable one before they are eventually submitted.

## **RQ2. Does the presence of videos in bug reports impact the overall bug resolution process, beyond helping developers understand the reports?**

This research question answers whether the inclusion of videos in bug reports may lead to benefits for those who report bugs. Especially since the practice of including videos in bug reports has been steadily increasing over the past years and with prior work encouraging reporters to submit videos along their bug reports to convey additional context for understanding bugs, this research question studies whether there is actually a tangible

incentive for reporters to spend the extra effort of creating and submitting videos alongside their bug reports.

If such benefits exist, they may provide an incentive for reporters to submit videos as part of their bug reports, because it would appear to increase their chances of getting their bug report resolved faster, with a higher chance of being resolved with a patch, and/or lower number of back-and-forth).

This research question is answered through a quantitative empirical study that assesses the impact of the presence of videos in bug reports on the overall resolution process. It examines 2,814,599 bug reports from five systems, namely Mozilla, Android, LibreOffice, IntelliJ, and Minecraft, and examines the impact of the presence of videos in bug reports in terms of a potential reduction in average time to resolution, a potential increment in the percentage of bug reports being resolved with a patch, and a potential reduction in the average number of back-and-forth.

In answering the research question, the analysis also includes a look at other factors that might play a role in and perhaps even mask the phenomena observed: whether it matters if videos are submitted by developers on the project or by end-users, whether the type of bug being reported with a video may have an impact, and whether the effect of including videos differs depending on the assigned priority and/or severity of the bug report. This deep dive again analyzed the observable impact of these factors in terms of time to resolution, the percentage being resolved as FIXED, and back-and-forth, as determined by examining Mozilla bug reports only.

Together, the findings give rise to a number of important research questions from the perspective of finding motivation for reporters to produce and submit videos along with their bug reports. Any additional incentive for bug reporters might increase the number of videos attached to bug reports, and thereby benefit everyone.

**RQ3. What sets videos that are more effective apart from videos that are less effective, in terms of their content?**

The third research question focuses on understanding the role of various characteristics of videos attached to bug reports on the potential benefits for bug reporters. If, for instance, bug reports with video that include certain specific characteristics (e.g., showing steps to reproduce, presence of voiceover, and highlighting with the mouse) are resolved more quickly or appeared to be perceived as helpful by developers, this may provide further guidance for reporters to produce videos with such characteristics as part of their bug reports. Moreover, if certain characteristics prove particularly helpful, it might be worthwhile to design tools that assist reporters in easily producing videos with these characteristics.

To answer this question, the contents of videos attached to 1,045 Mozilla bug reports were manually analyzed to catalogue a range of different characteristics. Then the back-and-forth after video submission were examined to understand how developers reacted publicly to the videos. Next, in a quantitative approach, the impact of videos that contain various video characteristics was assessed by correlating the presence and non-presence of each characteristic with whether or not the developers perceived the corresponding videos as helpful, as well as by comparing the time to resolution, percentage of being fixed with a patch, and amount of back-and-forth. Particularly, in terms of whether different characteristics of the video appeared to be perceived as helpful or not by developers and whether those characteristics may have observable effects on the bug report resolution process (i.e., in reducing time to resolution, leading to an actual fix, or reducing back-and-forth following a bug report submission).

To place the findings in context, the dissertation includes a survey that was conducted among IntelliJ developers. The survey was designed to get a first-hand understanding of how developers feel about different characteristics of videos as a part of bug reports. Particularly, the survey asked how developers characterize the content of the video attachments in relation to the corresponding textual descriptions and what kinds of video content they believe helps certain videos be more or less useful in understanding bugs.

Together the findings can serve as a basis for developing tools that support reporters in attaching videos with characteristics that provide incentives for both developers and bug reporters.

## **1.1 Dissertation Structure**

This dissertation is organized into six chapters. The remaining chapters are structured as follows:

### **Chapter 2 – Background**

This chapter presents an overview of the terminology used and introduces relevant background material in bug reporting research and the role of videos in bug reporting.

### **Chapter 3 – Bug Report Quality Prediction: Actionable versus Non-Actionable Bug Reports**

This chapter discusses the approach designed to answer RQ1, a classifier that can distinguish actionable from non-actionable bug reports with sufficiently high results. This chapter also presents the findings of RQ1 and places the results in the broader context of work to date. The chapter concludes with a discussion of the threats to validity, a summary of the key findings, and an outlook at future work.

### **Chapter 4 – An Analysis of Video Submissions in Bug Reporting**

This chapter presents the approach and the results from studying RQ2: whether the inclusion of video in bug reports impacts the bug resolution process in terms of time to resolution, resolution with a patch aiming to fix the reported bug, and the amount of back-and-forth. In this chapter, the key findings are also placed in the broader context of the literature to date. The chapter concludes with the threats to validity, a summary of key findings, and future work.

### **Chapter 5 – A Content-Based Analysis of Video Submissions in Bug Reporting**

This chapter presents the methodology designed to answer RQ3: what sets videos that are more effective apart from videos that are less effective, in terms of their content? The chapter

discusses the key findings and places the results in the broader context of the work to date. Then, the chapter presents the threats to validity, conclusion, and future work.

## **Chapter 6 – Conclusion**

This chapter takes a step back and holistically considers the lessons learned across all aspects of the studies presented in the prior chapters. It particularly discusses the high-level takeaways, key individual results, and overall implications. This chapter also summarizes the contributions and presents future work.



## **2 CHAPTER 2: BACKGROUND**

In this section, I first provide an overview of the bug reporting process and then discuss relevant prior research in the context of bug reports and their associated videos.

### **2.1 Terminology**

In this section, I explain key terms used in bug report analysis and describe the life cycle of bug reports.

#### **2.1.1 Bug**

A software bug is an error, flaw, or fault in a computer program or system that produces unexpected results or behavior. There is a distinction between a “bug” and an “issue”. An issue could be a bug but is not always a bug. It can also concern a feature request, task, missing documentation, and so on [2]. The process of finding and removing bugs is called “debugging”. Bugs may be caused by tiny coding errors, but the results of bugs can be serious, making finding and fixing bugs a rather challenging task.

#### **2.1.2 Bug Report**

A bug report is a software document describing a software bug and is submitted by a developer, a tester, or an end-user [50]. Bug reports have many other names, such as “defect reports”, “fault reports”, “failure reports”, “error reports”, “problem reports”, “trouble reports”, and so on [1]. Bug reports generally include some sort of description of the problem and may optionally include attachments, in the form of, e.g., patches, test cases, and screenshots.

### 2.1.3 Bug Tracking System

A bug tracking system (also called an issue tracking system) manages bug reports to support the developers who fix bugs [4]. Bug tracking systems are designed to track reported software bugs. Bugs are stored in a bug repository, which is the major component of a bug tracking system. To enable submission and the tracking of the resolution of bugs, developers of many popular open-source projects (e.g., Mozilla<sup>3</sup>, Eclipse<sup>4</sup>, and Linux kernel<sup>5</sup>) use bug tracking systems (e.g., Bugzilla<sup>6</sup>, Jira<sup>7</sup>, Mantis<sup>8</sup>). For example, Mozilla uses Bugzilla as its bug reporting system.

### 2.1.4 Bug Report Fields

Depending on the bug tracking system, a bug report contains a set of fields. Typically, a bug report has an identification number, title, date and time when it is reported, developer's information as to who is assigned to fix the bug, description of the bug, additional comments, attachments, and more. To make bug reports consistent, often default templates are provided in project repositories, where certain required or at least recommended fields are specified to be filled out by the reporters. For example, Figure 3 shows Bugzilla Helper [51], which is a custom bug entry form that guides users to include steps to reproduce, actual results, and expected results in Mozilla bug reports. Below the common fields in most bug tracking systems are introduced.

#### 2.1.4.1 Description

A bug report usually contains a detailed description of the failure. This description is a generic field that can contain various types of information such as build version, crash

---

<sup>3</sup> <https://mozilla.com/>

<sup>4</sup> <https://www.eclipse.org/>

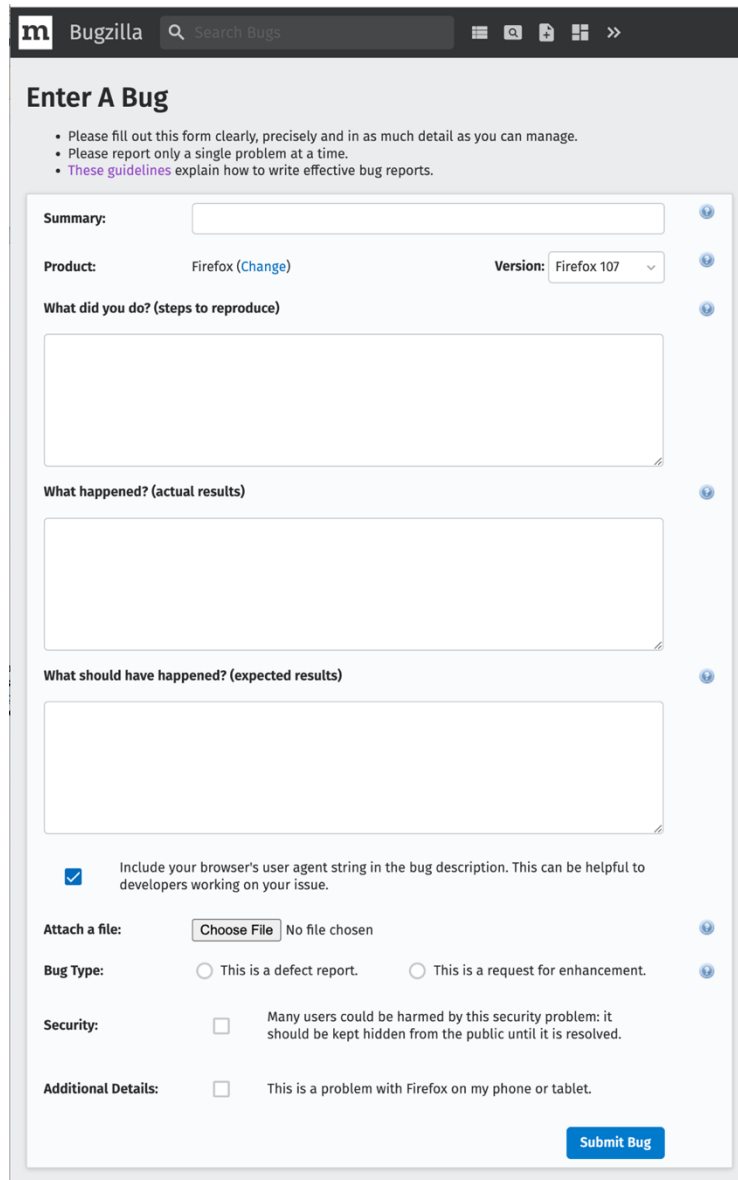
<sup>5</sup> <https://www.kernel.org/>

<sup>6</sup> <https://bugzilla.mozilla.org/>

<sup>7</sup> <https://jira.com/>

<sup>8</sup> <https://www.mantisbt.org/>

description, reproducing steps, and so on. Below are a set of fields that are recommended by previous studies to be included in the bug description:



The image shows a screenshot of the Mozilla Bugzilla 'Enter A Bug' form. The form is titled 'Enter A Bug' and includes a search bar at the top. Below the title, there are three bullet points providing instructions: 'Please fill out this form clearly, precisely and in as much detail as you can manage.', 'Please report only a single problem at a time.', and 'These guidelines explain how to write effective bug reports.' The form contains several sections: 'Summary:' with a text input field; 'Product:' set to 'Firefox' with a '(Change)' link; 'Version:' set to 'Firefox 107' with a dropdown arrow; 'What did you do? (steps to reproduce)' with a large text area; 'What happened? (actual results)' with a large text area; 'What should have happened? (expected results)' with a large text area; a checkbox for 'Include your browser's user agent string in the bug description. This can be helpful to developers working on your issue.' which is checked; 'Attach a file:' with a 'Choose File' button and 'No file chosen' text; 'Bug Type:' with two radio buttons: 'This is a defect report.' (selected) and 'This is a request for enhancement.'; 'Security:' with a checkbox and text 'Many users could be harmed by this security problem: it should be kept hidden from the public until it is resolved.'; and 'Additional Details:' with a checkbox and text 'This is a problem with Firefox on my phone or tablet.' At the bottom right, there is a blue 'Submit Bug' button.

Figure 3. Mozilla custom bug entry form.

**Steps to reproduce.** A clear set of instructions or a list of items that developers can use to reproduce the bug on their own machine. For example: “1) View any web page (such as <https://www.google.com/>); 2) While holding down the mouse button, drag the mouse pointer downwards from any point in the browser’s content region to the bottom of the browser’s content region.” [52]

**Observed behavior.** What the user saw happened in the application because of a bug. Users often have to provide at least a minimal amount of detail about what the application did after performing the steps to reproduce. For example: “The application crashed.” [52].

**Expected behavior.** What the user expected to happen, were the bug not present. For example: “The window should scroll downwards. Scrolled content should be selected. Or, at least, the application should not crash.” [52].

**Additional information.** Any other information that could help developers identify the root of the problem. For example, information about the platform, browser, operating system, release version of code being used, build information, etc.

#### **2.1.4.2 Severity**

Severity indicates how important the problem is to the health of the project. For example, in Bugzilla, the severity can vary from blocker (far ranging impact, limiting development and usage), to critical (known to cause crashes, can lead to loss of data), to major (certain functionality is not working properly), with other intermediate levels all the way down to trivial (minor cosmetic issue). Severity can also be set to enhancement to indicate a request for new functionality. The severity field is expected to be used by developers to classify bugs according to their importance. Not all important bugs are fixed right away, however.

#### **2.1.4.3 Priority**

While severity indicates the seriousness of the defect, priority defines the order in which a bug should be fixed. Priority can be first set by a bug reporter and then typically is adjusted by bug triagers. For instance, in Bugzilla, priority can range from P1 (to be fixed in the current release cycle), to P2 (to be fixed in the next release cycle), all the way down to P5 (no active plans to fix). An incorrect description can lead to a decreased bug priority.

#### 2.1.4.4 Status

The status of a bug report tracks its current state when it comes to processing it. The initial status for bug reports for the Mozilla projects is always UNCONFIRMED when it is first submitted and the final status is typically one of RESOLVED, VERIFIED, or CLOSED as set by the bug assignee once they are done [53].

Figure 4 shows the valid transitions along the various statuses that are possible in Bugzilla. When a bug is first reported, the bug report is marked as UNCONFIRMED. When a triager has verified that the bug is not a duplicate and indeed a new bug, the status is set to NEW. Then the triager assigns the bug report to a relevant developer, and the status is changed to ASSIGNED. The assigned developer reproduces the bug, localizes it, and tries to fix it. When the bug has been addressed, the bug report is marked as RESOLVED. After that, if a tester is not satisfied with the solution, the bug should be reopened with the status set to REOPEN; if a tester has verified that the solution worked, the status is changed to VERIFIED. The final status of a bug report is CLOSED, which is set when no occurrence of the bug is reported. This bug, however, might get REOPENED later if the resolution is deemed incorrect. For example, a WORKSFORME bug is REOPENED when more information shows up and the bug is now reproducible.

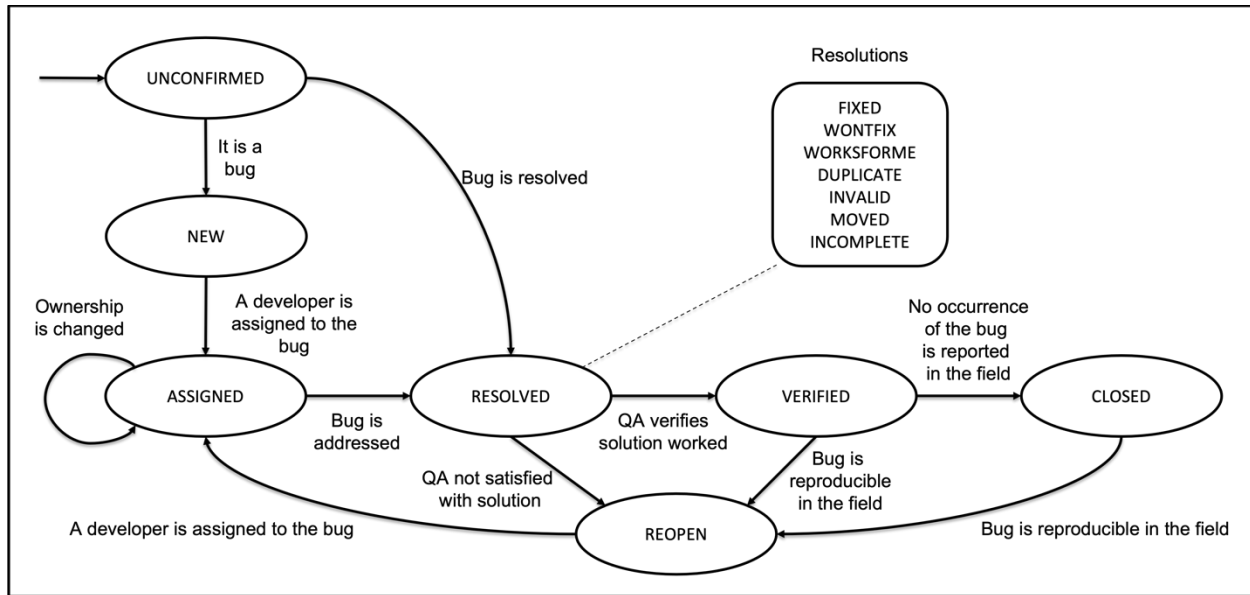


Figure 4. The life cycle of a bug report in Bugzilla<sup>9</sup>.

Note that the status RESOLVED has an important associated field that captures how it was resolved: the bug report could represent a real bug and it was FIXED; the bug report could be a DUPLICATE; the bug report could represent a possible real bug but it has been decided that it will not be fixed (WONTFIX); the bug report could represent a bug that cannot be reproduced by triagers (WORKSFORME); the issue described in the bug report could be not a bug (INVALID, e.g., the reporter is requesting a new or modified feature, what the reporter is believing is a bug is actually by design, or the bug is not Mozilla related and needs to be taken up with another organization); the bug report is now being worked on but has been relocated to another issue tracker (MOVED); and the bug report may describe a valid bug but there is not enough information at this point in the bug report to tell whether the bug is still present or not (INCOMPLETE).

Although other bug reporting systems use somewhat different terminologies and life cycles, Bugzilla and its life cycle are representative of what typically transpires. Of note is that many submitted bug reports tend to stay in the status NEW, when they are not yet processed or have been looked at but are not easily comprehensible, incomplete, or simply not of high

<sup>9</sup> Adapted from [2], [54].

priority. They stay in the bug tracking system and sometimes are known as UNCONFIRMED bug reports [55].

#### 2.1.4.5 Attachments

Depending on the nature of the bug, the developer may need different types of attachments to clearly depict the occurrence of the bug. Below are the commonly used ones:

**Stack Trace.** A stack trace is a list of the method calls that the application was in the middle of when an exception was thrown. A stack trace often is produced by the application when it crashes. Because a stack trace may contain hints as to the origin of the problem, reports that include stack traces can be very valuable to developers when debugging a problem. In many ways, a stack trace is a more specific form of an error report, which is itself a more specific form of observed behavior.

**Error Report.** An error report is often produced by an application when a bug occurs. Error reports can include stack traces, and any time when a quoted error message appears on screen, as well as any detailed logs (e.g., Java core dumps).

**Screenshot.** A screenshot of the application, while the bug is occurring, can also be useful to developers to decipher what the user was doing when the bug arises. Usually, only the screenshots of the application with the bug are considered, not of other applications or of proposed changes.

**Screen Recording.** A video file of the screen that shows the bug and what actions led up to it. A screen recording can be more useful than a static screenshot since it provides developers an opportunity to understand how a user interacts with the system, examine the behavior of the system in the context of what the user did, and comprehend any events that may have contributed to the manifestation of the bug.

**Test Case.** A test case demonstrates that the system is functioning incorrectly, or at least according to the user who reported the bug. A good test case will include steps, data, and the conditions for the test in order to understand and check the expected versus actual results.

**Patch.** The common format of a patch is the uniform diff format. A patch represents a small piece of software designed to update or fix problems with a computer program or its supporting data. Sometimes reporters attach a patch, which can make the life of the developer a lot easier because now they only need to check if the patch is of good quality and indeed fixes the bug being reported.

**Source Code.** Small to medium-sized code examples are sometimes used by reporters to illustrate a problem, describe the environment in which a problem occurred, or even represent a sample fix to the problem described in the report.

### 2.1.5 Discussions

After the submission of a bug report, developers and end-users frequently participate in a discussion by posting comments. These comments can include both discussions about possible solutions to the bug and comments such as a user posting “updating the status”. Because a significant number of comments can result, especially taking into account that a developer could be resolving multiple bug reports at the same time, in Bugzilla Mozilla<sup>10</sup> users can set up their accounts to filter out all emails about the bug report except the reports in which their name is explicitly mentioned.

**Number of Comments.** This represents the total number of times that developers and end-users posted comments in bug reports. These comments accumulate and can show the progression of how a bug is understood by developers.

**Number of Back-and-Forth.** This represents the number of times that the reporter was explicitly requested to answer a question plus the number of times that the reporter answered. For example, in Mozilla, the flag “Flags: needinfo?” is raised when the developer directly mentions the reporter’s name and asks for some information. It is lowered by the reporter when they provide the information. By raising and lowering the needinfo flag, developers and bug reporters signal their messages to one another, which gives a sense of

---

<sup>10</sup> <https://www.bugzilla.org/docs/2.18/html/hintsandtips.html>



their direct, intended interactions, for instance, to clarify an issue or obtain some auxiliary information.

Note that the number of comments is different than the number of back-and-forth. The former represents all communication for a bug report, the latter signifies direct communication. For example, Figure 5 highlights that Mozilla bug id 1790809<sup>11</sup> has six comments in total; whereas the number of times that the flag needinfo was raised and lowered is two.

---

<sup>11</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1790809](https://bugzilla.mozilla.org/show_bug.cgi?id=1790809)

**Closed** Bug 1790809 Opened 1 month ago Closed 1 month ago

### two private browsing shortcuts are sometimes created

**bhearsum@mozilla.com (bhearsum)** Reporter  
Description • 1 month ago

QA reported this in [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1789991#c10](https://bugzilla.mozilla.org/show_bug.cgi?id=1789991#c10), but it's actually a distinct issue. The problem boils down to the fact that we have two separate strings for the Private Browsing shortcuts (one for the installer, one for the browser), and if they don't match, we'll end up with two shortcuts. This can happen in at least two cases:

1. If a localized build has only translated one string -- in which we end up with a localized shortcut *and* an en-US one (ew)
2. If a localized build has translated the two strings differently - we end up with two slightly different localized shortcuts.

As far as I know, there's no way to share strings between the installer and the browser, so I think the best fix here is to simply remove shortcut creation from the installer (which was only added as an optimization). Doing so will cause a small amount of I/O at first run, but that's on a background thread these days anyways. Once the shortcut is created successfully once all of that I/O is skipped in subsequent launches.

The other downside to not doing it in the installer is that the shortcut will not exist until Firefox runs - but given that our primary install flow (the stub installer) automatically launches Firefox, this seems OK enough.

---

**Monica Chiorean** Assignee  
Comment 1 • 1 month ago

Attached file [Bug 1790809: stop creating private browsing shortcuts in the installer r?bytesized](#) -- Details

It's important that this shortcut exists to avoid [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1762994](https://bugzilla.mozilla.org/show_bug.cgi?id=1762994). We were creating it in the installer to avoid first run I/O, with a fallback at runtime to catch zip builds and updates. Due to some of this code being in the installer and some of it in the browser, we ended up with two different strings. Unfortunately, this has resulted in a bug where we sometimes create two private shortcuts. This happens in at least two cases:

1. A localization has only translated one of those strings -- in which case we get an en-US string and a localized string
2. A localization has translated the strings differently -- in which case we get two localized, but slightly different, stings

Since the installer creation of the shortcut is an optimization, and the first run I/O is now on a background thread anyways, let's just get rid of the installer shortcut rather than trying to come up with a more complex fix for this.

---

**Pulsebot**  
Comment 2 • 1 month ago

Pushed by [bhearsum@mozilla.com](mailto:bhearsum@mozilla.com):  
<https://hg.mozilla.org/integration/autoland/rev/b2b72ce4b4ff>  
stop creating private browsing shortcuts in the installer r=bytesized

---

**Andreea Pavel [apavel]**  
Comment 3 • 1 month ago  
bugherder

<https://hg.mozilla.org/mozilla-central/rev/b2b72ce4b4ff>

Status: ASSIGNED → RESOLVED

---

**Monica Chiorean** Assignee  
Comment 4 • 1 month ago

(In reply to [bhearsum@mozilla.com](mailto:bhearsum@mozilla.com) (bhearsum) from [comment-#1](#))

Do you know the exact languages that use different localizations like 1 and 2 mentioned above, so we can check both scenarios? Thank you.

Flags: needinfo?(bhearsum)

---

**bhearsum@mozilla.com (bhearsum)** Reporter  
Comment 5 • 1 month ago

(In reply to Monica Chiorean from [comment-#4](#))

I can't give you a complete list, but fr and es-ES are on the list given [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1789991#c10](https://bugzilla.mozilla.org/show_bug.cgi?id=1789991#c10).

Of note though - if my patch is correct there is only ever one place (and one string) for shortcuts now. So as long as the installer is not creating a Private shortcut, it should be impossible to have second or mismatched ones.

Flags: needinfo?(bhearsum)

---

**Monica Chiorean** Assignee  
Comment 6 • 21 days ago

Could not reproduce the issue using latest Beta 106.0b8(20221004185850). I used ES and FR localizations.

Status: RESOLVED → VERIFIED

Figure 5. Comparison between the number of comments and back-and-forth in Mozilla bug id 1790809 (cropped).

### **2.1.6 Time to Resolve**

Time to resolve captures the number of days between the date when a bug report was first created and the date when it was eventually resolved. The number of days to resolve a bug report includes: (1) the time spent waiting for the assignment of the bug report to a developer, (2) the number of days spent inspecting the bug, and (3) the number of days during which the assigned developer made code changes and verified the correctness of the changes. It can take longer to fix a bug lacking details or having a description that is not precise because it requires additional time for problem clarification.

### **2.1.7 Machine Learning**

Machine Learning (ML) is an application of Artificial Intelligence (AI) that provides systems the ability to learn and improve from experience without being explicitly programmed [56]. There are two types of ML algorithms: predictive (or supervised) and descriptive (or unsupervised). A predictive algorithm builds a model based on historical training data and uses this model to predict, from the values of input attributes, an output label (class attribute) for a new sample. A predictive task is called classification when the label value is discrete, or regression when the label value is continuous. On the other hand, a descriptive algorithm explores or describes a dataset. There is no output label associated with a sample. Data clustering and pattern discovery are two examples of descriptive tasks. In this dissertation, predictive algorithms are used.

#### **2.1.7.1 ML Algorithms**

An ML algorithm works over a dataset, which contains many samples or instances. Each instance is composed of input attributes or independent variables, and one output attribute or dependent variable. Input attributes are commonly named features or feature vectors, and the output attribute is commonly named class or category. Next, a brief description of each classification algorithm that is used in this dissertation is presented.

**Naïve Bayes (NB).** Naïve Bayes calculates the likelihood that every given data point falls into one or more of a set of categories [57]. It is a supervised learning approach for addressing classification issues that are based on the Bayes theorem. It is a probabilistic classifier, which means it makes predictions based on an object's likelihood [58].

**Support Vector Machine (SVM).** Support Vector Machine [59] is a supervised machine learning algorithm that can be used for both classification and regression challenges. However, it is mostly used in classification problems [60]. In the SVM algorithm, each data item is plotted as a point in n-dimensional space (where n is a number of features) with the value of each feature being the value of a particular coordinate. Then, the classification is done by finding the hyper-plane that differentiates the two classes as best as possible.

**Decision Tree (DT).** Decision Tree [61] is a classification algorithm that uses a process of division to split data into increasingly specific categories. It is called a decision tree because the classification process resembles a tree's branches when represented graphically. The algorithm works on a supervised model [62].

**Random Forest (RF).** Random Forest [63] is a supervised learning approach used in machine learning for classification and regression. It's a classifier that averages the results of many decision trees applied to distinct subsets of a dataset to improve the dataset's projected accuracy.

### 2.1.7.2 Performance Measures

**Accuracy.** Accuracy relates the number of correct classifications to the total number of classifications. Accuracy is how close a measured value is to the actual (true) value.

$$Accuracy(x) = \frac{(TP_x + TN_x)}{(TP_x + FP_x + TN_x + FN_x)}$$

where P is the total of positive class instances, N is the total of negative class instances, TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

**Precision.** Precision relates the number of true positives to the total number of instances classified as positives. Precision is how close the measured values are to each other.

$$\text{Precision}(x) = \frac{(TP_x)}{(TP_x + FP_x)}$$

where TP is the number of true positives, and FP is the number of false positives.

**Recall.** Recall (also known as sensitivity) identifies the fraction of positive events that were predicted correctly. Recall is the number of true positives divided by the total number of true positives and false negatives.

$$\text{Recall}(x) = \frac{(TP_x)}{(TP_x + FN_x)}$$

where TP is the number of true positives, and FN is the number of false negatives.

**F-score.** F-score (also known as F1-score or F-measure) is the harmonic mean of the precision and recall. A perfect model has an F-score of 1.

$$\text{F-score}(x) = 2 * \frac{\text{Precision}(x) * \text{Recall}(x)}{\text{Precision}(x) + \text{Recall}(x)} = \frac{TP_x}{TP_x + \frac{1}{2}(FP_x + FN_x)}$$

**ROC.** Receiver operating characteristic (ROC) is an alternative measure to evaluate binary classifiers. A ROC curve [64] is a bi-dimensional chart, where the X-axis represents false positives, and the Y-axis represents true positives. The Area Under ROC Curve (AUC), ranging between 0 and 1, is used to assess the performance of ML algorithms. An algorithm outperforms another one if its AUC value is closer to 1.

### 2.1.7.3 Hyper-Parameter Optimization

The tuning of hyper-parameter values is used to determine the values that will lead to the best algorithm performance in a specific context. Each algorithm has its own set of parameters and the adjustment of these values can greatly impact in algorithm performance.

The procedure that is used by researchers for setting hyper-parameters is to train the ML model with the hyper-parameters to get the optimal model accuracy. After changing the hyper-parameter candidate values, researchers should train the model again until the predicting model achieves satisfactory prediction accuracy. When this goal is reached, the machine learning model building process can be considered “complete” [65].

In this dissertation, grid search [66] is used as the tuning strategy. Basically, the domain of the hyperparameters is divided into a discrete grid. Then, every combination of values of this grid is tried by calculating performance metrics using cross-validation. The point of the grid that maximizes the average value in cross-validation is the optimal combination of values for the hyperparameters.

#### **2.1.7.4 Cross Validation**

The evaluation of a supervised method’s effectiveness is mainly based on two datasets with labeled samples, one for training the predictive model and the other for testing this model. In order to obtain more reliable predictive estimates, resampling methods can be used to split the entire dataset into a training dataset and a testing dataset. Cross validation, also known as k-fold CV, is a resampling method that is used to train and evaluate classifiers [67]. This validation approach randomly divides the manually classified data set into k groups of equal size. In each iteration, it saves a different fold for testing and uses all the others for training. The mean of the k executions is used as an estimation of the classifier's accuracy.

#### **2.1.7.5 Cross-Project Prediction**

Cross-project prediction concerns the portability of a learned prediction model among different projects [68]. This is important to avoid the cold-start problem, where prior information is not available for training purposes. Cross-project prediction can fail for projects drawn from the same domain. For example, Zimmermann et al. [69] tried to port models between two web browsers (Internet Explorer and Firefox) and found that cross-project prediction was not consistent: a model built on Firefox was useful for Explorer, but

not vice versa. Additionally, prediction performance may not even be generalizable within a project. Menzies et al. [70] found that a given project's data may have many local regions which, when aggregated at a global level, may offer a completely different conclusion in terms of both quality control and effort estimation.

### 2.1.8 Text Preprocessing

Common ML algorithms cannot directly process unstructured text such as a bug report's description, because it contains noise in various forms including punctuation, text written in numerical or special character form, and many commonly used words. Therefore, during a preprocessing step, these unstructured text fields are converted into more manageable representations. Text preprocessing is the process of converting unstructured text into a structure suited to analysis [71]. It is composed of three primary activities [72].

**Tokenization.** Tokenization is the action of parsing a character stream into a sequence of tokens by splitting the stream at delimiters. A token is a block of text or a string of characters (without delimiters such as spaces and punctuation), which is a useful portion of the unstructured data.

**Stop words removal.** Stop words removal eliminates commonly used words that do not provide relevant information to a particular context, including prepositions, conjunctions, articles, common verbs, nouns, pronouns, adverbs, and adjectives.

**Stemming.** Stemming is the process of reducing or normalizing inflected or sometimes derived words to their word stem, i.e., its base form (e.g., "working" and "worked" into "work").

### 2.1.9 Bag of Words

One of the most traditional ways of representing a document relies on the use of a bag of words (unigrams) [71]. In this approach all terms represent features, and thus the

dimension of the feature space is equal to the number of different terms in all documents (i.e., bug reports).

### **2.1.10 TF-IDF**

Term frequency-inverse document frequency (TF-IDF) is a statistical measure that evaluates how relevant a word is to a document in a collection or corpus [73]. The tf-idf technique is used in text mining to understand the significance of a word in a document and across multiple documents. The term frequency and the inverse document frequency are calculated as:

$$\text{Term Frequency (TF)} = \frac{\text{Number of occurrences of a word}}{\text{Total words in the document}}$$

$$\text{Inverse Document Frequency (IDF)} = \text{Log} \left( \frac{\text{Total number of documents}}{\text{Number of documents containing the word}} \right)$$

### **2.1.11 Statistical Tests**

Many scenarios require running several algorithms to choose the best model. Even though the performance of these algorithms may be shown to be different on specific datasets, it needs to be confirmed whether the observed differences are statistically significant and not merely coincidental [64]. In this situation, conducting statistical tests is a recommended practice for reliable comparison between predictive models under investigation [74]. In this dissertation, the t-test [64] is used, which is a parametric statistical hypothesis test to assess whether the means of two groups are statistically different from each other.

## **2.2 Quality of Bug Report**

Bug reports serve an important role in the software development process: they collect relevant information about faults or defects that end-users, testers, or other developers encounter when using or working with software [1]. Ideally, the information provided in a bug report is sufficient and clear for developers to triage, diagnose, and subsequently fix the



software bug that was encountered. However, several studies conclude that a significant portion of bug reports lack in that regard [13], [14]. Without some sort of major clarification or important additional information, the developer is unable to comprehend and process the report. As a result, the bug report is either ignored altogether or the developer has to engage in the extra effort of formulating a set of questions to the reporter that hopefully will lead to the insight that is needed to take the next step with the bug report [7], [16], [20], [75]. Interestingly, such requests are frequently made in vain, with Breu et al. finding in a study of Mozilla bug reports that 50% of requests to reporters went unanswered [15]. Usually, after several months with no response, one of the developers closes the bug report and marks it as either *RESOLVED* or *WORKSFORME*.

### 2.2.1 Attributes of High-Quality Bug Reports

What makes for high-quality (or good) bug reports has been the subject of extensive study. For instance, Bettenburg et al. [19] conducted a survey of Eclipse developers, finding that the inclusion of steps to reproduce and stack traces is considered most important, whereas reporter mistakes in specifying the steps to reproduce and incomplete information are most problematic in successfully triaging and addressing bug reports. A follow-up survey by the same authors includes Apache and Mozilla developers and confirms steps to reproduce and stack traces are seen as most important, with test cases, observed behavior, and screenshots next [76].

Zimmermann et al. [14] investigated the quality of bug reports from the perspective of developers. To find out which features matter the most, they surveyed 466 developers from the Apache, Eclipse, and Mozilla projects. The responses reveal that most developers consider steps to reproduce, stack traces, and test cases as helpful. Other surveys report similar results (e.g., [16], [55]), expand the list of information considered important with items such as code examples or an explanation of the difference between the expected and the observed behavior (e.g., [20], [21]), or find a different order of importance developers assign to different information (e.g., [13], [33]). Somewhat differently, Bhattacharya et al. identify a long textual description of the problem as an indicator of a good bug report [21].

Laukkanen and Mantyla [13] surveyed 74 developers from six industrial software development companies. Their results showed that steps to reproduce and observed behavior are very important to understanding bug reports and significant for reproducing bugs.

In 2020, Soltani et al. [77] interviewed 35 developers to gain a more detailed perspective into the importance of various contents of bug reports, followed by a survey applied to 305 developers. The authors concluded that the essential elements are crash description, steps to reproduce, test cases, and stack traces. They also evaluated the quality of bug reports of the 250 most popular projects on Github. Their analysis showed that steps to reproduce, stack traces, and fix suggestions have a statistically significant impact on bug resolution times for 33% to 76% of the projects.

From a bug reporter's perspective, Karim et al. investigated the key features of high-impact bug reports (HIB) [20]. A HIB is defined as a bug that can significantly affect the software development process and resulting product quality. The researchers discovered that observed behavior, expected behavior, and code examples are the most frequent features provided by bug reporters in HIB reports. Chaparro et al. also detected important bug report elements, such as observed behavior, expected behavior, and steps to reproduce [33]. They manually analyzed 2,912 bug reports from nine systems (e.g., Eclipse, Firefox, and LibreOffice) and found that, while 93.5% of bug reports contain observed behavior (i.e., 93.5%), only 35.2% and 51.4% explicitly described expected behavior and steps to reproduce, respectively.

### **2.2.2 Missing Information and Clarification**

As opposed to identifying what information is important in high-quality bug reports, some studies focus on what information is missing from the typical bug report. While an early survey suggests that approximately half of the developers feel that bug reports are nearly always complete [12], a later study at Microsoft shows that the majority of bug reports are missing information and that a significant proportion contains inaccurate information [3]. Ko and Chilana [4] equally concluded that the majority of bug reports in the Mozilla bug

tracking system contribute little-to-no useful information. Through a survey, Bettenburg and colleagues found that screenshots, stack traces, and code examples are items that are frequently missing when they should have been included given the nature of the bug report [16][22].

Another reason for incomplete or inaccurate bug reports is inadequate tool support for bug reporting [7], a finding that is strongly corroborated by a petition from the developers of over a thousand open-source projects asking for improvements to GitHub’s infrastructure to ensure that essential information is reported by the users<sup>12</sup>. Beyond predefined templates, which have been shown to help [33] (but unfortunately are ignored by reporters all too often), advanced tool support remains lacking. Reporters simply fill out suggested fields on a form and submit their reports. For example, even though Bugzilla Helper [51] (see Figure 3) aims to guide users in including steps to reproduce, actual results, and expected results, it does not guarantee that the reporters necessarily provide this information [33].

Antoniol et al. acknowledged the importance of bug tracking systems and source repositories and suggested a unified framework to manage information extracted from source code, version histories, and bug reports [79]. In such a setting a much richer set of features would be available to users, such as links between bug reports and the complexity of the associated source code as well as change histories for files and components.

### **2.2.3 Techniques for Predicting the Quality of Bug Reports**

To address the problems surrounding bug reports, prediction techniques have been suggested as a potential solution. Some prediction techniques aim to assist in improving aspects of new bug reports prior to submission. For instance, Wu et al. developed a tool called BUGMINER [29], a tool that derives useful information from a historic bug report database via data mining. Using the information, the tool performs a completion check and redundancy check on a newly submitted bug report. It first predicts any missing data (e.g., product, operating system, and component) fields of a new bug report by mining historic

---

<sup>12</sup> <https://github.com/dear-github/dear-github>

data and applying an SVM model. BUGMINER, then, checks if the bug report is a duplicate by measuring the similarity between a new bug report and all the bug reports in its database using KL Divergence [80]. The evaluation of BUGMINER on the bug report repositories of Apache Tomcat, Eclipse, and Linux Kernel showed that it can determine the missing product, operating system, and component field with an F-score of 99%, 81%, and 79%, respectively. Moreover, results showed that BUGMINER can determine the product, operating system, and component fields with an accuracy of 99%, 68%, and 67%, respectively.

Chapparo et al. [33] created a tool called DeMIBuD to automatically detect the absence (or presence) of expected behavior and steps to reproduce in bug descriptions. DeMIBuD uses linear N-grams and SVM to classify the description of bug reports. With its best setting, DeMIBuD was able to detect missing expected behavior (steps to reproduce) with 85.9% (69.2%) average precision and 93.2% (83%) average recall. In their next study, Chapparo et al, [34] proposed Euler, an approach that automatically identifies and assesses the quality of the steps to reproduce in a bug report and provides quality annotations to the reporters, which they can use to improve the bug report. Euler uses an engine called Crashescope [81] which detects steps to reproduce missing in the report. The evaluation results on 24 bug reports of six Android applications showed that Euler correctly identified 98% of the existing steps to reproduce and 58% of the missing ones, while 73% of its quality annotations were correct.

Some researchers have investigated ways to improve the quality of bug reports by identifying the quality of the individual fields in a bug report [41]. For example, Ko et al. [30] performed a linguistic analysis of 200,000 bug report titles from Eclipse, Firefox, Apache, Linux, and OpenOffice to study how people describe software problems and suggested new designs for more structured report forms. In particular, they applied a probabilistic part-of-speech tagger to all report titles, by counting the nouns, verbs, adverbs, and adjectives that appeared in those titles, and proposed several design ideas motivated by their results, such as soliciting more structured titles.

Gromova et al. [82] introduced Nostradamus to evaluate the quality of bug reports by calculating the probability of priority levels, time to resolve, reject, and wontfix resolution. The authors used a combination of SVM and SMOTE [83], a method that can improve the

accuracy of classifiers for a minority class, on 5,242 bug reports from the JBOSS project. Their best model could achieve a mean F-score of 88% and a mean accuracy of 86%.

Considering the bug report as a whole, Zimmermann et al. trained a new tool by building a supervised learning model called CUEZILLA. CUEZILLA predicts the overall quality level of a new bug report (i.e., bad, neutral, or good) using a readability measurement, pattern matching, and analysis of the bug reports' attachments [14]. It also randomly provides relevant facts to bug reporters that were mined from a prebuilt bug database in order to encourage better reports, such as "Bug reports with stack traces are fixed sooner". Evaluating CUEZILLA on APACHE bug reports, using SVM regression with a linear kernel provides the top results, but with only 50 percent accuracy. Schuegerl et al. similarly assessed the overall quality of bug reports on a five point scale of very good, good, average, poor, and very poor [32]. The authors used Decision Tree and Naïve Bayes to automatically flag weak descriptions. The result of their evaluation on a random data sample of 178 bug reports revealed an accuracy of 44% for Decision Tree and for Naive Bayes.

Zanetti et al. [84] conducted a case study on the position of bug reporters in the collaboration networks of four Open Source Software communities (Mozilla Firefox, Eclipse, and Netbeans) to identify valid bug reports (bug reports that are actual software bugs, not duplicates, and contain enough information to be processed right away). The authors found that the position of bug reporters in communities is indeed indicative of whether bug reports are valid or invalid (are duplicates or incomplete). More specifically, they found that the number of bug reports that a bug reporter submitted and could be fixed in the month preceding the submission of a new report can positively influence the attention received by developers, thus affecting the chance of the bug report being taken seriously, prioritized, and eventually fixed. Based on this finding, they developed an automated bug report classification mechanism using nine topological measures at the level of bug reporters (e.g., eigenvector, betweenness, and closeness) and an SVM classifier and could achieve an F-score of 92%.

## 2.2.4 Bug Resolution Time

The quality of bug reports influences the time of bug resolution [26]. Karim et al. [20] found that bug reports for which developers require additional information have a significantly longer fixing time compared to bugs that do not. Many researchers investigated ways to assist in processing newly submitted bug reports by predicting whether a bug report can be resolved within a given time [25]. For example, Hooimeijer and Weimer [25] presented a model to measure bug report quality based on 27,000 Mozilla Firefox reports. They divided bug reports into “cheap” and “expensive” by predicting whether bug reports can be resolved within a given time. Their research was performed based on the assumption that “in general, reports of higher quality are dealt with more quickly than those of lower quality”. Many bug reports are addressed quickly not because of their quality, but because of the urgent problems they describe. While their results showed that the presence of an attachment tends to lead to a faster bug fixing time and that the comment count suggests that bug reports that receive more attention get fixed faster. Zimmermann et al. [14] also investigated which bug reports have a faster fixing time. The authors found that bug reports that contain stack traces and are easier to read get fixed sooner.

Zaman et al. [85] compared bug fix times of security and performance bug reports in Mozilla Firefox. They found that security bugs are fixed faster than performance bugs. Bhattacharya et al. [21] performed a similar analysis for security versus non-security bugs and showed that the quality of security bug reports is higher compared to non-security bugs, though security bugs are fixed slower compared to other bugs, which contradicts the finding of Zaman et al. [85]. They also found that Google Code’s bug tracker<sup>13</sup>, which is used by most open-source Android apps, offers less bug management support (e.g., for bug triaging) compared to other trackers such as Bugzilla or Jira. The authors mention this lack of information limits empirical analyses and might hinder the bug-fixing process on Google Code-based projects.

Kuramoto et al. [86] investigated the impact of including images and videos on bug report resolution time. Their study on 1,230 videos from 226,286 bug reports revealed that the

---

<sup>13</sup> <https://issuetracker.google.com>

resolution time in bug reports without videos is longer than in bug reports with videos, but no statistically significant differences were observed.

## **2.2.5 Triaging**

Triaging is the process of reviewing bug reports to ensure they represent a real bug and have accurate information that allows them to be resolved and tested [87]. Triagers and assigned developers spend a significant portion of their time comprehending the submitted bug reports and asking for missing or additional information [75]. In response, many researchers have focused on automating the triaging of bug reports. For example, Antoniol et al. [88] applied text mining techniques to the description of bug reports to determine whether a bug report is a real bug or a feature request. The authors used techniques such as decision trees, logistic regression, and a Naive Bayes classifier to achieve this purpose. The performance of this approach on three projects (Mozilla, Eclipse, and JBoss) indicated that reports can be predicted to be a bug or an enhancement with an accuracy between 77% and 82%.

Bug triaging also includes the process of checking whether the reported bug is a duplicate of an existing bug, prioritizing bug reports, deciding which developer should work on the bug reports, and so on. In the following, the relevant work in each category is discussed.

### **2.2.5.1 Predicting Severity**

The severity of a bug report is an important factor in determining how fast it will take to fix the bug it represents. Menzies and Marcus [89] are among the first to predict the severity of bug reports. The authors adopted a rule learning technique that uses the textual descriptions of reported bugs [20]. The approach analyzes the textual contents of bug reports and outputs fine-grained severity levels used internally in NASA's Independent Verification and Validation Facility. Their approach reported precision and recall values that varied a lot, with precision between 0.08 and 0.91 and recall between 0.59 and 1.00. Lamkanfi et al. [90] extended the work of Menzies and Marcus to predict whether or not a bug report is severe in Bugzilla. They used five severity labels out of six severity labels in Bugzilla and drop bug

reports belonging to the category “normal”. The remaining five categories were grouped into two groups: severe and non-severe. They explored four classifiers including Naive Bayes, Naive Bayes Multinomial, SVM, and Nearest-Neighbor classification, and found that the Naive Bayes Multinomial performed the best on a dataset consisting of 29,204 bug reports, with the highest accuracy of 93% [91].

Yang et al. [92] performed feature selection schemas such as chi-square, correlation coefficient, and information gain to choose suitable features. The features help extract potential severe (e.g., crash) and non-severe (e.g., typo) indicators in bug reports. The authors then applied the Multinomial Naive Bayes (NB) classification approach. The experimental results showed that these feature selection schemes perform well for severity prediction by achieving accuracies of over 0.77 and 0.84 for bug reports in Eclipse and Mozilla, respectively. The prediction results also demonstrated that the reports of Eclipse tend to use more severity-consistent words to describe the bug such that the indicators have high specificity. In contrast, the reporters of Mozilla tend to use less severity consistent words to describe the bugs, which might be because of “its reporters’ diversified background”, as the authors explain.

Tian et al. [93] used three open-source software systems (i.e., OpenOffice, Mozilla, and Eclipse) and found that around 51% of the duplicate bug reports have inconsistent human-assigned severity labels, even though they refer to the same software problem. To address this problem, the authors proposed a new approach for assigning severity that agrees between 77% and 86% of the time with human-assigned severity labels.

Recently, Tan et al. [94] used a somewhat different approach to predict the severity of bug reports. They collected all question-and-answer pairs from the posts related to bug repositories in Stack Overflow and then used BM25<sup>14</sup> to get a similarity score between a sentence from Stack Overflow and a bug report. Then they applied logical regression to predict the severity of bug reports. They evaluated their approach by comparing the result of the severity prediction with a previous study, Word2Vec [96]. The results showed that their approach could achieve an average F-measure of over 82% and improve the average F-

---

<sup>14</sup> BM25 [95] is a ranking system that helps search engines identify the most relevant documents in a given set of documents.



measure in Mozilla, Eclipse, and GCC by 23.03%, 21.86%, and 20.59% compared to Word2Vec.

### 2.2.5.2 Predicting Priority

To aid bug triagers in assigning priority, automated techniques have been proposed to recommend priority levels of bug reports using information available in the bug reports. In one of the early studies, Tian et al. [97] proposed DRONE to predict the priority levels of bug reports in Bugzilla. The authors considered multiple factors available in bug reports, including bug description, reporter's information, and severity, to train a machine learning model, built by combining linear regression with a thresholding approach to address the issue with imbalanced data. The result on a dataset consisting of more than 100,000 bug reports from Eclipse showed an average F-measure of 29%.

Abdelmoez et al. [98] used a Naive Bayes classifier to predict the priority of bug reports in three projects: Mozilla, Eclipse, and GNOME. They prioritized the bug reports according to their mean time into 3 quartiles: from fast bug reports that developers start with, to slow ones that developers exclude and defer them. The results showed varying average precision (0.44-1.00) and recall (0.2-0.99).

Alenezi and Banitaan [23] used Naive Bayes, Decision Tree, and Random Forest to predict the priority of bug reports. They use two feature sets, one based on TF-weighted words of bug reports and a second based on the classification of bug report attributes. The results of their evaluation on bug reports of Eclipse and Firefox showed that the usage of the second feature set performed better with the highest F-score of 0.63.

Umer et al. [99] recently used a Convolutional Neural Network (CNN) based model to predict the priority of bug reports. They first converted each word into a vector using a word2vec model. Then, they performed a software engineering domain-specific emotion analysis on bug reports and compute the emotion value for each of them. Finally, they passed both the vectors and emotions of bug reports to the CNN-based classifier. Results of the cross-project evaluation suggested that their approach could improve the performance of previous studies by increasing the average F-score by 24%.

### 2.2.5.3 Automatic Bug Assignment

Machine learning techniques also have been used to predict the most appropriate developer for resolving a new incoming bug report. This way, bug triagers are assisted in their task. For example, Cubranic et al. [100] trained a Naive Bayes classifier with the history of the developers who solved the bugs as the category and the corresponding descriptions of the bug reports as the data. This classifier is subsequently used by triagers to manually select the most appropriate developer (i.e., assignee) from the generated list for a newly reported bug. The results of an evaluation revealed that over 30% of the incoming bug reports of the Eclipse project are assigned to the correct developer using this approach. Anvik et al. [101] used the SVM classifier and text categorization to assign bug reports to an appropriate developer. In this study, they could obtain an overall classification accuracy of 57% and 64% for the Eclipse and Firefox projects respectively.

Zhang et al. [102] introduced a hybrid bug triaging method which is a combination of a probability and experience model for assigning developers to a new bug report. In their study, they used a smoothed Unigram model (UM) and a probability model based on Social Network analysis to search for bug reports which are similar to the newly reported bug. The evaluation result on automatic bug triage of bug reports in the JBoss and Eclipse projects showed the highest F-score of 71% for JBoss and 67% for Eclipse, when six developers are recommended (an F-score between 20% to 25% when one developer is recommended).

Yadav et al. [103] proposed an approach that ranks developers based on their expertise in triaging bugs. This approach generates developer expertise scores using the average fixing time, priority-weighted fixed issues, and index metrics. It then determines feature-based, cosine, and Jaccard similarities to compute the expertise scores. Finally, a ranked list of developers for new incoming bug reports is produced. They evaluated the approach on the Mozilla, Eclipse, NetBeans, Firefox, and Freedesktop projects covering 41,622 bug reports and could achieve an F-score of 89%. As a result, their approach could reduce the average length of bug tossing sequences by 20%.

Instead of applying machine learning techniques, Peng et al. [104] proposed a method based on relevant search techniques. Their method constructs a search engine for bug reports and ranks a list of relevant bug reports for resolving a newly reported bug. It then recommends its assigned developer. They compared their results with two existing machine learning-based developer recommendation methods (Naive Bayes and SVM) and could improve the recall rate, between 0.1-0.2, for both Mozilla and Eclipse projects.

#### **2.2.5.4 Duplicate Bug Report Detection**

Duplicate bug reports cause an overestimation of the number of bug reports and increase the cost of triaging. Some researchers use textual analysis to detect duplicate bug reports. For example, Runeson et al. [105] used text similarity techniques to help automate the detection of duplicate bug reports. By comparing the similarities between the description of bug reports they could detect 66% of the duplicate reports of Sony Ericson Mobile Communications.

Aggarwal et al. [106] used hierarchical contextual information extracted from software-engineering textbooks, project-related software literature, and project documentation. They then applied a BM25 similarity method to detect duplicate bug reports in the Android, Eclipse, Mozilla, and OpenOffice projects. The experimental results showed the importance of domain-specific context as it improved accuracy to over 92% for all four projects. The Kappa scores also improved by at least 3.8% to 10.8% compared to the base study of Alipour et al. [107].

Machine learning is also a line of approach for the automatic detection of duplicate bug reports. For instance, Bettenburg et al. [78] developed an SVM and a Naive Bayes to filter out duplicate bug reports. They triaged bug reports based on a word vector representation of the report titles and descriptions. Their approach could obtain roughly 65% accuracy.

Sun et al. [108] used SVM and information retrieval (IR) techniques to retrieve similar bug reports from a bug repository. They considered duplicate bug report detection as a binary classification problem, that is, given a new report, the retrieval process is to classify all existing reports into two classes: duplicate and non-duplicate. They computed 54 types of

textual similarities between reports and used them as features for training and classification. The results of an evaluation of bug repositories from OpenOffice, Firefox, and Eclipse showed that the approach could outperform existing state-of-the-art techniques by a relative improvement of 17–31%, 22–26%, and 35–43%, respectively.

Learning to Rank (L2R) is another useful machine learning approach to detect duplicate bug reports. Based on L2R, Zhou et al. [109] introduced BugSim which is based on learning to rank concepts. It identifies textual and statistical features and uses a stochastic gradient descent algorithm over the training set. For a new bug report, BugSim then retrieves candidate duplicate reports using the trained model. The results of an evaluation of 45,100 bug reports of twelve Eclipse projects showed an average recall rate for the top 10 retrieved reports of 76.11%. The proposed approach also works better than a previous SVM-based method of Sun et al. [108] by 15.41% and the BM25-based method of Sun et al. [110] by 3.71% on the recall rate of the top 10 results.

## **2.2.6 Video Submissions in Bug Reports**

Videos can demonstrate complex contexts about bugs and thereby can offer developers a new opportunity to collect context-rich bug information [36]. Videos help developers understand how users interact with the system, process the current behavior of the system, and comprehend any events that contributed to the bug. Several studies encourage reporters to submit relevant videos as part of their bug reports to convey additional context for understanding bugs [39]–[41].

### **2.2.6.1 Studies on Video Bugs in Game Applications**

Game developers attempt to learn more about the problems that gamers encounter by studying gameplay videos, as they often offer important context surrounding a bug [42]. Lewis et al. [39] created a taxonomy of video game bugs by deducing patterns from bug videos on YouTube. They identified the bug videos as “a rich resource that mixes creativity,

subversiveness, and pure chance. The videos available provide a startling amount of coverage.”

Lin et al. [36] proposed an approach to automatically identify game videos that showcase a bug using the metadata available online. Their process uses a random forest classifier to rank gameplay videos based on their likelihood of being a video containing a bug. Jacob et al. [88] and Lou et al. [89] presented approaches to automatically extract game logs from game videos, with the logs then used by game designers to study the bugs. Jacob et al. [88] used image processing techniques and predefined image patterns to extract game logs from gameplay videos of the Super Mario Bros. game. Luo et al. [89] used convolutional neural networks and transfer learning to extract game logs from gameplay videos of the Super Mario Bros., Megaman, and Skyrim games.

### **2.2.6.2 Studies on Video of Bugs in Mobile Applications**

In mobile applications, the practice of sharing videos containing screen recordings to convey additional context for understanding bugs has been steadily increasing over the past few years [111]. Many mobile applications allow users to report bugs in a graphical form (not necessarily video) in order to simplify the reporting of bugs by end-users and crowd-testers [112]–[114]. The reporting of visual data is also supported by many bug reporting services for mobile applications [115]–[119].

To assist developers in effectively taking in video content, approaches have been introduced for automatically analyzing videos to understand and use in bug resolution [47]. Krieter et al. [43] used video analysis to extract high-level descriptions of events from user video recordings on Android apps. Their approach generates log files that describe what events are happening at the app level. Lin et al. [120] proposed an approach called Screenmilk<sup>15</sup> to automatically extract screenshots of videos of sensitive information (e.g., a user entering a password) by using the Android Debug Bridge<sup>15</sup>. This technique focuses on the extraction of keyboard inputs from “real-time” screenshots.

---

<sup>15</sup> The Android Debug Bridge is a programming tool used for the debugging of Android-based devices.

Bao et al. [121] and Frisson et al. [122] performed a behavioral analysis of developers during programming tasks. They used computer vision techniques to extract the user interactions. In a similar line of research, Bernal-Cárdenas et al. [46] extracted generic user actions on mobile apps in order to translate video recordings of Android app usages into replayable scenarios. Cooper et al. [44] developed TANGO, which analyzes both video submissions and textual information present in mobile screen recordings to find duplicate video-based bug reports. TANGO combines computer vision and text retrieval techniques to retrieve the video-based reports that are most similar to the incoming report.

Hu et al. [45] developed AppFlow, which leverages machine learning techniques to analyze Android screens and categorize types of test cases that could be performed on them (i.e., a sign-in screen whose test case would be a user attempting to sign in). AppFlow is focused on the generation of semantically meaningful test cases in conjunction with automated dynamic analysis.

### **3 CHAPTER 3: Bug Report Quality Prediction: Actionable versus Non-Actionable Bug Reports**

Bug reports are vital to any software development project, as they inform developers of potential problems encountered with the software. An ideal bug report contains sufficient and clear information to be actionable: from the bug report, the developer can determine whether the report raises a valid issue and, if so, start the process of figuring out what might be wrong and how to fix it. Some minimal back-and-forth may be necessary to clarify a small issue or obtain some auxiliary information that might be helpful later, but fundamentally an actionable bug report is one where the developer feels comfortable proceeding. Many newly submitted bug reports are non-actionable [3], [4]. This is undesirable since it slows down the process of triaging and resolving bugs and incurs significant additional effort on all parties involved [17].

It is within this context that I believe a significant opportunity exists for tools that intervene before a non-actionable bug report is submitted. Such tools could flag newly composed bug reports if they are deemed non-actionable and give reporters the chance to improve these reports, ideally with some guidance, before they are eventually submitted. A necessary precursor to the design of these kinds of tools is a classifier that can distinguish actionable from non-actionable bug reports. However, the accuracy of existing classifiers is insufficient. Among them, Cuezilla [35] and the approach of Schuegerl et al. [32] provide the best results, with the former achieving an accuracy of 50% in classifying bug reports as good, neutral, or bad and the latter achieving an accuracy of 44% in classifying bug reports on a scale of 1 (very high quality) to 5 (very low quality). In addition to this being too low an accuracy to be practically useful in the kind of tool that is envisioned, such a tool also merely needs a simpler, binary classification, rather than one that offers a more fine-grained scale offered by prior work.

This chapter therefore revisits the problem of bug report quality prediction and answers the following research questions:

- 1- Is it possible to predict the overall quality of bug reports in terms of whether they are actionable or non-actionable with a sufficiently high accuracy?

2- Can performance results be further improved by adding auxiliary features (e.g., whether the report was submitted by an end-user or developer, whether it has an attachment)?

3- To what extent is the best-performing model portable?

To answer the research questions, I manually classified 1,423 bug reports from all the Mozilla Firefox projects (e.g., Firefox, Firefox Build System, Firefox for FireTV) as actionable or non-actionable, relying on existing findings regarding the distinctive characteristics of high-quality and low-quality bug reports to do so (e.g., [48], [49]). I then conducted supervised learning with four different machine learning classifiers, including Naïve Bayes, Support Vector Machine, Decision Tree, and Random Forest, on solely the description field from the bug reports. I compared the performance of each classifier, selected the one that performed best, and trained it again with a variety of additional features (e.g., whether the report was submitted by an end-user or developer, whether it has an attachment). After that, I conducted cross-project prediction by applying the classifier with the best results on an existing dataset that was studied previously containing bug reports for Apache (90) and Eclipse (100).

In the following subsections, I first explain the terminology of actionable and non-actionable bug report. Then, I detail the methodology and results on how to achieve a sufficiently high accuracy so the model could be used in downstream tools.

### **3.1. Actionable and Non-actionable Bug Report**

It is useful to clarify the terminology of actionable and non-actionable, especially in the context of prior work, which uses, among other things, a three-point scale of good, neutral, and bad [14] and a five-point scale from very high quality to very low quality [32]. In this dissertation, the terms actionable and non-actionable are preferred because they more accurately reflect what happens to bug reports: they are either sufficiently complete and clear for a developer to know what to do with them, or they are not; the distinction between understanding the essence of a bug report and what to do with it versus not understanding that essence is the difference between a bug report being actionable or not.



For example, Figure 6 (part 1) shows the description of Mozilla bug id 544833<sup>16</sup>, which is an actionable bug report. The report contains specific information about what happened, what the user expected to happen, and how to reproduce it. By reading it, another developer could also reproduce the bug on another system, see Figure 6 (part 2).

Figure 6. Example of an actionable bug report, bug id 544833 (cropped).

Figure 7 (part 1) shows that the assignee of the bug report could confirm the bug as well and found another way to replicate it “...click location Bar, then press Shift+Tab and then

<sup>16</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=544833](https://bugzilla.mozilla.org/show_bug.cgi?id=544833)

pressing *Ctrl+Shift+Tab* or *Ctrl+Tab*.” and suggested a patch for it (Figure 7 (part 2)). The patch is approved by QA and added to the central codebase, see Figure 7 (part 3).

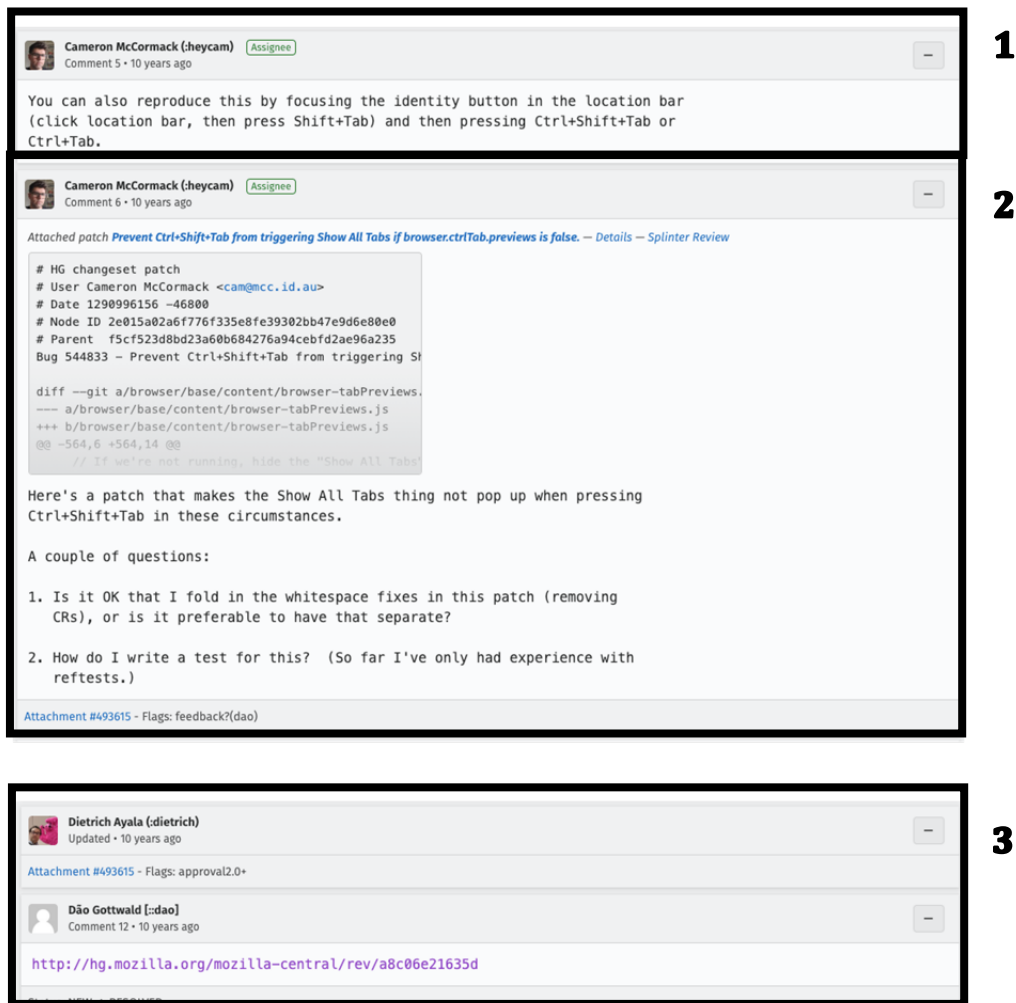


Figure 7. Comments from bug id 544833 (cropped).

A non-actionable bug report lacks in providing the developer with the information needed to determine what to do next. In such cases, the developer must obtain significant clarification or important additional information before they can even figure out what issue the bug report attempts to raise. Mozilla bug id 1497399<sup>17</sup>, shown in Figure 8, is an example of a non-actionable bug report.

<sup>17</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1497399](https://bugzilla.mozilla.org/show_bug.cgi?id=1497399)

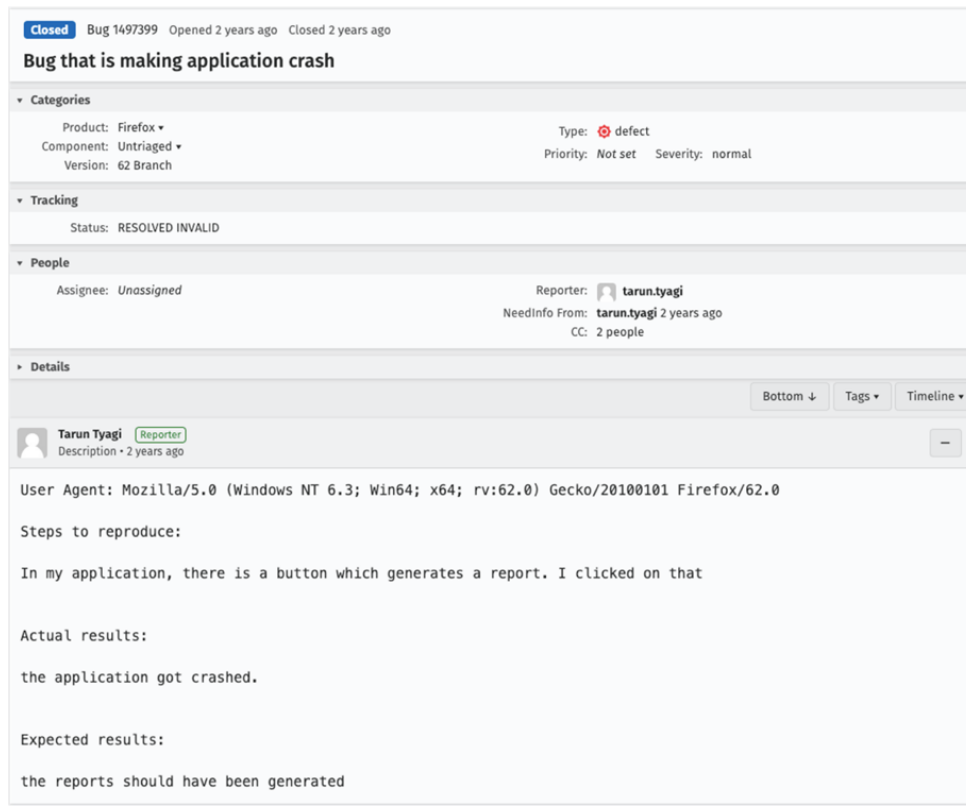


Figure 8. Example of a non-actionable bug report, bug id 1497399 (cropped).

Even though it may seem that its description has all the information needed to resolve the bug, the developer assigned to the bug report had to ask its reporter to provide a test case (*"Could you please provide a test case which shows the issue?"*), to which the reporter replied with a file containing a graphical illustration of the steps they took that led to the crash (Figure 9 (part 1)). However, the developer still could not reproduce the bug (*"Would you have a test case to reproduce it? Do you have a crash report in about:crashes?"*). A few months later, in light of no further response, the developer chose to close the bug report (*"Closing for now as we don't have enough information to act on it."*), see Figure 9 (part 2).

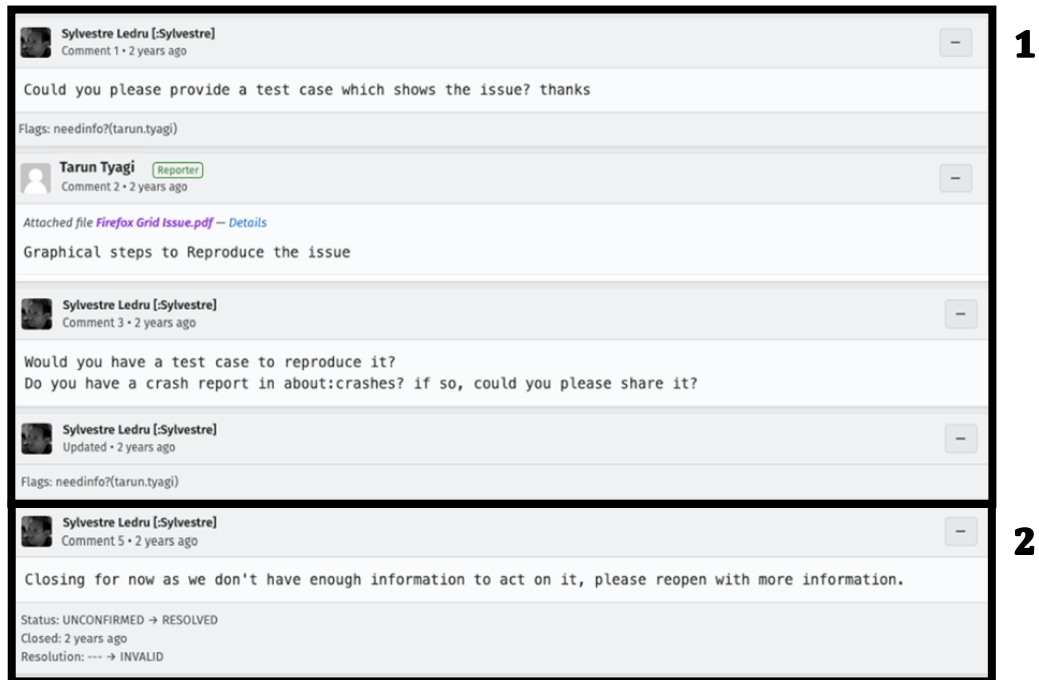


Figure 9. Comments from bug id 1497399 (cropped).

Not all bug reports are straightforward to categorize as actionable or non-actionable. For example, Mozilla bug id 1265173<sup>18</sup>, is an example of a bug report that is harder to categorize as actionable or non-actionable. As shown in Figure 10, the description of the bug was about a change in the color of active tabs and URL bar background. Initially the bug report looks actionable since it got triaged. However, a closer look reveals that the assignee of the bug commented *"please attach a screenshot showing how those colors are broken on your system."* It seems that the developer either wanted to obtain some auxiliary information to further see how the colors were broken in the screenshot of reporter's system, or they did not have enough information to understand the problem in the report.

<sup>18</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1265173](https://bugzilla.mozilla.org/show_bug.cgi?id=1265173)

**Closed** Bug 1265173 Opened 5 years ago Closed 5 years ago

### Broken background colors active tab and urlbar

**Categories**

Product: Firefox  
Component: Theme  
Version: 46 Branch

Type: defect  
Priority: Not set Severity: normal

**Tracking**

Status: RESOLVED FIXED  
Milestone: Firefox 49

| Tracking Flags: | Tracking | Status  |
|-----------------|----------|---------|
| firefox46       | ---      | wontfix |
| firefox47       | ---      | fixed   |
| firefox48       | ---      | fixed   |
| firefox49       | ---      | fixed   |

**Attachments**

**patch**  
5 years ago Dão Gottwald [:dao] •  
948 bytes, patch

mikedeboer : review+  
ritu : approval-mozilla-aurora+  
ritu : approval-mozilla-beta+

[Details](#) | [Diff](#) | [Splinter Review](#)

[Attach New File](#) [Show Obsolete](#)

[Add Comment](#) [Tags](#) [Timeline](#)

**J** Reporter  
Description • 5 years ago

User Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:46.0) Gecko/20100101 Firefox/46.0  
Build ID: 20160414152344

Steps to reproduce:

Default

Actual results:

Broken background colors active tab and urlbar

Expected results:

Active (current) tab background and urlbar background should be lighter

**Dão Gottwald [:dao]** Assignee  
Comment 1 • 5 years ago

J, please attach a screenshot showing how those colors are broken on your system.

Flags: needinfo?(yeeger9@gmail.com)

Figure 10. Example of a more difficult bug report to label, bug id 1265173 (cropped).

After the reporter provided the screenshot, another developer explained that the color of active tabs was actually an intended change *"This was changed in bug 1244500 in response to people with exactly the opposite complaint: that the lightness was inappropriate on dark themes"* and then asked, *"Did you file this "just" because there is a change, or do you have concrete problems with the new design?"*, (Figure 11 (part 1)). As the reporter confirmed that *"All of this is a bad change"*, the assignee provided a patch and modified the colors to fix the bug report (Figure 11 (part 3)). This bug report is an example of a non-actionable report, because a critical piece of information was missing in the initial submission of the report.

The developers did not know if the reporter did not like the recent change in the color or if the color was actually broken and needed their attention to be fixed.

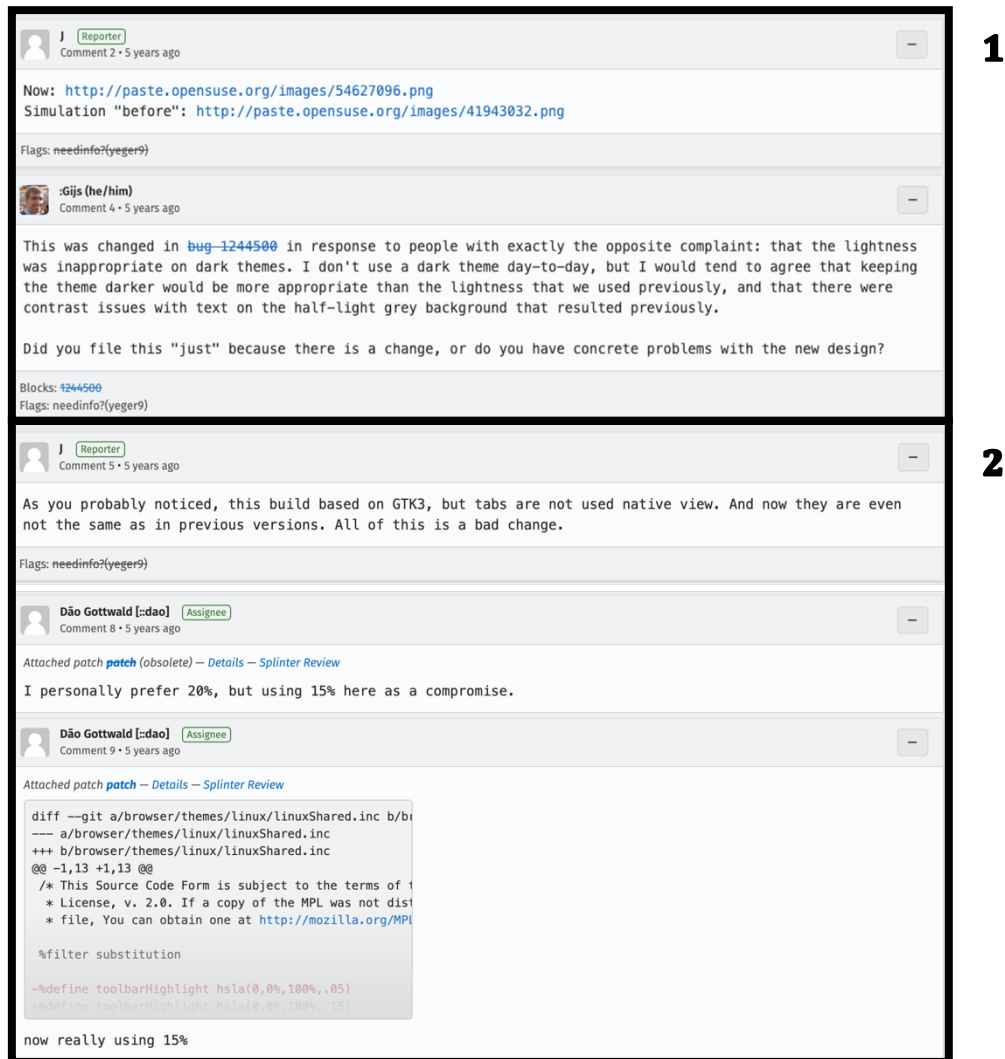


Figure 11. Comments from bug id 1265173 (cropped).

Mozilla bug id 621660<sup>19</sup> is another example of a bug report for which at first glance it is not obvious if it is actionable or non-actionable, see Figure 12. The reporter started with “*While I was able to successfully upgrade to this version from Beta 6, I cannot get it to work for Beta 8.*”, continued with “*I did not see this on the list of known issues*”, provided the steps to

<sup>19</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=621660](https://bugzilla.mozilla.org/show_bug.cgi?id=621660)

reproduce, actual results, and expected results, and even came up with some workaround for whom may have the same problem.

**Closed** Bug 621660 Opened 10 years ago Closed 10 years ago

### Stylish 1.1b1 breaks the extension manager which in turn breaks application update

**Categories**

Product: Firefox  
Component: Extension Compatibility  
Platform: x86 Windows XP

Type: defect  
Priority: Not set Severity: minor

**Tracking**

Status: RESOLVED FIXED  
Milestone: Firefox 4.0b6

**Terrell Kelley** Reporter  
Description 10 years ago

User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:2.0b7) Gecko/20100101 Firefox/4.0b7  
Build Identifier: Mozilla/5.0 (Windows NT 5.1; rv:2.0b7) Gecko/20100101 Firefox/4.0b7

While I was able to successfully upgrade to this version from Beta 6, I cannot get it to work for Beta 8. When ever I use the About Firefox dialog, I get the spinner, and it never determines whether there is a new version or not.

I did not see this on the list of known issues, and, for all I know, you fixed it Beta 8.

Reproducible: Always

Steps to Reproduce:

1. Click Help on either the Firefox button or the menu bar.
2. Click About Firefox.

Actual Results:  
Dialog box continually says "Checking for updates."

Expected Results:  
Dialog box should say "Update Found," "Click here to update," and/or similar.

I'm using a copy of a profile I've been using since Firefox 3.0.

And do remember that this worked properly back on Beta 6.

And please do not ignore because it's from Beta 7, as your feedback button seems to claim you will. It would be improper to say the bug was in Beta 8.

The "easy workaround is to install Beta 8 manually."

Figure 12. Example of a more difficult bug report to categorize, bug id 621660 (cropped).

At first glance, it seems that the bug's description is clear and complete, and thus represents an actionable bug report. However, the bug report involved an exchange between a pair of developers and the reporter that reveals a different story, see Figure 13 (part 1). The assigned developer asked for additional information “... *Post your Error Console Output*”, see Figure 13 (part 2). Once the reporter responded by submitting the requested information, a second developer asked for yet more information “*Please post a list of the extensions you have installed.*” Once the reporter provided the information, the developers could start to figure out what was wrong and resolved the bug, see Figure 13 (part 3). While the developers may have had an initial hunch, they did need significant additional information to actually be

able to diagnose where they should begin the exploration of this particular bug report. As such, I labeled this bug report non-actionable while labeling the training data.

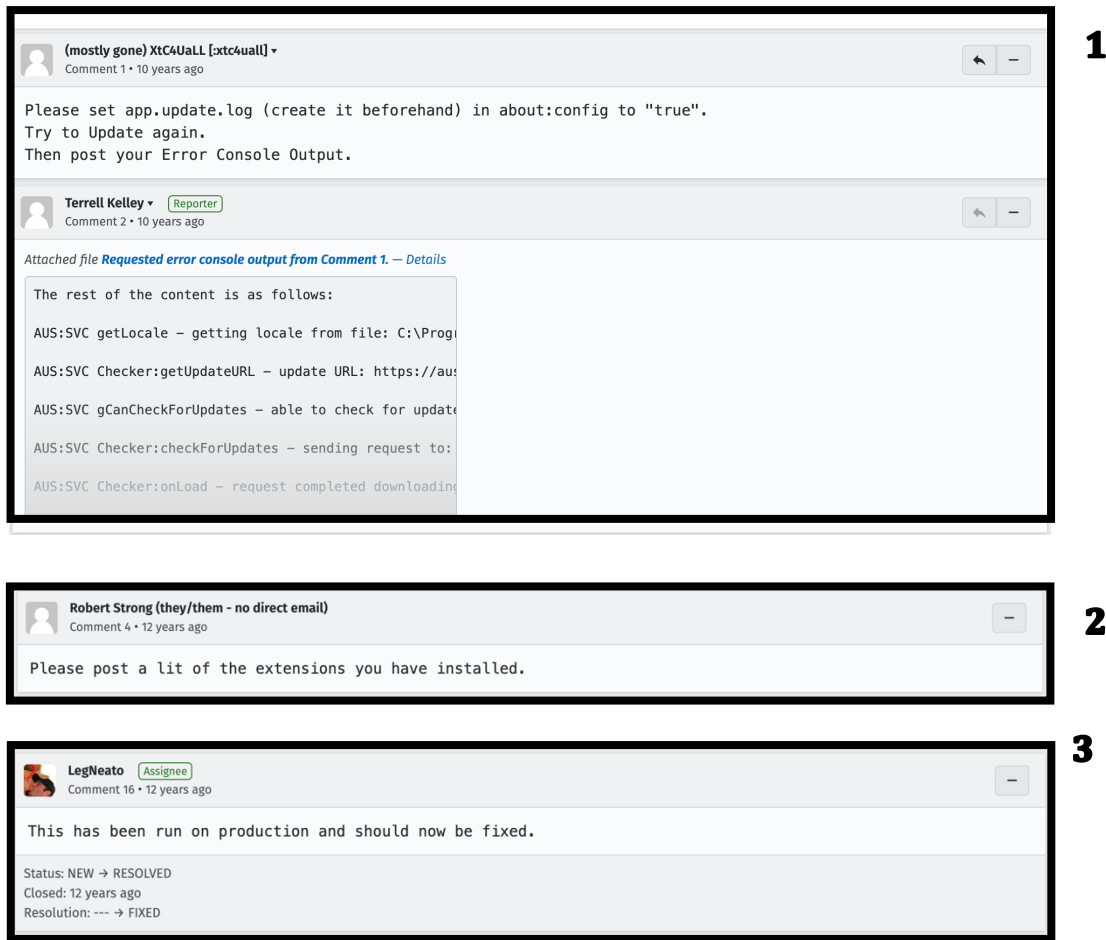


Figure 13. Comments from bug id 621660 (cropped).

### 3.2. Methodology

Compared to the current state of the art, this work differs in two ways: (1) it aims to achieve a significantly higher level of accuracy so to be able to use the classifier as part of a tool that could flag newly composed bug reports as actionable or non-actionable and give reporters the guidance to improve the reports, and (2) it focuses on actionable versus non-actionable, which requires a binary classification rather than one that is more fine-grained as offered by prior work. Instead of a finer-grained classification, a binary one is used because for



purposes of developing a tool that assists reporters in submitting bug reports that impose less work on developers to understand and process them, a binary distinction is enough.

Figure 14 illustrates the process that was used to study whether newly submitted bug reports can be predicted to be actionable or non-actionable. The process starts by collecting a sample of bug reports across all Mozilla Firefox projects, out of which a subset was manually labeled as actionable or non-actionable. Then typical pre-processing steps were applied on the bug reports' descriptions by applying stemming and removing special characters and stop words. After that, the bag of words model was used and calculated term frequency-inverse document frequency (tf-idf) [73] to account for the frequency of words based on their occurrence in the entire corpus of bug report descriptions, which formed the basis for supervised learning with four different machine learning classifiers: Naïve Bayes (NB), Support Vector Machine (SVM), Random Forest (RF), and Decision Tree (DT).

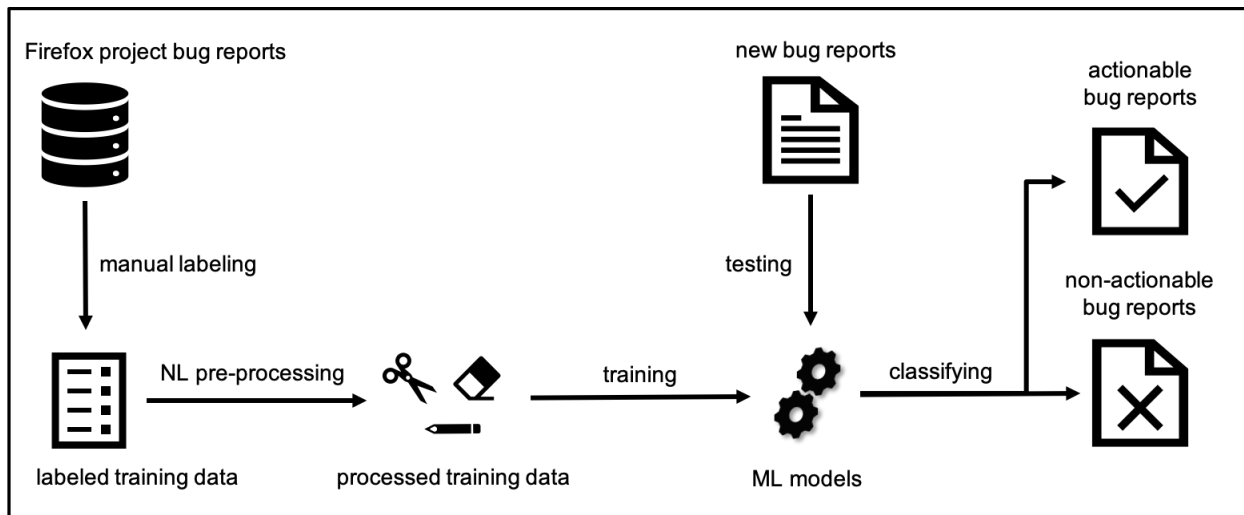


Figure 14. Process of classifying the bug reports.

Next, the performance of these four classifiers was compared and the one with the best results was selected, expanding the analysis to not just be based on the descriptions of bug reports, but additional factors as well, including, among others, whether bug reports were submitted by developers or end-users, whether bug reports included attachments, and readability scores. Then the ability of the best classifier was examined to perform cross-

project prediction by applying it on a previously existing dataset containing bug reports from Eclipse and Apache. In the following sections, I describe each of these steps in more details.

### 3.1.1 Data Collection

I collected 1,423 bug reports across all Mozilla Firefox projects (Firefox, Firefox Build System, Firefox Android, Firefox Echo Show, Firefox FireTV, Firefox iOS, Firefox Friends, and Firefox Private Network) that were labeled as “resolved”, but not as “wont-fix” or “duplicate”. The focus was specifically on resolved bug reports because, for purposes of training the classifiers, these have the most available information to decide whether a bug report appeared actionable or non-actionable to the developer. This information includes the eventual resolution (fixed, invalid, wontfix, moved, duplicate, worksforme, incomplete) as well as any back-and-forth conversations with the reporter, which tends to provide valuable clues as to how a developer might have interpreted a bug report on first submission.

For unresolved bug reports, making the determination as to whether they are actionable or non-actionable is much more uncertain, as an unresolved bug report could simply not have been read yet; been read, deemed actionable, but not updated in the system yet; or read, deemed non-actionable, and ignored – with no information available as to which is which. This uncertainty led to select resolved bug reports only.

Bug reports with a resolution of “wontfix” appear to include some that are simply not a priority, some that appear not relevant, and some that are not actionable. Because I could not often tell the reason with certainty, I excluded these bug reports from the dataset. For obvious reasons, I equally excluded “duplicate” bug report.

I selected Mozilla Firefox since it is a large-scale open-source project on multiple platforms, and has been widely used in empirical software engineering research [123]. For studying portability of the best model, a publicly available dataset from Cuezilla was used [14], which includes labeled bug reports for Apache (90) and Eclipse (100). The dataset also includes bug reports for Mozilla, which was ignored for obvious reasons in examining portability. However, those bug reports were used in a separate analysis in Section 3.2.1.1 to examine the impact of hyper-parameter optimization.

### 3.1.2 Manual Labeling

Two researchers manually labeled all 1,423 bug reports as actionable or non-actionable. For labeling purposes, the researchers analyzed: (1) the description of the bug reports, (2) if present, associated comments from discussions with other developers, and (3) any interactions that took place between the developer and the reporter. Particularly, the researchers assessed whether the developer or set of interacting developers could come to an understanding about the bug report on their own, without further interaction with the reporter. If so, the bug report was labeled as actionable. If interaction with the reporter did take place, the nature of the interaction was examined: does it appear that the developer is asking for a minor clarification or minimal additional information, clearly within the context of showing an understanding of the bug being reported, or does it appear that the developer is asking for significant clarification or important additional information that shows that they do not yet have a grasp of what the bug report entails? In other words, has the primary diagnosis already taken place, or is more needed before it can take place? In the former case, the bug report was labeled as actionable; in the latter as non-actionable.

The manual classification was conducted using the “negotiated agreement” method [124] by two researchers in three phases. In the first phase, researcher one collected the elements of good and bad bug reports as documented in prior research and discussed in Section 2.2.1. As a reminder, past studies for instance identified the importance of current behavior, steps to reproduce, and expected behavior [33], [125]. Next, researcher one manually analyzed 219 bug reports, categorizing and grouping them in order to determine a set of features for identifying and differentiating actionable bug reports from non-actionable ones. Once a high-level understanding emerged, the researcher created a set of guidelines, as follows:

- Whether or not a bug report is actionable should be judged based on developers' actions. Could they understand the bug report on their own and move ahead with a determination and action without interacting with the reporter? If so, the bug report should be labeled as actionable.

- The presence of fields indicating quality bug reports (e.g., current behavior, steps to reproduce, expected behavior) should be taken as an indication that a bug report might be actionable, but does not necessarily guarantee that it is. It is important to verify if the information provided appears clear and complete and examine it in the context of the next guideline.
- The number and nature of messages between reporters and developers should be closely examined: does it appear that the developer is asking a straightforward question regarding a minor clarification or minimal additional information, or is it a fundamental question and is it related to the nature of the bug report? In the former case, the bug report should be labeled actionable, in the latter case, non-actionable.
- Depending on the nature of a bug, some information is always required because otherwise developers will ask for it. For example, if a bug reports a crash, the developer needs the crash ID or stack trace. If this information is not provided in the initial bug report, it should be labeled non-actionable.

In the second phase, 440 bug reports were randomly selected and independently labeled by each of the two researchers using the above guidelines. The resulting labels were in agreement for 403 of the bug reports, which led to an inter-rater reliability Cohen's Kappa [126] of 0.82, which indicates “almost perfect agreement” [127]. For the remaining 37 bug reports, both researchers reviewed them together, and for each bug report discussed their respective reasoning and the source of disagreement. This led to one additional guideline:

- If a bug report was resolved, its secondary label should not be directly mapped to actionable or non-actionable (e.g., “fixed” to actionable, “incomplete” to “non-actionable”).

Our classification, after all, is about the initial bug report, not about the state it eventually finds itself in. It might be, for instance, that a bug was fixed, but involved extensive back and forth. It might equally be that a bug was deemed “incomplete”, but only after the developers engaged in a detailed exploration of the source code based on a very clear bug report that eventually was deemed to point to a different issue altogether. In the former case, the initial bug report should be labeled non-actionable; in the latter actionable.

Using the adjusted guidelines, both researchers independently labeled the 37 bug reports again, resulting in an inter-rater reliability Cohen's Kappa of 0.99 across all 440 bug reports, which indicates “perfect agreement” [128]. At this point, the researchers felt sufficiently confident to move to the third phase to label the remaining 983 bug reports, half by one researcher and the other half by the other. In total, 41% of bug reports (583) were classified as non-actionable and 59% (840) as actionable.

Afterwards, the corpus was queried to count the number of messages (in GitHub terms: comments) attached to each bug report. For 95% of the actionable bug reports, the number of messages was three or less, with the average across all actionable bug reports seven.<sup>20</sup> Compared to the non-actionable bug reports, for which only 30% had three or fewer comments and the average was 13, this is an indication that the manual classification might indeed reasonably separate actionable from non-actionable bug reports, since one would expect a longer thread of comments for the latter.

The additional bug reports from Cuezilla already were labeled by the authors of the Cuezilla study. The labels, however, were good, neutral, and bad. I mapped good to actionable, dropped the neutral bug reports from the dataset, and mapped bad to non-actionable. As a result, this corpus reduced to 69 bug reports for Mozilla (34 actionable), 54 for Apache (25), and 67 for Eclipse (34).

### 3.1.3 Text Mining

The primary classifiers that were explored are based on the description field of the bug reports. That is, for each bug report, the description was extracted and other aspects were ignored, such as priority, name of the reporter, title, resolution, secondary labels, and so on. Since the descriptions are primarily written in natural language, several common pre-processing steps were applied to clean the data before applying the four machine learning classifiers that were chosen for comparison. Then the nltk (Natural Language ToolKit)

---

<sup>20</sup> Within the remaining 5%, several outliers contained very lengthy exchanges among developers to, among others, identify the right developer to address the bug report, find a meeting time for when to discuss it, or plan for which future release should incorporate a fix. These represent auxiliary discussions that skew the average, but do not detract from the bug report being actionable.

library<sup>21</sup> was used to normalize the data and apply Lemmatization. Stop words, special characters, numbers, and single characters were removed as well. Finally, all textual descriptions were converted to lower-case.

Next, every sentence in the dataset was split into a list of words, with each distinct word corresponding to a feature, the value of which was set to the number of times the word occurs in a given bug report. The bag of words model was used from the *scikit-learn* library [129] and then applied tf-idf [73] to account for the fact that words may be common across the entire corpus of bug reports.

### 3.1.4 Machine Learning Classifiers

Using the processed bug report descriptions, four different machine learning classifiers were trained, namely Naïve Bayes (NB), Decision Tree (DT), Support Vector Machine (SVM), and Random Forest (RF). These four classifiers were selected because they produced the best results in classifying bug reports in prior studies [14], [32], [33]. Then, the *scikit-learn* library [129] was employed to implement the classifiers. Next, hyper-parameter optimization was performed using grid search [66], utilizing the *scikit-learn GridSearchCV* function to determine the optimal parameter settings for each of the four classifiers. To compare the performance of the various models that resulted, the standard measures of precision, recall, and F-measure were used.

Using the best resulting model, I then expanded the analysis to not just be based on the descriptions of bug reports, but additional factors. For each factor, a score was awarded to the bug report, which was either binary (e.g., attachment present or not) or continuous (e.g., readability score), and examined if any of the factors, alone or in combination with others, could further improve the results:

- 1- Presence of attachments. A binary score was assigned to this feature depending on whether the initial bug report included an attachment, regardless of the type (e.g., screenshot, stack trace, patch).

---

<sup>21</sup> <https://www.nltk.org/>

- 2- Readability. The style tool<sup>22</sup> was used to assign a readability score (called the Flesch score [130]) to each bug report.
- 3- Submitted by an end-user or developer. To determine whether a reporter is an end-user or developer, the number of patches that the reporter of each bug report previously submitted to the chosen Mozilla Firefox projects was extracted. If the number was zero, the reporter was identified as an end-user, otherwise as a developer.
- 4- Length of bug description. After text-preprocessing, each bug report was assigned a length based on the number of words remaining in the resulting description.
- 5- Experience of the reporter. To determine the experience level, the number of prior bug reports submitted by a reporter was extracted and set this value as their experience level.

These factors were extracted because they have been observed in previous research to impact the quality of bug reports (e.g., [14], [21]) or, from the experience of manually labeling the bug reports, seemed like they may influence actionability (particularly whether a bug report is submitted by a developer as opposed to an end-user).

### 3.1.5 Cross-Project Prediction

As explained in Chapter 2 Section 2.1.7.4, cross-project prediction concerns the portability of a learned prediction model among different projects [68]. This study examines to what extent the best-performing model that resulted from the training on Mozilla Firefox projects preserves its performance on other projects, specifically to bug reports from Apache (54) and Eclipse (67) as taken from the Cuezilla dataset. To train and evaluate the classifiers [67], k-fold cross validation was used. This validation approach randomly divides the manually classified data set into k groups of equal size. The first group is treated as a validation set, and the classifier is fit on the remaining nine groups. The mean of the k executions is used as an estimation of the classifier's accuracy.

---

<sup>22</sup> <https://pypi.org/project/readability>

### 3.1.6 Survey

In order to place the results in context, I conducted a short survey among Mozilla developers. The goal was to obtain an understanding as to how they feel the quality of bug reports has or has not changed over the years and, if it has changed, what they believe may have caused that change. Additionally, the survey asked whether developers felt that the quality of bug reports has increased over the years, and what might have caused such improvements (and vice versa if they felt that quality decreased, what may have caused the decline). The following three questions were specifically asked:

- 1- How many years have you worked on the Mozilla project?
- 2- How would you characterize the quality of newly submitted bug reports today as compared to the quality of newly submitted bug reports in the early days of you working on the Mozilla project (significantly better, somewhat better, remained about the same, somewhat worse, significantly worse).
- 3- Please expand on your answer. Why do you believe the quality has or has not changed? Can you give examples of what aspects of newly submitted bug reports have or have not changed? If the quality has changed, what do you believe has caused the change in quality over time?

First, the survey was piloted with two researchers and one Ph.D. student with experience in the area to get feedback on the questions and their corresponding answers, difficulties faced answering the survey, and time to finish it. Several iterations of the survey were conducted and the questions were rephrased and others were removed to make the survey easier to understand and answer. The responses from the pilot survey were used solely to improve the questions and these responses were not included in the final results. The survey was kept anonymous but, at the end, the respondents could provide their email to receive a summary of the study. Then the survey was sent to developers who were listed as 'assignee' for 10,000 random bug reports that were submitted (and assigned) in the years 2018 and 2019. Given that assignees can repeat, I removed duplicates and thus sent the survey to 2,839 unique developers. Excluding 324 e-mail addresses that bounced, in total 2,515 invites were



delivered. Participation was voluntary, anonymous, and participants were allowed to discontinue at any time; participants did have to consent to participating. I received 53 responses (2.1% response rate), which is below other studies in the field (e.g., 5.7% [97], 7.9% [98]). Because the survey was intentionally kept extremely short, I had hoped for a higher response rate. The survey hit, however, right at the time that Mozilla laid off 25% of its workforce<sup>23</sup>. Further, I also received an e-mail from the Mozilla security team, as a number of developers felt their privacy had been violated, which led to a long internal discussion at Mozilla. Both factors, I suspect, negatively influenced the response rate.

## 3.2 Results and Discussion

In this section, I present the results of the study. I first investigate the ability to predict the quality of bug reports in terms of whether or not they are actionable. Next, I investigate additional features and whether they can further improve the results. Finally, I examine the potential of portability of our best model.

### 3.2.1 Predicting Actionable versus Non-Actionable

I trained four machine learning classifiers: Naive Bayes (NB), Decision Tree (DT), Support Vector Machine (SVM), and Random Forest (RF). For SVM, a sigmoid kernel with standard values was used. To compare the performance of the various models that resulted, the standard measures of precision, recall, and F-measure were used. Table 1 summarizes the results of the classifiers on the 1,423 bug reports of the corpus, reporting the precision, recall, accuracy, and F-measure of each classifier with and without applying hyperparameter optimization.

Table 1. Performance of the classifiers.

| MODEL | PRECISION | RECALL | ACCURACY | F-MEASURE |
|-------|-----------|--------|----------|-----------|
|-------|-----------|--------|----------|-----------|

<sup>23</sup> <https://www.cnet.com/news/mozilla-cutting-250-jobs-after-coronavirus-pandemic-cuts-revenue>

|                     |      |      |      |      |
|---------------------|------|------|------|------|
| <b>SVM + tuning</b> | 0.94 | 0.89 | 0.92 | 0.91 |
| <b>RF + tuning</b>  | 0.85 | 0.93 | 0.86 | 0.89 |
| <b>DT + tuning</b>  | 0.81 | 0.92 | 0.96 | 0.86 |
| <b>NB + tuning</b>  | 0.63 | 0.92 | 0.86 | 0.74 |
| <b>NB</b>           | 0.65 | 0.76 | 0.84 | 0.69 |
| <b>DT</b>           | 0.96 | 0.46 | 0.66 | 0.62 |
| <b>RF</b>           | 0.81 | 0.38 | 0.84 | 0.50 |
| <b>SVM</b>          | 0.84 | 0.31 | 0.62 | 0.43 |

Interestingly, while SVM performs worst without hyper-parameter optimization, it performs best with, attaining an F-measure of 0.91 (with  $\gamma$  of 0.01 and C of 10). RF + tuning and DT + tuning follow closely with an F-measures of 0.89 (with max\_depth of 90) and 0.86 (with max\_leaf\_nodes of 100), respectively.

In all four cases, hyper-parameter optimization improves the predictive capability, which in many ways is not surprising given its observed impact in past studies (e.g., [133]). Especially for SVM, which is very sensitive to the choice of parameter tuning, hyper-parameter optimization can make a very significant difference [134]. The typical values were studied for  $c$  ( $0.1 < c < 100$ ) and gamma ( $0.0001 < \gamma < 10$ ), as recommended by previous studies [68][69] to tune the SVM classifier. Note that for all four classifiers, tuning improves the accuracy, F-measure, and recall. Precision on the other hand was always high, regardless of applying hyper-parameter optimization, except in NB + tuning where precision drops but recall significantly increases. For completeness, the precision and recall of all four classifiers is reported with hyper-parameter optimization for both classes, actionable and non-actionable bug reports, in Table 2.

Table 2. Results per class (hyper-parameter optimization).

| MODEL               | LABELS         | PRECISION | RECALL | ACCURACY | F-MEASURE |
|---------------------|----------------|-----------|--------|----------|-----------|
| <b>SVM + tuning</b> | actionable     | 0.92      | 0.88   | 0.97     | 0.92      |
|                     | non-actionable | 0.94      | 0.90   | 0.85     | 0.91      |
| <b>RF + tuning</b>  | actionable     | 0.86      | 0.92   | 0.88     | 0.88      |
|                     | non-actionable | 0.85      | 0.93   | 0.85     | 0.88      |
| <b>DT + tuning</b>  | actionable     | 0.81      | 0.92   | 0.96     | 0.86      |
|                     | non-actionable | 0.81      | 0.91   | 0.88     | 0.85      |
| <b>NB + tuning</b>  | actionable     | 0.68      | 0.88   | 0.83     | 0.76      |
|                     | non-actionable | 0.60      | 0.95   | 0.85     | 0.73      |

Compared to the current state of the art, SVM + tuning achieves the best results to date, with Cuezilla [14] and Schuegerl et al. [32] only achieving an accuracy of 50% and 44%, respectively. Note that Cuezilla and Schuegerl et al. did not report precision, recall, and F-measure. So, the performance results of the models can be only compared in terms of accuracy.

### 3.2.1.1 Impact of Hyper-Parameter Optimization

To better contextualize the sizeable impact of hyper-parameter optimization on the results, the best-performing classifier (SVM + tuning) was chosen and trained on the Cuezilla dataset (Apache, Eclipse, and Mozilla data – the only time the Mozilla data from the Cuezilla dataset was used, since this analysis is entirely independent). The classifier was trained both on the original dataset for three outcomes (good, neutral, bad) and on the reduced dataset with two

outcomes (good, bad), to control for the fact that prediction with fewer outcomes tends to nearly always perform better than prediction with more outcomes. Table 3 presents the results (only in terms of accuracy, since the original Cuezilla paper did not report F-measure). Note that, for all three of Apache, Eclipse, and Mozilla, results improved compared to the original Cuezilla results, signifying that hyper-parameter optimization indeed made a difference. At the same time, the accuracy remains below the one reported in Table 1. The impact of predicting just two outcomes presented an interesting result, in that the results improved for Apache and Eclipse when compared to predicting three outcomes, but results were worse for Mozilla. This may be because of the relatively small data sets involved.

Table 3. Accuracy of the best performing model and Cuezilla's model.

| <b>MODEL</b>                             | <b>APACHE</b> | <b>ECLIPSE</b> | <b>MOZILLA</b> |
|--|---------------|----------------|----------------|
| <b>ORIGINAL CUEZILLA RESULTS</b>         | 0.50          | 0.45           | 0.43           |
| <b>SVM + tuning (good, neutral, bad)</b> | 0.62          | 0.76           | 0.82           |
| <b>SVM + tuning (good, bad)</b>          | 0.83          | 0.85           | 0.72           |

A significant difference between this approach and that of Cuezilla is that my classifiers are based on the bag of words model, whereas Cuezilla uses fewer, higher-level features (e.g., certain keywords appearing in the description field, attachments, readability score). In addition to using a significantly larger dataset, and hyper-parameter optimization, the difference in underlying approach chosen could be contributing to my improved performance as well. That said, because of Cuezilla's relative success in only using high-level features, I investigate whether adding similar features to our classifier may result in even better performance (Section 3.2.2).

### 3.2.1.2 Feature Importance

I studied feature importance in the SVM + tuning model. Simple coefficient statistics between each feature and the target variable could not be calculated, since for the SVM classifier, such calculations only work with a linear kernel and our best results were with sigmoid. Hence, permutation feature importance was used for classification [137], which calculates relative importance scores and is independent of the model used. Then, 8,677 unique words were extracted (after text mining) from the descriptions of our 1,423 bug reports, with Table 4 presenting the 20 words with the highest scores. From the low scores of importance, I believe that none of these are outliers that unduly dominate the classification results on their own, especially given the broad set of words involved across all the bug reports.

Table 4. Feature importance (sorted by relative importance).

| FEATURE (WORD)  | IMPORTANCE |
|-----------------|------------|
| <b>mozilla</b>  | 0.0542     |
| <b>bug</b>      | 0.0136     |
| <b>switch</b>   | 0.0109     |
| <b>attach</b>   | 0.0105     |
| <b>testcase</b> | 0.0101     |
| <b>ubuntu</b>   | 0.0096     |
| <b>firefox</b>  | 0.0089     |
| <b>buildid</b>  | 0.0080     |
| <b>night</b>    | 0.0063     |

|                  |         |
|------------------|---------|
| <b>step</b>      | 0.0062  |
| <b>will</b>      | 0.0054  |
| <b>there</b>     | 0.0031  |
| <b>product</b>   | 0.0026  |
| <b>broke</b>     | 0.0016  |
| <b>com</b>       | 0.0015  |
| <b>http</b>      | 0.0015  |
| <b>internet</b>  | 0.0011  |
| <b>runtim</b>    | 0.0010  |
| <b>stop</b>      | 0.0010  |
| <b>searchbox</b> | 0.00099 |

With respect to words such as ‘mozilla’ and ‘bug’ appearing in the top 20, one can actually be positive about these kinds of terms having as low a feature importance as they do. It would not have been surprising had the importance been higher since they can easily be expected to be part of many Mozilla bug reports.

### 3.2.2 Additional Features

It is possible to predict the overall quality of bug reports as actionable or non-actionable with a high level of accuracy and high F-measure. However, the goal was to understand whether results could be further improved, especially since the results were based on the relatively straightforward application of machine learning using a bag of words model with hyper-

parameter optimization. Moreover, in prior research, adding auxiliary features such as, for instance, length of bug report description or presence of stack traces was shown to have a positive impact on the predictive capability of the resulting models [14], [21]. In the below, five additional features were studied, both individually and combined: presence of attachments, readability, submitted by an end-user or developer, length of bug description, and experience of the reporter.

Each factor was awarded a score to the bug report, which was either binary (e.g., attachment present or not) or continuous (e.g., readability score). Then I examined if any of the factors, alone on or in combination with others, could further improve the results.

As the base model for doing so, SVN + tuning was used, since it had the best results. Table 5 presents the augmented results. Foreshadowing the discussion below in the following sections, the base classifier outperforms any of the others. The additional features do not seem to provide benefits, which is contrary to prior studies.

Table 5. Performance of the SVN + tuning model augmented with additional features, top six and bottom six.

| MODEL                                | PRECISION | RECALL | ACCURACY | F-MEASURE |
|--------------------------------------|-----------|--------|----------|-----------|
| <b>SVM + tuning</b>                  | 0.94      | 0.89   | 0.92     | 0.91      |
| <b>SVM + tuning + D<sup>24</sup></b> | 0.91      | 0.90   | 0.92     | 0.90      |
| <b>SVM + tuning + A<sup>25</sup></b> | 0.89      | 0.87   | 0.92     | 0.87      |
| <b>SVM + tuning + D + A</b>          | 0.91      | 0.83   | 0.81     | 0.86      |
| <b>SVM + tuning + R<sup>26</sup></b> | 0.94      | 0.77   | 0.85     | 0.84      |
| <b>SVM + tuning + E<sup>27</sup></b> | 0.79      | 0.80   | 0.82     | 0.79      |

<sup>24</sup> Developer or end-user

<sup>25</sup> Attachment

<sup>26</sup> Readability

<sup>27</sup> Reporter's experience

| ...  | ...  | ...  | ...  | ...  |
|--|------|------|------|------|
| <b>SVM + tuning + E + A + L<sup>28</sup></b> | 0.69 | 0.80 | 0.84 | 0.74 |
| <b>SVM + tuning + E + R + L</b>              | 0.66 | 0.83 | 0.80 | 0.73 |
| <b>SVM + tuning + E + A + D + L</b>          | 0.65 | 0.79 | 0.83 | 0.72 |
| <b>SVM + tuning + R + A + D + L</b>          | 0.76 | 0.64 | 0.83 | 0.69 |
| <b>SVM + tuning + E + R + A + D + L</b>      | 0.68 | 0.67 | 0.77 | 0.67 |
| <b>SVM + tuning + E + R + A + L</b>          | 0.67 | 0.59 | 0.73 | 0.62 |

As backdrop for the analyses I present next, I also extracted 180,000 bug reports randomly across all Mozilla projects from the years 2002 to 2019, 10,000 each year, and used them to determine potential trends as related to the five auxiliary features that I analyze. By placing the performance of the augmented classifiers in the context of these features' historical trends, I can better hypothesize about why I might be seeing the kinds of results I found.

### 3.2.2.1 Presence of Attachment

The trend of bug reports with attachments is shown in Figure 15. Over the years, there clearly is an increase to, today, over half of the bug reports including an attachment. Given that attachments are generally considered a sign of a good bug report [5][44], I thus extended and re-trained the SVM + tuning model to incorporate a binary feature: does a bug report include an attachment such as a screenshot, stack trace, or patch? Contrary to prior research [14], this did not lead to an increase in performance. The resulting accuracy was 92% (the same) and the F-measure 0.87 (a drop of 0.04).

---

<sup>28</sup> Length of bug description



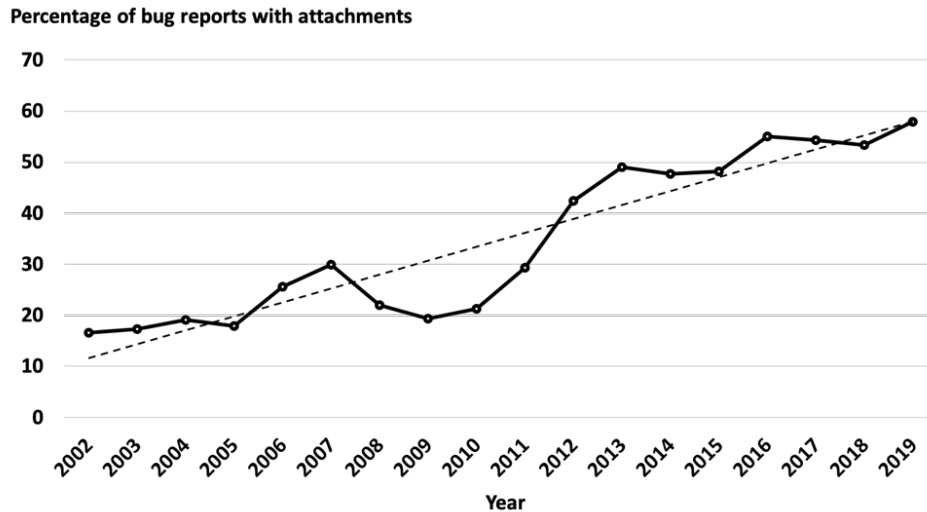


Figure 15. Percentage of bug reports which have attachments.

The decrease in F-measure is too small to draw strong conclusions. Nonetheless, I observe that the presence of attachments may not be as important a feature when the base model produces sufficiently strong results on its own. Its apparent non-importance might also be because a higher number of bug reports now include attachments. A possible reason could be that before, the inclusion of an attachment seemed to be a practice amongst those who truly cared about high-quality bug reports and thus were diligent in not only writing clear explanations but also including as much other evidence. Today, however, including attachments appears to be more of a rote activity practiced by many more reporters, a good many of whom may still be producing descriptions that lack significantly. A feature that covers whether a bug report has an attachment, then, may have lost its defining value as compared to 10 years ago.

### 3.2.2.2 Readability

The readability of a bug report can clearly have an impact on its actionability or non-actionability. Indeed, as part of the experiments surrounding Cuezilla, readability was included as an extra feature [14]. I mimicked this by using the style tool<sup>29</sup> to assign the Flesch

<sup>29</sup> <https://pypi.org/project/readability>

readability score [130] to each bug description and augment the base model of SVM + tuning with this score. Both in terms of accuracy (a drop of 0.07) and F-measure (a drop of 0.07), the resulting model did not perform as well.

Placed in the context of how readability scores have evolved over the years (see Figure 16), no clear reason emerges. The readability score has been relatively constant over the years, meaning that it should not have altered the effect that was seen in Cuezilla. If anything, the results suggest that readability is not a good proxy for how informative a bug report's description is.

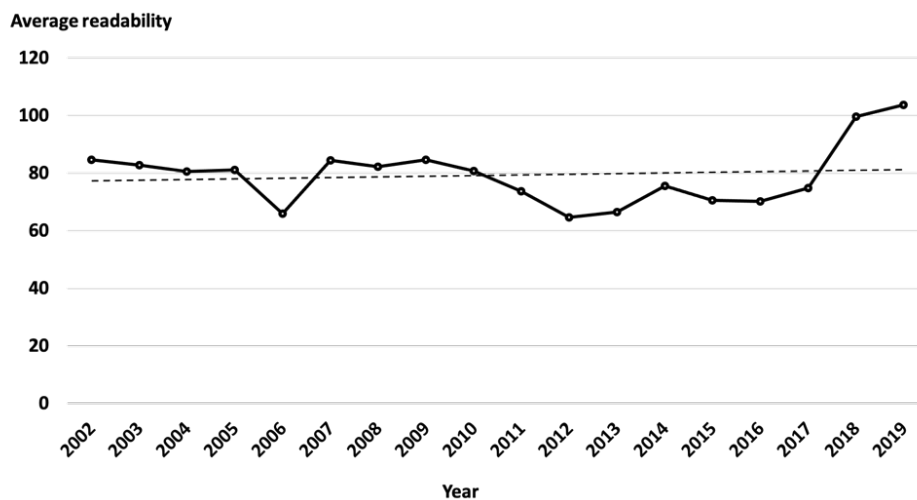


Figure 16. Average readability score of bug descriptions.

### 3.2.2.3 End-user versus Developer

One possible factor, not previously studied, is whether bug reports are submitted by end-users or developers. Possibly, since developers are generally more experienced than end-users, they might be more likely to submit bug reports more frequently. Moreover, since they are also regularly the ones having to process bug reports submitted by others, one might expect them to have an understanding of what actionable bug reports look like and make the effort to submit such bug reports themselves. Especially in the context of the percentage of bug reports being submitted by developers having significantly increased over the years

from 12% to 52% (see Figure 17), I studied the inclusion of a feature as to whether a bug report is submitted by an end-user or developer.

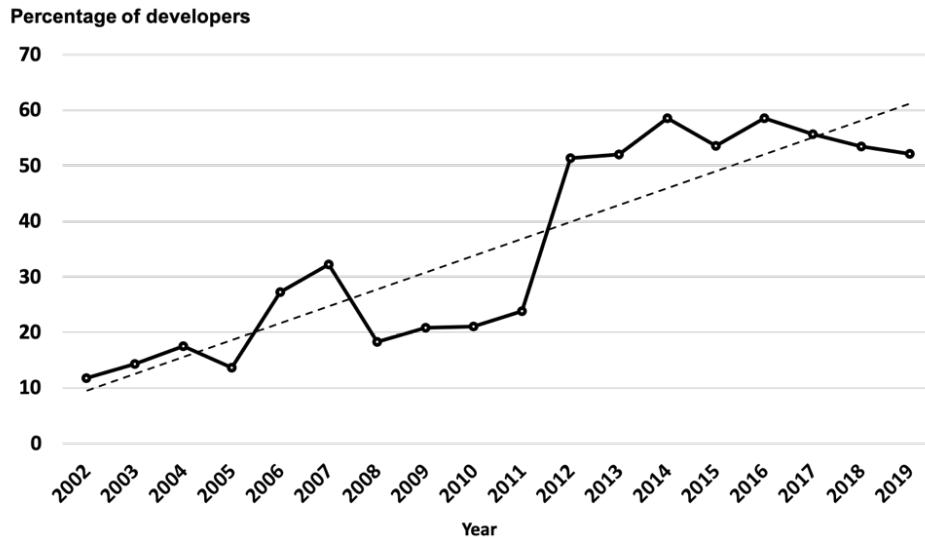


Figure 17. Percentage of bug reporters who are developers.

The results, once again, were not what was believed might happen. The accuracy was 92% (the same) and the F-measure dropped by a mere 0.01. These effects are negligible, and thus no conclusions can be drawn regarding the impact of including the end-user versus developer feature in the prediction model.

#### 3.2.2.4 Length of Bug Description

Prior work observed that a good bug report often has a long textual description of the problem [138]. Especially in light of the average meaningfully declining over the years (see Figure 18), which does show an interesting “recovery” over the past two years), it is worthwhile examining this factor. I once again constructed a new model based on the SVM + tuning model and examined the effect. Out of all the features I considered, this model performed the worst. Accuracy dropped to 82% (-0.10%) and F-measure to 0.76 (-0.15).

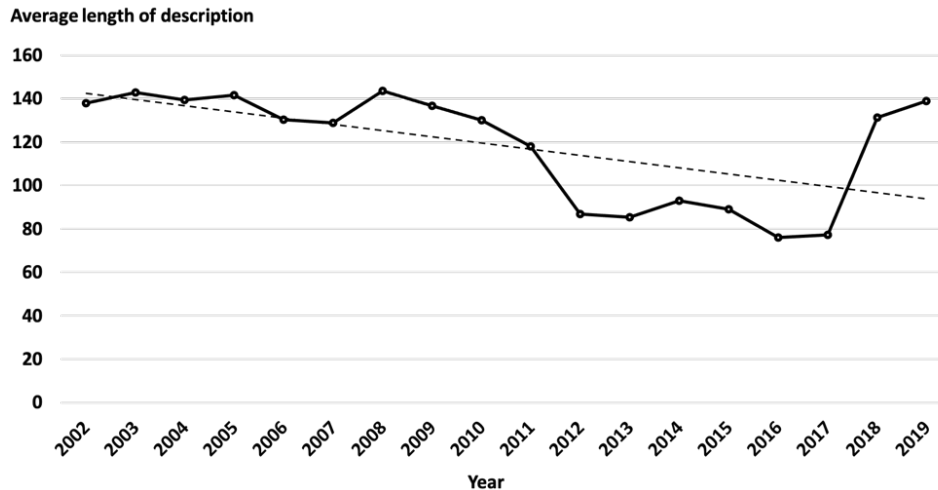


Figure 18. Average length of bug descriptions.

The inclusion of length, thus, had a negative impact on performance, contrary to what the prior work found. I speculate that a factor could be the difference in study. Bhattacharya et al. [138] drew their conclusion based on the assertion that bug reports that are resolved more quickly are good bug reports. Obtained by examining actual textual content of bug reports, the results show that length is not a good predictor, thus indicating that the assertion made by Bhattacharya et al. is inaccurate and that using time to resolution as a proxy for quality of a bug report appears not appropriate. This corroborates what the authors of Cuezilla found, as they already had observed that bug resolution time is independent of quality and is instead dominated by the severity/urgency of the bug at hand.

I also calculated the average length of bug descriptions submitted by developers versus end-users. As Figure 19 shows, for each of the past 18 years, developers submitted shorter bug descriptions on average. This is particularly interesting in the context of the discussion in Section 3.2.2.3. Recent years saw bug reports by developers with an average of a mere 50 words (after pre-processing). This is very short and could well explain why the distinction between developers and end-users as a feature does not help the classifier. With bug reports that short, developers may not be submitting bug reports that are as clear and informative, and thus as actionable, at a rate as one would expect.

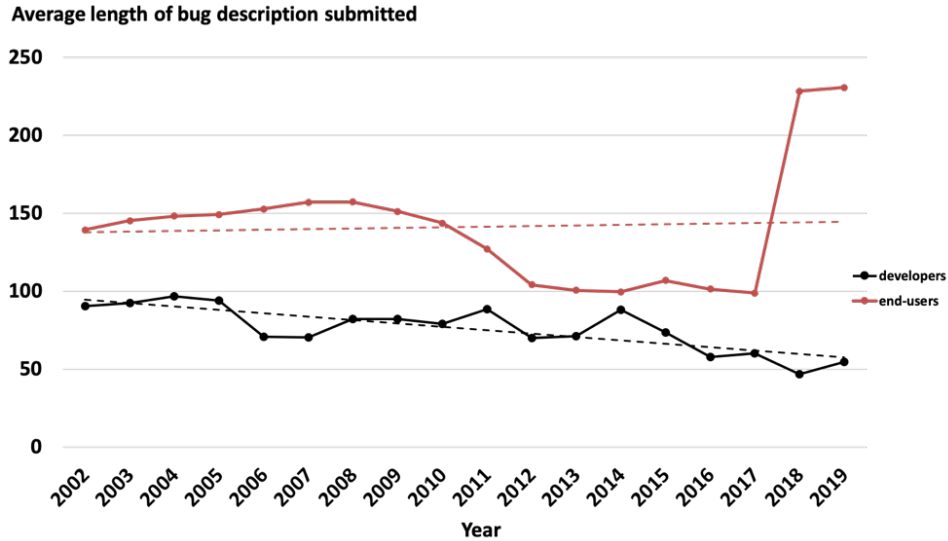


Figure 19. Average length of bug descriptions submitted by developers and end-users.

There is not a good explanation for why the average length of bug descriptions increased so dramatically in the past two years for end-users. I took an informal look at these bug reports to check for any potential outliers or anomalies in the dataset. However, no particular pattern could be found.

### 3.2.2.5 Experience

Finally, I studied the impact of the experience of reporters. One may expect that having more experience would lead to improvements in the bug descriptions being submitted over time, as past interactions may have taught the reporters the kinds of information that developers typically need. Experience was approximated with a feature in the model that counts the number of bug reports a reporter previously submitted (to the same Mozilla Firefox projects of the corpus, so to not only limit the search for experience but more importantly represent experience in interaction with the developers of these systems who have their particular expectations that may differ from those working on other systems). The revised model performed only marginally better than our worst model thus far (length of bug descriptions), with an accuracy of 82% and an F-measure of 0.79.

In the context of Figure 20, this is perhaps not a surprising result. The average experience is very low and over the years has become even lower, which in and of itself is surprising, since the percentage of bug reports being submitted by developers is increasing and one could have expected them to have a higher level of experience, especially over time. This likely implies that a great many reporters are one-time submitters, with this fact probably drowning out any positive effect that experience may otherwise have on the results.

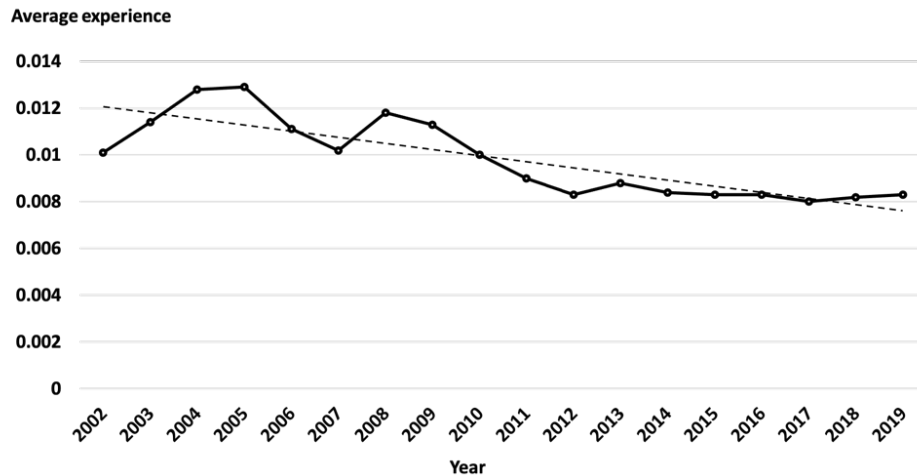


Figure 20. Average experience of bug reporters.

I also studied the impact of experience on the average number of back-and-forth exchanges between developers and reporters in both actionable and non-actionable bug reports. More specifically, I examined how much the experience of the bug reporter (whether the reporter is a developer or an end-user) affects the average number of back-and-forth exchanges in actionable and non-actionable bug reports. The results are presented in Table 6. To calculate the average number of back-and-forth exchanges, I used the same 180,000 bug reports that I extracted across all of Mozilla, from the years 2002 to 2019, 10,000 each year, and counted the number of times that developers raised the flag “Flags:needinfo?(reporter’s email)”, to either clarify a small issue or obtain some auxiliary information, and the number of times that the reporter resolved the flag and responded.

As Table 6 shows, interesting results emerged. First, actionable bug reports, regardless of being submitted by a developer or end-user and regardless of the submitter being

experienced or one-time, had the lowest average number of back-and-forth exchanges throughout the years. This is considered as another indication that the manual classification used in this study might reasonably separate actionable from non-actionable bug reports, as one would expect a lower number of back-and-forth exchanges for the former, and a higher number for the latter.

Table 6. Average number of back-and-forth exchanges over 18 years.

|   |      | <b>BUG REPORTS</b>         |          |                           |          |                            |          |                           |          |
|---|------|----------------------------|----------|---------------------------|----------|----------------------------|----------|---------------------------|----------|
|   |      | actionable                 |          |                           |          | non-actionable             |          |                           |          |
|   |      | submitted by<br>developers |          | submitted by<br>end-users |          | submitted by<br>developers |          | submitted by<br>end-users |          |
| <b>AVERAGE NUMBER OF BACK-AND-FORTH<br/>EXCHANGES IN YEAR</b> | year | experienced                | one-time | experienced               | one-time | experienced                | one-time | experienced               | one-time |
|   | 2002 | 0.0                        | 0.0      | 0.0067                    | 0.0015   | 0.0                        | 0.0056   | 0.0096                    | 0.0058   |
|   | 2003 | 0.0307                     | 0.0008   | 0.0045                    | 0.0028   | 0.0013                     | 0.0008   | 0.0091                    | 0.0016   |
|   | 2004 | 0.0                        | 0.0006   | 0.0009                    | 0.0044   | 0.0042                     | 0.0048   | 0.0072                    | 0.0064   |
|   | 2005 | 0.0                        | 0.0      | 0.0008                    | 0.0009   | 0.0011                     | 0.0058   | 0.0067                    | 0.0088   |
|   | 2006 | 0.0                        | 0.0      | 0.0053                    | 0.0010   | 0.0086                     | 0.0005   | 0.0084                    | 0.0095   |
|   | 2007 | 0.0                        | 0.0005   | 0.0011                    | 0.0010   | 0.0043                     | 0.0033   | 0.0041                    | 0.0097   |
|   | 2008 | 0.0                        | 0.0025   | 0.0074                    | 0.0080   | 0.0048                     | 0.0009   | 0.0072                    | 0.0060   |
|   | 2009 | 0.0                        | 0.0205   | 0.0355                    | 0.0148   | 0.0080                     | 0.0013   | 0.0232                    | 0.0075   |

|      |        |        |        |        |        |        |         |        |
|------|--------|--------|--------|--------|--------|--------|---------|--------|
| 2010 | 0.0    | 0.0230 | 0.0209 | 0.0491 | 0.0084 | 0.01   | 0.0323  | 0.0359 |
| 2011 | 0.0    | 0.0490 | 0.0947 | 0.0889 | 0.0380 | 0.0611 | 0.0916  | 0.0770 |
| 2012 | 0.0    | 0.0473 | 0.1781 | 0.4934 | 0.08   | 0.05   | 0.2932  | 0.4161 |
| 2013 | 0.0    | 0.1817 | 0.5503 | 0.9861 | 0.3723 | 0.6386 | 0.7892  | 0.9577 |
| 2014 | 0.0    | 0.2489 | 0.6487 | 1.0287 | 0.3590 | 0.5    | 0.7473  | 1.0902 |
| 2015 | 0.75   | 0.4299 | 0.7230 | 1.0039 | 0.3190 | 0.95   | 0.9732  | 1.2528 |
| 2016 | 0.1666 | 0.4221 | 0.4379 | 0.9161 | 0.4237 | 0.75   | 0.8365  | 1.0896 |
| 2017 | 0.0    | 0.4139 | 0.3574 | 0.7114 | 0.2979 | 0.5    | 0.49315 | 0.9627 |
| 2018 | 0.0    | 0.3859 | 0.3440 | 1.0197 | 0.3225 | 0.55   | 1.0909  | 0.9852 |
| 2019 | 0.4469 | 0.3730 | 0.4356 | 0.8244 | 0.4345 | 0.8181 | 1.0172  | 1.3333 |

Second, experienced developer who submitted actionable bug reports had the lowest average number of back-and-forth exchanges, when compared with the rest of the results. In most of the years (14 out of 18), the average was 0 and could go as high as 0.7 in 2015. The next group with the lowest average number of back-and-forth was one-time submitters who were developers, with the lowest of 0 and highest of 0.42 in 2015.

Third, the developers who submitted non-actionable bug reports had a smaller number of back-and-forth exchanges than end-users who submitted non-actionable bug reports. That is, regardless of their experience level, developers were asked a smaller number of questions about their non-actionable bug reports than end-users. It might be that when bug submitters were asked to provide additional information, developers could provide enough and better information than end-users and ultimately could avoid further back-and-forth exchanges.



### 3.2.2.6 Combinations

Apart from studying the addition of individual features, I exhaustively examined all their combinations. These results are also presented in Table 5. None of the combinations improved over our base model, with the combination of attachment and whether the reporter is an end-user or developer being the closest to the base model after a drop of 0.11 in accuracy (81% versus 92%) and 0.05 in F-measure (0.86 versus 0.91). Most other combinations fared poorly, with combinations of four features and the combination of all five features near the bottom of performance (including all five features led to an accuracy of 77% and an F-measure of 0.67). From this, one can only conclude that combinations of features end up detracting from the model performance. This is ultimately not too surprising, given that each of the individual features did not improve the base model either.

### 3.2.3 Cross-Project Prediction

Both because performing well on one dataset does not necessarily mean performing well on another, and because the performance of DT + tuning and RF + tuning was quite close to that of SVM + tuning, I studied portability of all four classifiers (with tuning) as trained on Mozilla data and applied to the dataset from the Cuezilla study.

As described in Section 3.1.2, I removed the bug reports that were labeled neutral in Cuezilla, so to create a correspondence between good (actionable) and bad (non-actionable). I also did not include the Mozilla dataset in the cross-project analysis, for obvious reasons (potential of overlap and conceptually being too close).

Table 7 shows the results, as applied to the Apache bug reports and Eclipse bug reports. Clearly, the performance degrades, though the results seem cautiously optimistic for three reasons. First SVM + tuning remained the top performing model, providing consistency. Second, I inspected the incorrect predictions (i.e., false positives and false negatives in the confusion matrix) and manually went through the bug reports that SVM + tuning wrongfully classified. No pattern of misclassification was found, meaning the model did not make predictions based on noise and is not overfitted. Third, precision is quite high. While recall

still is significantly lower than it should be for portability to be practical, I believe these results provide a good starting point for further exploration.

Table 7. Performance results of cross-project training (trained on Mozilla).

| MODEL        | TESTING |           |        | F-MEASURE |          |
|--------------|---------|-----------|--------|-----------|----------|
|              | ON      | PRECISION | RECALL |           | ACCURACY |
| SVM + tuning | Apache  | 0.95      | 0.51   | 0.75      | 0.66     |
| DT + tuning  |         | 0.76      | 0.46   | 0.65      | 0.57     |
| RF + tuning  |         | 0.53      | 0.58   | 0.63      | 0.55     |
| NB + tuning  |         | 0.81      | 0.38   | 0.78      | 0.51     |
| SVM + tuning | Eclipse | 0.80      | 0.67   | 0.77      | 0.73     |
| RF + tuning  |         | 0.60      | 0.75   | 0.73      | 0.66     |
| DT + tuning  |         | 0.60      | 0.73   | 0.67      | 0.63     |
| NB + tuning  |         | 0.81      | 0.38   | 0.62      | 0.51     |

To be exhaustive in the comparison, I performed two additional analyses using the Cuezilla dataset. First, I performed an analysis where, instead of ignoring all neutral bug reports, all neutral bug reports in the Cuezilla dataset were labeled as non-actionable (to mimic an ultra-conservative approach). Second, I performed an analysis where the Cuezilla tags were ignored and instead its bug reports were classified using our labeling guidelines into actionable and non-actionable bug reports (to assess portability more directly based on the understanding of actionable and non-actionable, rather than a heuristic mapping among two labeling systems). Table 8 shows the results of testing the best-performing SVM + tuning classifier on both new sets of labels. Note that in both cases and for both Apache and Eclipse, results in terms of F-measure improved, increasing the confidence that the results of Table

7 are robust, due to the model, and not accidental to the choice of mapping of Cuezilla's labels to the notions of actionable and non-actionable.

Table 8. Portability with alternative label mappings.

| MODEL                       | TESTING ON | MAPPING APPROACH                  | PRECISION | RECALL | ACCURACY | F-MEASURE |
|-----------------------------|------------|-----------------------------------|-----------|--------|----------|-----------|
| <b>SVM<br/>+<br/>tuning</b> | Apache     | Cuezilla labels, neutral ignored  | 0.95      | 0.51   | 0.75     | 0.66      |
|                             |            | Neutral labeled as non-actionable | 0.83      | 0.72   | 0.71     | 0.73      |
|                             |            | With new labels                   | 0.80      | 0.72   | 0.70     | 0.73      |
|                             | Eclipse    | Cuezilla labels, neutral ignored  | 0.80      | 0.67   | 0.77     | 0.73      |
|                             |            | Neutral labeled as non-actionable | 0.77      | 0.70   | 0.75     | 0.75      |
|                             |            | With new labels                   | 0.81      | 0.72   | 0.75     | 0.76      |

### 3.2.4 Survey Results

The results showed that it is possible to predict the overall quality of bug reports as actionable or non-actionable with a significantly higher level of accuracy and F-measure compared to the best results to date [14], [32]. However, other than using the bag of words model and hyper-parameter optimization, the primary cause of the increase in performance results could not be found.

To see if there is a potential trend that developers have observed collectively, the results from the second question of the survey was analyzed – whether quality of newly submitted bug reports has increased, stayed about the same, or decreased. As Table 9 shows, 56% of those surveyed felt that the quality of bug reports has remained about the same, with 26% feeling that the quality of bug reports has become somewhat better.

Table 9. Results of the survey.

| <b>Experience</b>            | <b>Quality has become significantly worse</b> | <b>Quality has become somewhat worse</b> | <b>Quality has remained about the same</b> | <b>Quality has become somewhat better</b> | <b>Quality has become significantly better</b> |
|------------------------------|---|--|--|---|--|
| <b>Less than 2 years</b>     | 2%  | 2%                                       | 8%   | 2%  | 0  |
| <b>Between 2 and 4 years</b> | 0%  | 2%                                       | 12%  | 6%  | 0  |
| <b>Between 4 and 6 years</b> | 0%  | 2%                                       | 14%  | 6%  | 0  |
| <b>More than 6 years</b>     | 2%  | 6%                                       | 22%  | 12%                                       | 2%   |
| <b>Total</b>                 | 4%  | 12%                                      | 56%  | 26%                                       | 2%   |

To determine whether these results align with a possible real trend in the quality of bug reports in terms of whether they are actionable or non-actionable over the years, the percentage of actionable bug reports from 2002 to 2019 was measured. Specifically, the SVM + tuning model was run on the same 180,000 bug reports that were used previously, 10,000 each year. As Figure 21 shows, the number of actionable bug reports increased noticeably as the years pass, from about 50% to about 80%, though the later years appear flatter. This could explain why the percentage of developers saying that the quality of bug reports has somewhat increased is greater the longer their tenure with Mozilla (see Table 9). Most of those who surveyed, however, felt that the quality of bug reports has remained about the same.

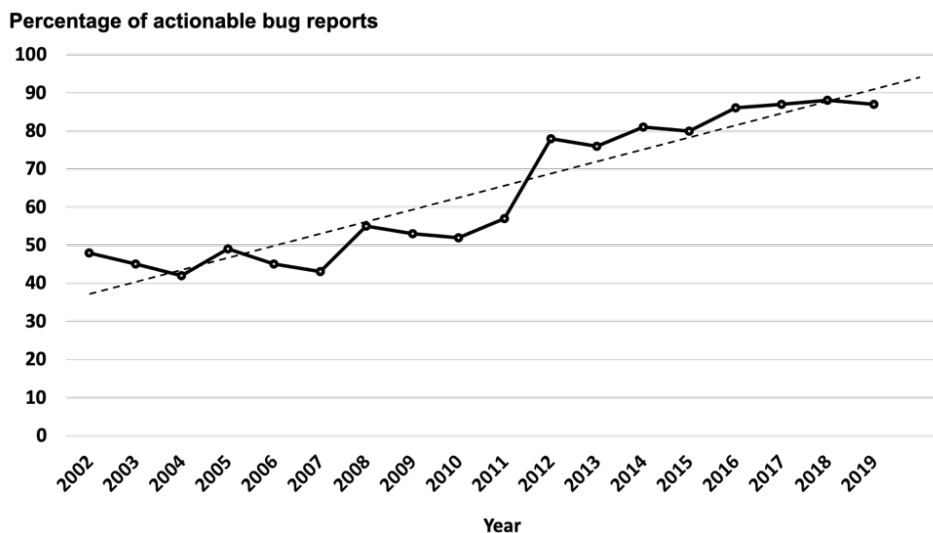


Figure 21. Percentage of Actionable Bug Reports over 18 Years.

Several of the answers to the open-ended question corroborate this observation, for instance: *“In recent years, looking across the many projects I am involved in, I am seeing a higher proportion of bugs being reported including at least the basic background information (software version, OS, detail of events leading to issue being reported) and a higher proportion being reported with enough information to allow reproduction.”* Not everybody agrees, however. Another respondent believed that, because more end-users are submitting bug reports, the number of non-actionable bug reports is actually increasing: *“I think the quality of bug reports has changed because more non-technical users and members of the general public are submitting more bug reports. They often ignore the instructions and do not fill out any of the sections except saying what happened and telling the developers to fix it.”* The results, however, contradict this perception. First, as Figure 21 shows, over the years the number of actionable bug reports was actually increasing. Second, the results suggested that the percentage of end-users who submit bug reports was actually decreasing (see Figure 17).

Whether end-users or developers submit bug reports matters to those who process them. One respondent stated: *“The origin makes a big difference. Mozillians are largely consistent in reporting bugs with clear repro steps (or a comment about why they aren't available). They sometimes attach relevant logs or config data (maybe 50% of the time). Community*

users, instead, usually mention what they were doing when the bug appeared (e.g., the web address, their actions with the input device, but not repro steps). They may mention what OS they are running but that is usually the extent of their initially supplied config data.” While it does not matter from a prediction performance point of view according to the results (Section 3.2.2.3), clearly to those processing bug reports, they prefer developers to report bugs because the resulting reports in their eyes seem more complete.

One respondent provided a humane counterpoint: *“Generally I look at performance bugs. Often the quality of a bug reported by someone new to reporting perf bugs is just expectedly low, because they are not familiar with any of the tools, we use to get more information about performance problems. This is honestly fine. If we were asking them to commit to, for example, profiling the application while it is being slow, prior to even submitting a bug report, we might discourage them from submitting. However, if they submit the bug with vague information and then a friendly real human asks them for a little information, it feels like they are having an impact and the exchange is more personal, and they will be more likely to provide the necessary information”*. Note that they mention “a little information”, implying that the initial report should at least be actionable.

Interestingly, the increase in actionable bug reports corresponds nicely with the increase in percentage of bug reporters who are developers (see Figure 17). One observation especially stood out in this regard: *“Many reports of bugs in the browser (as opposed to feature requests, task-planning bugs, etc) are generated by Continuous Integration (e.g., intermittent tests, tests run on rare architectures). Nearly all others are reported by Mozillians, and a handful are from the community”*. This comment not only affirms that many bug reports are submitted by developers (“Mozillians”), but also points out that some percentage of the bug reports being submitted by ‘developers’ is actually being submitted by bots. What percentage that is, and what impact this has on the predictability of actionable versus non-actionable bug reports is a subject of future study.

Several respondents mentioned that Mozilla has already undertaken steps in an attempt to improve the quality of bug reports. In particular, the Bugzilla Helper<sup>30</sup> offers a template that

---

<sup>30</sup> <https://www-archive.mozilla.org/events/dev-day2001/community-testing/bugzilla-helper>

specifically asks users about steps to reproduce, actual results, and expected results. While bug reporters are not required to fill out these fields, their mere presence is intended to guide them to write a more effective bug description. This form was not always met with a uniformly positive reaction. One developer said: *“A major problem in bug reports from non-expert contributors is always getting details about what exactly they did and what they expected to happen that didn't happen. This has become less of a problem because we added a form for new Bugzilla accounts to fill out that specifically addresses those questions. Getting sufficient detail to diagnose the problem is still an issue, though”*. Another complained: *“A simple report form leads to simple reports, lacking necessary detail”*.

To see if bug reporters are actually filling out the fields suggested by Bugzilla Helper, an analysis was performed to calculate the percentage of bug reports that had a combination of “steps to reproduce:”, “Actual results:”, and “Expected results:” strings in their descriptions. The search strings were case sensitive and specifically included colon symbols to replicate the way Bugzilla Helper lists the fields. Figure 22 presents the result. Clearly, the template has had impact, though the fact that, today, only about 25% of newly submitted bug reports use it is disappointing. On the other hand, one respondent said: *“We *might* have started pasting in a markdown template for new bugs about 2 or 3 years ago -- I'm not sure -- but many new bugs clear out the template. That's what I do when I submit a bug, is to clear out the template”*. They continue on: *“Once a user has internalized what makes a useful bug report, and the template reinforces that, that user seems to create useful reports whether or not the template is retained”*. This may in fact be true. While the template use is low, Figure 21 indicates that the number of actionable bugs has gone up, nonetheless. This may be from the learning effect from the template.

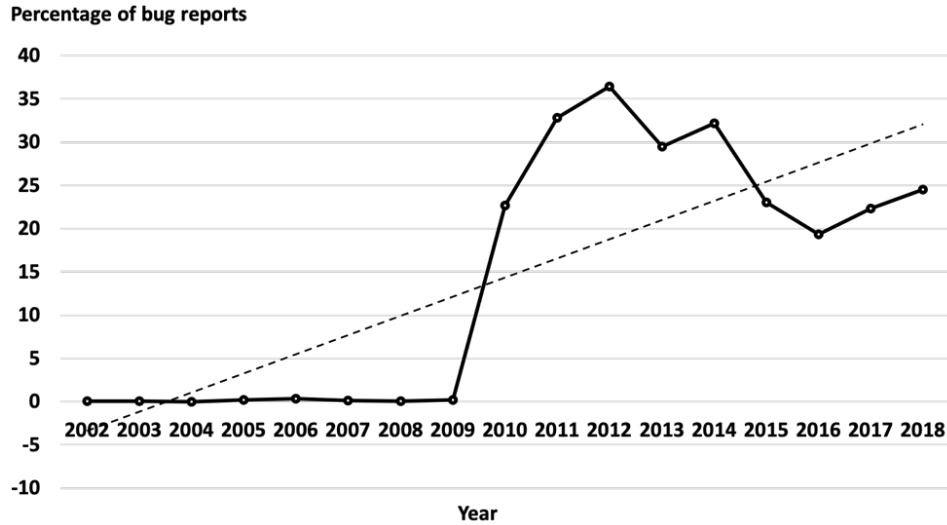


Figure 22. Percentage of Bug Reports that Contained “Steps to Reproduce:”, “Actual Results:”, and “Expected Results:”.

A final interesting observation comes from a developer on the accessibility team: *“I work in a rather exotic department, in accessibility. Our bug reports either come from us, the team members, or web accessibility experts. Those have usually done some research, provide code samples to reproduce the bug, etc. And that hasn't changed since the early days.”* The respondent continues with: *“We don't often get reports from end users. They usually contact us through other means (social media, e-mail, etc.), and we convert them into actionable bug reports”*. Note that the word actionable was used. This might explain the negligible effects of developers submitting bug reports on the performance results (Section 3.2.2.3). Also, it would be interesting to study these kinds of translations from social media and e-mail in more detail; what, if any, interactions with the users take place and what is the resulting quality of the bug reports?

### 3.3 Discussion

This chapter focuses on overall bug report quality prediction. Among previous works, Cuezilla [35] and the approach of Schuegerl et al. [32] provide the best results in predicting bug report quality, with the former achieving an accuracy of 50% in classifying bug reports as good, neutral, or bad and the latter achieving an accuracy of 44% in classifying bug reports



on a scale of 1 (very high quality) to 5 (very low quality). This work improves the results by achieving a significantly higher level of accuracy so to be able to use the classifier as part of a future tool that assists reporters in improving the quality of bug reports that are deemed non-actionable.

Further exploration of the results showed that, similar to Menzies et al. [89], hyperparameter optimization had an important role in the improvements of the results. This was further confirmed when the best model (SVM + tuning with an accuracy of 92%) was run on Cuezilla's training data (i.e., only Mozilla bug reports) and achieved an accuracy of 75%, which was 25% higher than before, with an F-measure of 0.64. This reiterates the importance for optimization and how the community should pay more attention.

The improvement in the results are also explained by the fact that prediction with fewer outcomes tends to nearly always perform better than prediction with more outcomes. This work focuses on two outcomes, actionable versus non-actionable, which requires a binary classification rather than one that is more fine-grained as used in prior work. The envisioned tool merely needs a simpler, binary classification, rather than one that offers a more fine-grained scale offered by prior work.

This work further employed many experiments in an attempt to improve the results of the best model (SVM + tuning). The analyses were both based on previous research to account for the factors observed to be essential to the quality of bug reports (see Section 2.2.1) and intuitively based on the experience in manually labeling bug reports (see Chapter 5). For instance, experience of bug reporters was explored as a potentially important factor when it comes to bug report being actionable or non-actionable, since inexperienced reporters often submitting ambiguous or difficult to understand bug reports [5], [6], [13]. However, the results indicated that adding input features to the bag of words actually decreased the performance of the best model (SVM + tuning). These decrements in results were small enough to prevent drawing conclusions out of these experiments, though they indicate that some of these input features might not be as important as they are mentioned in previous research. For example, the model which included the length of bug reports as another feature had the second lowest F-measure of 0.84, even though length of description was considered very important to the quality of bug reports [138].

These input features were additionally analyzed to see how they have evolved over time. By placing the performance of the augmented classifiers in the context of these features' historical trends, the reason why such results were achieved could be better understood. The results showed that from 2002 to 2019, more reporters are including attachments, the readability of bug reports has not really changed, more developers are reporting bugs rather than end-users, the average length of bug descriptions has not changed, and the experience of bug reporters has slightly reduced. The increase in number of developers reporting bugs rather than end-users can be because Mozilla is a mature product and more bug reports are reported internally. Based on one of the responses from the survey, there are some departments in Mozilla that do not often get reports from end-users: *"I work in a rather exotic department, in accessibility. Our bug reports either come from us, the team members, or web accessibility experts."*

An interesting final observation was the increase in the percentage of actionable bug reports over the years. This might be because of the learning effect from the use of Bugzilla helper, as shown in Section 3.2.4. Additional investigations are needed to understand the effect of the template on the quality of bug reports.

This study differs from the study by Zanetti et al. [84], which predicts whether bug reports are valid or invalid. In their approach, they automatically mapped bug reports with the resolution status of fixed and wontfix to valid, and duplicate, incomplete, and invalid to invalid. Their approach, however, introduces limitations. First, it is unclear as to whether the bug itself is invalid or the bug report is invalid. It might be that the bug report was high quality and helped developers discover that the bug itself is invalid. For example, in some projects such as Mozilla, a resolution status of invalid can mean that the problem that the report is explaining is actually a feature request rather than a real bug. Second, bug reports with the resolution status of fixed might not necessarily be high quality. For example, a bug report might have a low quality when submitted, but after much back-and-forth with developers and providing additional information it can end up being resolved as fixed.

Taken together, the results are sufficiently accurate that they can serve as a reliable basis for developing a tool that assists reporters in improving bug reports that are deemed non-actionable. Future investigations include exploring such tools and studying additional

directions related to quality prediction. First and foremost is the question of portability: is it possible to develop models that more reliably transfer to bug reports for unseen software systems? This is an important desirable property and, despite our results being stronger than any to date in terms of portability, the results have room for improvement. Second is the question of whether the results can be further improved. A particularly interesting direction in this regard is the recent trend to include videos in bug reports: whether video-based bug reports are more actionable and, if so, whether this can be leveraged in the model. Finally, it is necessary to dive deeper into the properties of bug reports that the machine learning classifiers pick up on. Identifying these features could lead to new insights as to what makes bug reports actionable.

### **3.4 Threats to Validity**

While this study was structured to avoid introducing bias and worked to eliminate the effects of random noise, it remains possible that the mitigation strategies may not have been effective. This section reviews the threats to validity to the study.

A threat to construct validity exists with respect to the concept of actionable versus non-actionable bug reports. The assessment of what is actionable in this study may differ from what a professional developer might determine it to be. In response, note that, first, the two researchers involved in the manual labeling each have multiple years of experience as professional developers. Second, the labeling is based on what the Mozilla developers actually—and—publicly did when they processed bug reports as part of their work, as opposed to merely their perceptions that one might obtain if one were to ask developers to label sample bug reports as actionable or non-actionable. Because this concept is rooted in real actions displayed, I feel that this risk is significantly reduced.

The dataset used for the study contained bug reports from a set of projects that are all related to Mozilla Firefox. As a result, the findings may not be generalizable to all OSS projects, or commercial projects, which represents a threat to external validity. Sampling bug reports from a number of Firefox projects, rather than a single one, however, means that the dataset includes software in multiple languages and across multiple operating systems. Together

with the large number of bug reports included in the corpus, I believe this ameliorates this threat to a degree. The portability of the classifier to other projects, with results that improve over the original study of Cuezilla, provides further evidence that this may be less of a problem. That said, further study is needed to assess whether similarly accurate results can be obtained for a broader range of projects.

A primary threat to the internal validity of this study is the possibility of faults in the Python code that was implemented to perform the study. This threat was addressed by extensively testing the implementation and verifying its results against a smaller dataset for which the correct results were manually determined.

The training and testing data were labeled manually, which represents a threat to internal validity as it could have introduced bias or mistakes due to a lack of domain expertise. To address this concern, two researchers used the negotiated agreement method to label a significant portion of the data. Because of the high inter-rater reliability that resulted, I believe that this risk is reduced considerably. Further, the stark difference in average number of messages posted for actionable bug reports compared to non-actionable bug reports provides further confidence in the labeling.

The choice to only include resolved bug reports in the dataset may represent a threat to internal validity if the nature of the content of these bug reports differs significantly from those of resolved bug reports. Given the discussion in Section 3.1.1, I believe this to be a necessary tradeoff for labeling (and thus training) accuracy.

In the analysis of the impact of reporters' experience, experience was verified by treating each unique e-mail address as one person (e.g., jack@mozilla.org and jack@gmail.com are counted as two persons). A common step is to unify these two addresses based on the assumption that they represent a single person, Jack. this study did not do so, since the analysis showed relatively few common e-mail names across multiple domains for the reporters in the dataset and, in those cases, the domains were sufficiently different and specialized that it was difficult to speculate on whether they represented the same person. Hence, the possibility remains that experience was under-assessed in a few cases. Across the entire dataset, I believe the impact is relatively minimal, however.

The SVM + tuning model, that was trained on only Mozilla Firefox bug reports, was run on 180,000 bug reports from across all Mozilla projects. To make sure that the results of the classifier can be trusted, the SVM + tuning model was checked to see if it is "internally portable". That is if a model that is trained on some parts of a project still works with high accuracy if tested on the other parts. To do that, 50 bug reports were randomly selected from 180,000 bug reports from across all Mozilla projects, excluding Firefox, manually classified them to actionable or non-actionable, and compared the results with the labels that SVM + tuning model had already assigned. The results suggested that the model is "internally portable" and can predict the overall quality of the bug reports in terms of whether or not they are actionable with a F-measure of 0.96. I assume the risk of our classification model not being internally portable is minimal.

Finally, since all survey respondents were related to Mozilla, the survey responses may not be representative across OSS and commercial projects as the characteristics of developers in other OSS projects may be different from those included in this study. This risk can be reduced by repeating the survey across other constituencies of developers.

### **3.5 Conclusion**

This chapter revisited an older problem, that of bug report quality prediction. Particularly the study manually labeled the bug reports as actionable or non-actionable, since the concept of actionable more accurately reflects what happens to bug reports: they are either sufficient and clear to know what to do with them, or they are not. The results significantly improved over the state-of-the-art and that, through detailed analyses of whether auxiliary features can further improve the results, challenge long-held beliefs on the value of incorporating those features in the prediction model. The primary results are:

- A significant improvement over the state of the art in bug quality prediction to a precision of 94%, recall of 89%, and F-measure of 0.91. The primary cause of this improved performance appears to be a combination of needing a binary classifier only, using a bag-of-words approach rather than higher-level features, hyperparameter optimization, and a large training dataset.

- Portability of the resulting model that has better predictive capabilities than the best models to date that were specifically trained on those other systems.
- An absence of improvement in performance with the inclusion of auxiliary features (alone or together) that, to date, had improved classification accuracy. With a sufficiently strong base model, the impact of including these features seems to disappear.
- A survey among Mozilla developers to get an understanding as to how they feel the quality of bug reports has or has not changed over the years, with the results showing that they believe the quality stayed the same with a small percentage saying it has become better.

## 4 CHAPTER 4: An Analysis of Video Submissions in Bug Reporting

Effective bug reporting and resolution is essential to any software project, since a significant portion of the overall software development life cycle is spent addressing bugs [75]. Being efficient in this process is crucial, especially when projects mature and large numbers of bug reports are submitted on a regular basis. The resulting volume creates challenges in processing them all quickly and with the right outcomes [11], which in turn leads to wasted effort by developers, delays overall progress, and can have a detrimental effect on the overall quality of the software [35].

In practice, bug reports often lack information [8], [9], containing only a stack trace, overly detailed logs but little else, or just a brief, non-structured, natural language description of the bug [139]. Previous studies show that the information that is most useful for a developer in resolving a bug report is often the most difficult information for reporters to provide [7].

Within this context, an interesting phenomenon has emerged: that of the inclusion of videos in bug reports [111]. Videos can visually communicate bugs and what actions led up to them, offering developers a new opportunity to obtain and inspect context-rich bug information [36]. Videos help developers understand how users interact with the system, process the current behavior of the system, and comprehend any events that may have contributed to the manifestation of the bug [35], [140]–[142]. Several studies conclude that reporters should be encouraged to submit relevant videos as part of their bug reports to convey additional context for understanding bugs [39]–[41].

An important question that has not been answered to date is whether the presence of videos in bug reports may have a further effect on the overall bug resolution process? That is, does the inclusion of videos in bug reports also lead to benefits for those who report bugs? If, for instance, bug reports that include video are resolved more quickly, this may provide tangible incentive for reporters to submit videos as part of their bug reports. The extra work they would incur in producing a video would then not only benefit the developers processing the bug report, but also provide a tangible benefit to the reporter themselves in getting their bug report resolved faster.

In this chapter, the results from the study of 2,814,599 bug reports from five systems (Mozilla, Android, LibreOffice, IntelliJ, and Minecraft) are presented. The study assesses the impact of the presence of videos in bug reports on the overall resolution process. The study is quantitative, comparing three outcomes for bug reports that include videos with bug reports that do not, answering the following questions:

**RQ1:** Does including videos in bug reports lead to a reduction in the average time to resolution? If so, this would be a potential benefit for bug reporters who include video, since their chances of having closure on their bug report sooner increase compared to those who do not include video.

**RQ2:** Does including videos in bug reports lead to a higher percentage of bug reports being resolved with a patch, that is, actually fixed? If so, this would be a potential benefit for bug reporters who include video, as their chances of having their issue actually fixed increase compared to those who do not include video.

**RQ3:** Does including videos in bug reports lead to a reduction in the average back-and-forth that occurs once the report has been submitted? If so, this would be a potential benefit for bug reporters who include video, as they are more likely to have to answer fewer requests by triagers and developers, which means fewer interruptions of their regular work and less mental effort to re-engage with the bug.

Beyond answering these questions, the study performs a deep dive to understand whether other factors may have an effect on overall bug resolution process. The deep dive particularly examines whether it matters if videos are submitted by developers on the project or by end-users, whether the type of bug being reported with a video may have an impact, and whether the effect of including videos differs depending on the assigned severity of the bug report. This part of the study only focuses on Mozilla, with the exception of analysis of severity, for two reasons. First, only Mozilla publicly provides adequate user statistics (e.g., number of patches submitted) to determine if the video submitter is a developer or end-user. Second, in Chapter 5, Mozilla bug reports were already studied and provided necessary information to determine the type of the bugs. To classify the type, manually labeling bug reports was



selected rather than applying an automatic approach [143], as the automatic approach wrongfully classified many bug reports.

In the remainder of this chapter, I detail the methodology and report the results. Then I place the results in the broader context of the literature to date.

#### 4.1. Methodology

The goal is to understand whether the presence of videos in bug reports has a potential impact on factors that may be of concern to those submitting bug reports: time to resolution, bug reports being successfully resolved with a patch (i.e., bugs being fixed), and the amount of back-and-forth discussion. To do so, a multi-year selection of bug reports was downloaded for five different systems, extracted relevant metadata, determined for each bug report whether it had one or more videos attached as well as if any videos appeared in the subsequent back-and-forth, and performed a range of statistical analyses. I detail the methodology below. Figure 23 illustrates the overall process that was used for the analysis.

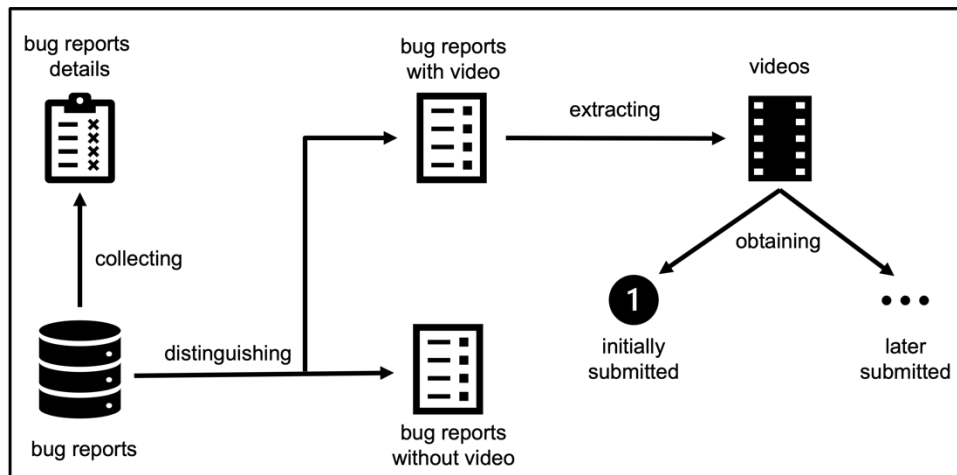


Figure 23. Overall process of studying the role of videos in bug reporting.

### 4.1.1 Data Collection

I collected 2,814,599 bug reports across five different systems: Mozilla, Android, LibreOffice, IntelliJ, and Minecraft, from 2010 to 2021. These projects were selected because they represent different domains, including a web browser, an operating system, an office suite, a development environment, and a game, providing necessary breadth for the findings to be at least somewhat generalizable. Moreover, they use different bug tracking systems, including Bugzilla, YouTrack, and Google Issue Tracker. All available metadata was downloaded either through the API associated with the issue tracker (e.g., the BugZilla REST API<sup>3</sup>) or by creating a custom web scraping tool. The metadata was used to detect the presence of video attachments.

Table 10 displays the details of the experimental dataset. As shown in the table, the dataset spans from 2010 to 2021. The total number of downloaded issues was 2,814,599. Out of those, 2,039,221 were closed. Bug reports with a resolution status of NEW, UNCONFIRMED, REOPENED, and ASSIGNED were considered as open, and bug reports with a resolution status of RESOLVED, VERIFIED, and CLOSED as closed (while certain individual bug reports may constitute an exception to this interpretation of open and closed, the number tends to be small and the convention set by other studies was followed, e.g., [144], [145]).

Table 10. Experimental Dataset.

| Ecosystem | Domain                                       | Bug tracking system | Number of bug reports downloaded | Number of closed bug reports <sup>31</sup> | Number of closed bug reports with videos | Date of first bug report |
|-----------|--|---------------------|----------------------------------|--|--|--------------------------|
| Mozilla   | web browser with many significant extensions | Bugzilla            | 893,073                          | 794,383                                    | 10,594                                   | 2010-01-01               |

<sup>31</sup> (RESOLVED, VERIFIED, CLOSED)

|                  |                         |                           |           |         |        |            |
|------------------|-------------------------|---------------------------|-----------|---------|--------|------------|
| Android platform | operating system        | Google code issue tracker | 308,273   | 174,782 | 3,706  | 2010-01-01 |
| LibreOffice      | office suit             | Bugzilla, Redmine         | 172,088   | 81,050  | 1,488  | 2010-09-30 |
| Minecraft        | game                    | Mojang studios (Jira)     | 402,990   | 388,689 | 24,011 | 2012-09-06 |
| IntelliJ         | development environment | YouTrack                  | 1,038,175 | 600,317 | 9,862  | 2010-01-01 |

#### 4.1.2 Identifying Videos

A custom web scraping tool was created using Python that visited the webpages of all bug reports from our dataset to identify the bug reports that included video attachments. The crawler, in particular, searched for attachments with the following extensions most commonly used for videos files: 'webm', 'mpg', 'mp2', 'mpeg', 'mpe', 'mpv', 'ogg', 'mp4', 'm4p', 'm4v', 'avi', 'wmv', 'mov', 'qt', 'flv', 'swf', 'avchd'. In total, 49,661 bug reports were identified that contained a video attachment, distributed across the five systems per the second to the last column of Table 10.

#### 4.1.3 Collecting Bug Report Details

Table 11 shows an example of some of the metadata associated with one of the bug reports involved in our study (note that its history of status transitions is not shown).

Table 11. Excerpt of some relevant bug report data.

|                           |                       |
|---------------------------|-----------------------|
| Bug report ID             | 1329633 <sup>32</sup> |
| Creation year             | 2017                  |
| Number of days to resolve | 4                     |
| Priority                  | Not set               |
| Severity                  | normal                |
| Resolution status         | RESOLVED INVALID      |
| Reporter ID               | 258347                |
| Number of comments        | 1                     |

It is important to note that a bug report being resolved does not always mean that a fix was produced. Resolution also can concern bug reports that are not fixed (as in the case of the example) or that are deemed to work for the development team. As such, the time to bug fix, which is frequently studied from the perspective of prediction as to how long it may take (e.g., [146]–[148]) is different from the time to resolution, since the latter applies to a broader category of bug reports. In this study, bug resolution time was used.

Also note that bug reports with a status of VERIFIED or CLOSED in almost all cases have a preceding status of RESOLVED and, in cases when not, still should have been tagged prior as RESOLVED before applying VERIFIED and/or CLOSED. For this reason, they were included in the set of bug reports being studied.

Across the five systems that were studied, standard priority levels used in the bug tracker were different:

- Android: P0, P1, P2, P3, P4.

---

<sup>32</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1329633](https://bugzilla.mozilla.org/show_bug.cgi?id=1329633)

- IntelliJ: showstopper, critical, blocker, major, high, medium, minor, nice to have, unclassified.
- Minecraft: blocker, critical, important, normal, low, blank.
- LibreOffice: highest, high, medium, low, lowest.
- Mozilla: not set, P1, P2, P3, P4, P5.

The Mozilla levels were used as the base upon which to map the others. All but one (IntelliJ) had five levels and were therefore trivial to map (if a priority had not been assigned yet, it was mapped to not set). For IntelliJ, showstopper and critical was mapped to P1, blocker and major to P2, high to P3, medium and minor to P4, nice to have to P5, and unclassified to not set. Then the bug reports were grouped into two priority groups: high priority (P1, P2, P3) and low priority (P4, P5, not set).

Apart from the details related to bug reports shown in Table 11, the following information was collected:

- Number of back-and-forth. This represents the number of times that the reporter was requested to answer plus the number of times that the reporter answered. For example, for Mozilla, the number of times that the flag “Flags: needinfo?” was raised in a bug report were counted plus the number of times it was lowered when the respondent provided the info. By raising and lowering the needinfo flag, developers and bug reporters signal their messages to one another, which gives us a sense of their direct, intended interactions, for instance to clarify an issue or obtain some auxiliary information. Note that the number of back-and-forth is different than the number of comments. The former represents direct communication, the latter all communication for that bug report.
- Video submitted initially or later. To determine at what stage of bug resolution a video was submitted, the metadata about whether the video was attached to the initial bug report or attached to a comment later during the bug report resolution process were collected. The latter is sometimes at the prompting of the developer, other times the reporter decides to include a video as part of the exchange.

- Presence of steps to reproduce, actual results, and expected results. Similar to Davies et al. [55], pattern matching was used to determine whether the search strings “steps to reproduce”, “actual results”, and/or “expected results” were present in the bug report. The search strings were case insensitive.

#### 4.1.4 Statistical Analysis

In order to answer the three research questions identified in the introduction, after checking for the normality assumption of the data, the Welch Two Sample t-test was conducted between the bug reports with and without video with respect to bug resolution time, percentage of bug reports being successfully resolved with a patch (i.e., as FIXED), and the back-and-forth discussion following a bug report submission. Since multiple tests were performed, the significance value was adjusted accordingly to account for multiple hypothesis correction. I used the Benjamini–Hochberg corrections [149], which resulted in an adjusted p-value of 0.14.

Three Generalized Linear Regression (GLM) [150] models were also built. The dependent variables (i.e., time to resolution, bug reports resolved as FIXED, and number of back-and-forth) follow a Poisson distribution. Therefore, a Poisson regression model with a log linking function was used. In order to build each model, factors that the existing literature has already identified as correlating with bug fix time were used, including operating system [151], product, component, priority, severity, status, platform, resolution, version, cc count, votes [152], whether it is confirmed (only in LibreOffice), confirmation status (Confirmed, Null, Plausible, and Unconfirmed; only in Minecraft), labels, number of duplicates, number of comments, number of unique participants [153], number of attachments, number of developers [138], number of unique words in title and bug description [154], and readability score (Flesch score [130]) [25]. Note, once more, that this study is about bug resolution time, not bug fix time, as studied by this existing work. A corollary to this work, then, is the determination of whether these factors correlate with bug resolution time beyond bug fix time.

To this set, the presence of actual results, presence of expected results, presence of steps to reproduce, number of videos in each bug report, number of back-and-forth, and video submitted initially or later were added. Finally, each bug reporter's reputation, a common feature which has been studied in previous work in mining bug reports, was added. The intuition is that poor issue reports may take longer to resolve and reporters who frequently write such poor reports will accumulate a poor reputation, and vice versa. I used the reporter reputation formula of Hooimeijer [25].

After collecting these metrics, multi-collinearity was checked using the Variance Inflation Factor (VIF) of each predictor in the model [155]. VIF describes the level of multi-collinearity (correlation between predictors). A VIF score between 1 and 5 indicates a moderate correlation with other factors, so the predictors with a VIF score threshold of 5 were selected. This step was necessary since the presence of highly correlated factors forces the estimated regression coefficient of one variable to depend on other predictor variables that are included in the model. Next, using the selected factors, the best model was identified using Akaike Information Criterion (AIC), which estimates the information loss between models in comparison to the original [156]. It ultimately selects the best model based on both the fit of the model and the information lost. Then the model, identified by AIC as the final model, was selected. This process was repeated independently for each of the three regression models.

#### **4.1.5 Mozilla-Specific Data Collection**

For one system, Mozilla, several additional analyses were performed that sought to investigate whether other factors might play a role in the results that were present across all five systems. For these analyses, additional data were used (see Figure 24).

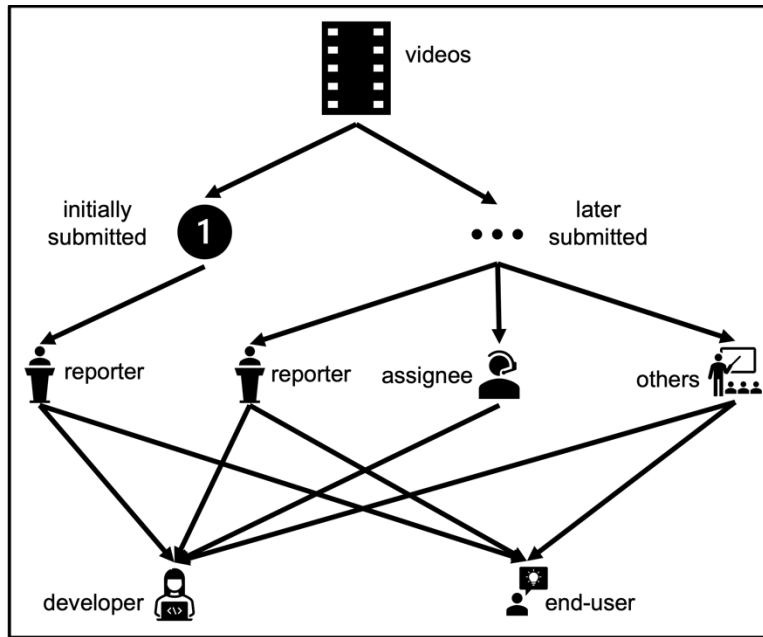


Figure 24. Additional steps to study the role of videos in Mozilla bug reports.

First, two additional pieces of information were collected: who submitted the video and what role they had. Mozilla bug reports were selected for this part of the study as they provided adequate user information to find out if the video submitter is a developer or end-user. To do so, the user-ID of each video submitter was cross-checked with the user-ID of bug reporters and bug assignees across all bug reports. In case there was no match, the video submitter was labeled as “others”. Also, to determine the role, the number of patches that the video submitter previously submitted to the Mozilla projects was extracted. If the number was zero, the video submitter was identified as an end-user, otherwise as a developer.

Second, the results from the labeling of 1,045 videos (see Chapter 5) were used to determine the type of bug being illustrated. These 1,045 videos were randomly selected from all 10,594 Mozilla videos and labeled by two researchers (achieving a “perfect agreement” [127] with an IRR of 0.87; see Section 5.1.2). For the type of bug, the following categories emerged: (1) UI — problems related to the user interface, (2) system settings — problems related to the configuration settings, (3) access — problems with functionality for logging in or signing up, (4) failure to load — problems related to the software not being able to start, and (5) functionality — problems related to faulty results being produced by the software.



Third, to understand if the video is helpful for critical bugs, the severity levels were grouped based on the Least Significant Difference (LSD) test [157], which determines if the means of the groups are statistically different. Then, high severity Mozilla bug reports with and without video were studied. This analysis was extended to study Android and LibreOffice bug reports as well, but not for IntelliJ and Minecraft, as the severity field in their bug reports were not visible to readers.

## **4.2 Results**

In this section, the results are presented. To set the stage, first, the prevalence of including videos as part of bug reports is briefly reviewed for all five systems of interest. Then the three research questions are answered. Finally, a deep dive into Mozilla was performed to examine several other factors that may be at play. Because of the possibility of sensitivity to the period of time studied, all the analyses were repeated with shorter time windows (last nine, six, and three years).

### **4.2.1 Prevalence**

First, the 2,039,221 closed bug reports were examined to determine how many of them contained videos. Then, their trend was plotted over the years. The result is shown in Figure 25. While the total number of bug reports with videos, 49,661, is low at around 2.43 percent, a clear upward trend exists, with the year 2021 seeing almost seven percent of, for instance, Minecraft bug reports involving videos (Mann-Kendall trend test,  $p\text{-value} < 0.01$ ). This confirms the informal observations made in prior studies that videos appear to be becoming more prevalent in and important to the bug reporting process (e.g., [42], [46]).

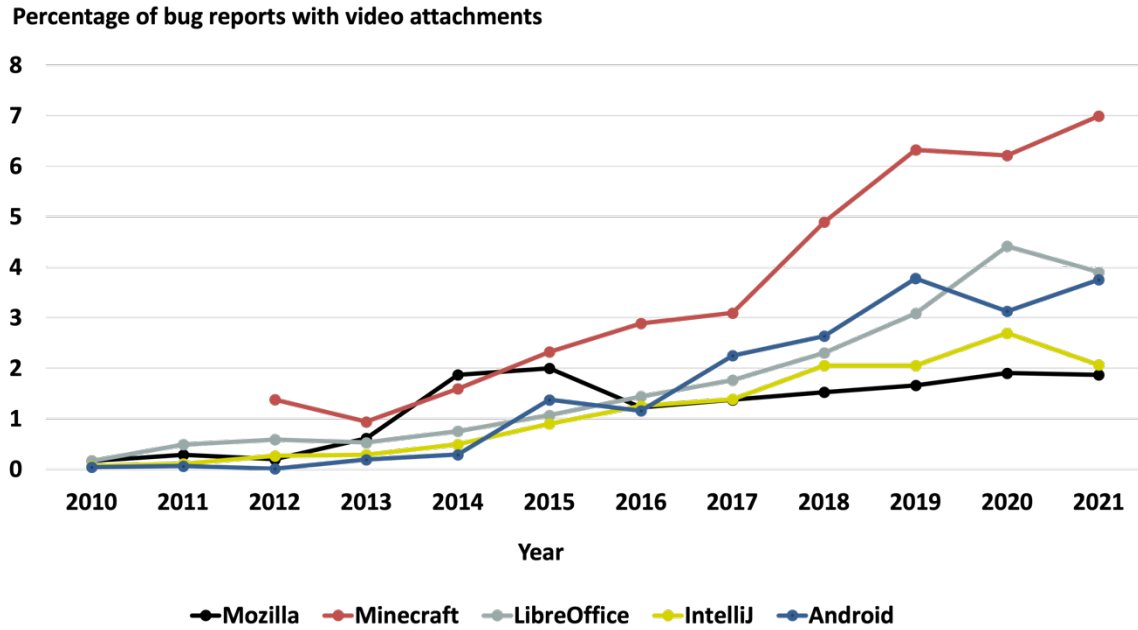


Figure 25. Percentage of bug reports with video attachments.

**Observation 1. Across all projects, an upward trend is present in bug reports including videos.**

Each video was labeled as to whether it was submitted initially as part of the original bug report being filed, or later (irrespective of whether by the reporter, the assigned developer, or someone else who chimed in). Figure 26 shows the difference between the percentage of bug reports with video attachments submitted initially and the percentage submitted later, on a year by year basis. The majority of videos were submitted initially at the time the bug report was filed in four of the projects, with Mozilla moving strongly from the early years in which more videos were submitted later to the later years in which a majority were submitted with the initial bug report. LibreOffice is very different, however. It consistently sees more videos submitted at a later time as compared to at the initial submission of the bug report.

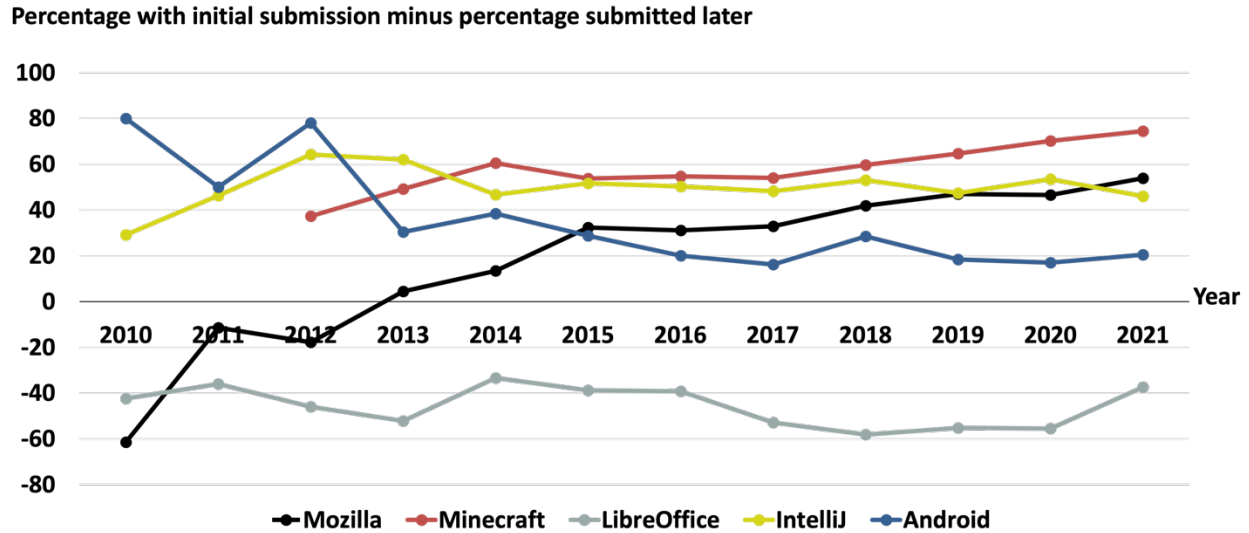


Figure 26. Difference between percentage of bug reports that were submitted initially and percentage submitted later.

The means were compared for both groups across all five projects and the difference was statistically significant, even with the presence of LibreOffice (Welch Two Sample t-test, 11.241,  $df = 21.998$ ,  $p\text{-value} < 0.14$ ). This confirms that the percentage of videos initially submitted is higher than the percentage of videos submitted later, but individual differences must be respected as it is clear, in this case, that the pattern is the opposite for LibreOffice (Welch Two Sample t-test,  $t = -2.11$ ,  $df = 13.02$ ,  $p\text{-value} < 0.14$ ).

In Mozilla, the balance of videos that were submitted along with the initial bug report versus as part of the follow-on back-and-forth has shifted drastically over the years, from 17% initially and 83% later to 79% initially and 21% later. That could be because of several reasons, including a growing belief among reporters of the potential usefulness of including video, the ubiquitous availability of cell phones with cameras, or even the broader societal phenomenon of short videos being an accepted part of life.

**Observation2. Across all projects, more videos are submitted with the initial bug report than during the follow-on back-and-forth. Individual differences exist, however, and LibreOffice is an exception in exhibiting the opposite pattern.**

## 4.2.2 Potential Benefits to Reporters

This section focuses on the main objective of this chapter: to understand the impact of including videos on the resolution process in terms of time to resolution (RQ1), percentage being resolved with a patch and thus fixed (RQ2), and amount of back-and-forth (RQ3). For each research question, a statistical analysis was performed using Welch Two Sample t-test and a regression analysis as explained in Section 4.1.4. The Welch Two Sample t-test was performed on 49,661 bug reports that included video and 1,989,560 bug reports that did not include video. For regression analysis, the dataset included 49,661 bug reports that included video and 99,322 bug reports that did not include video; the latter being a smaller subset than all bug reports without video that was selected: (a) to be twice the number of bug reports with videos, and (b) to match the distribution of resolution outcomes and priorities of the set with videos.

### 4.2.2.1 RQ1: Time to Resolution

The impact of including a video (or multiple videos), regardless of whether it was included initially or later, on the time to resolution was assessed by comparing the average days to resolve the bug reports with and without video(s). For each resolved bug report, the time to resolution was calculated as the number of days between the initial submission of the bug report and when the status of the bug report was changed to RESOLVED. It is possible that a bug report was resolved and then reopened, meaning that its history includes the status RESOLVED multiple times. In such cases, the time to resolution was measured until the day it was last changed to RESOLVED. In order to handle potential outliers, the bug reports with time to resolution outside of the interval formed by the 5th and 95th percentiles were removed (e.g., bug reports in IntelliJ with time to resolution more than 1,135 days and less than 7 days were removed from the dataset).

Figure 27 shows the trend for the average number of days to resolve bug reports with video minus the average days without video, on a year by year basis. Note that bug reports with video took longer to resolve than those without when comparing the mean (with 342.7 days, without 295.8 days). The difference, however, is not statistically significant (Welch Two

Sample t-test,  $t = 0.57$ ,  $df = 20.30$ ,  $p\text{-value} > 0.14$ ). The difference in how much longer becomes smaller over time, with the average over the three most recent years being just 43 days, which given the trend is unsurprisingly also statistically not significant (Welch Two Sample t-test,  $t = 1.05$ ,  $df = 3.75$ ,  $p\text{-value} > 0.14$ ).

Note that only for the last six and three years, Android and IntelliJ bug reports with video had a longer time to resolution than those without. The difference in how much longer was statistically significant only for IntelliJ (Welch Two Sample t-test,  $t = 2.55$ ,  $df = 9.54$ ,  $p\text{-value} < 0.14$ ).

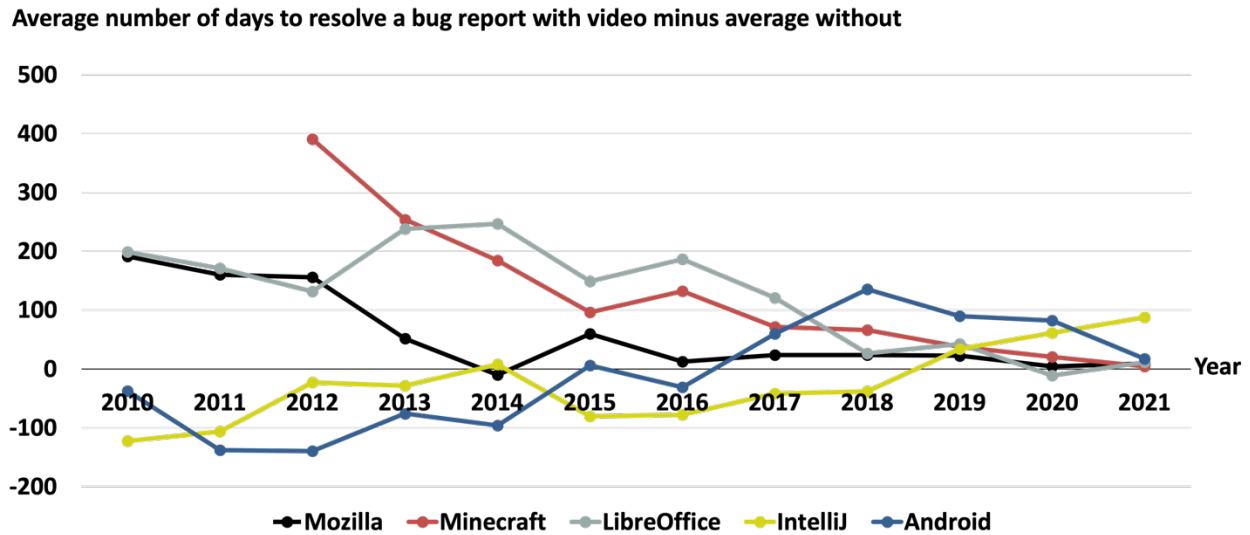


Figure 27. Comparing the average days to resolve bug reports with and without video attachments.

**Observation 3. Across all five projects it took on average a higher number of days to resolve the bug reports that included video attachments than the ones that did not, but the difference is not statistically significant, except for IntelliJ in later years. Moreover, the difference becomes considerably lower in the later years.**

Because initially submitted videos may bring clarity to the bug resolution process earlier than videos that are submitted later, I studied whether a difference may exist in impact on time to resolution for bug reports with videos that were submitted at the onset versus bug reports for which videos that were submitted as part of the subsequent interactions among the assigned developer, reporter, and others. Figure 28 shows the difference between average days to resolve a bug report with a video submitted initially versus the average for those with a video submitted later.

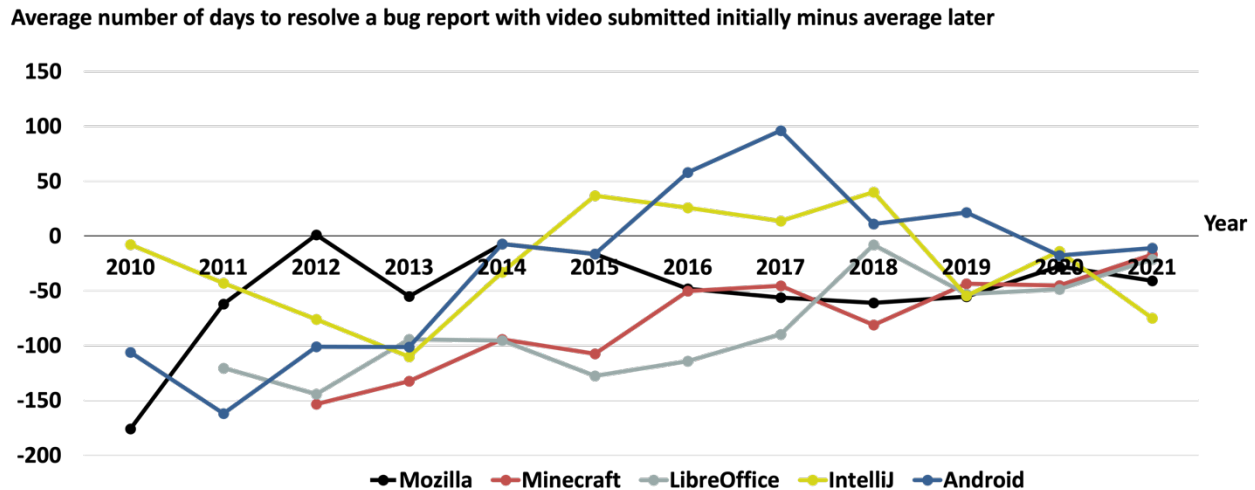


Figure 28. Difference between average number of days to resolve bug reports with videos attached initially and with videos attached later.

The average number of days to resolve bug reports with initially submitted videos is lower than those with later submitted videos across all years, with the mean for initially submitted 333.5 days and for later submitted 435.1 days. The difference is not statistically significant (Welch Two Sample t-test,  $t = -0.82$ ,  $df = 22$ ,  $p\text{-value} > 0.14$ ). Note that the difference is becoming lower over the years, with the difference changing from 102 days in 2010 to 48 days in 2016 and 26 days in 2021.

**Observation 4.** The difference between the average number of days to resolve bug reports with videos initially submitted versus those with videos submitted later is not statistically significant. Over the years, the difference has become smaller.

#### 4.2.2.1.1 Priority

High-priority bug reports tend to get more attention from developers than low-priority bugs reports; if the typical priority of bug reports with videos is low compared to those without videos, it might explain why bug reports with videos take longer to resolve. It might be that

videos attached to low-priority bug reports that are resolved slowly overshadow their importance for high-priority bug reports in the analyses thus far. Therefore, the bug reports were grouped into two priority groups: high priority (P1, P2, P3) and low priority (P4, P5, not set), as per Section 4.1.3. Then the average number of days to resolve for each group was compared, with and without video. The difference was not statistically significant (Welch Two Sample t-test,  $t = 1.44$ ,  $df = 18.17$ ,  $p\text{-value} > 0.14$ ) between the average number of days to resolve the high priority bug reports with video (mean = 454) and the high priority bug reports without video (mean = 301), indicating that priority likely does not play a role in the non-effects seen thus far. Because of the possibility of sensitivity to the period of time studied, the analysis was repeated for the last nine, six, and three years. The average time to resolution was still higher for the high priority bug reports with video (mean = 135.36) than high priority bug reports without video (mean = 72.41) for the three most recent years, yet the difference was still not statistically significant (Welch Two Sample t-test,  $t = 1.18$ ,  $df = 2.60$ ,  $p\text{-value} > 0.14$ ).

#### **4.2.2.1.2 Generalized Linear Model**

To further explore whether video may have an impact on bug resolution time, a Generalized Linear Model was built following the procedure outlined in Section 4.1.4. The resulting best model used only 11 factors out of the factors that were initially provided, and consisted of priority, resolution, readability, presence of steps to reproduce, presence of actual results, presence of expected results, number of unique words in title, number of unique words in description, number of back-and-forth, number of videos, and video submitted initially or later. The predicted value is the average number of days to resolve. I calculated McFadden's Adjusted  $R^2$  [158] as a quality indicator of the model. The McFadden Adjusted  $R^2$  of the model is 0.66, which means that the independent variables used in building the regression model only contribute to 66% of accurate prediction. The number of videos was a significant factor in the model ( $p < 0.14$ ), with an estimate of -0.02, which means that, while adding videos to bug reports reduces the time to resolution, it does so by only a minimal amount: 0.02 days, which is a mere 15 minutes. This, clearly, does not make a real difference from the perspective of the reporter.



The coefficient estimate for priority in the last 11 years ranged from lowest of  $-7.957e-02$  for P0 to highest of  $5.887e-01$  for P5. Moreover, the estimate for resolution varied from  $-8.011e-01$  (inactive) to  $9.932e-01$  (worksforme). Table 12 shows the estimates for the remaining factors used in the model.

Because of the possibility of sensitivity to the period of time studied, this analysis was repeated with shorter time windows. Results indicate a similarly small impact across all time windows, though it is interesting to observe that, for the time window of the past three years, a positive instead of negative coefficient exists for the number of videos: time to resolution is predicted to increase, though only by about 10 minutes, which again means there is no meaningful difference from the perspective of the reporter.

Table 12. Generalized Linear Model predicting the impact on average number of days to resolve bug reports.

|  | <b>2010-2021</b> | <b>2013-2021</b> | <b>2016-2021</b> | <b>2019-2021</b> |
|--|------------------|------------------|------------------|------------------|
| <b>McFadden's Adjusted R<sup>2</sup></b>     | 0.66             | 0.66             | 0.71             | 0.84             |
| <b>Number of back-and-forth</b>              | 3.020e-03        | 2.498e-03        | 1.046e-03        | 5.071e-04        |
| <b>Expected results</b>                      | 3.803e-02        | 8.121e-02        | 3.558e-01        | -2.305e-01       |
| <b>Steps to reproduce</b>                    | 1.037e-01        | 1.208e-01        | 2.024e-01        | 1.600e-01        |
| <b>Actual results</b>                        | -4.261e-01       | -4.471e-01       | -6.784e-01       | 3.468e-01        |
| <b>Readability</b>                           | -6.248e-05       | -6.591e-05       | -7.958e-04       | -5.064e-04       |
| <b>Number of unique words in title</b>       | 1.547e-02        | 1.518e-02        | 1.042e-02        | 4.422e-03        |
| <b>Number of unique words in description</b> | 3.604e-04        | 3.981e-04        | 1.357e-04        | -1.857e-04       |
| <b>Video submitted</b>                       | -1.153e-01       | -1.506e-01       | -1.225e-01       | 1.002e-01        |

Estimate

| <b>Initially</b>        |           |           |           |          |
|-------------------------|-----------|-----------|-----------|----------|
| <b>Number of videos</b> | -2.50E-02 | -6.81E-02 | -2.31E-02 | 1.61E-02 |
| <b>P-value</b>          | < 0.14    | < 0.14    | < 0.14    | < 0.14   |

**Observation 5. The impact of including videos on time to resolution is minimal.**

The following subsections detail the results of the Generalized Linear Model predicting the time to resolution of bug reports for individual systems.

#### **4.2.2.1.3 Android**

Table 13 shows the results of a Generalized Linear Model for Android. The resulting best model used only 11 factors consisted of priority, severity, status, readability, presence of steps to reproduce, presence of actual results, presence of expected results, number of unique words in title, number of attachments, number of videos, and video submitted initially or later. The McFadden Adjusted  $R^2$  of the model is 0.43, which means this set of factors is insufficient to build an accurate model. The number of videos is a significant factor in the model ( $p < 0.14$ ), with an estimate of 0.09. So, from the perspective of end-users, no beneficial incentive exists to add videos to bug reports, as it increases the time to resolution, though only by a minimal amount: 0.09 days. Across all time windows, as Table 13 shows, a similarly small impact exists for the results.

Table 13. Generalized Linear Model predicting the impact on average number of days to resolve Android bug reports.

|  | <b>2010-2021</b> | <b>2013-2021</b> | <b>2016-2021</b> | <b>2019-2021</b> |
|--|------------------|------------------|------------------|------------------|
|  |                  |                  |                  |                  |

|                 |  |            |            |            |            |
|-----------------|--|------------|------------|------------|------------|
| <b>Estimate</b> | <b>McFadden's Adjusted R<sup>2</sup></b> | 0.43       | 0.42       | 0.41       | 0.33       |
|                 | <b>Number of attachments</b>             | -2.730e-02 | -3.129e-02 | -1.423e-02 | -2.798e-03 |
|                 | <b>Expected results</b>                  | -5.051e-02 | -1.327e-02 | -2.192e-01 | -1.594e-01 |
|                 | <b>Steps to reproduce</b>                | -5.233e-02 | -1.259e+01 | 1.319e-01  | 7.317e-02  |
|                 | <b>Actual results</b>                    | 2.850e-02  | -1.154e-01 | -1.547e-01 | 4.258e-01  |
|                 | <b>Readability</b>                       | -4.125e-03 | -3.442e-03 | -5.105e-02 | -2.100e-02 |
|                 | <b>Number of unique words in title</b>   | 6.925e-03  | 7.925e-03  | 1.261e-02  | 1.060e-02  |
|                 | <b>Video submitted Initially</b>         | 3.325e-01  | 3.191e-01  | 2.913e-01  | 1.229e-01  |
|                 | <b>Number of videos</b>                  | 9.123e-02  | 1.005e-01  | 5.655e-02  | 2.523e-02  |
|                 | <b>P-value</b>                           | < 0.14     | < 0.14     | < 0.14     | < 0.14     |

#### 4.2.2.1.4 IntelliJ

The resulting best model for IntelliJ used only 10 factors, including priority, readability, presence of steps to reproduce, presence of actual results, presence of expected results, number of unique words in title, number of attachments, number of comments, number of videos, and video submitted initially or later. Table 14 shows the results of a Generalized Linear Model for IntelliJ. The McFadden Adjusted R<sup>2</sup> of the model is 0.89 but decreases to 0.74 for the last three years. The number of videos is a significant factor in the model ( $p < 0.14$ ), with an estimate of only 0.05, which means adding videos to bug reports increases the time to resolution by only 0.05 days. The analysis was repeated with shorter time windows and similarly minimal correlations were found.

Table 14. Generalized Linear Model predicting the impact on average number of days to resolve IntelliJ bug reports.

|  | 2010-2021  | 2013-2021  | 2016-2021  | 2019-2021  |
|--|------------|------------|------------|------------|
| <b>McFadden's Adjusted R<sup>2</sup></b> | 0.89       | 0.78       | 0.78       | 0.74       |
| <b>Number of attachments</b>             | -6.592e-02 | -3.802e-02 | -4.791e-02 | 1.066e-02  |
| <b>Number of comments</b>                | 1.159e-02  | 8.535e-03  | 9.501e-03  | 4.701e-03  |
| <b>Expected results</b>                  | -6.332e-02 | 1.996e-01  | -7.498e-02 | -5.812e-01 |
| <b>Steps to reproduce</b>                | -6.251e-01 | -5.299e-01 | -3.479e-01 | -3.574e-02 |
| <b>Actual results</b>                    | -4.356e-01 | -4.711e-01 | -1.431e-01 | 4.915e-01  |
| <b>Readability</b>                       | -3.377e-02 | -3.858e-04 | 1.578e-03  | 7.821e-03  |
| <b>Number of unique words in title</b>   | 1.639e-02  | 2.158e-02  | 2.995e-02  | 2.849e-02  |
| <b>Video submitted Initially</b>         | 4.395e+00  | 4.045e+00  | 3.597e+00  | 2.742e+00  |
| <b>Number of videos</b>                  | 5.161e-02  | 6.152e-02  | 8.783e-02  | 1.191e-01  |
| <b>P-value</b>                           | < 0.14     | < 0.14     | < 0.14     | < 0.14     |

Estimate

#### 4.2.2.1.5 LibreOffice

Table 15 shows the results of a Generalized Linear Model for LibreOffice. The resulting best model used 17 factors, including product, priority, platform, component, severity, resolution, version, readability, number of unique words in title, number of unique words in description, number of attachments, number of participants, status, number of videos, and video submitted initially or later. The McFadden Adjusted R<sup>2</sup> of the model is 0.52 and the number of videos is a statistically significant factor (-0.03). Thus, the model cannot be used to explain the effect of the number of videos on number of days to resolve, see Table 15. The analysis was repeated with shorter time windows and results were similar.

Table 15. Generalized Linear Model predicting the impact on average number of days to resolve LibreOffice bug reports.

|                 |  | <b>2010-2021</b> | <b>2013-2021</b> | <b>2016-2021</b> | <b>2019-2021</b> |
|-----------------|--|------------------|------------------|------------------|------------------|
| <b>Estimate</b> | <b>McFadden's Adjusted R<sup>2</sup></b>     | 0.52             | 0.51             | 0.47             | 0.42             |
|                 | <b>Number of attachments</b>                 | -4.058e-02       | 1.800e-02        | 1.299e-02        | 3.622e-02        |
|                 | <b>Number of participants</b>                | 7.884e-03        | 1.618e-01        | 1.609e-01        | 1.756e-01        |
|                 | <b>Readability</b>                           | 7.884e-03        | 7.966e-03        | 2.886e-03        | -8.605e-03       |
|                 | <b>Number of unique words in title</b>       | 6.968e-03        | 2.850e-03        | 4.674e-03        | 7.567e-03        |
|                 | <b>Number of unique words in description</b> | 2.403e-05        | 4.911e-05        | 6.611e-05        | 1.985e-04        |
|                 | <b>Video submitted Initially</b>             | 2.614e-01        | 1.438e-01        | 2.323e-01        | 2.270e-01        |
|                 | <b>Number of videos</b>                      | -2.521e-02       | -9.007e-02       | -9.630e-02       | -1.158e-01       |
|                 | <b>P-value</b>                               | < 0.14           | < 0.14           | < 0.14           | < 0.14           |

#### 4.2.2.1.6 Minecraft

The resulting best Generalized Linear Model for Minecraft used six factors, including product, priority, component, resolution, number of videos, and video submitted initially or later. Table 16 shows the results. The McFadden Adjusted  $R^2$  of the model is 0.61, which means that the independent variables used in building the regression model only contribute to 61% of accurate prediction. The number of videos is not a significant factor in the model ( $p > 0.14$ ), with an estimate of -1.64. It is interesting to observe that, for the shorter time windows, a positive instead of negative coefficient exists for the number of videos: time to resolution is predicted to increase by about 1.8 days, which is not a beneficial incentive for end-users to create videos.

Table 16. Generalized Linear Model predicting the impact on average number of days to resolve Minecraft bug reports.

|          |   | 2010-2021 | 2013-2021 | 2016-2021 | 2019-2021 |
|----------|---|-----------|-----------|-----------|-----------|
| Estimate | <b>McFadden's Adjusted <math>R^2</math></b> | 0.61      | 0.59      | 0.52      | 0.48      |
|          | <b>Video submitted</b>                      |           |           |           |           |
|          | <b>Initially</b>                            | 4.3       | 1.30      | 3.813     | 4.201     |
|          | <b>Number of videos</b>                     | -1.64     | 1.89      | 3.075     | 1.80      |
|          | <b>P-value</b>                              | 0.50      | 0.22      | < 0.14    | 0.36      |

#### 4.2.2.1.7 Mozilla

Table 17 shows the results of a Generalized Linear Model for Mozilla. The resulting best model used 15 factors that were initially provided, and included product, priority, severity, component, status, number of duplicates, number of comments, resolution, readability, presence of actual results, presence of expected results, number of videos, video submitted initially or later, bug reporter's reputation, and whether bug reporter is a developer or an

end-user. The McFadden Adjusted R<sup>2</sup> of the model is 0.54, which means that the independent variables used in building the regression model only contribute to 54% of accurate prediction. The number of videos was a significant factor in the model ( $p < 0.14$ ), with an estimate of -0.02 which means that, while adding videos to bug reports reduces the time to resolution, it does so by only a minimal amount: 0.02 days, which is a mere 15 minutes. Results indicate a similarly small impact across all time windows, though it is interesting to observe that, for the time window of the past three years, a positive instead of negative correlation exists: time to resolution is predicted to increase. Note that across all time windows, if the bug reporter was a developer, rather than end-user, time to resolution dropped, but only by a minimal amount: 0.05 days.

Table 17. Generalized Linear Model predicting the impact on average number of days to resolve Mozilla bug reports.

|  | <b>2010-<br/>2021</b> | <b>2013-<br/>2021</b> | <b>2016-<br/>2021</b> | <b>2019-<br/>2021</b> |
|--|-----------------------|-----------------------|-----------------------|-----------------------|
| <b>McFadden's Adjusted R<sup>2</sup></b>     | 0.54                  | 0.62                  | 0.6                   | 0.53                  |
| <b>Number of duplicates</b>                  | 1.294e-01             | -1.080e-01            | 3.181e-02             | -1.080e-01            |
| <b>Number of comments</b>                    | 2.534e-03             | 1.086e-02             | 7.735e-03             | 1.086e-02             |
| <b>Expected results</b>                      | -5.522e-02            | -3.818e-01            | 3.549e-01             | -3.818e-01            |
| <b>Actual results</b>                        | 8.127e-02             | 2.486e-01             | -2.903e-01            | 2.486e-01             |
| <b>Bug reporter's reputation</b>             | 1.123e-02             | 9.949e-02             | 3.181e-02             | 9.949e-02             |
| <b>Developer or end-user<br/>(developer)</b> | -5.256e-02            | -2.729e-01            | -1.840e-01            | -2.729e-01            |
| <b>Readability</b>                           | -2.173e-05            | -4.059e-04            | -2.550e-04            | -4.059e-04            |
| <b>Video submitted<br/>Initially</b>         | -6.008e-02            | 1.843e-01             | -6.628e-02            | 1.843e-01             |

Estimate

|                         |           |           |           |          |
|-------------------------|-----------|-----------|-----------|----------|
| <b>Number of videos</b> | -2.41E-02 | -1.47E-02 | -5.47E-02 | 2.82E-02 |
| <b>P-value</b>          | < 0.14    | < 0.14    | < 0.14    | < 0.14   |

#### 4.2.2.2 RQ2: Resolution with a Patch (FIXED)

A reduced time to resolution is not the only possible desirable outcome; if the presence of videos were to lead to a higher percentage of bug reports being resolved with a patch (i.e., resolution status of FIXED), that might serve as an important incentive for reporters to submit videos along with their bug reports since it increases their chances of having their issue be addressed.

Figure 29 shows the difference in percentage of bug reports successfully resolved with video and without. Comparing this difference on all 49,661 bug reports with video and 99,322 without reveals that, across all 11 years, a higher percentage of bug reports without videos were resolved with a resolution status of FIXED. This difference is statistically significant (Welch Two Sample t-test,  $t = 6.85$ ,  $df = 21.81$ ,  $p\text{-value} < 0.14$ ). The difference, however, is no longer statistically significant over the last three years (Welch Two Sample t-test,  $t = 3.73$ ,  $df = 2.16$ ,  $p\text{-value} > 0.14$ ).



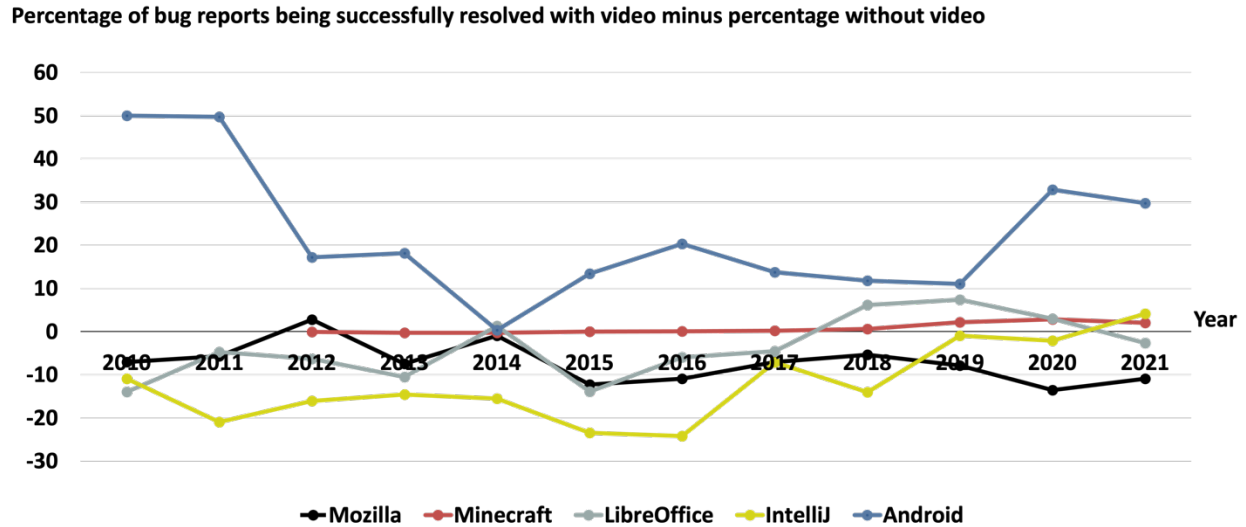


Figure 29. Difference between percentage of bug reports with and without video attachments that were successfully resolved with a patch.

Examining the difference on a system-by-system basis, the results are different for different systems. For Android and Minecraft, a higher percentage of fixed and resolved bug reports have video attachments, whereas for Mozilla and IntelliJ the opposite is true (all results are statistically significant over 11 years and three years, except IntelliJ where the last three years are no longer statistically significant). The difference in LibreOffice is not statistically significant. These results show that individual systems can experience differences in outcomes. Android in particular stands out, which is perhaps not surprising because it is a very visual platform. It is also interesting to observe that IntelliJ has clearly trended away from videos having a negative impact.

#### 4.2.2.2.1 Priority

Considering only high priority bug reports (as defined in Section 4.1.3), results showed that a higher but non-statistically significant percentage of high priority bug reports without videos were resolved with a patch (Welch Two Sample t-test,  $t = -1.45$ ,  $df = 18.81$ ,  $p\text{-value} > 0.14$ ). The difference remained not statistically significant (Welch Two Sample t-test,  $t =$

0.45, df = 2.5, p-value > 0.14) when only considering the last three years. Together, these results indicate that priority does not appear to affect findings that were observed thus far.

#### 4.2.2.2.2 Generalized Linear Model

To further examine the effect, a Generalized Linear Model was built. Out of the 26 factors introduced in Section 4.1.4, nine factors remained to be included in the selected best model, namely priority, readability, number of unique words in title, presence of actual results, presence of steps to reproduce, number of back-and-forth, time to resolution, number of videos, and video submitted initially or later.

The predicted value is bug reports being resolved with a patch, yes or no. The McFadden Adjusted R<sup>2</sup> of the model is 0.29, which means that this set of factors is insufficient to build an accurate regression model (see Table 18). Note that priority ranged from -5.753e-01 (P5) to 9.255e-01 (Not set). The same model was built for the last nine, six, and three years and encountered the same situation. Because the p-values are low, the model itself is inaccurate and thus the effect of number of videos on resolution with a patch cannot be interpreted or explained. Overall, then, this means that other factors than the 26 are at work. Until further study, the results only show that a meaningful difference exists per Figure 29 and the Welch Two Sample t-test.

Table 18. Generalized Linear Model predicting percentage of bug reports with and without videos being resolved as FIXED.

|          |  | 2010-2021  | 2013-2021  | 2016-2021  | 2019-2021 |
|----------|--|------------|------------|------------|-----------|
| Estimate | <b>McFadden's Adjusted R<sup>2</sup></b> | 0.29       | 0.36       | 0.42       | 0.55      |
|          | <b>Number of back-and-forth</b>          | 1.672e-03  | 1.618e-03  | 1.899e-03  | 3.98E-03  |
|          | <b>Steps to reproduce</b>                | -5.591e-02 | -4.962e-02 | -2.650e-02 | 1.19E-02  |
|          | <b>Actual results</b>                    | -8.609e-02 | -9.119e-02 | -5.011e-02 | -1.66E-02 |

|  |            |            |            |           |
|--|------------|------------|------------|-----------|
| <b>Readability</b>                     | 1.541e-04  | 1.389e-04  | 2.219e-04  | 5.25E-04  |
| <b>Number of unique words in title</b> | 8.568e-03  | 8.840e-03  | 1.377e-02  | 1.18E-02  |
| <b>Time to resolution</b>              | -4.453e-04 | -5.492e-04 | -3.501e-04 | -3.01E-04 |
| <b>Video submitted<br/>Initially</b>   | 1.539e-01  | 1.438e-01  | -6.206e-02 | -1.80E-01 |
| <b>Number of videos</b>                | 3.947e-02  | 3.970e-02  | 1.951e-02  | 2.39E-02  |
| <b>P-value</b>                         | < 0.14     | < 0.14     | < 0.14     | < 0.14    |

**Observation 6. Overall, including videos does not have a statistically significant impact on bug reports being fixed and resolved. However, individual systems differ, with Android and Minecraft seeing a positive impact on the number being fixed and Mozilla and IntelliJ a negative impact (though for IntelliJ this impact becomes minimal in the later years).**

The following subsections present the results of the Generalized Linear Model predicting the percentage of bug reports being resolved with a patch for individual systems.

#### **4.2.2.2.3 Android**

Table 19 shows the results of a Generalized Linear Model for Android. The resulting best model used 11 factors consisted of severity, time to resolution, status, readability, presence of steps to reproduce, presence of actual results, presence of expected results, number of unique words in title, number of attachments, number of videos, and video submitted initially or later. The McFadden Adjusted  $R^2$  of the model is 0.63 which means these factors were not sufficient to build an accurate regression model. The number of videos is not a significant factor in the model ( $p > 0.14$ ), with an estimate of  $1.955e-15$ , which means that the impact of adding videos to Android bug reports on the chance of getting resolved with a

patch is negligible. This, clearly, does not make a real difference from the perspective of the reporter. The results of the analysis with shorter time windows (the last nine, six, and three years) indicated a similarly small impact across all time windows, see Table 19.

Table 19. Generalized Linear Model predicting percentage of Android bug reports with and without videos being resolved as FIXED.

|  | <b>2010-<br/>2021</b> | <b>2013-<br/>2021</b> | <b>2016-<br/>2021</b> | <b>2019-<br/>2021</b> |
|--|-----------------------|-----------------------|-----------------------|-----------------------|
| <b>McFadden's Adjusted R<sup>2</sup></b> | 0.63                  | 0.58                  | 0.56                  | 0.54                  |
| <b>Number of attachments</b>             | -9.332e-17            | -2.571e-02            | -9.332e-17            | 1.633e-17             |
| <b>Expected results</b>                  | 1.027e-12             | 2.596e-01             | -1.298e-12            | 3.373e-13             |
| <b>Steps to reproduce</b>                | 1.257e-13             | 1.650e-01             | 1.257e-13             | 9.816e-14             |
| <b>Actual results</b>                    | -1.298e-12            | -7.430e-01            | 1.027e-12             | -3.919e-13            |
| <b>Readability</b>                       | -3.533e-17            | -5.397e-02            | -3.533e-17            | -5.327e-17            |
| <b>Number of unique words in title</b>   | -3.931e-15            | 1.145e-02             | -3.931e-15            | -3.521e-15            |
| <b>Time to resolution</b>                | 5.955e-16             | 5.955e-16             | 5.955e-16             | 5.425e-16             |
| <b>Video submitted Initially</b>         | 1.062e-14             | 2.316e-01             | 1.062e-14             | 5.042e-15             |
| <b>Number of videos</b>                  | 1.955e-15             | 1.304e-02             | 1.955e-15             | 1.488e-16             |
| <b>P-value</b>                           | > 0.14                | > 0.14                | > 0.14                | > 0.14                |

Estimate

#### 4.2.2.2.4 IntelliJ

The resulting best Generalized Linear Model for IntelliJ only used 10 factors and consisted of priority, time to resolution, presence of steps to reproduce, presence of expected results,

number of unique words in title, number of unique words in description, readability, number of comments, number of videos, and video submitted initially or later. The McFadden Adjusted  $R^2$  of the model is 0.89, which means that the independent variables used in building the regression model contribute to 89% of accurate prediction. The number of videos is a significant factor in the model ( $p < 0.14$ ), with an estimate of 0.027 (Table 20). That means adding videos to bug reports reduces the chance of getting resolved with patch, by around 3%. Though it is interesting to observe that, while the number of videos had a similarly small impact in shorter time windows, it was no longer a statistically significant factor in the model (see Table 20).

Table 20. Generalized Linear Model predicting percentage of IntelliJ bug reports with and without videos being resolved as FIXED.

|                 |  | <b>2010-2021</b> | <b>2013-2021</b> | <b>2016-2021</b> | <b>2019-2021</b> |
|-----------------|--|------------------|------------------|------------------|------------------|
| <b>Estimate</b> | <b>McFadden's Adjusted <math>R^2</math></b>  | 0.89             | 0.9              | 0.91             | 0.9              |
|                 | <b>Time to resolution</b>                    | -1.053e-04       | 1.953e-05        | 2.572e-05        | 8.989e-05        |
|                 | <b>Steps to reproduce</b>                    | -1.556e-01       | -8.511e-02       | -9.574e-02       | -4.518e-02       |
|                 | <b>Expected results</b>                      | 1.501e-01        | 2.153e-01        | 2.402e-01        | 1.579e-01        |
|                 | <b>Number of unique words in title</b>       | 1.168e-02        | 1.095e-02        | 1.047e-02        | 7.104e-03        |
|                 | <b>Number of comments</b>                    | -3.762e-03       | 5.038e-04        | 1.018e-04        | -2.364e-03       |
|                 | <b>Number of unique words in description</b> | 2.544e-04        | 1.367e-04        | 5.565e-05        | 4.156e-05        |
|                 | <b>Readability</b>                           | 4.374e-03        | 2.664e-03        | 2.755e-03        | 1.775e-03        |
|                 | <b>Number of videos</b>                      | 2.734e-02        | 7.874e-03        | 5.513e-03        | 1.330e-02        |
|                 | <b>Video submitted</b>                       | 5.457e-01        | 7.823e-01        | 8.972e-01        | 9.172e-01        |

| <b>Initially</b> |        |        |        |        |
|------------------|--------|--------|--------|--------|
| <b>P-value</b>   | < 0.14 | > 0.14 | > 0.14 | > 0.14 |

#### 4.2.2.2.5 LibreOffice

Table 21 shows the results of a Generalized Linear Model for LibreOffice. The resulting best model used 11 factors including severity, time to resolution, readability, number of unique words in title, presence of actual results, presence of expected results, presence of steps to reproduce, number of attachments, status, number of videos, and video submitted initially or later. The McFadden Adjusted R<sup>2</sup> of the model is 0.53, which means that this set of factors is insufficient to build an accurate regression model. The same model was built for the last nine, six, and three years and the same situation occurred (see Table 21). While the p-values are low (< 0.14), the number of videos did not impact the percentage of bug reports being resolved with patch (1.955e-15).

Table 21. Generalized Linear Model predicting percentage of LibreOffice bug reports with and without videos being resolved as FIXED.

|                 |  | <b>2010-2021</b> | <b>2013-2021</b> | <b>2016-2021</b> | <b>2019-2021</b> |
|-----------------|--|------------------|------------------|------------------|------------------|
| <b>Estimate</b> | <b>McFadden's Adjusted R<sup>2</sup></b> | 0.53             | 0.53             | 0.53             | 0.52             |
|                 | <b>Time to resolution</b>                | 5.955e-16        | 1.596e-13        | 2.033e-15        | 2.033e-15        |
|                 | <b>Steps to reproduce</b>                | 1.257e-13        | -2.839e-13       | -4.563e-14       | 1.806e-14        |
|                 | <b>Expected results</b>                  | 1.027e-12        | 2.993e-13        | 3.030e+01        | 1.679e-01        |
|                 | <b>Actual results</b>                    | -1.298e-12       | 1.256e-13        | -3.766e-12       | -3.274e-12       |
|                 | <b>Number of unique words in title</b>   | -3.931e-15       | 2.529e-15        | 1.374e-15        | 1.530e-03        |
|                 | <b>Readability</b>                       | -3.533e-17       | 1.058e-14        | 8.648e-15        | -1.062e-03       |

|                                      |            |            |            |            |
|--------------------------------------|------------|------------|------------|------------|
| <b>Number of attachments</b>         | -9.332e-17 | 4.895e-14  | 5.083e-14  | 3.162e-02  |
| <b>Number of videos</b>              | 1.955e-15  | -1.665e-13 | -1.605e-13 | -1.006e-01 |
| <b>Video submitted<br/>Initially</b> | 1.062e-14  | 3.533e-13  | 2.300e-13  | 4.624e-01  |
| <b>P-value</b>                       | < 0.14     | < 0.14     | < 0.14     | < 0.14     |

#### 4.2.2.2.6 Minecraft

Table 22 illustrates the results of a Generalized Linear Model for Minecraft. The resulting best model used only five factors out of the factors that were initially provided, including component, confirmation status, labels, number of videos, and video submitted initially or later. The McFadden Adjusted R<sup>2</sup> of the model is 0.52, which means that these five factors are insufficient to build an accurate regression. Building the same model for the shorter time windows led to similar results (see Table 22). The number of videos was a significant factor in the model ( $p < 0.14$ ), with an estimate of 0.012, which is a minimal amount from the perspective of bug reporters.

Table 22. Generalized Linear Model predicting percentage of Minecraft bug reports with and without videos being resolved as FIXED.

|                                      | 2010-2021 | 2013-2021 | 2016-2021 | 2019-2021 |
|--------------------------------------|-----------|-----------|-----------|-----------|
| <b>McFadden's<br/>Adjusted R2</b>    | 0.52      | 0.53      | 0.53      | 0.52      |
| <b>Video submitted<br/>Initially</b> | 3.115e-02 | 3.550e-02 | 3.609e-02 | 3.543e-02 |
| <b>Number of<br/>videos</b>          | 1.19E-02  | 9.78E-03  | 1.27E-02  | 1.23E-02  |

|  |                |          |          |          |          |
|--|----------------|----------|----------|----------|----------|
|  | <b>P-value</b> | $< 0.14$ | $< 0.14$ | $< 0.14$ | $< 0.14$ |
|--|----------------|----------|----------|----------|----------|

#### 4.2.2.2.7 Mozilla

Table 23 shows some of the results of a Generalized Linear Model for Mozilla. The resulting best model used only eight factors out of the 26 total factors that were initially provided, and included priority, status, number of duplicates, number of comments, number of videos, video submitted initially or later, bug reporter's reputation, and whether bug reporter is a developer or an end-user. The McFadden Adjusted  $R^2$  of the model is 0.03, which means that this set of factors is insufficient to build an accurate regression model. The same model was built for shorter time windows and the same situation occurred. While the p-values are low, the model itself is inaccurate and thus the effect of number of videos on resolution with a patch cannot be interpreted or explained. Overall, then, this means that other factors than the ones identified are at work. Until further study, results only show that a meaningful difference exists per Figure 29 and the Welch Two Sample t-test.

Table 23. Generalized Linear Model predicting percentage of Mozilla bug reports with and without videos being resolved as FIXED.

|  | <b>2010-<br/>2021</b> | <b>2013-<br/>2021</b> | <b>2016-<br/>2021</b> | <b>2019-<br/>2021</b> |
|--|-----------------------|-----------------------|-----------------------|-----------------------|
| <b>McFadden's Adjusted <math>R^2</math></b>  | 0.03                  | 0.09                  | 0.12                  | 0.12                  |
| <b>Number of duplicates</b>                  | 0.048                 | 0.075                 | 0.053                 | 0.024                 |
| <b>Number of comments</b>                    | 0.0017                | 0.001                 | 0.004                 | 0.007                 |
| <b>Bug reporter's reputation</b>             | 0.021                 | 0.058                 | 0.056                 | 0.041                 |
| <b>Developer or end-user<br/>(developer)</b> | -0.14                 | -0.39                 | -0.37                 | -0.40                 |
| <b>Video submitted</b>                       | 0.074                 | 0.029                 | -0.059                | -0.074                |

Esti



| <b>Initially</b>        |        |        |        |        |
|-------------------------|--------|--------|--------|--------|
| <b>Number of videos</b> | 0.067  | 0.092  | 0.084  | 0.034  |
| <b>P-value</b>          | < 0.14 | < 0.14 | < 0.14 | < 0.14 |

**Observation 7. The impact of including videos on percentage of bug reports being resolved with a patch is minimal; and in some systems non-existence (e.g., Android, LibreOffice).**

#### 4.2.2.3 RQ3: Back-and-forth

Another way in which videos could possibly have a positive impact on the resolution process is if they reduced the back-and-forth, that is, if their presence led to fewer requests from developers to reporters for clarification or additional information (see Section 4.1.3). Recall that the dataset only considers bug reports that have been resolved, which might imply that, if bug reports with video induce fewer requests by the developer to the reporter, the videos may answer questions that developers otherwise would be asking. A reduction in back-and-forth would be beneficial to both the reporter and the developer and potentially increase the likelihood of the bug report being resolved because some portion of the requests to reporters go unanswered [15].

Figure 30 shows that bug reports with videos had a consistently higher average number of back-and-forth over the years, except for LibreOffice in 2010 and 2013. The difference was statistically significant for each of the five projects individually and for all together (Welch Two Sample t-test,  $t = 5.54$ ,  $df = 11.63$ ,  $p\text{-value} < 0.14$ ). The average back-and-forth for bug reports with videos stayed higher, even when considering just the three most recent years (an average of 4 extra back-and-forth, Welch Two Sample t-test,  $t = 4.75$ ,  $df = 3.17$ ,  $p\text{-value} < 0.14$ ).

Average number of back-and-forth with video minus average without

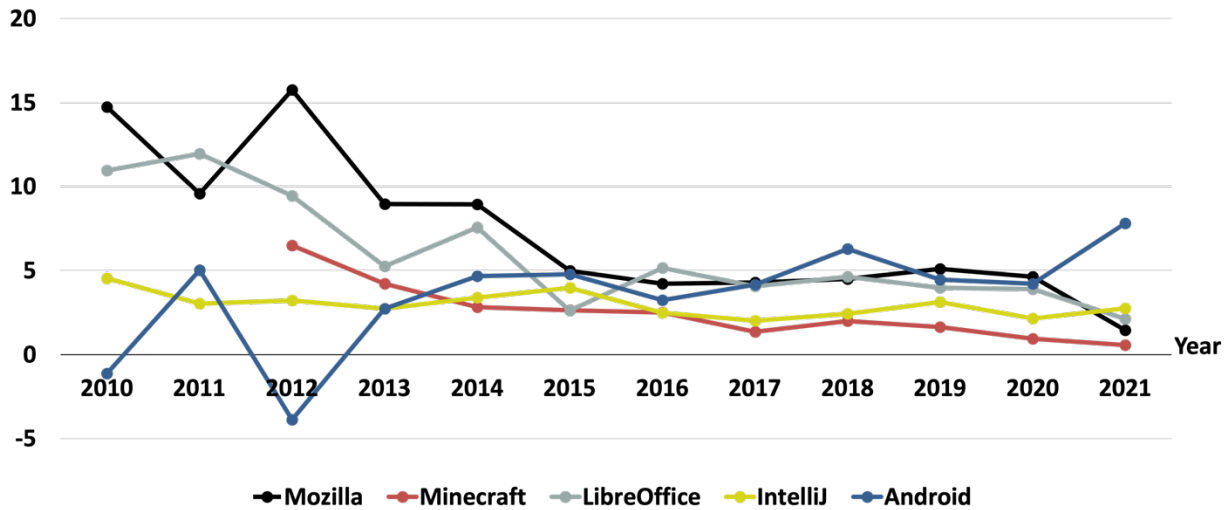


Figure 30. Difference between average number of back-and-forth in bug reports with and without video attachments.

**Observation 8. Over the years, bug reports with video had a higher average back-and-forth than bug reports without video.**

Next, videos that were submitted initially or later were examined to see if a difference might exist between the two groups. Videos submitted later can be indicative of a variety of factors, including the initial textual bug report being of poor quality or the possibility that requests for video are made by developers because these bug reports concern more difficult bugs. Figure 31 compares the averages, showing that bug reports with videos initially submitted on average had a lower number of back-and-forth throughout the years than videos that were submitted later. This difference is statistically significant for all five projects individually and also across all of them together (Welch Two Sample t-test,  $t = -10.48$ ,  $df = 14.82$ ,  $p\text{-value} < 0.14$ ).

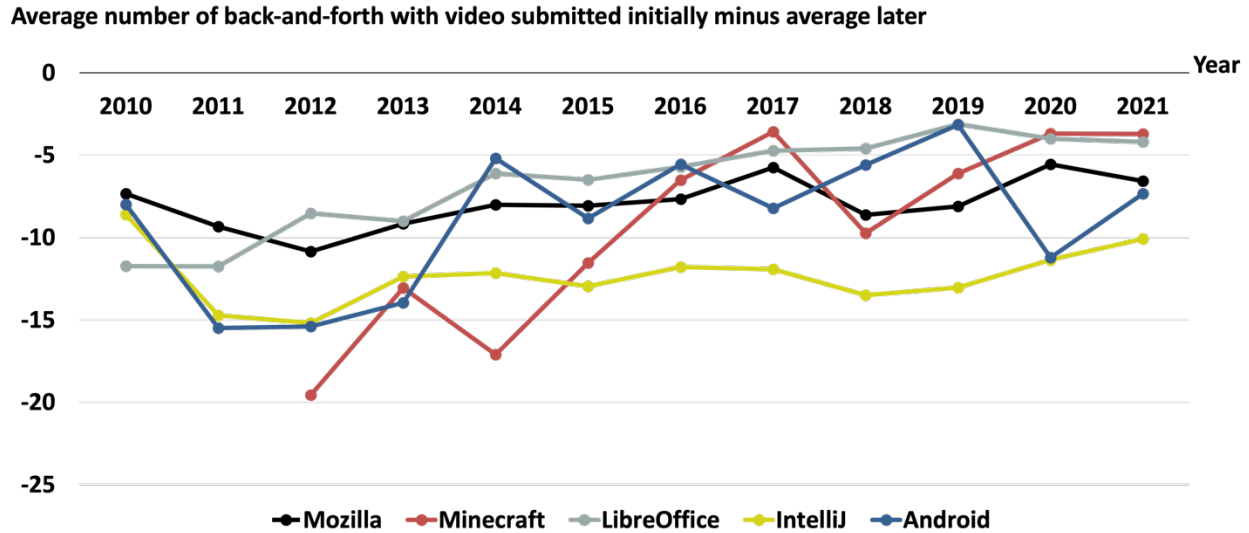


Figure 31. Difference between average number of back-and-forth for bug reports with videos submitted initially and those submitted later.

While changing slightly, it also remains statistically significant when considering just the last three years (Welch Two Sample t-test,  $t = -9.09$ ,  $df = 3.67$ ,  $p\text{-value} < 0.14$ ). Interestingly, a statistically significant difference remains even if all back-and-forth, that takes place before videos were submitted, was ignored (Welch Two Sample t-test,  $t = 1.17$ ,  $df = 15.10$ ,  $p\text{-value} < 0.14$ ). That is, for bug reports with videos submitted later, most of the back-and-forth happens after the video is submitted and this amount of back-and-forth is on average still greater than that for videos initially submitted. Combined with the fact that bug reports with video tend to take longer to resolve (Section 4.2.2.1), this may be a possible indication of more difficult videos leading developers to reach out and request videos to help clarify the issue (see Section 4.3 for more discussion) .

**Observation 9.** Solely counting back-and-forth after videos are submitted, the average back-and-forth for videos submitted later remains higher than for videos submitted initially.

#### **4.2.2.3.1 Priority**

Focusing only on high priority bug reports (as defined in Section 4.1.3), results remained the same: bug reports with video had a higher average number of back-and-forth (Welch Two Sample t-test  $t = 5.91$ ,  $df = 11.03$ ,  $p\text{-value} < 0.14$ ). The difference was statistically significant (Welch Two Sample t-test  $t = 5.91$ ,  $df = 11.03$ ,  $p\text{-value} < 0.14$ ). Considering just the three most recent years, the average back-and-forth for bug reports with videos stayed higher and the difference remained statistically significant (Welch Two Sample t-test,  $t = 3.52$ ,  $df = 2.01$ ,  $p\text{-value} < 0.14$ ).

#### **4.2.2.3.2 Generalized Linear Model**

To further study the effect of including video on the number of back-and-forth associated with bug reports, a Generalized Linear Model was built following the process outlined in Section 4.1.4. Out of the 26 factors that were initially considered, 10 factors were selected for inclusion in the resulting best model, namely priority, resolution, readability score, number of unique words in title, number of unique words in description, presence of actual results, presence of expected results, time to resolution, number of videos, and video submitted initially or later. The predicted value is the average number of back-and-forth.

The McFadden Adjusted  $R^2$  of the model is 0.64, which means that this set of 11 factors together is able to explain 64% of the variability. The same model was built for the last nine, six, and three years, with the result shown in Table 24. The high McFadden Adjusted  $R^2$  value, especially for the last three years (0.82), indicates that these 11 factors are strongly associated with the average number of back-and-forth. However, the relatively low but statistically significant regression coefficient of number of videos indicates that this factor alone is not a major factor for number of back-and-forth, with the increase a mere 0.15 per video (last three years).

Table 24. Generalized Linear Model predicting the impact on average back-and-forth by the number of videos included in a bug report.

|                 |  | <b>2010-2021</b> | <b>2013-2021</b> | <b>2016-2021</b> | <b>2019-2021</b> |
|-----------------|--|------------------|------------------|------------------|------------------|
| <b>Estimate</b> | <b>McFadden’s Adjusted R<sup>2</sup></b>     | 0.64             | 0.69             | 0.75             | 0.82             |
|                 | <b>Readability</b>                           | -3.669e-04       | -3.356e-04       | -4.003e-04       | -8.182e-04       |
|                 | <b>Number of unique words in description</b> | 2.079e-04        | 2.035e-04        | 1.339e-04        | 3.074e-05        |
|                 | <b>Number of unique words in title</b>       | -5.176e-03       | -4.996e-03       | -5.473e-03       | -1.431e-02       |
|                 | <b>Actual results</b>                        | 3.683e-02        | 2.375e-02        | 2.787e-02        | -2.623e-02       |
|                 | <b>Expected results</b>                      | -8.722e-02       | -8.942e-02       | -1.216e-01       | -1.274e-01       |
|                 | <b>Tine to resolution</b>                    | 2.326e-04        | 1.928e-04        | 1.528e-04        | 3.020e-04        |
|                 | <b>Number of videos</b>                      | 1.929e-01        | 1.953e-01        | 1.885e-01        | 1.482e-01        |
|                 | <b>Video submitted Initially</b>             | 4.057e-01        | 4.082e-01        | 5.931e-01        | 6.977e-01        |
|                 | <b>P-value</b>                               | < 0.14           | < 0.14           | < 0.14           | < 0.14           |

In the following subsections, the results of the Generalized Linear Models predicting the average number of back-and-forth were detailed for each systems.

**4.2.2.3.3 Android**

The resulting best Generalized Linear Model for Android used 11 factors consisted of priority, severity, status, readability, presence of steps to reproduce, presence of actual results, presence of expected results, number of unique words in title, number of

attachments, number of videos, and video submitted initially or later. The McFadden Adjusted R<sup>2</sup> of the model is 0.43 (see Table 25). That means this set of factors is insufficient to build an accurate regression model. The same model was built for the last 9, 6, and 3 and the results were similar. While the number of videos is a significant factor in the model (p < 0.14), the model itself is inaccurate and thus the effect of number of videos cannot be further explained.

Table 25. Generalized Linear Model predicting the impact on average back-and-forth by the number of videos included in a Android bug report.

|                 |  | <b>2010-2021</b> | <b>2013-2021</b> | <b>2016-2021</b> | <b>2019-2021</b> |
|-----------------|--|------------------|------------------|------------------|------------------|
| <b>Estimate</b> | <b>McFadden's Adjusted R<sup>2</sup></b> | 0.45             | 0.47             | 0.53             | 0.65             |
|                 | <b>Number of attachments</b>             | 0.024            | 0.023            | 0.022            | 0.021            |
|                 | <b>Steps to reproduce</b>                | -0.004           | -0.084           | -0.077           | 0.052            |
|                 | <b>Expected results</b>                  | -0.0099          | -0.041           | -0.18            | 0.027            |
|                 | <b>Actual results</b>                    | 0.51             | 0.22             | 0.35             | -0.12            |
|                 | <b>Number of unique words in title</b>   | 0.014            | 0.014            | 0.016            | 0.010            |
|                 | <b>Readability</b>                       | -0.017           | -0.016           | -0.019           | -0.021           |
|                 | <b>Number of videos</b>                  | 0.05             | 0.056            | 0.061            | 0.025            |
|                 | <b>Video submitted Initially</b>         | -0.86            | -0.82            | -0.84            | -0.77            |
|                 | <b>P-value</b>                           | < 0.14           | < 0.14           | < 0.14           | < 0.14           |

#### 4.2.2.3.4 IntelliJ

Table 26 shows the results of a Generalized Linear Model for IntelliJ. The resulting best model used only nine factors including priority, readability, presence of steps to reproduce, presence of expected results, number of unique words in title, number of attachments, Time to resolution, number of videos, and video submitted initially or later. The McFadden Adjusted  $R^2$  of the model is 0.69. The number of videos is a significant factor in the model ( $p < 0.14$ ), with an estimate of 0.24, which indicates that this factor alone is not a major factor for number of back-and-forth and does not make a beneficial incentive from the perspective of the bug reporter.

Table 26. Generalized Linear Model predicting the impact on average back-and-forth by the number of videos included in an IntelliJ bug report.

|                 |   | 2010-2021  | 2013-2021  | 2016-2021  | 2019-2021 |
|-----------------|---|------------|------------|------------|-----------|
| <b>Estimate</b> | <b>McFadden's Adjusted <math>R^2</math></b> | 0.69       | 0.67       | 0.66       | 0.68      |
|                 | <b>Steps to reproduce</b>                   | -1.478e-01 | -1.674e-01 | -1.503e-01 | -0.072    |
|                 | <b>Expected results</b>                     | 2.424e-01  | 2.385e-01  | 2.516e-01  | 0.13      |
|                 | <b>Number of unique words in title</b>      | -2.516e-02 | -2.126e-02 | -1.405e-02 | -0.004    |
|                 | <b>Time to resolution</b>                   | 2.993e-04  | 2.940e-04  | 4.038e-04  | 0.0009    |
|                 | <b>Readability</b>                          | -2.878e-03 | 5.765e-03  | 7.904e-03  | 0.007     |
|                 | <b>Number of attachments</b>                | 1.135e-01  | 1.088e-01  | 1.058e-01  | 0.1       |

|                         |           |           |           |           |
|-------------------------|-----------|-----------|-----------|-----------|
| <b>Number of videos</b> | 2.343e-01 | 2.445e-01 | 2.415e-01 | 2.415e-01 |
| <b>Video submitted</b>  |           |           |           |           |
| <b>Initially</b>        | 1.39      | 1.31      | 8.99      | 0.88      |
| <b>P-value</b>          | < 0.14    | < 0.14    | < 0.14    | < 0.14    |

#### 4.2.2.3.5 LibreOffice

Out of all factors that were initially considered, the resulting best Generalized Linear Model for LibreOffice used 17 factors: product, priority, platform, component, severity, resolution, version, readability, number of unique words in title, number of unique words in description, presence of steps to reproduce, number of attachments, number of participants, status, whether it is confirmed, number of videos, and video submitted initially or later. The McFadden Adjusted  $R^2$  of the model is 0.40, which means that this set of factors is insufficient to build an accurate regression model. The same model was built for shorter time windows and results indicated a similar impact across all time windows (see Table 27). Again, the model cannot be used to interpret the effect of number of videos on number of back-and-forth.

Table 27. Generalized Linear Model predicting the impact on average back-and-forth by the number of videos included in a LibreOffice bug report.

|   | 2010-2021 | 2013-2021 | 2016-2021 | 2019-2021 |
|---|-----------|-----------|-----------|-----------|
| <b>McFadden's Adjusted <math>R^2</math></b> | 0.40      | 0.41      | 0.39      | 0.38      |
| <b>Number of unique words in title</b>      | 4.806e-03 | 1.578e-03 | 4.478e-03 | 2.492e-03 |
| <b>Presence of steps to reproduce</b>       | 4.448e-02 | 1.588e-02 | 3.223e-03 | 1.612e-02 |

Estimate



|  |            |            |            |            |
|--|------------|------------|------------|------------|
| <b>Number of unique words in description</b> | 9.601e-05  | 8.103e-05  | 9.678e-05  | 1.411e-04  |
| <b>Readability</b>                           | -3.194e-03 | 6.877e-04  | 5.799e-05  | -1.474e-03 |
| <b>Number of participants</b>                | 1.156e-01  | 1.357e-01  | 1.455e-01  | 1.610e-01  |
| <b>Number of attachments</b>                 | 5.181e-02  | 9.620e-02  | 9.986e-02  | 1.234e-01  |
| <b>Number of videos</b>                      | 4.208e-02  | -6.907e-03 | -2.218e-02 | -3.208e-02 |
| <b>Video submitted Initially</b>             | 2.153e-01  | 1.974e-01  | 1.852e-01  | 1.514e-01  |
| <b>Is confirmed</b>                          | 2.649e-01  | 2.072e-01  | 1.703e-01  | 1.462e-01  |
| <b>P-value</b>                               | < 0.14     | < 0.14     | < 0.14     | < 0.14     |

#### 4.2.2.3.6 Minecraft

For Minecraft, the best Generalized Linear Model used only seven factors out of the total factors that were initially provided and included priority, confirmation status, time to resolution, labels, resolution, number of videos, and video submitted initially or later. The McFadden Adjusted  $R^2$  of the model is 0.32, which means that the regression model is not accurate enough to explain the effect of number of videos on number of back-and-forth. The same model was built for the last nine, six, and three years and the McFadden Adjusted  $R^2$  of the model dropped (see Table 28). The number of videos is a significant factor in the model ( $p < 0.14$ ), but the model itself was inaccurate and thus we cannot interpret or explain the effect of number of number of back-and-forth. Overall, then, this means that other factors than the ones identified are at work.

Table 28. Generalized Linear Model predicting the impact on average back-and-forth by the number of videos included in a Minecraft bug report.

|                 |  | <b>2010-2021</b> | <b>2013-2021</b> | <b>2016-2021</b> | <b>2019-2021</b> |
|-----------------|--|------------------|------------------|------------------|------------------|
| <b>Estimate</b> | <b>McFadden's Adjusted R<sup>2</sup></b> | 0.32             | 0.28             | 0.22             | 0.23             |
|                 | <b>Video submitted</b>                   |                  |                  |                  |                  |
|                 | <b>Initially</b>                         | 2.99             | 2.95             | 2.84             | 1.56             |
|                 | <b>Number of videos</b>                  | 1.83             | 1.92             | 1.95             | 2.58             |
|                 | <b>P-value</b>                           | < 0.14           | < 0.14           | < 0.14           | < 0.14           |

**4.2.2.3.7 Mozilla**

Table 29 shows some of the results of a Generalized Linear Model for Mozilla. The resulting best model used only 14 factors out of the total factors that were initially provided and included component, severity, number of comments, readability score, presence of actual results, presence of expected results, bug reporter's reputation, number of videos, video submitted by developer or end-user, and video submitted initially or later. The predicted value is the average number of back-and-forth. The McFadden Adjusted R<sup>2</sup> of the model is 0.01, which means that this set of factors is insufficient to build an accurate regression model. The same model was built for shorter time windows and the same situation occurred (see Table 29). The model could not be used to interpret or explain the effect of number of videos on number of back-and-forth. Further study will need to discover what other factors besides the 14 factors studied may be causing the difference; for now, the results can only show a statistically significant difference but cannot explain it.

Table 29. Generalized Linear Model predicting the impact on average back-and-forth by the number of videos included in a Mozilla bug report.

|  | <b>2010-<br/>2021</b> | <b>2013-<br/>2021</b> | <b>2016-<br/>2021</b> | <b>2019-<br/>2021</b> |
|--|-----------------------|-----------------------|-----------------------|-----------------------|
| <b>McFadden’s Adjusted R<sup>2</sup></b>     | 0.01                  | 0.02                  | 0.03                  | 0.05                  |
| <b>Number of duplicates</b>                  | -4.381e-03            | -3.992e-03            | -7.872e-03            | -1.370e-02            |
| <b>Number of comments</b>                    | 3.313e-04             | 1.359e-04             | -1.311e-03            | -1.942e-03            |
| <b>Expected results</b>                      | -6.749e-02            | -9.135e-02            | -2.535e-02            | -8.431e-03            |
| <b>Actual results</b>                        | 5.480e-02             | 9.328e-02             | 4.459e-02             | 1.910e-02             |
| <b>Bug reporter's reputation</b>             | 2.079e-04             | -1.910e-02            | -7.651e-03            | -1.150e-02            |
| <b>Developer or end-user<br/>(developer)</b> | -5.176e-03            | -1.039e-02            | -3.824e-03            | 3.656e-02             |
| <b>Readability</b>                           | -3.669e-04            | -3.320e-05            | -1.491e-05            | 8.843e-06             |
| <b>Video submitted<br/>Initially</b>         | 4.057e-01             | 1.791e-02             | 3.994e-02             | 8.998e-03             |
| <b>Number of videos</b>                      | 2.76E-03              | 2.44E-03              | -1.05E-02             | -4.45E-03             |
| <b>P-value</b>                               | < 0.14                | < 0.14                | > 0.14                | > 0.14                |

Estimate

Overall, with the relatively low and limited statistical significance regression coefficient of number of videos, additional evidence would need be gathered to be able to understand the effect as this factor alone is not a major factor for number of back-and-forth (see Section 4.3).

**Observation 10. The impact of including videos on number of back-and-forth is minimal.**

### **4.2.3 A Deeper Dive into Mozilla**

The results thus far were surprising, especially since, as discussed in Section 2.2.6, developers seem appreciative of videos being present and one could therefore perhaps have expected that this might translate into tangible benefits for the reporters along the kinds of effects that were examined. This section, therefore, presents the results of the deep dive that was performed to examine other aspects that potentially could have influenced the findings. More specifically, this section explores whether it matters if videos are submitted by developers on the project or by end-users, whether the type of bug being reported with a video may have an impact, and whether the effect of including videos differs depending on the assigned severity of the bug report. For two factors, role and type, Mozilla bug reports were analyzed, as detailed in Section 4.1.5. The last analysis examines the potential role of the severity that is assigned to bug reports in Mozilla and two other systems, namely Android and LibreOffice.

#### **4.2.3.1 Role of the Video Submitter**

Because developers can perhaps be presumed to be more skilled at writing bug reports for colleagues on the project, they may mostly resort to textual bug reports and only include video when truly necessary. As a result, a different effect may exist for developers versus end-users submitting videos. To study whether the role of video submitter may have an effect on the overall bug resolution process, an analysis was conducted only on Mozilla, since it had all the information about who actually submitted the videos. The information was examined by mapping the reporter, assignee, and others to whether they are developers or end-users using the approach detailed in Section 4.1.5. Such others could be other developers who have an interest, might have some relevant knowledge, or are the owner of relevant code. It could also be end-users other than the reporter. Figure 32 illustrates the resulting counts broken down into these categories.

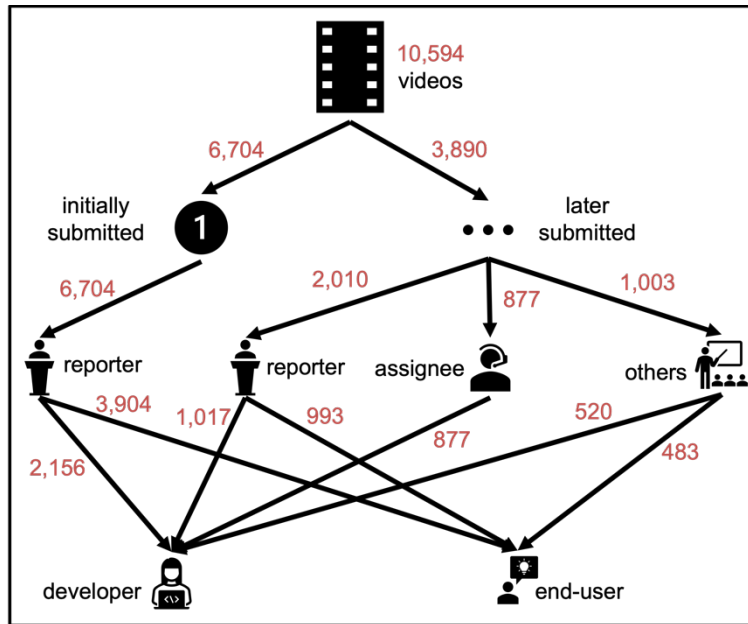
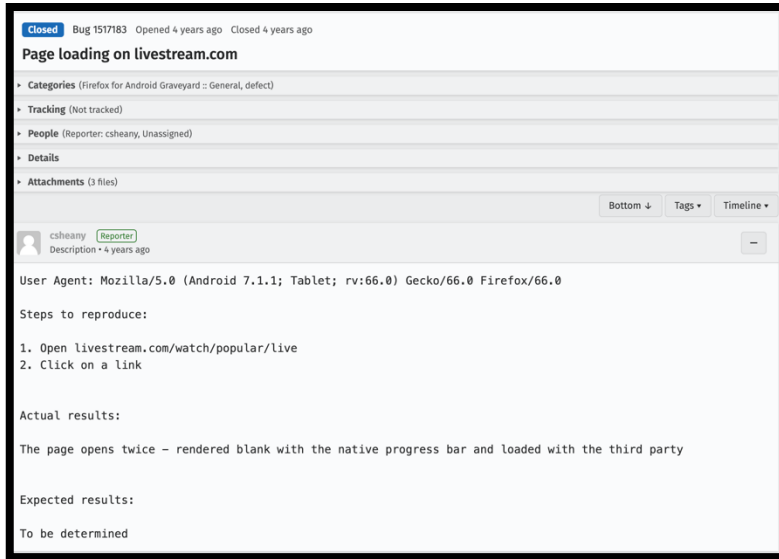


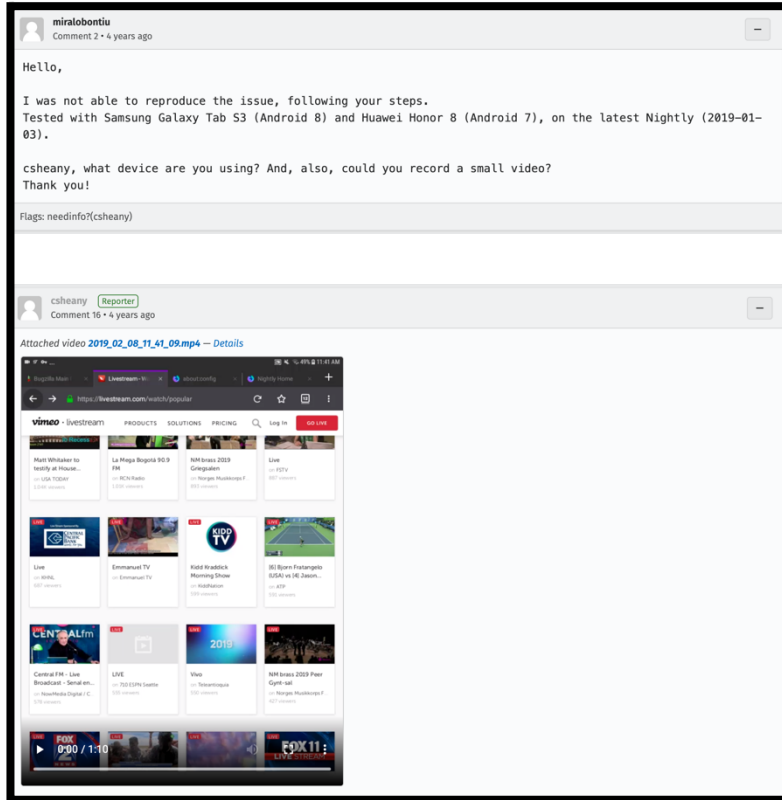
Figure 32. Categorization of videos. Note that “reporter” is listed twice so to distinguish reporters as developers versus end-users for videos that were initially submitted versus later submitted.

The figure shows that the majority of videos, 6,704, were submitted with the initial bug report, as already shown in Figure 26. The remaining 3,890 were submitted after the initial bug report was filed, typically after some back-and-forth between the assigned developer and the reporter. Often, the assigned developer would explicitly ask for a video so that they could obtain additional context that may help them in deciding what to do with the bug report. For instance, Figure 33 (part 1) shows the description of Mozilla bug id 1517183<sup>33</sup>, in which a developer immediately asked the bug reporter to “record a small video”, and the reporter attached a video in response to (Figure 33 (part 2)).

<sup>33</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1517183](https://bugzilla.mozilla.org/show_bug.cgi?id=1517183)



1



2

Figure 33. Example of video submitted later because of developer's request, bug id 1517183 (cropped).

In some cases, however, the reporter would submit the video unsolicited in an effort to bring clarity to the back-and-forth discussion. Bug id 1636769<sup>34</sup> is an example of such situations (Figure 34). Even though it may seem that its description has all the information needed to resolve the bug, two developers had to ask its reporter to provide more information (note: generally, not as a video per se), to which the reporter replied by attaching a video illustrating the needed information (Figure 35).

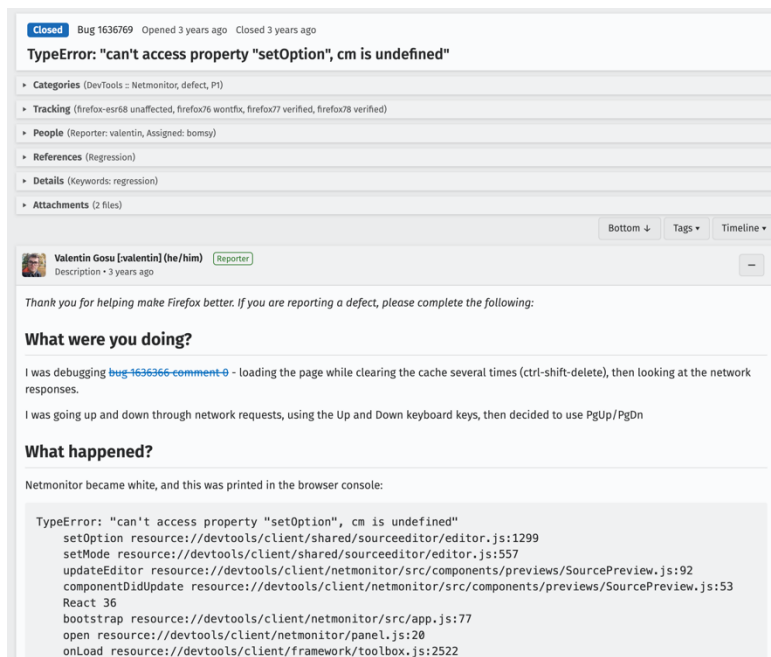
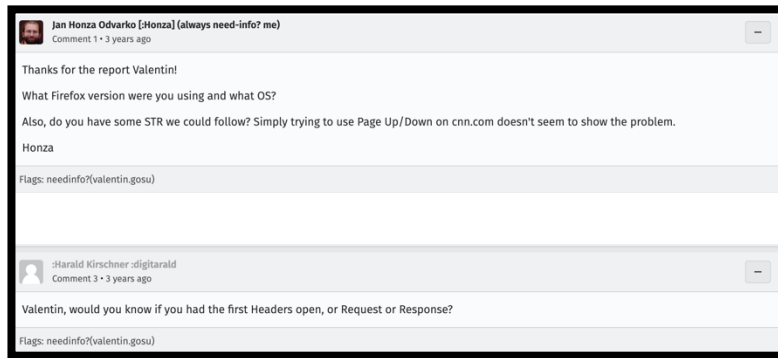
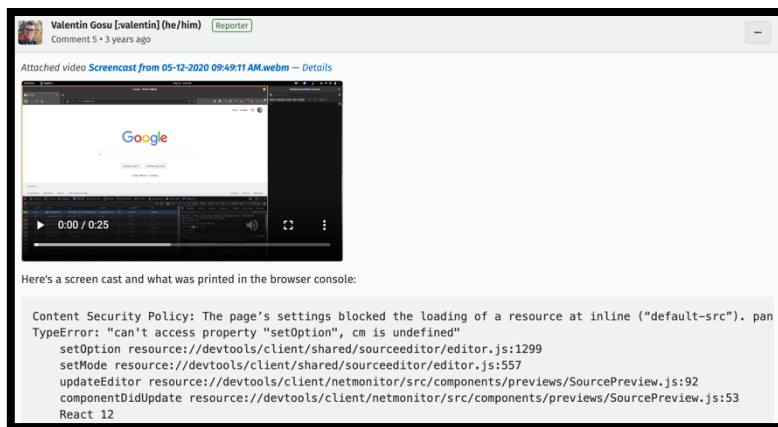


Figure 34. Example of video submitted later voluntarily by bug reporter, bug id 1636769 (cropped).

<sup>34</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1636769](https://bugzilla.mozilla.org/show_bug.cgi?id=1636769)



1



2

Figure 35. Comments from bug id 1636769 (cropped).

Next, the data about who actually submitted the videos was examined, based on the bottom two rows of Figure 32. Clearly, all initially submitted bug reports (6,704) were submitted by reporters. Of the 3,890 videos that were submitted later, 2,010 (51%) were submitted by the reporter, 877 (23%) were submitted by the assigned developer, and another 1,003 (26%) were submitted by others. Mapping the reporter, assignee, and others to whether they are developers or end-users, in total 4,570 videos (45%) were submitted by developers on the projects and 5,380 (55%) by end-users. Note that in a few cases the bug reporter was also the assigned developer for the bug, meaning that the developer themselves filed a bug report that they not only worked on at a later time, but also contributed a video as part of the discussion. In these cases, they were categorized as bug reporters, as a sampling seemed to indicate that often the assignee would post the video for other developers to understand the bug report better.



Potential trends were examined in these distributions over the years. Figure 36 shows the year-by-year distribution of whether later videos were submitted by reporters, assigned developers, or other persons. Using the Mann-Kendall Test together with Sen’s Slope Estimator for the determination of trend and slope magnitude [159], no discernible pattern is detected. Nonetheless, the role of others is interesting: more than a fifth of later submitted videos are submitted by others, consistently.

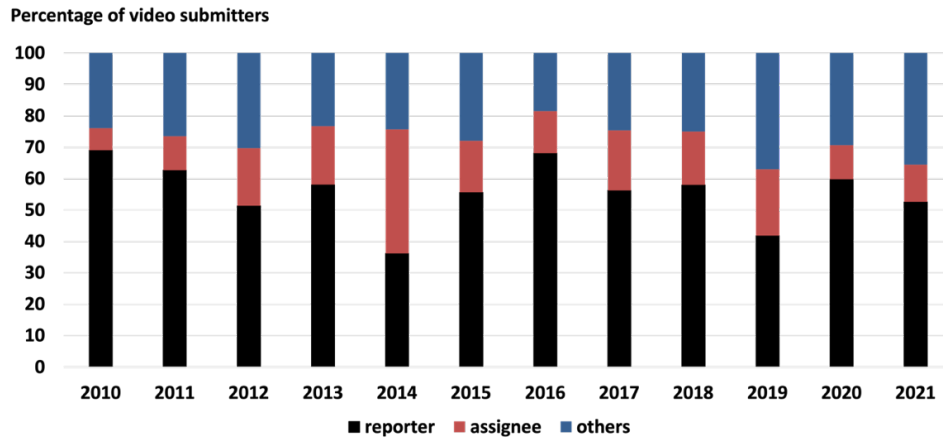


Figure 36. Who attached videos later in the process.

Figure 37 explores this phenomenon in more detail, separating reporters and others from Figure 36 into whether they are developers or end-users. With the refinement, no particular pattern or changes over time are apparent. The distribution of developer versus end-user for reporters fluctuates but does not exhibit a trend (Mann-Kendall trend test, p-value < 0.14); the distribution of developer versus end-user for others similarly equally fluctuates and equally does not exhibit an underlying trend (Mann-Kendall trend test, p-value < 0.14). That said, on the surface one would not expect this many end-users as there is no good reason for other end-users to be monitoring the bug database. That said, one possible explanation might be that this concerns bugs that many people experience and, once they have been reported, others seeking to file a bug report find the existing bug report and add information, including videos.

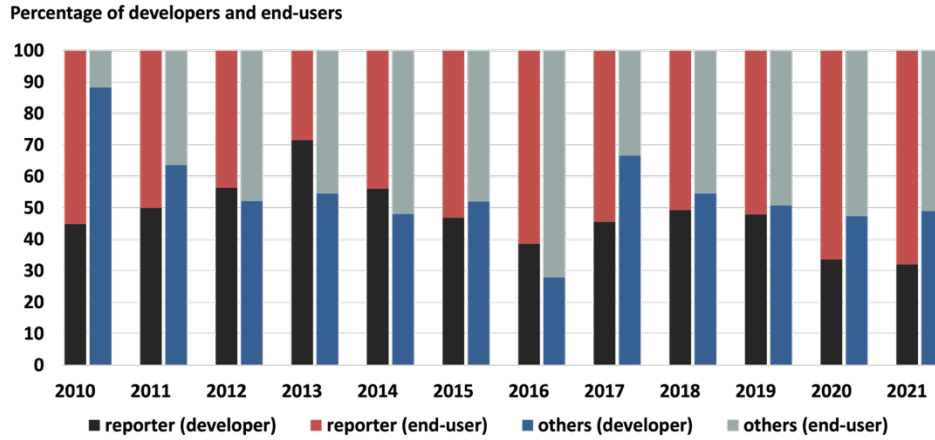


Figure 37. Percentage of developers and end-users who submitted video attachments later in the resolution of bug reports.

Moreover, the role of bug reporters who submitted videos initially, attached to the original bug report, was studied. Figure 38 shows a strong shift (Mann-Kendall trend test,  $p$ -value  $< 0.14$ ) from primarily developers to primarily end-users. This shift may be because early videos by developers may have set an example for end-users to follow, an effect that could be slowly cascading to other end-users. It may also be because video is much easier to capture and submit today than it was a decade ago.

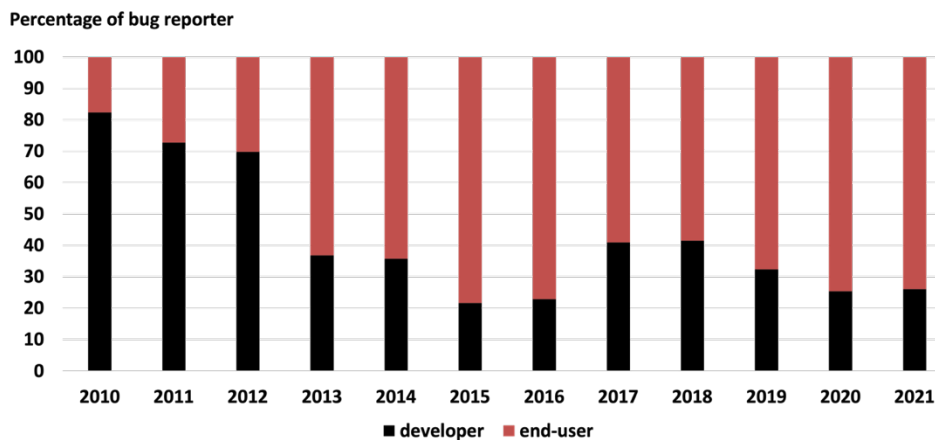


Figure 38. Percentage of bug reporters who are developers or end-users.

Next, the impact of bug reports with videos submitted by developers was analyzed as compared to those with videos submitted by end-users in terms of the three outcomes of

time to resolution, percentage being resolved as fixed, and back-and-forth. Several effects were found. First, bug reports with videos submitted by developers took longer to resolve than those submitted by end-users (comparing the mean); the difference however was not statistically significant (Welch Two Sample t-test,  $t = -0.98$ ,  $df = 18.65$ ,  $p\text{-value} > 0.14$ ).

Second, a higher percentage of bug reports submitted by developers were resolved with a resolution status of FIXED than those by end-users, a statistically significant difference (Welch Two Sample t-test,  $t = 4.11$ ,  $df = 22$ ,  $p\text{-value} < 0.14$ ), with developers having a mean of 58 and end-users 42. Over the last three years, however, the difference becomes smaller and is no longer statistically relevant.

Additionally, the bug reports submitted by developers with videos have less back-and-forth compared to bug reports submitted by end-users. The difference was statistically significant (Welch Two Sample t-test,  $t = 2.75$ ,  $df = 21.56$ ,  $p\text{-value} < 0.14$ ). It gets lower over time, with the difference in the three most recent years being three days (Welch Two Sample t-test,  $t = 6.04$ ,  $df = 3.54$ ,  $p\text{-value} < 0.14$ ).

Overall, these results may point to developers submitting more difficult bug reports when video is included (leading to a longer time to resolution compared to end-users) with higher quality bug reports and/or videos (higher percentage resolved, less back-and-forth). With limited statistical significance, however, additional evidence would need be gathered to be able to conclude this more strongly (see Section 4.3).

**Observation 11. Over the years, a higher percentage of Mozilla bug reports submitted by developers was resolved with a resolution status of FIXED. These bug reports on average also had a longer time to resolution and less back-and-forth, compared to bug reports submitted by end-users.**

### 4.2.3.2 Bug Type

To understand if the type of the bug has an impact on the bug report resolution process, the results from manually labeling Mozilla bug reports were used (see Chapter 5). As detailed in Section 5.1.2, the bugs were categorized into the five categories of User interface, System settings, Failure to load, Functionality, and Access. From the 1,045 labeled videos, videos of potential UI bugs dominated at 69.86%, which was not entirely unexpected given that video is an excellent medium for capturing UI behavior. Therefore, this analysis is focused on this 69.86%, comparing UI bug reports with and without video.

Figure 39 shows the trend for the average number of days to resolve for UI bug reports with and without video. Note that UI bug reports with video take longer to resolve than those without (the difference however is not statistically significant (Welch Two Sample t-test,  $t = 0.68518$ ,  $df = 19.044$ ,  $p\text{-value} > 0.14$ ), which aligns with the general trend discussed in Section 4.2.2.1.

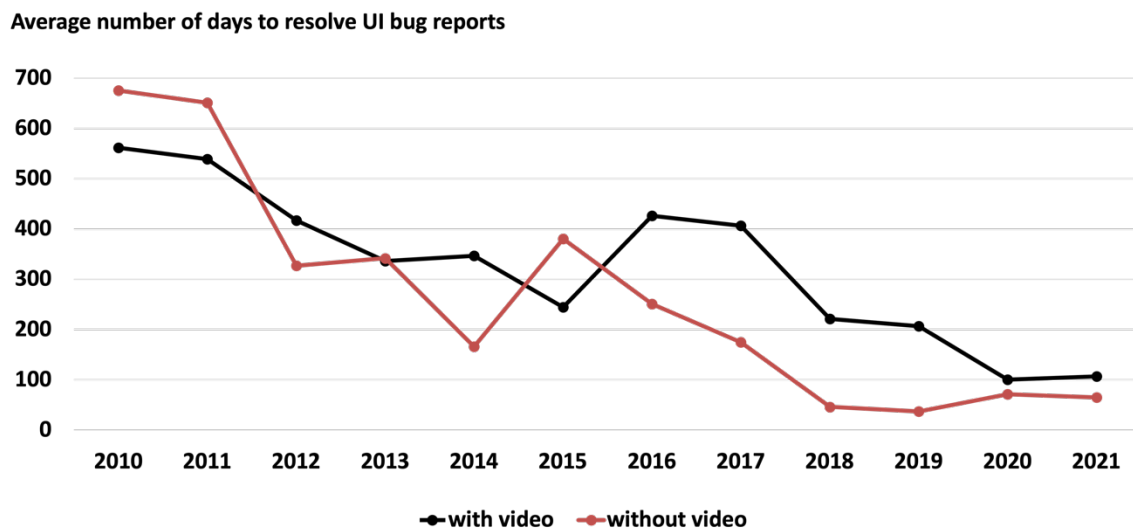


Figure 39. Comparing the average days to resolve of UI bug reports with and without video attachments.

Interestingly, a higher percentage of UI bug reports without video was resolved with a resolution status of FIXED (see Figure 40). This difference is statistically significant (Welch

Two Sample t-test,  $t = -2.8504$ ,  $df = 21.258$ ,  $p\text{-value} < 0.14$ ) and remains so when examining just the last three years (Welch Two Sample t-test,  $t = -2.2785$ ,  $df = 2.5173$ ,  $p\text{-value} < 0.14$ ).

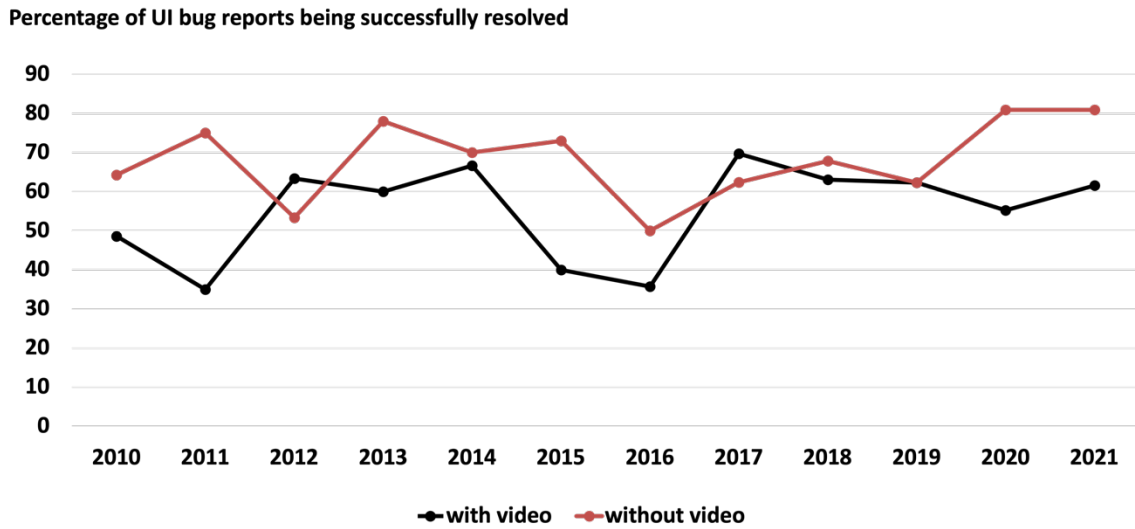


Figure 40. Percentage of UI bug reports with and without video attachments that were successfully resolved with a patch.

Assuming that video is good at illustrating the important parts of UI bugs, the bug reports with videos would have fewer number of back-and-forth and shorter discussion, compared to bug reports without videos. Figure 41 shows that UI bug reports with video incurred less back-and-forth (Welch Two Sample t-test,  $t = 3.0622$ ,  $df = 17.341$ ,  $p\text{-value} < 0.14$ ).

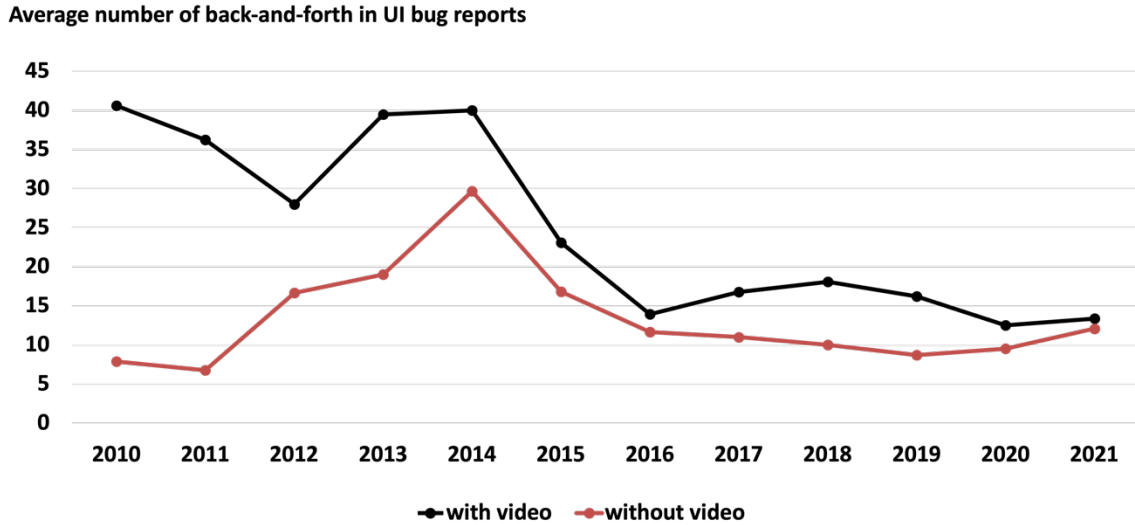


Figure 41. Average number of back-and-forth in UI bug reports with and without video attachments.

A possible explanation for UI bug reports with video being fixed less often might be that the inclusion of video is a signal of the bug being more difficult. At the same time, the lower back-and-forth for UI bug reports with video does not align with this, as one might reasonably assume more difficult bug reports might involve more back-and-forth. Therefore, another possible explanation is that a higher percentage of UI bug reports do not represent actual bugs and thus are closed quickly.

**Observation 12.** Over the years, Mozilla UI bug reports with videos had fewer number of back-and-forth, compared to bug reports without videos.

#### 4.2.3.3 Severity

The last analysis examines the potential role of the severity that is assigned to bug reports: it could be that bug reports with video have a different severity than those without, and this may impact the bug report resolution process because of the implied importance of fixing severe bugs.

### 4.2.3.3.1 Mozilla

As Figure 42 shows, the majority of Mozilla bug reports had a normal severity over the years. Moreover, the severity of bug reports with and without video attachments appears similar over the years. The mean for each level of severity was calculated and the difference between the average number of days to resolve, percentage of bug reports being fixed, and average number of back-and-forth for the bug reports with and without videos was compared. No statistically significant difference was found for any of the possible severity levels.

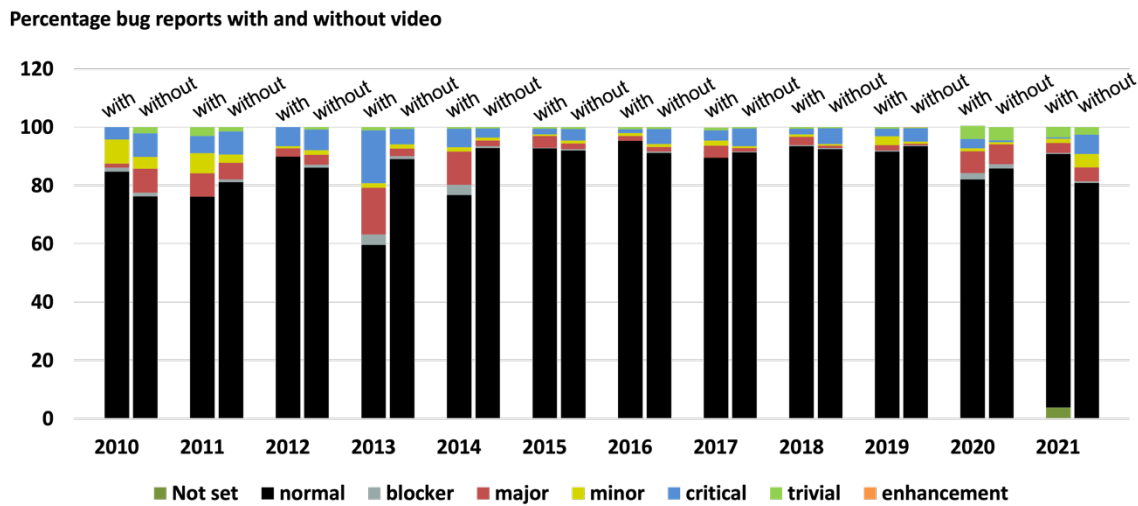


Figure 42. Severity of bug reports with and without video attachments.

The severity levels in Mozilla were grouped based on Least Significant Difference (LSD) test [157], which determines if the means of the groups are statistically different, and obtained three severity groups: high severity (group 1: blocker, critical, and major), low severity (group 2: normal, trivial, minor, Not set), and request for enhancement (group 3: enhancement).

The bug reports were analyzed in group 1, comparing those with video to those without. The results showed that a higher percentage of high severity bug reports with video was fixed (not statistically significant over the last three years), that they were resolved slower (180.19 days compared to 111.44, Welch Two Sample t-test,  $t = -1.69$ ,  $df = 17.23$ ,  $p\text{-value} < 0.14$ ), and that they involved less back-and-forth (statistically significant even over the last three years, Welch Two Sample t-test,  $t = -4.30$ ,  $df = 115.36$ ,  $p\text{-value} < 0.14$ ).

This might perhaps be the strongest evidence that videos could help, given that all three outcomes improved and that severity itself is not a factor in this improvement. This might mean that the reporters of severe bug reports know how to convey the bug clearly, so it is swiftly understood and dealt with. In the context of the earlier results that showed no conclusive evidence, it is possible that the noise of the many other bug reports that are not high severity obfuscates this effect.

**Observation 13. Over the years, a higher percentage of high severity Mozilla bug reports with video was resolved with a status of FIXED, compared to the ones without video. Moreover, high severity bug reports with video took a smaller number of days to resolve and had fewer number of back-and-forth.**

#### **4.2.3.3.2 Android and LibreOffice**

To understand if video is helpful for critical bugs in other systems as well, the same analysis was repeated for the two other systems: Android and LibreOffice. Since the severity levels were not publicly available for IntelliJ (e.g., bug id IDEA-307583<sup>35</sup>) and Minecraft bug reports (e.g., bug id MC-258893<sup>36</sup>), they were not included in the further analyses.

LibreOffice bug reports were grouped into high severity (group 1: blocker, critical, and major), low severity (group 2: normal, trivial, minor), and request for enhancement (group 3: enhancement). The bug reports that were analyzed were in group 1, comparing those with video to those without. The results showed that over the years, for the group of bug reports with high severity, a higher percentage with video was fixed (not statistically significant), that they took a greater number of days to resolve (not statistically significant), and that they involved a higher number of back-and-forth (statistically significant, Welch Two Sample t-test  $t = 4.81$ ,  $df = 21.9$ ,  $p\text{-value} < 0.14$ ).

---

<sup>35</sup> <https://youtrack.jetbrains.com/issue/IDEA-307583>

<sup>36</sup> <https://bugs.mojang.com/projects/MC/issues/MC-258893>



The severity levels of Android were grouped into high severity (group 1: critical, and high) and low severity (group 2: moderate, low, negligible security impact). The results showed that over the years, bug reports of high severity and with video took longer to resolve (not statistically significant), a higher percentage was fixed (not statistically significant), and involved a higher number of back-and-forth than those with high severity and no video (not statistically significant).

It is interesting to contrast these results with the main results of Mozilla bug reports with high severity. For bugs of high severity, the inclusion of video is a helpful practice in Mozilla bug reports, as compared to Android and LibreOffice where it is not. It is unclear what may be the reason, other than possibly bug reports with video in Android (operating system) and LibreOffice (office suite) may be different kinds of bugs. For example, LibreOffice may have more UI bugs which may be less of a severity.

**Observation 14. Over the years, comparing bug reports of high severity with video and without video, only three of the results were statistically significant: bug reports of high severity with video in Mozilla had a slower time to resolution and less back-and-forth, and in LibreOffice had more back-and-forth. That means, severity may play a small role in individual systems, but overall does not appear to affect the non-effects that were seen.**

### 4.3 Discussion

This chapter began by citing two common beliefs surrounding the inclusion of videos in bug reports: that videos offer an opportunity to share context-rich bug information with developers [36], [39], [42] and that, in doing so, they help developers in understanding users' interactions, see the actual behavior of the system in response to some input, and internalize what may have contributed to the manifestation of the bug [35], [140]–[142]. This study does not question this belief, but asks a complementary question: does the inclusion of videos in bug reports lead to other, externally observable effects in terms of the bug resolution process that may offer further benefits? With the clear increase in video attachments over the past

10 years (Figure 25), but the overall percentage of bug reports with videos still remaining low (just 2.43%), it is particularly important to understand if any additional incentive could make a difference and bring the percentage up, since they could provide an important incentive for bug reporters to include videos if they themselves would experience tangible benefits.

In this context, the results answer three research questions:

- *RQ1:* Does including videos in bug reports lead to a reduction in the average time to resolution? The results showed that including videos incurs on average a longer time to resolution. The difference has shrunk considerably, however, and over the last six years the impact has become minimal. This holds true when videos are submitted as part of the initial bug report submission and when they are submitted at a later time.
- *RQ2:* Does including videos in bug reports lead to a higher percentage of bug reports being resolved with a patch, that is, actually fixed? Mixed results were found: while for some systems a higher percentage of bug reports ends up being fixed if a video is included, for other systems the opposite occurs in fewer bug reports being fixed if videos are included.
- *RQ3:* Does including videos in bug reports lead to a reduction in the average back-and-forth that occurs once the report has been submitted? On average bug reports with videos incur a statistically significant higher average back-and-forth than bug reports without. For bug reports that are submitted later, even if only the back-and-forth after the submitted video is counted, a statistically significant difference remains in the back-and-forth still being higher as compared to bug reports without videos.

Taken together, from the perspective of perhaps providing motivation for reporters to produce and submit videos along with their bug reports, these findings suggest that the only tangible incentive that exists is an increase in the potential for the bug report to be fixed for two of the systems (Android and Minecraft). For the other systems, no tangible incentive seems to exist and, even for Android and Minecraft, the potential for the bug report to be

fixed comes with the potential drawback of a slower resolution process and more back-and-forth.

These results should not be interpreted as discouraging submission of videos as part of bug reports. The altruistic motivation of making the work of bug assignees easier remains as important as ever since bug triaging, diagnosing, and fixing is often still a difficult and thankless process. Any small bit of help will be appreciated. If anything, the results give rise to a number of important research questions.

First, beyond the factors we explored in Section 4.2.3, other factors should be studied. Section 4.2 already began to discuss the potential role of bug difficulty: could the non-benefit seen in the results in actuality be an overlap of two effects: a benefit along the lines that were studied that is negated by videos typically being included for bugs that are more difficult to resolve? This probably requires an entire study in and of itself: how is difficulty determined, is it a binary factor or more nuanced, can it be reliably classified based on bug report and code changes, and is it possible to perform an analysis at scale? The current best classifier achieves an AUC of 0.612 [160], which would be too low for an automated analysis.

Second, is the question of what sets aside videos that are more effective than others in reducing time to resolution, leading to an actual fix, or reducing back-and-forth (ideally all three)? The study reports on averages, which means that some videos do contribute to beneficial effects and others detract. The deep dive into the bug types of Mozilla represents a first, small step into whether the content of videos might provide clues as to useful differences. The look at bug types, and particularly at user interface related bugs, was a reasonable place to start in this regard, since user interface bugs are visual in nature and thus fit videos well. Yet, the absence of a meaningful difference implies that bug type may not be a determinant. On the other hand, the fact that Mozilla bug reports with videos submitted by developers are fixed at a higher rate and lead to less back-and-forth is a potentially promising starting point. And, perhaps most important, the fact that, within the class of high severity Mozilla bug reports, videos improve on all three outcomes seems to present a first clear sense that videos actually might be serving an important role and impact externally visible outcomes. To truly sort this out, a deep empirical study is needed, in which the content of videos is juxtaposed with the written part of bug reports, the subsequent

discussions, the nature of the bugs, and the kinds of fixes that were involved. Among others, factors to study are whether videos are concise, show all the steps leading up to the bug manifestation, guide the developer via clear highlighting by mouse or via annotations, and contain useful voice-overs (see Chapter 5).

Third, it is important to perform a field study with developers to study how they process videos as part of bug report resolution. Such a study has the same goal as the empirical study outlined in the prior point: to uncover factors that help certain videos be more effective than others. A field study would observe developers at work and talk with them about their experiences and perceptions, offering an important complementary perspective.

A fourth important direction for research is to broaden the study beyond the five systems that were studied. It might be that some of the individual differences are related to the domain of the systems studied. Video games are a particularly interesting example in this regard, since they are highly visual. Including additional domains and multiple systems per domain would help to identify whether such domain-driven differences exist.

#### **4.4 Threats to Validity**

While the study is structured so to avoid introducing bias and worked to eliminate the effects of random noise, it remains possible that the mitigation strategies may not have been effective. This section reviews the threats to validity.

Given that bug reports from five open source systems were examined, the findings may not generalize broadly. It is possible that the systems studied are outliers in the free and open-source software (FOSS) community because of their size or other properties. That said, several of these systems are frequently studied in the literature and a system such as Mozilla is often used as an exemplar, or role model, for other FOSS projects. Replication of the study to other systems and particularly to commercial projects is imperative.

A threat to the internal validity of the study is the possibility of faults in the Python code that was implemented to perform web scraping and crawling. This threat was addressed by extensively testing the implementation and verifying its results against a smaller dataset for which the correct results were manually determined.

The choice to only include resolved bug reports to study the impact of videos in bug resolution may represent a threat to internal validity if the nature of the content of these bug reports differs significantly from those of unresolved bug reports. The results may also be affected if any of the unresolved bugs are reopened in the future. Fortunately, the bug report repositories of the five systems studied are of large size and have long histories, meaning that what used to be unresolved bug reports have moved on to become resolved. Moreover, the results were analyzed in shorter time windows to study the effect of the bug reports that were reopened recently. As such, this threat is believed to be fairly minimal.

For one of the Mozilla-specific analyses, the bug reporters were categorized into end-users and developers. For this categorization, the number of patches, that were previously submitted by each user ID to Mozilla projects were extracted. If the number was zero, the submitter was identified as an end-user, otherwise as a developer. This may have led to some misclassifications, for instance, if a developer used multiple user IDs, one for development and one for submitting reports as an actual end-user of the product.

For Mozilla, the presence of the flag “Flags: needinfo?(reporter’s email)” was used to calculate the number of back-and-forth, and similar heuristics for the other systems (see Section 4.1.3). However, it is possible that developers did not use the flag when replying to a reporter, and this may have resulted in under-counting back-and-forth. Given that the outcomes of the back-and-forth analyses showed increases in back-and-forth as compared to bug reports without videos, such potential under-counting would not alter the nature of the observations, but merely increase the found effects.

## **4.5 Conclusions**

This chapter contributes a first empirical study that examines whether externally observable effects exist in the bug report resolution process that can offer a potential incentive for reporters to submit videos along with their bug reports. That is whether including videos in bug reports could have potential benefits to the reporters of those bug reports. The chapter specifically studies whether the inclusion of video in bug reports impacts the bug resolution

process in terms of time to resolution, resolution with a patch aiming to fix the reported bug, and the amount of back-and-forth.

The primary finding is that, on the whole, the inclusion of videos does not appear to have a positive impact, except in the case of Android and Minecraft, where a higher percentage of bug reports with videos is resolved with a patch, and thus fixed, as compared to bug reports without videos. Otherwise, time to resolution is barely impacted, no discernible differences exist in the percentage of bug reports being fixed for the other three systems studied, and the average amount of back-and-forth is higher for bug reports that include video than those that do not across all five systems. Yet, a deep dive into a select set of potential factors for Mozilla shows that further study is needed. Especially the fact that bug reports with videos submitted by developers are resolved at a higher rate with less back-and-forth implies that studying the contents of those videos can provide important clues as to the desirable properties of videos attached to bug reports.

## 5 CHAPTER 5: A Content-Based Analysis of Video Submissions in Bug Reporting

A recent trend is for bug reports to include videos as attachments. Videos can easily demonstrate complex contexts about bugs which offers developers a new opportunity to collect context-rich bug information [36]. The interpretation of videos provides a different perspective to help developers understand how users interact with the system, process the current behavior of the system, or comprehend any events that contributed to the bug, which might have been hard to capture using words [39].

Several studies encourage reporters to submit relevant videos along with their bug reports to convey additional context for understanding bugs [40], [41], [111]. To assist developers in effectively taking in the content of videos, approaches have been introduced for automatically analyzing videos [37], [38] and turning those videos into replayable scenarios that are easier to understand and use in bug resolution [46], [47].

This chapter complements the existing work on videos and bug reports to date with an important new perspective: what sets videos that are more effective apart from videos that are less effective, in terms of their content? It specifically focuses on the detailed characteristics of videos, by which I mean things such as whether the video shows steps to reproduce, whether the video contains a voice over, and whether the creator of video highlights the bug with mouse movement. Examining the video characteristics refines the analysis from Chapter 4, which only looked at the presence of videos, with a focus on the content of videos.

The study first examines the contents of 1,045 videos and their associated bug reports from Mozilla projects (a qualitative approach), examining different characteristics of videos. It then, in a quantitative approach, assesses the impact of videos that contain various video characteristics in comparison to those that do not. Particularly, this study examines how developers may react publicly to various characteristics of videos attached to bug reports (whether they appear to be perceived as helpful or not helpful by developers) and whether those characteristics may have observable effects on the bug report resolution process (i.e., in reducing time to resolution, leading to an actual fix, or reducing back-and-forth).

This chapter also includes a survey that was conducted to understand how developers feel about different characteristics of videos as a part of bug reports. The survey was centered around three main questions to collect developers' perspectives on: (1) how they characterize the content of the video attachments in relation to the corresponding textual descriptions; (2) what are the kinds of video content that they believe help certain videos be more or less useful in understanding bugs; and (3) whether the inclusion of video, in their opinion, is effective in reducing time to resolution, leading to an actual fix, or reducing back-and-forth.

In the remainder of this chapter, I detail the methodology and report the results. I then present the analysis of the results, describe the threats to validity, and then conclude with a summary of the key findings and future work.

## **5.1 Methodology**

The goal is to understand how developers may react to various characteristics of videos attached to bug reports and whether video characteristics may have beneficial effects on the bug report resolution process (i.e., in reducing time to resolution, leading to an actual fix, or reducing back-and-forth). In order to do so, 1,045 videos and their associated bug reports from Mozilla projects were randomly selected. The videos were labeled by two researchers to get a better understanding of their content. Next, to form the basis of the analysis, all the available metadata of the videos and bug reports were extracted from Chapter 4. Figure 43 illustrates the overall process that was used to study the impact of including different video characteristics on the videos appearing to be perceived as helpful by developers and the bug report resolution process.



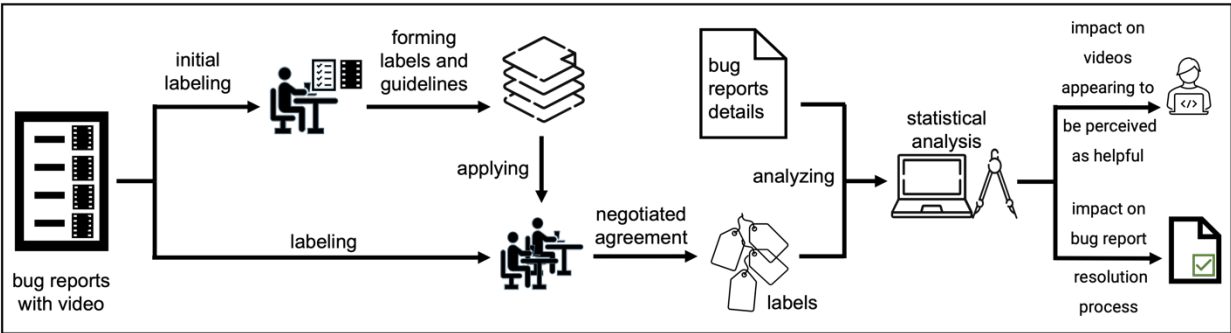


Figure 43. The overall process of studying the role of different characteristics of videos on videos appearing to be perceived as helpful by developers and bug report resolution process.

### 5.1.1 Data Collection

From 10,594 Mozilla videos that were collected in Chapter 4, 1,045 videos were randomly selected (around 10%). The experiments were performed on Mozilla since it is a large-scale open-source project on multiple platforms, and has been widely used in empirical software engineering research [123]. The dataset spans from 2010 to 2021 and only contains bug reports with video attachments and resolution status of RESOLVED, VERIFIED, and CLOSED. Sample details related to bug reports in the dataset include creation year, number of days to resolve, component, priority, product, severity, number of comments, number of back-and-forth, readability score, number of videos, and presence of steps to reproduce, actual results, and expected results.

### 5.1.2 Manual Labeling

Two researchers manually labeled the content of 1,045 videos and their associated bug reports. For labeling purposes, the researchers analyzed the characteristics of the videos, description of the bug reports (if present), associated comments from discussions with other developers, and any interactions between the developer and the reporter.

The manual classification was conducted in three phases. In the first phase, researcher one, through repeated examining of the bug reports and watching the videos collected for the

study, identified a set of video characteristics found numerous times in the dataset and created a set of labels, following an open coding approach. During this phase, the labels were continuously compared to contradict, expand upon, or support the existing labels. Next, 100 bug reports were randomly selected and independently labeled by each of the two researchers according to the defined labels. Then the researchers went through all the videos and associated bug reports, compared, and discussed disagreements in order to reconcile and arrive at a final version of the labels.

In the second phase, using the “negotiated agreement” method [124], both researchers then independently labeled 300 bug reports (including the first 100) using the set of labels, achieving an inter-rater reliability Cohen's Kappa of 0.87, which indicates “almost perfect agreement” [127]. Both researchers together then reviewed the labels that were not in agreement, and for each bug report discussed their respective reasoning and the source of disagreement, which mostly led to resolution. At this point, the researchers felt sufficiently confident to move to the third phase to label the remaining 745 bug reports, 373 by one researcher and 372 by the other.

The following is the set of labels and guidelines that emerged during the labeling of the videos:

- 1- Bug details (Content): what aspect of the bug does the video show? This can include:
  - Steps to reproduce. The video shows the steps on how to reproduce the bug
  - Actual results. The video shows the occurrence of the bug
  - Expected results. The video shows what the application should have done if the bug was not present
  - Platform. The video shows OS, version of application, component, etc.
  - Workaround. The video shows how to work around the bug
  - Solution. The video shows how to resolve the bug
  - Crash report. The video shows the crash report produced when the bug happened

- Verified as fixed. The video shows that the bug has been fixed now
- Cannot reproduce. The video shows that the bug cannot be reproduced
- Occurrence of a new bug. The video shows another bug

2- Bug type (Category): what aspects of the software does the video illustrate? This can include:

- User interface. The video shows problems related to the user interface
- System settings. The video shows problems related to the configuration settings
- Access. The video shows problems with functionality for logging in or signing up
- Failure to load. The video shows problems related to the software not being able to start
- Functionality. The video shows problems related to the performance of the back-end (UI is behaving correctly, but the underlying software produces faulty results)

3- Sound: whether the video includes any sound. This can include:

- Noise. The video includes mouse clicks and background noise that are not related to the bug
- Voiceover. The reporter explains the bug in video

4- Annotation: whether the video is annotated or highlighted. This can include:

- Pointer movement. The video shows a pointer going back and forth or circulating
- Highlighting with the cursor. The video shows highlighting like **this**
- Pointer amplified or highlighted. The video shows any kind of change to the cursor's appearance to amplify the location of the cursor on the screen

- Labels. The video shows labels such as text boxes to add more information about the events happening in different parts of the video
  - Pointing with a finger. The video shows the steps to reproduce through touch or moving of fingers – in videos in mobile applications and videos taken by another device
- 5- Length: what is the length of the video in seconds?
- 6- Repetition: how many times does the reporter show the bug in the video? This can be:
- Once
  - More than once
- 7- Reception: in the comment section of the bug report, what was the immediate reaction to the video?
- Perceived helpful. The video appears to be perceived as helpful by a developer. Several indicators are used. One indicator is that right after the video is submitted in the comment section a developer specifically states that the video is helpful and goes on to fixing the bug (e.g., bug id 1383406<sup>37</sup>). Another indicator is that right after the video was submitted, the bug report gets resolved or a patch is submitted (which means the bug is fixed), even if the developer does not specifically comment that the video is helpful; or the thread(s) of discussion is not about the video and is about the ways to resolve the bug (e.g., bug id 1546326<sup>38</sup>), indicating the discussion has shifted because the bug report is understood.
  - Reproducible. The video is effective in enabling a developer to reproduce the bug. For example, a developer comments: "I confirm that the bug is reproducible." (e.g., bug id 1518050<sup>39</sup>)

---

<sup>37</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1383406](https://bugzilla.mozilla.org/show_bug.cgi?id=1383406)

<sup>38</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1546326](https://bugzilla.mozilla.org/show_bug.cgi?id=1546326)

<sup>39</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1518050](https://bugzilla.mozilla.org/show_bug.cgi?id=1518050)

- Non-reproducible. The video is not effective in enabling a developer to reproduce the bug. For example, a developer states: "still cannot reproduce the bug". (e.g., bug id 1612460<sup>40</sup>)
- Question about video. The video triggers a question about the video itself by a developer. For example, the developer comments: "how can I play this screencast?", "how it was produced?", "when did you create this video?", etc. (e.g., bug id 1576990<sup>41</sup>)
- Question about bug. The video triggers questions about the bug and its nature by a developer. For example, right after the video was submitted, the developer comments: "do you see that with latest nightly? Please also try different backends (WebRedner/Basic compositors)." (e.g., bug id 633108<sup>42</sup>)

8- Change in activity: in the activity section of the bug report, was there any immediate change in the status (lifecycle of bug report) that was caused by the video submission.

This can be a change in:

- Status (e.g., bug id 1517183<sup>43</sup>)
- Resolution (e.g., bug id 1517548<sup>44</sup>)
- Assignee (e.g., bug id 1517805<sup>45</sup>)
- Priority (e.g., bug id 1523844<sup>46</sup>)
- Severity (e.g., bug id 1633246<sup>47</sup>)
- Component (e.g., bug id 1607810<sup>48</sup>)

9- Flags: in the activity section of the bug report, was any flag raised immediately after video submission?

---

<sup>40</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1612460](https://bugzilla.mozilla.org/show_bug.cgi?id=1612460)

<sup>41</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1576990](https://bugzilla.mozilla.org/show_bug.cgi?id=1576990)

<sup>42</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=633108](https://bugzilla.mozilla.org/show_bug.cgi?id=633108)

<sup>43</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1517183](https://bugzilla.mozilla.org/show_bug.cgi?id=1517183)

<sup>44</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1517548](https://bugzilla.mozilla.org/show_bug.cgi?id=1517548)

<sup>45</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1517805](https://bugzilla.mozilla.org/show_bug.cgi?id=1517805)

<sup>46</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1523844](https://bugzilla.mozilla.org/show_bug.cgi?id=1523844)

<sup>47</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1633246](https://bugzilla.mozilla.org/show_bug.cgi?id=1633246)

<sup>48</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1607810](https://bugzilla.mozilla.org/show_bug.cgi?id=1607810)

- Flags that confirm the effectiveness of video. For example, “Flags: ever confirmed: true”
- Flags that request for information from the video submitter. For example, “Flags: needinfo?(*email of the video submitter*)”

### 5.1.3 Statistical Analysis

The goal is to determine the impact of the various characteristics of videos on the bug report resolutions process (bug report resolution time, percentage of bug reports being successfully resolved as FIXED, and the back-and-forth discussion following a bug report submission) and percentage of videos appearing to be perceived as helpful by developers. To do so, the Welch Two Sample t-test was conducted between subsets of videos that contain various characteristics of videos and the ones that do not. Since multiple tests were performed, the significance value was adjusted accordingly to account for multiple hypothesis corrections. Using the Benjamini–Hochberg correction [149] the p-value was adjusted to 0.06.

Since the t-tests only identify potential correlations, Generalized Linear Regression Models (GLM) [150] were built for four dependent variables: videos appearing to be perceived as helpful by developers, time to resolution, bug reports resolved as FIXED, and the number of back-and-forth. The dependent variables follow a Poisson distribution. Therefore, a Poisson regression model was used with a log linking function.

In order to build the model, all the available metadata of the videos and bug reports were used from Chapter 4 (Section 4.1.1) including severity, operating system, priority, product, component, status, platform, resolution, version, cc count, votes, number of duplicates, number of comments, number of unique participants, number of attachments, number of developers, number of unique words in title and bug description, and readability score (Flesch score [130]). To this set, the results from manual labeling were also added, such as the presence of actual results in the video, the presence of steps to reproduce in the video, annotation, length of the video, video submitted initially, and video submitted voluntarily later.

After collecting these metrics, multi-collinearity was checked using the Variance Inflation Factor (VIF) [150] of each predictor in the model. VIF describes the level of multi-collinearity (correlation between predictors). A VIF score between 1 and 5 indicates a moderate correlation with other factors, so the predictors with a VIF score threshold of 5 were selected. This step was necessary since the presence of highly correlated factors forces the estimated regression coefficient of one variable to depend on other predictor variables that are included in the model.

Next, using the selected factors, the best model was identified through Akaike Information Criterion (AIC), which estimates the information loss between models in comparison to the original. It ultimately selects the best model based on both the fit of the model and the information lost. Then, the model identified by AIC as the final model was identified. This process was repeated independently for each of the four regression models.

#### **5.1.4 Survey**

In order to understand developers' experiences with and perceptions of different characteristics of the video attachments, I conducted a survey among IntelliJ developers. The survey was centered around three main questions: (1) how they characterize the content of the video attachments in relation to the corresponding textual descriptions; (2) what are the kinds of video content that they believe help certain videos be more or less useful in understanding bugs; and (3) whether the inclusion of video, in their opinion, is effective in reducing time to resolution, leading to an actual fix, or reducing back-and-forth.

The survey was sent to developers who were listed as 'assignee' for IntelliJ bug reports with videos that were submitted (and assigned) in the year 2021. Given that assignees can repeat, ultimately, out of the 469 bug reports with video, the survey was sent to 17 unique developers. Participation was voluntary and participants were allowed to discontinue at any time; participants did have to consent to participate. I received 9 responses (52.9% response rate) from developers with the highest 30 to lowest 5 assignments to bug reports with videos.

## 5.2 Results

In this section, the results of the study are reported and discussed in four stages: (1) what characteristics do the videos that are attached to bug reports have, (2) how different characteristics of videos may have impact on the videos appearing to be perceived as helpful by developers, (3) how they may have observable effects on the bug report resolution process (i.e., in reducing time to resolution, leading to an actual fix, or reducing back-and-forth), and (4) what developers who were surveyed actually think about different characteristics of videos as a part of bug reports.

### 5.2.1 Video Characteristics

As discussed in Section 5.1.2, 1,045 videos and their associated bug reports were manually labeled to specifically examine different characteristics of videos (e.g., whether the video is annotated, what aspects of the software the video illustrates) and to understand how developers publicly reacted to various characteristics of videos. In the following, I explain the results.

**Bug details (Content).** Table 30 presents the count and percentage of the videos showing different categories of bug details. In the majority of cases (88.04%), the videos were minimal and focused on showing just what went wrong (actual results). Almost half of the videos (51.87%) showed the steps to reproduce the bug. The percentage of the rest of the categories dropped significantly to only around 3% of videos showing expected results, platform, workaround, and solution and just 0.1% showing a crash report. The combinations of the labels for bug details were also studied. 47.65% of videos included both actual results and steps to reproduce. 40.47% of videos showed only the actual results and not the steps to reproduce the bug; and around 4% of videos only illustrated the steps to reproduce and not the actual results. Only 1% of videos showed actual results, steps to reproduce, and expected results; the three most important attributes in high quality textual bug reports.



Table 30. Count and percentage of the videos that contained different bug details.

| Bug details (content)   | Count | Percentage |
|-------------------------|-------|------------|
| Actual results          | 920   | 88.04      |
| Steps to reproduce      | 542   | 51.87      |
| Expected results        | 40    | 3.83       |
| Platform                | 38    | 3.64       |
| Workaround              | 34    | 3.25       |
| Solution                | 34    | 3.25       |
| Verified as fixed       | 22    | 2.11       |
| Occurrence of a new bug | 18    | 1.72       |
| Cannot reproduce        | 17    | 1.63       |
| Crash report            | 2     | 0.19       |

**Bug type (Category).** The categories user interface and functionality dominated (69.86% and 21.72%, respectively). This was not unexpected as sometimes it is easier to show what is wrong than attempting to write it in text. The other categories, however, appeared not as frequent. For example, 2.78% of the videos showed problems with system settings, which typically are textual and can be easily copied and pasted into a bug report. Also, 0.38% of the problems were about access and 2.78% were about failure to load.

**Sound.** Meaningful audio was absent in most of the videos (97.13%): 92.82% did not have any sound and 4.31% had what appeared to be accidental background noise that was clearly audible, but no meaningful audio otherwise (e.g., bug id 1428870<sup>49</sup>). One could assume it would be relatively easy and a low hurdle to either directly narrate while a video is being

<sup>49</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1428870](https://bugzilla.mozilla.org/show_bug.cgi?id=1428870)

recorded that shows a bug, or even to overlay audio at a later time. However, as Table 31 shows, only 2.87% of the videos had a voiceover describing what was being shown (e.g., bug id 1029058<sup>50</sup>).

Table 31. Count and percentage of the videos with different type of sounds.

| Sound     | Count | Percentage |
|-----------|-------|------------|
| Noise     | 45    | 4.31       |
| Voiceover | 30    | 2.87       |

**Annotation.** 37.8% of the videos included annotations, broken down into pointer movement (18.18%), highlighting with the cursor (8.23%), pointer amplified (11.39%), adding labels (1.05%), and pointing with a finger (2.68%). Other than only a few, none of the videos included a combination of annotation categories. The combination of pointer movement and pointer amplified represented the most frequent combination of individual factors, but amounted to only 4.21% (e.g., 1298011<sup>51</sup>). The next highest was 1% for the combination of pointer movement and highlighting with the cursor (e.g., 1346164<sup>52</sup>).

**Length.** As Table 32 shows, the majority of videos (78.08%) were less than 30 seconds. That is followed by only 17.99% of videos having a length between 30 and 60 seconds and only 3.92% being longer than 60 seconds. Overall, the average length of videos was 21.3 seconds. Bug id 1250866<sup>53</sup> had the longest in length of all videos (136 seconds), which shows the actual result and steps to reproduce the bug more than once and includes a combination of pointer movement and pointer amplified for annotation. The shortest video (1 second) was included in bug id 1310293<sup>54</sup> which seemed to be only attached to the bug report to represent a video format (and not to show the bug).

<sup>50</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1029058](https://bugzilla.mozilla.org/show_bug.cgi?id=1029058)

<sup>51</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1298011](https://bugzilla.mozilla.org/show_bug.cgi?id=1298011)

<sup>52</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1346164](https://bugzilla.mozilla.org/show_bug.cgi?id=1346164)

<sup>53</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1250866](https://bugzilla.mozilla.org/show_bug.cgi?id=1250866)

<sup>54</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1310293](https://bugzilla.mozilla.org/show_bug.cgi?id=1310293)

Table 32. Count and percentage of the videos with different lengths.

| Length                | Count | Percentage |
|-----------------------|-------|------------|
| Less than 30 sec      | 816   | 78.08      |
| Between 30 and 60 sec | 188   | 17.99      |
| More than 60 sec      | 41    | 3.92       |

**Repetition.** In terms of how many times the reporter shows the bug in the videos, 36.93% of videos showed the bug more than once. Bug id 974643<sup>55</sup> represents an example and is furthermore interesting as the video was not a screen recording, as most of the videos were, but instead was filmed using an external device.

**Reception.** In terms of how the developers publicly reacted to videos, 48.9% of the videos appeared to be perceived as helpful, 22% of the videos enabled the developers to reproduce the bug, though 8.33% was ineffective to do so, 10.81% triggered a question about the bug, and 1.72% triggered question about the video. Out of all the videos, 6.88% did not seem to have an immediate reaction from the developers at all, and 1.38% resulted in an immediate response from end-users. Regarding any possible combinations of these response types in a single comment, 12.05% of videos both seemed to result in being perceived as helpful and also appeared to enable developers to reproduce the bugs. 3.92% of videos appeared to be perceived as helpful but also triggered questions about the bugs. Lastly, 2.58% of videos did not enable developers to reproduce the bug and also resulted in developers having questions about the bug.

Some additional results emerged during the labeling of the videos. First, 34.45% of videos were submitted initially, as part of the original bug report being filed. Out of 65.55% of the videos that were submitted later, 60.38% were voluntary and 5.17% were by request, as part

<sup>55</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=974643](https://bugzilla.mozilla.org/show_bug.cgi?id=974643)

of the subsequent back-and-forth between the reporter, the assigned developer, and others who may chime into the discussion.

Moreover, around 10% of bug reports contained a second video (1.33% contained three or more). 39.81% of the second videos were alternatives (showing another way to reproduce the bug such as bug id 1517198<sup>56</sup>), 15.98% showed expected results (e.g., bug id 1439051<sup>57</sup>), 13.59% were refinement (a better quality video or a better example to reproduce the bug, such as bug id 1607954<sup>58</sup>), 10.02% showed a solution to resolve the bug (e.g., bug id 1383593<sup>59</sup>), 9.95% illustrated that the bug is fixed (e.g., bug id 1384290<sup>60</sup>), 6.8% were duplicates (not the same video but a similar one using a new system setup or operating system, such as bug id 1511215<sup>61</sup>), and 3.85% showed actual results (e.g., bug id 1665294<sup>62</sup>).

## 5.2.2 Impact on Videos Appearing to Be Perceived as Helpful by Developers

To examine the impact of including various characteristics of video on the video of the bug report appearing to be perceived as helpful by developers, the results from manual labeling were used. That is, the labeled videos were divided into two groups. The first group was videos with the labels “perceived helpful”, “reproduceable”, “change of activity”, or “Flags: ever confirmed: true”, and was additionally labeled as “helpful”. The second group was the videos with the labels of “question about bug”, “question about video”, “non-reproducible”, or “Flags: needinfo?(*email of the video submitter*)”, which was labeled as “not helpful”.

The dataset included 1,045 videos out of which 618 were labeled as “helpful” and 427 as “not helpful”. Figure 44 shows that across all 11 years a higher percentage of videos were “helpful”. This difference was statistically significant (Welch Two Sample t-test,  $t = 6.16$ ,  $df = 18.98$ ,  $p\text{-value} < 0.06$ ) and remained so when examining just the last three years (Welch Two Sample t-test,  $t = 3.37$ ,  $df = 3.67$ ,  $p\text{-value} < 0.06$ ). This confirms the observations made

---

<sup>56</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1517198](https://bugzilla.mozilla.org/show_bug.cgi?id=1517198)

<sup>57</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1439051](https://bugzilla.mozilla.org/show_bug.cgi?id=1439051)

<sup>58</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1607954](https://bugzilla.mozilla.org/show_bug.cgi?id=1607954)

<sup>59</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1383593](https://bugzilla.mozilla.org/show_bug.cgi?id=1383593)

<sup>60</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1384290](https://bugzilla.mozilla.org/show_bug.cgi?id=1384290)

<sup>61</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1511215](https://bugzilla.mozilla.org/show_bug.cgi?id=1511215)

<sup>62</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1665294](https://bugzilla.mozilla.org/show_bug.cgi?id=1665294)

in prior studies that videos offer developers context-rich bug information [36] and help them understand any events that may have contributed to the manifestation of the bug [35], [140]–[142].

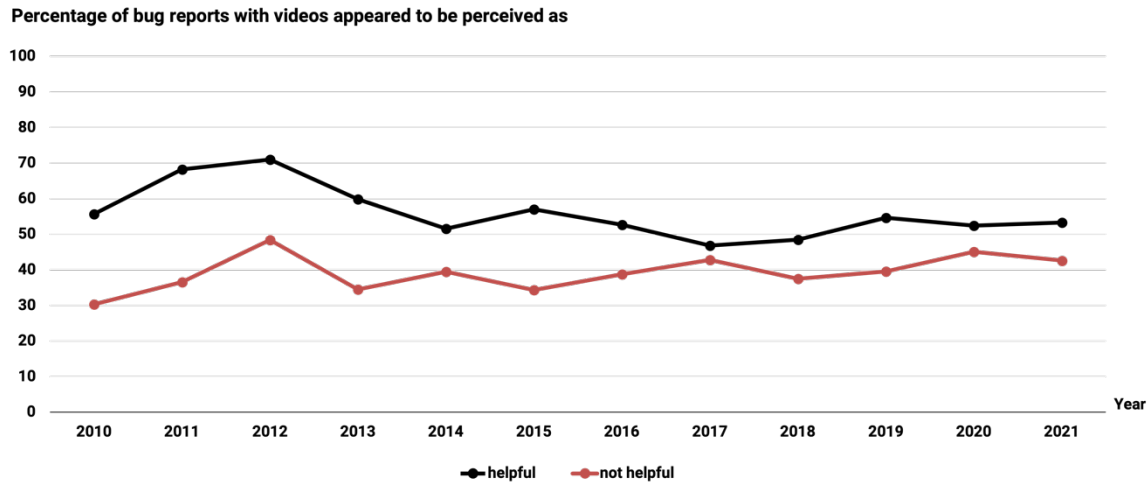


Figure 44. Percentage of bug reports with videos that appeared to be perceived as helpful and not helpful.

**Observation 1. Across all 11 years on average a statistically significant higher percentage of videos appeared to be perceived as helpful by developers.**

In the following subsections, the potential impact of including various characteristics of video, detected during the manual labeling, on the videos appearing to be perceived as “helpful” are presented. A few video characteristics such as sound and repetition were not further included in the analyses as only a small percentage of videos that were studied contained them (see Section 5.2.1).

### 5.2.2.1 Annotations

If the presence of annotation were to lead to a higher percentage of videos appearing to be perceived as helpful by developers, that would represent a potential motivation for reporters to annotate the videos they submit. As explained in Section 5.2.1, 37.8% of the videos were labeled as having annotations such as pointer movement or highlighting with the cursor.

Figure 45 shows the trend for the percentage of videos with and without annotation that appeared to be perceived as helpful by developers across 11 years. Comparing the means, the videos without annotations had a higher percentage of being perceived as helpful than the ones with annotations throughout the years. The difference was statistically significant (Welch Two Sample t-test,  $t = -4.30$ ,  $df = 12.37$ ,  $p\text{-value} < 0.06$ ), except in the last three years where the difference becomes smaller and no longer statistically significant (Welch Two Sample t-test,  $t = -1.12$ ,  $df = 1.54$ ,  $p\text{-value} < 0.06$ ).

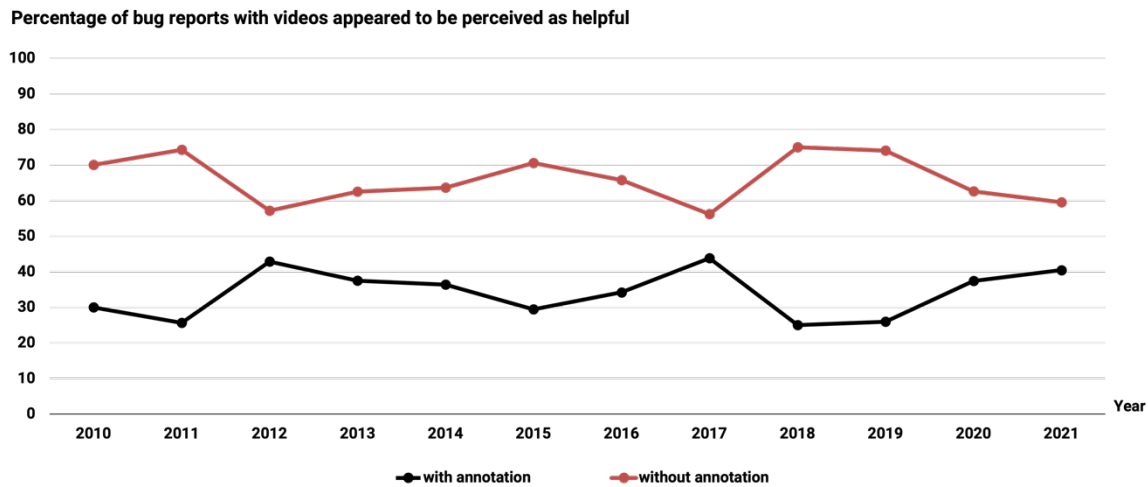


Figure 45. Percentage of bug reports with videos appeared to be perceived as helpful with and without annotation.

In a way, the result is surprising as one would expect annotations help developers to locate the problem and understand the bug, such as bug id 1518086<sup>63</sup>, which included a video with annotation (highlighting with cursor) and enabled a developer to reproduce the bug (appeared to be perceived as helpful). However, it seems that some annotated videos were still not able to cover all aspects of a bug and appeared to be as not helpful. That might be because of the nature of bugs: for the bugs that are more difficult, including annotations does not necessarily contribute to the video becoming helpful. For example, the video in bug id 1267379<sup>64</sup> had annotations (combination of pointer amplified and labels) but still triggered questions about the bug itself by the developers (and thus appeared to be perceived as not

<sup>63</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1518086](https://bugzilla.mozilla.org/show_bug.cgi?id=1518086)

<sup>64</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1267379](https://bugzilla.mozilla.org/show_bug.cgi?id=1267379)

helpful). Another possible reason might be that annotations are often misused by end-users and cause confusion, and therefore developers may generally prefer the videos without annotations.

**Observation 2. The difference between the percentage of videos with and without annotation that appeared to be perceived as helpful by developers was statistically significant. Over the last three years, the difference became smaller and no longer statistically significant.**

### 5.2.2.2 Length

Length of the videos was another characteristic that was studied. If developers often appear to perceive videos with a specific (or a range of) length as helpful, it would be important to guide reporters in making videos of a certain length. To examine the impact of the length of the videos on video appearing to be perceived as helpful, the videos were split based on their length into three groups: (1) less than 30 seconds, (2) between 30 and 60 seconds, and (3) more than 60 seconds. Since no prior study was found that grouped bug videos in terms of their length, these groups were formed by the researchers after labeling the videos.

Figure 46 shows the difference in the percentage of bug reports with videos seemed to be perceived as helpful in each group. Note that a higher percentage of videos that were shorter in length (less than 30 seconds) appeared to be perceived as helpful by developers than the ones that were longer (between 30-60 and more than 60). The difference was statistically significant between videos with length of less than 30 seconds and between 30-60 seconds (Welch Two Sample t-test,  $t = -0.91$ ,  $df = 18.94$ ,  $p\text{-value} < 0.06$ ); and less than 30 seconds and more than 60 seconds (Welch Two Sample t-test,  $t = 3.22$ ,  $df = 9.53$ ,  $p\text{-value} < 0.06$ ).

The difference, however, was not statistically significant between videos with a length between 30-60 and more than 60 seconds (Welch Two Sample t-test,  $t = -0.53$ ,  $df = 18.71$ ,  $p\text{-value} > 0.06$ ) and, while changing slightly, that remained so when considering just the last three years (Welch Two Sample t-test,  $t = 0.53$ ,  $df = 1.99$ ,  $p\text{-value} > 0.06$ ).

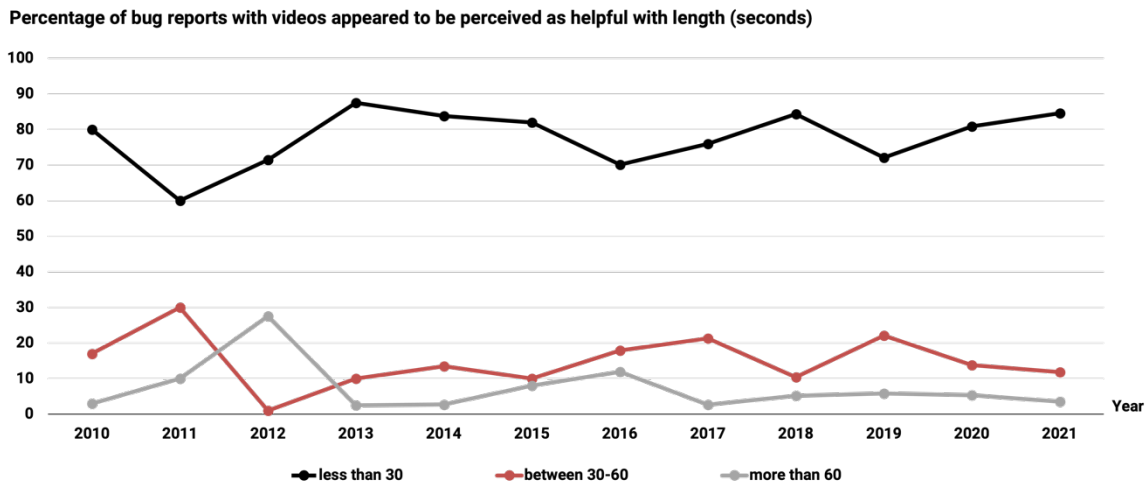


Figure 46. Percentage of bug reports with videos appeared to be perceived as helpful with different lengths.

One could have expected that developers would prefer videos to be longer rather than shorter, especially in the context of prior work that indicated that a good bug report often has a long textual description of the problem [21]. It is unclear what may be causing this, other than possibly the same reason as for annotation: the length of video might depend on the nature of the bug. It could also be that, as the video becomes longer, its “quality” gets lower and that bores developers rather than helping them to understand the bug. One could also assume that a video with the length up to 30 seconds is long enough to show the bug, as software failures typically manifest quickly. Further study is needed to understand what may be causing this.

**Observation 3. The videos with a length of less than 30 seconds appeared to have a statistically significant higher percentage of being perceived as helpful by developers than the ones that are of length of between 30-60 and more than 60. The difference in percentage of being perceived as helpful by developers between videos with lengths of 30-60 seconds and more than 60 seconds, however, was not statistically significant.**



### 5.2.2.3 Steps to Reproduce

Several studies revealed that most developers consider steps to reproduce in the description of bug reports as helpful (e.g., [16], [55]), are very important to understand bug reports (e.g., [13], [33]), and significant for reproducing bugs (e.g., [13]). Presumably, that would be the case for videos as well. Figure 47 shows the trend for the percentage of bug reports with videos that appeared to be perceived as helpful with and without showing steps to reproduce. There was not a statistically significant difference between the two groups (Welch Two Sample t-test,  $t = 0.59$ ,  $df = 22$ ,  $p\text{-value} > 0.06$ ).

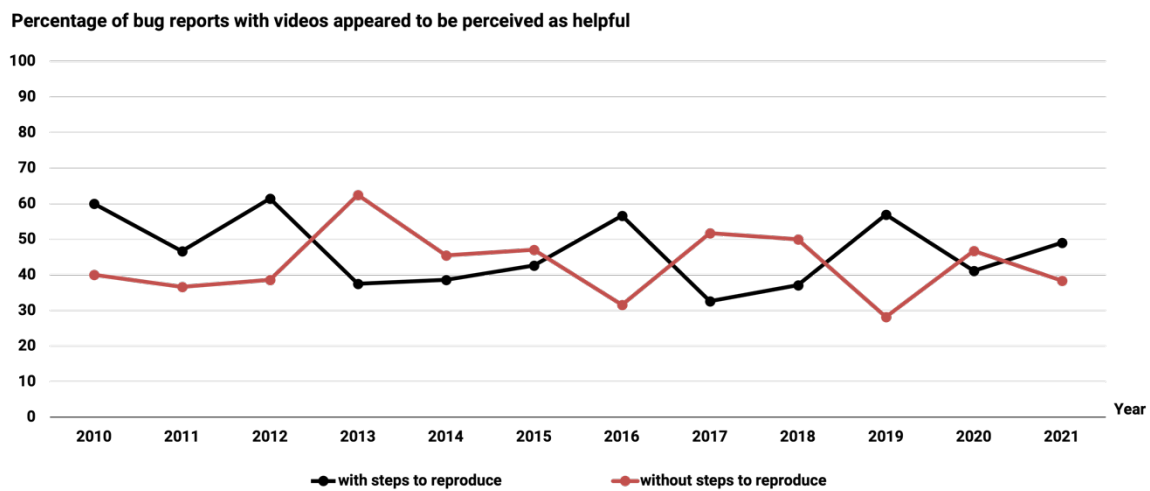


Figure 47. Percentage of bug reports with videos appeared to be perceived as helpful with and without showing steps to reproduce.

This result is surprising, given that the presence of steps to reproduce in the description of bug reports is one of important attribute of high quality bug reports (see Section 2.2.1). One could have expected that including steps to reproduce in a video would similarly appear to be perceived as helpful by developers (e.g., 1177822<sup>65</sup>). It might be that developers generally prefer steps to reproduce to be written down in the bug description rather than video to

<sup>65</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1177822](https://bugzilla.mozilla.org/show_bug.cgi?id=1177822)

replicate the bug (e.g., 1611887<sup>66</sup>). Further investigation is needed to understand what might have caused this, such as interviewing the developers.

**Observation 4. The difference between the percentage of videos being perceived as helpful with videos showing steps to reproduce versus those that did not was not statistically significant.**

#### 5.2.2.4 Actual Results

Similar to steps to reproduce, including actual results in the description of bug reports is considered as one of the most important factors to help developers understand the bug (see Section 2.2.1). Supposedly, showing actual results in video may also be helpful for developers. Examining the difference between the percentage of videos with actual results and without, the results are promising. A higher percentage of videos showing actual results appeared to be perceived as helpful by developers, see Figure 48. The difference was statistically significant (Welch Two Sample t-test,  $t = -3.88$ ,  $df = 11.93$ ,  $p\text{-value} < 0.06$ ) and remained so when examining just the last three years (Welch Two Sample t-test,  $t = -3.47$ ,  $df = 1.29$ ,  $p\text{-value} < 0.06$ ). This possibly confirms that including actual results is a very important factor for a video to be perceived as helpful by developers.

---

<sup>66</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1611887](https://bugzilla.mozilla.org/show_bug.cgi?id=1611887)

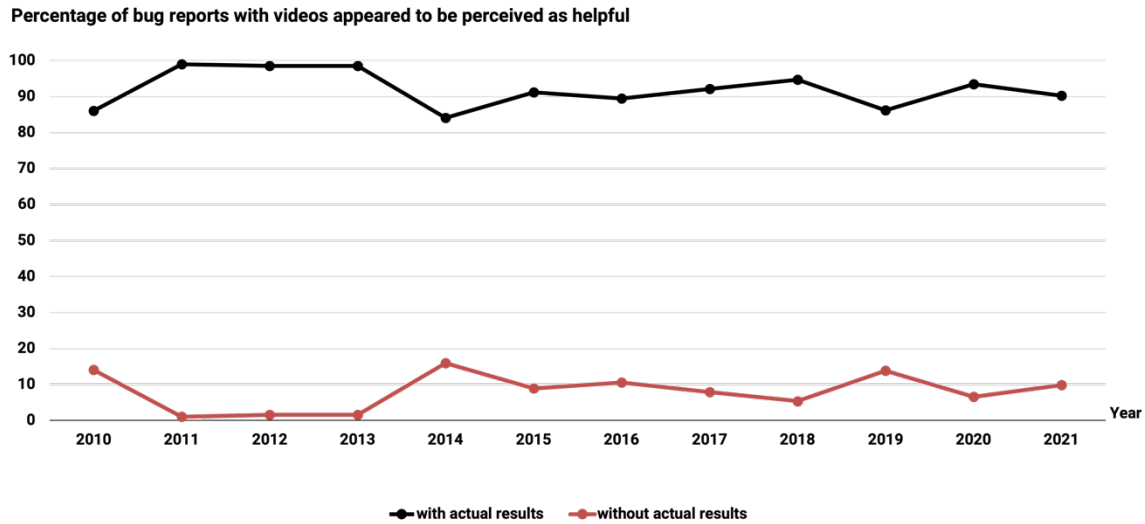


Figure 48. Percentage of bug reports with videos perceived as helpful with and without showing actual results.

One could have expected that video might often include both actual results and steps to reproduce. However, overall, out of all the videos that included actual results, only 47.65% also included steps to reproduce (see Section 5.2.1). The reason might be that reporters did not know or remember how to again reproduce the bug.

**Observation 5. There was a statistically significant higher percentage of videos showing actual results that appeared to be perceived as helpful by developers than those that did not. Moreover, the difference remained statistically significant when examining just the last three years.**

### 5.2.2.5 Priority

The potential role of the priority that was assigned to bug reports with videos was also examined. Priority defines the order in which a bug should be fixed (see section 2.1.4.3). It could be that bug reports with video with various characteristics have a different priority than those without, which could impact the percentage of videos appearing to be perceived as helpful by developers. To do so, the bug reports with videos were grouped into two priority groups: high priority (P1, P2, P3) and low priority (P4, P5, not set). Then the

percentage of videos that appeared to be perceived as helpful for each group was compared, with and without various characteristics. The difference was statistically significant (Welch Two Sample t-test,  $t = 1.96$ ,  $df = 9.82$ ,  $p\text{-value} < 0.06$ ) between the percentage of videos that appeared to be perceived a helpful with annotations and the ones without annotations. A higher percentage of videos without annotations appeared to be perceived as helpful than the ones with annotations throughout the years. The difference was also statistically significant when examining only last three years.

The results also showed that the difference between the percentage of videos that appeared to be perceived as helpful and included steps to reproduce and the ones that did not include steps to reproduce was not statistically significant (Welch Two Sample t-test,  $t = 1.2$ ,  $df = 15.29$   $p\text{-value} > 0.06$ ). The difference remained not statistically significant when assessing only the past three years (Welch Two Sample t-test,  $t = 0.44$ ,  $df = 7.08$ ,  $p\text{-value} > 0.06$ ).

For the three video length groups, the difference between the percentage of videos that appeared to be perceived as helpful with videos of less than 30 seconds and between 30-60 seconds was statistically significant (Welch Two Sample t-test,  $t = 0.98$ ,  $df = 13.4$ ,  $p\text{-value} < 0.06$ ). The difference between the percentage of videos with length of less than 30 seconds and more than 60 seconds (Welch Two Sample t-test,  $t = 1.04$ ,  $df = 10.05$ ,  $p\text{-value} > 0.06$ ), and between the percentage of videos with the length of between 30-60 seconds and more than 60 seconds (Welch Two Sample t-test,  $t = 1.55$ ,  $df = 6.74$ ,  $p\text{-value} > 0.06$ ) was not statistically significant.

The results also revealed that the difference between the percentage of videos in high priority bug reports that appeared to be perceived as helpful with videos showing actual results and not showing actual results was statistically significant (Welch Two Sample t-test,  $t = -2.11$ ,  $df = 13.02$ ,  $p\text{-value} < 0.06$ ). The difference remained statistically significant when only examining the past three years.

The combination of these findings with the overarching results implies that the correlation of including video characteristics with the percentage of videos appearing to be perceived as helpful in high priority bug reports, is similar to all bug reports with different priority levels.

### 5.2.2.6 Severity

Next, the potential role of the severity that is assigned to bug reports was assessed. It might be that bug reports with videos with various characteristics have a different severity than those without and that could impact the percentage of videos appearing to be perceived as helpful by developers. The severity levels were grouped into three severity groups: high severity (group 1: blocker, critical, and major), low severity (group 2: normal, trivial, minor, Not set), and request for enhancement (group 3: enhancement). Then the percentage of videos that appeared to be perceived as helpful by developers was compared for group 1 with and without annotation, steps to reproduce, actual results, and three video length groups. The results showed that the difference between the percentage of high severity bug reports with videos that appeared to be perceived as helpful with annotation and without annotation, was statistically significant (Welch Two Sample t-test,  $t = 4.26$ ,  $df = 10.76$ ,  $p\text{-value} < 0.06$ ). Only assessing the last three years revealed similar results (Welch Two Sample t-test,  $t = 5.95$ ,  $df = 9.54$ ,  $p\text{-value} < 0.06$ ).

The difference was not statistically significant between videos that appeared to be perceived as helpful and show steps to reproduce versus the ones that do not show steps to reproduce in high severity bug reports (Welch Two Sample t-test,  $t = 4.93$ ,  $df = 5.36$ ,  $p\text{-value} > 0.06$ ). The difference still was not statistically significant when only examining the past three years.

The results also showed that in high severity bug reports, the length of video did not correlate with the percentage of videos appearing to be perceived as helpful. There were statistically significant differences between the percentage of videos, that appeared to be perceived as helpful and were of length of less than 30 seconds versus the ones between 30-60 seconds (Welch Two Sample t-test,  $t = 0.37$ ,  $df = 10.69$ ,  $p\text{-value} > 0.06$ ), and between videos with length of less than 30 seconds and more than 60 seconds (Welch Two Sample t-test,  $t = 0.82$ ,  $df = 8.59$ ,  $p\text{-value} > 0.06$ ).

The results, overall, indicate that the inclusion of annotation, steps to reproduce, and actual results in high severity bug reports resulted in similar results to when bug reports with various severity levels were studied. The only exception concerns video length: studying all severity levels revealed that videos that were shorter in length (less than 30 seconds)

appeared to have a higher percentage of being perceived as helpful by developers than the ones that were longer (between 30-60 or more than 60). However, for high severity bug reports the length of video did not seem to correlate with video appearing to be perceived as helpful.

### 5.2.2.7 Generalized Linear Model

To examine whether different characteristics of video may have an impact on video appearing to be perceived as helpful, a Generalized Linear Model was built. The resulting best model consisted of 10 factors out of all factors extracted from the bug report (mentioned in Section 5.1.3) including time to resolution, component, priority, severity, product, number of comments, version, operating system, readability, and number of videos; and seven factors from the results of the manual labeling including annotation, length of the video, showing actual results in video, showing steps to reproduce in the video, video submitted initially, video submitted voluntarily later, and video submitted by request later.

The predicted value is whether video appeared to be perceived as helpful or not by developers. The McFadden Adjusted ( $R^2$ ) of the model is 0.50, which means that the independent variables used in building the regression model only contribute to 50% of accurate prediction. Table 33 shows the estimates for some of the factors used in the model. Because of the apparent sensitivity to the period of time studied, this analysis was repeated with shorter time windows (2010-2021, 2013-2021, 2016-2021, and 2019-2021). Results overall indicate a similarly small impact across all time windows. While the p-values are low, the model itself is not accurate enough and thus the effect cannot be interpreted or explained.

Table 33. Generalized Linear Model predicting the percentage of videos appeared to be perceived as helpful by developers by different video characteristics included in the video.

|   | <b>2010-2021</b> | <b>2013-2021</b> | <b>2016-2021</b> | <b>2019-2021</b> |
|---|------------------|------------------|------------------|------------------|
| <b>McFadden's Adjusted <math>R^2</math></b> | 0.50             | 0.52             | 0.52             | 0.65             |

|          |  |            |            |            |          |
|----------|--|------------|------------|------------|----------|
| Estimate | <b>Annotation</b>                        | -6.508e-02 | -7.454e-02 | -7.454e-02 | -0.11    |
|          | <b>Length</b>                            | 3.349e-04  | 3.349e-04  | 3.650e-04  | 0.0027   |
|          | <b>Steps to reproduce</b>                | -1.071e-02 | 2.010e-03  | 2.010e-03  | -0.018   |
|          | <b>Actual results</b>                    | -1.954e-02 | -9.912e-03 | -9.912e-03 | -0.12    |
|          | <b>Video submitted initially</b>         | 1.741e-01  | 4.385e-02  | 4.385e-02  | 0.56     |
|          | <b>Video submitted voluntarily later</b> | -3.800e-02 | 5.869e-02  | 5.869e-02  | 0.50     |
|          | <b>Video submitted by request later</b>  | 1.235e-01  | 7.633e-03  | 7.633e-03  | 0.29     |
|          | <b>Time to resolution</b>                | 1.516e-04  | 2.124e-04  | 2.124e-04  | 0.00061  |
|          | <b>Number of comments</b>                | -1.208e-03 | -2.261e-03 | -2.261e-03 | -0.0053  |
|          | <b>Readability</b>                       | 9.798e-05  | 1.275e-04  | 1.275e-04  | 0.000044 |
|          | <b>Number of videos</b>                  | -4.341e-02 | -3.586e-02 | -3.586e-02 | -0.023   |
|          | <b>P-value</b>                           | < 0.06     | < 0.06     | < 0.06     | < 0.06   |

### 5.2.3 Impact on Bug Report Resolution Process

To understand the impact of including various characteristics of video on the bug report resolution process, analyses were performed along three questions: (1) what is the impact on the length of time to resolution, (2) what is the impact on bug report being resolved with a patch (i.e., resolution status of FIXED), and (3) what is the impact on the back-and-forth between the assigned developer and reporter? Each analysis, because of the seemingly sensitivity to the period of time studied, was repeated with shorter time windows.

### 5.2.3.1 Impact on Time to Resolution

To assess the impact of including different characteristics of video on the time to resolution, the average number of days to resolve the bug reports with videos were compared, between the ones that included the video characteristic and the ones that did not. For each bug report, the time to resolution was calculated as the number of days between the initial submission of the bug report and when the status of the bug report was changed to RESOLVED. It is possible that a bug report was resolved and then reopened, meaning that its history includes the status RESOLVED multiple times. In such cases, the day it was last changed to RESOLVED was measured.

Figure 49 shows the trend for the average number of days to resolve bug reports with videos with various characteristics. The bug reports with videos including annotation took a smaller number of days to resolve (Welch Two Sample t-test,  $t = -4.1227$ ,  $df = 13.167$ ,  $p\text{-value} < 0.06$ ), see Figure 49(a). Comparing the mean, the difference was statistically significant. The difference, however, did get lower over time, where in the last three years it was not statistically significant (Welch Two Sample t-test,  $t = -1.8518$ ,  $df = 3.3428$ ,  $p\text{-value} > 0.06$ ).

Figure 49(b) shows similar results for showing steps to reproduce, where the difference between the average number of days to resolve bug reports with videos and the ones that did not, was statistically significant (Welch Two Sample t-test,  $t = -2.6883$ ,  $df = 13.024$ ,  $p\text{-value} < 0.06$ ). The difference for the last three years, however, was no longer statistically significant (Welch Two Sample t-test,  $t = -0.64221$ ,  $df = 3.9909$ ,  $p\text{-value} > 0.06$ ).

For the three video length groups, the differences in the average number of days to resolve were not statistically significant. The average number of days to resolve the bug reports with video longer than 60 seconds (mean = 281) was higher than the ones with video with a length between 30-60 seconds (mean = 240) and the ones shorter than 30 seconds (mean = 208), see Figure 49(c). The differences remained not statistically significant when only considering the last three years (Welch Two Sample t-test,  $t = -0.4903$ ,  $df = 8.1972$ ,  $p\text{-value} > 0.06$ ).



Finally, the difference between the average number of days to resolve bug reports with videos showing actual results and the ones that did not, was statistically significant (Welch Two Sample t-test,  $t = -3.62$ ,  $df = 13.20$ ,  $p\text{-value} < 0.06$ ); and remained so when examining just the last three years (Welch Two Sample t-test,  $t = -3.363$ ,  $df = 3.0163$ ,  $p\text{-value} = 0.04329$ ), see Figure 49(d).

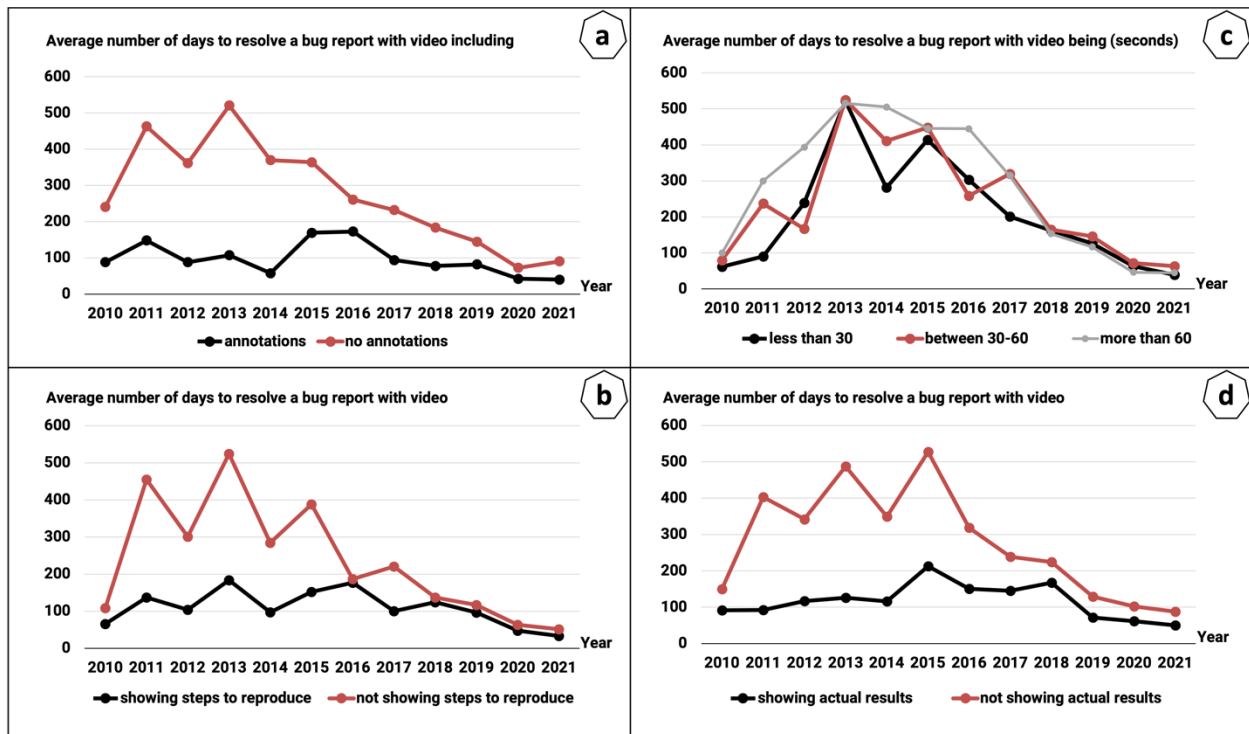


Figure 49. Comparing the average number of days to resolve bug reports with videos with and without different characteristics.

**Observation 6.** Across 11 years it took on average a statistically significant lower number of days to resolve the bug reports with videos showing actual results or including annotation. For the three video length groups or steps to reproduce, the differences in the average number of days to resolve were not statistically significant.

### 5.2.3.1.1 Priority

If the typical priority of bug reports with videos with annotation, steps to reproduce, and actual results were high compared to those without, it might explain why they took statistically significant less time to resolve on average. To assess this, the bug reports with videos were grouped into high priority (P1, P2, P3) and low priority (P4, P5, not set). Then the average number of days to resolve for each group was compared. The difference was statistically significant (Welch Two Sample t-test,  $t = -2.08$ ,  $df = 2038.4$ ,  $p\text{-value} < 0.06$ )

between the average number of days to resolve high priority bug reports with video with annotation and without annotation. The average time to resolution was lower for the high priority bug reports with video with annotation than high priority bug reports with video without annotation. The difference, however, was no longer statistically significant when only studying the last three years (Welch Two Sample t-test,  $t = 1.36$ ,  $df = 171.7$ ,  $p\text{-value} > 0.06$ ).

Similar results were achieved when examining steps to reproduce and actual results. The average time to resolution was smaller for the high priority bug reports with video showing steps to reproduce than those not showing steps to reproduce (Welch Two Sample t-test,  $t = -3.33$ ,  $df = 3.03$ ,  $p\text{-value} < 0.06$ ); and videos showing actual results and those not showing actual results (Welch Two Sample t-test,  $t = -31.03$ ,  $df = 4$ ,  $p\text{-value} < 0.06$ ). The results remained so when examining just the last three years.

The results where all priority levels were considered showed that differences were not statistically significant for the three video length groups. But, for high priority bug reports the differences in average time to resolution were statistically significant. That is, the difference in the average time to resolution for high priority bug reports with videos of less than 30 seconds and between 30-60 seconds was statistically significant (Welch Two Sample t-test,  $t = -0.61$ ,  $df = 3.99$ ,  $p\text{-value} < 0.06$ ); and also videos of less than 30 seconds and more than 60 seconds (Welch Two Sample t-test,  $t = -0.61$ ,  $df = 3.99$ ,  $p\text{-value} < 0.06$ ).

It is interesting to pair these findings with the overarching result of bug reports with video with different characteristics taking less time. The combination of the two results implies that bug reports with video with various characteristics take less average time to resolution, regardless of the bug report's priority. Also, shorter videos (less than 30 seconds) attached to high priority bug reports appeared to lead to a statistically significant smaller number of days to resolve.

#### **5.2.3.1.2 Severity**

The potential role of the severity that is assigned to bug reports was also assessed. It could be that bug reports with videos with various characteristics have a different severity than

videos without, and that might impact the time to resolution. The severity levels were grouped into high severity (group 1: blocker, critical, and major), low severity (group 2: normal, trivial, minor, Not set), and request for enhancement (group 3: enhancement). Then the time to resolution was compared for group 1 with and without annotation, steps to reproduce, actual results, and for the three video length groups. The results showed that there was no statistically significant difference between the time to resolution for the high severity bug reports with video with annotation and without (Welch Two Sample t-test,  $t = -0.01$ ,  $df = 21.57$ ,  $p\text{-value} > 0.06$ ). Similar results persisted when only assessing the last three years (Welch Two Sample t-test,  $t = -2.66$ ,  $df = 3.56$ ,  $p\text{-value} > 0.06$ ).

The results were similar for steps to reproduce and actual results. The difference between the time to resolution for high severity bug reports with videos showing steps to reproduce and videos not showing steps to reproduce, was not statistically significant (Welch Two Sample t-test,  $t = 6.68$ ,  $df = 21.98$ ,  $p\text{-value} > 0.06$ ), as well as for videos showing actual results and videos not showing actual results (Welch Two Sample t-test,  $t = 1.97$ ,  $df = 4.35$ ,  $p\text{-value} < 0.06$ ). The difference still was not statistically significant when only examining the past three years. The results also showed that there was not a statistically significant difference in the time to resolution of high severity bug reports with videos shorter than 30 seconds and between 30-60 seconds (Welch Two Sample t-test,  $t = -5.85$ ,  $df = 12.26$ ,  $p\text{-value} > 0.06$ ). The difference also was not statistically significant between videos with the length of 30 seconds and more than 60 seconds (Welch Two Sample t-test,  $t = -3.65$ ,  $df = 3.96$ ,  $p\text{-value} > 0.06$ ).

This means that, for bugs of importance, the inclusion of annotation, steps to reproduce, actual results, and having different video length appeared to have no correlation with average number of days to resolve.

### **5.2.3.1.3 Generalized Linear Model**

To explore whether the video characteristics may have an impact on bug resolution time, I built a Generalized Linear Model. The resulting best model consisted of 19 factors out of all factors mentioned in Section 5.1.3, including operating system, component, severity,

priority, version, resolution, number of unique participants, number of comments, presence of actual results in bug report description, presence of steps to reproduce in bug report description, number of videos, presence of actual results in video, presence of steps to reproduce in the video, length of the video, annotation in the video, repetition, video submitted initially, video submitted voluntarily later, and video submitted by request later. The predicted value is the average number of days to resolve. The McFadden Adjusted ( $R^2$ ) of the model is 0.78, which means that the independent variables used in building the regression model contribute to 78% of accurate predictions. Results indicate a small impact across all time windows (2010-2021, 2013-2021, 2016-2021, and 2019-2021), see Table 34. It is interesting to observe that, for the time window of the past six and three years, a positive instead of negative correlation exists for actual results: time to resolution is predicted to increase. The difference, though, is small.

Table 34. Generalized Linear Model predicting the impact on the average number of days to resolve bug reports by different video characteristics included in the video.

|                 |   | <b>2010-2021</b> | <b>2013-2021</b> | <b>2016-2021</b> | <b>2019-2021</b> |
|-----------------|---|------------------|------------------|------------------|------------------|
| <b>Estimate</b> | <b>McFadden's Adjusted <math>R^2</math></b> | 0.78             | 0.79             | 0.79             | 0.87             |
|                 | <b>Annotation</b>                           | 1.090e-01        | 7.049e-02        | -1.072e-02       | -3.726e-01       |
|                 | <b>Length</b>                               | -3.583e-03       | -1.738e-03       | -2.224e-03       | 8.881e-03        |
|                 | <b>Steps to reproduce</b>                   | 3.857e-02        | 7.247e-02        | 1.122e-01        | -3.510e-01       |
|                 | <b>Actual results</b>                       | -1.362e-01       | -1.163e-01       | 3.580e-02        | 5.160e-01        |
|                 | <b>Video submitted initially</b>            | 1.741e-01        | 9.340e-02        | 4.510e-01        | -1.433e+00       |
|                 | <b>Video submitted voluntarily later</b>    | -3.800e-02       | -5.658e-02       | 7.077e-02        | -1.344e+00       |

|   |            |            |            |            |
|---|------------|------------|------------|------------|
| <b>Video submitted by request later</b>                         | 1.235e-01  | 4.522e-02  | -1.401e-01 | -1.319e+00 |
| <b>Number of unique participants</b>                            | 5.154e-02  | 5.466e-02  | 5.927e-02  | 2.963e-03  |
| <b>Number of comments</b>                                       | 7.196e-03  | 2.304e-03  | 9.442e-03  | 4.198e-02  |
| <b>Presence of steps to reproduce in bug report description</b> | -5.775e-01 | -4.289e-01 | -3.497e-01 | 2.234e-01  |
| <b>Presence of actual results in bug report description</b>     | 3.091e-01  | 3.652e-01  | 4.683e-01  | -1.516e-01 |
| <b>Number of videos</b>   | 1.929e-01  | 2.316e-01  | -1.690e-03 | 5.684e-01  |
| <b>P-value</b>  | < 0.06     | < 0.06     | < 0.06     | < 0.06     |

**Observation 7. The impact of including certain video characteristics on bug report's time to resolution, is minimal.**

### 5.2.3.2 Impact on Being Resolved with a Patch

If the presence of a video characteristic in videos were to lead to a higher percentage of bug reports being resolved with a patch (i.e., resolution status of FIXED), that would represent a different kind of positive influence. As Figure 50(a) shows, across all 11 years, a higher percentage of bug reports with videos that did not include annotation were resolved with a resolution status of FIXED. The difference was statistically significant (Welch Two Sample t-test,  $t = -13.674$ ,  $df = 22$ ,  $p\text{-value} < 0.06$ ) and remained so when only considering the last three years (Welch Two Sample t-test,  $t = -31.043$ ,  $df = 4$ ,  $p\text{-value} < 0.06$ ).

As Figure 50(b) shows, showing the steps to reproduce in the videos did not make a difference in the percentage of bug reports being successfully resolved. The difference between videos showing steps to reproduce and the ones that did not, was not statistically significant (Welch Two Sample t-test,  $t = 1.8146$ ,  $df = 22$ ,  $p\text{-value} > 0.06$ ).

For the three video length groups, a higher percentage of bug reports with videos less than 30 seconds were resolved successfully with a patch, see Figure 50(c). The differences between the percentage of bug reports being successfully resolved were statistically significant for all three groups. A higher percentage of bug reports with videos shorter than 30 seconds were resolved with a patch than the ones between 30-60 seconds (Welch Two Sample t-test,  $t = 20.405$ ,  $df = 18.279$ ,  $p\text{-value} < 0.06$ ) and the ones longer than 60 seconds (Welch Two Sample t-test,  $t = 4.48$ ,  $df = 16.56$ ,  $p\text{-value} < 0.06$ ); and a higher percentage of bug reports with videos between 30 and 60 seconds were resolved with a patch than the ones longer than 60 seconds (Welch Two Sample t-test,  $t = 3.06$ ,  $df = 18.65$ ,  $p\text{-value} < 0.06$ ).

Results also showed that a higher percentage of bug reports with videos showing actual results were resolved successfully with a patch, see Figure 50(d). The difference between the percentage of bug reports being successfully resolved with videos showing actual results and the ones that did not, was statistically significant. The videos that showed the actual results had a higher percentage of being resolved with a patch (Welch Two Sample t-test,  $t = 19.623$ ,  $df = 22$ ,  $p\text{-value} < 0.06$ ) and remained so when examining just the last three years (Welch Two Sample t-test,  $t = 115.49$ ,  $df = 4$ ,  $p\text{-value} < 0.06$ ).

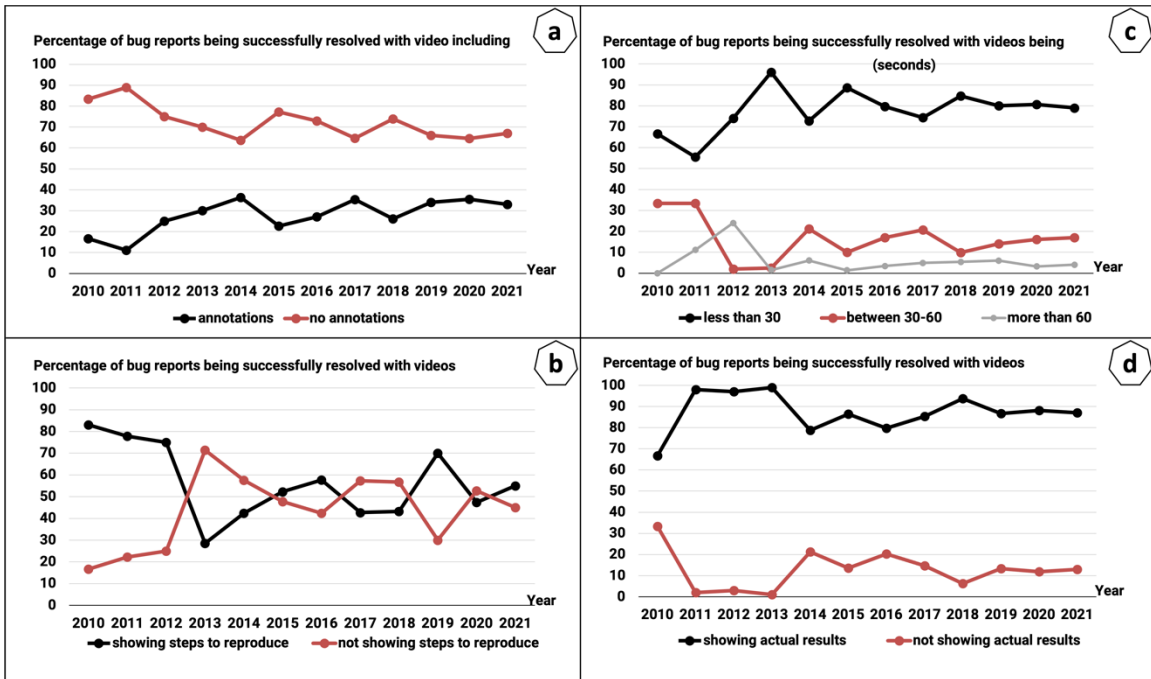


Figure 50. Comparing the percentage of bug reports with videos with different characteristics being resolved with a patch.

**Observation 8.** Overall, bug reports with videos that included annotation, showed actual results, or were shorter than 30 seconds had a statistically significant higher chance of being successfully resolved with a patch. However, no statistically significant difference existed between the bug reports with videos showing steps to reproduce and the ones that do not show them.

### 5.2.3.2.1 Priority

Next, I examined the potential role of the priority that is assigned to bug reports with videos. It could be that bug reports with video with different characteristics have a different priority than those without, and this may impact the percentage of bug reports being resolved with a patch. Again, bug reports with videos were grouped into high priority (P1, P2, P3) and low priority (P4, P5, not set). Then the percentage of bug reports being resolved with a patch



with videos with certain video characteristics was compared to the videos without those characteristics. The difference was statistically significant (Welch Two Sample t-test,  $t = -21.34$ ,  $df = 38.08$ ,  $p\text{-value} < 0.06$ ) when examining annotation: a higher percentage of high priority bug reports with videos that did not include annotation were resolved with a resolution status of FIXED. The difference remained statistically significant (Welch Two Sample t-test,  $t = -18.4$ ,  $df = 32.14$ ,  $p\text{-value} < 0.06$ ) when only considering the last three years.

Showing the steps to reproduce in the videos of high priority bug reports did not make a statistically significant difference in the percentage of bug reports being successfully resolved, when compared with videos that did not include steps to reproduce in high priority bug reports (Welch Two Sample t-test,  $t = 2.46$ ,  $df = 4$ ,  $p\text{-value} > 0.06$ ).

For the three video length groups, a higher percentage of high priority bug reports with videos less than 30 seconds were resolved successfully with a patch than those between 30-60 and more than 60 seconds. The differences, however, were not statistically significant (comparing high priority bug reports with videos shorter than 30 seconds with the ones between 30-60 seconds: Welch Two Sample t-test,  $9.29$ ,  $df = 3.97$ ,  $p\text{-value} > 0.06$ , and comparing high quality bug reports with videos shorter than 30 seconds with those longer than 60 seconds: Welch Two Sample t-test,  $14.703$ ,  $df = 21.974$ ,  $p\text{-value} > 0.06$ ). Similar results were achieved when considering only the past three years.

The results also showed that a higher percentage of high priority bug reports with videos showing actual results were resolved successfully with a patch: the difference between the percentage of high priority bug reports being successfully resolved with videos showing actual results and the ones that did not was statistically significant (Welch Two Sample t-test,  $t = 9.87$ ,  $df = 4.76$ ,  $p\text{-value} < 0.06$ ) and remained so when examining just the last three years (Welch Two Sample t-test,  $t = 63.46$ ,  $df = 3.53$ ,  $p\text{-value} < 0.06$ ).

Overall, the results appeared to be similar to the overarching results when bug reports were not grouped based on their priority. An exception concerns only for the three groups of video length: while results from exploring all priority levels showed that a higher percentage of bug reports with videos being less than 30 seconds were resolved successfully with a patch,

examining only high priority bug reports indicated that the length of video did not make a difference in percentage of bug reports being resolved with a patch.

#### **5.2.3.2.2 Severity**

Next, the potential role of the severity that is assigned to bug reports was examined. It could be that bug reports with video with various characteristics have a different severity than those without, and that might impact the percentage of bug reports being resolved with a patch. The severity levels were grouped into three severity groups: high severity (group 1: blocker, critical, and major), low severity (group 2: normal, trivial, minor, Not set), and request for enhancement (group 3: enhancement). Then the percentage of bug reports being resolved with a patch was compared for group 1 with and without annotation, steps to reproduce, actual results, and for three video length groups. The results showed that there was no statistically significant difference between the percentage of high severity bug reports with video being successfully resolved with annotation and without (Welch Two Sample t-test,  $t = 2.9$ ,  $df = 4.61$ ,  $p\text{-value} > 0.06$ ). Similar results persisted when only assessing only the last three years (Welch Two Sample t-test,  $t = 2.94$ ,  $df = 4.62$ ,  $p\text{-value} > 0.06$ ).

The difference between the percentage of bug reports being successfully resolved for high severity bug reports with videos showing steps to reproduce and the videos not showing steps to reproduce was not statistically significant (Welch Two Sample t-test,  $t = 14.68$ ,  $df = 3.57$ ,  $p\text{-value} > 0.06$ ). The difference still was not statistically significant when only examining the past three years.

The results also showed that there was not a statistically significant difference in the percentage of bug reports being successfully resolved with videos shorter than 30 seconds and between 30-60 seconds (Welch Two Sample t-test,  $t = 9.92$ ,  $df = 3.15$ ,  $p\text{-value} > 0.06$ ); and videos shorter than 30 seconds and more than 60 seconds (Welch Two Sample t-test,  $t = 14.3$ ,  $df = 21.53$ ,  $p\text{-value} > 0.06$ ).

Finally, the results showed that a higher percentage of high severity bug reports with videos showing actual results were resolved successfully with a patch: the difference between the

percentage of bug reports being successfully resolved with videos showing actual results and the ones that did not was statistically significant (Welch Two Sample t-test,  $t = 9.87$ ,  $df = 4.76$ ,  $p\text{-value} < 0.06$ ) and remained so when examining just the last three years (Welch Two Sample t-test,  $t = 63.46$ ,  $df = 3.53$ ,  $p\text{-value} < 0.06$ ).

Overall, the results indicated that for bugs of importance, the inclusion of annotation, steps to reproduce, and actual results, and having different video length did not correlate with the percentage of bug reports being successfully resolved.

### 5.2.3.2.3 Generalized Linear Model

To examine the effect, I also built a Generalized Linear Model (GLR). The resulting best GLR model consisted of 17 factors out of all factors mentioned in Section 5.1.3, including time to resolution, priority, whether the bug was confirmed, readability, number of unique words in the title, number of unique words in the description, number of unique participants, presence of actual results in the bug description, presence of actual results in video, presence of steps to reproduce in the video, annotation in the video, length of the video, count, video submitted initially, video submitted voluntarily later, video submitted by request later, and voiceover. The predicted value is whether bug reports are resolved with a patch, or not. The McFadden Adjusted  $R^2$  of the model is 0.28, which means that this set of factors is insufficient to build an accurate regression model. I built the same model for shorter time windows (2010-2021, 2013-2021, 2016-2021, and 2019-2021) and encountered the same situation, see Table 35. While the p-values are low, the model itself is inaccurate and thus the effect of including a video characteristic on resolution with a patch cannot be explained. Overall, then, this means that the other 17 factors are at work.

Table 35. Generalized Linear Model predicting the impact on the percentage of bug reports being resolved as FIXED by different video characteristics included in the attached video.

|   | 2010-2021 | 2013-2021 | 2016-2021 | 2019-2021 |
|---|-----------|-----------|-----------|-----------|
| <b>McFadden's Adjusted <math>R^2</math></b> | 0.28      | 0.29      | 0.31      | 0.34      |

|          |   |            |            |            |            |
|----------|---|------------|------------|------------|------------|
| Estimate | <b>Annotation</b>   | 2.931e-03  | 1.224e-02  | -5.226e-03 | 2.139e-02  |
|          | <b>Length</b>   | -5.527e-04 | -5.677e-04 | -5.975e-04 | -1.235e-03 |
|          | <b>Steps to reproduce</b>                                   | 3.272e-02  | 3.304e-02  | 6.932e-03  | -7.589e-03 |
|          | <b>Actual results</b>                                       | -4.733e-02 | -4.541e-02 | -5.907e-02 | -3.371e-02 |
|          | <b>Video submitted initially</b>                            | -2.428e-01 | -2.637e-01 | -1.816e-02 | -9.102e-02 |
|          | <b>Video submitted voluntarily later</b>                    | -2.063e-01 | -2.149e-01 | 1.268e-02  | -2.010e-01 |
|          | <b>Video submitted by request later</b>                     | -2.070e-01 | -2.392e-01 | -4.475e-03 | -6.304e-02 |
|          | <b>Time to resolution</b>                                   | -5.530e-04 | -5.920e-04 | -6.493e-04 | -1.011e-03 |
|          | <b>Whether the bug was confirmed</b>                        | 3.143e-01  | 3.310e-01  | 3.405e-01  | 2.330e-01  |
|          | <b>Number of unique words in the title</b>                  | 1.553e-03  | 1.729e-03  | 4.083e-03  | 3.586e-04  |
|          | <b>Number of unique words in the description</b>            | -1.108e-04 | -1.095e-04 | -1.830e-05 | 7.677e-05  |
|          | <b>Readability</b>  | 2.194e-05  | 2.804e-05  | 1.653e-05  | 1.085e-04  |
|          | <b>Number of unique participants</b>                        | 1.923e-02  | 1.849e-02  | 1.963e-02  | 3.187e-02  |
|          | <b>Presence of actual results in bug report description</b> | 1.350e-02  | 1.822e-02  | -1.532e-02 | -5.529e-02 |
|          | <b>Voiceover</b>  | 4.396e-02  | 4.243e-02  | 4.373e-02  | 2.030e-01  |

|                |        |        |        |        |
|----------------|--------|--------|--------|--------|
| <b>P-value</b> | < 0.06 | < 0.06 | < 0.06 | < 0.06 |
|----------------|--------|--------|--------|--------|

### 5.2.3.3 Impact on Back-and-Forth

The back-and-forth exchanges represent times that developers explicitly reached out to the reporter, to clarify something in the bug report or obtain auxiliary information (see Section 2.1.5). Below, I explain whether including various characteristics of video made a statistical difference in the average number of back-and-forth.

The difference between the average number of back-and-forth for bug reports with videos that included annotations (mean 26) and the ones that did not (mean 27), was not statistically significant (Welch Two Sample t-test,  $t = -0.20371$ ,  $df = 21.949$ ,  $p\text{-value} > 0.06$ ), see Figure 51(a).

Figure 51 (b) shows similar results for showing steps to reproduce in the video. The difference between the average number of back-and-forth was not statistically significant (Welch Two Sample t-test,  $t = -0.66451$ ,  $df = 20.286$ ,  $p\text{-value} > 0.06$ ).

The bug reports with videos shorter than 30 seconds had a lower average number of back-and-forth than the ones with the length of between 30-60 seconds (Welch Two Sample t-test,  $t = -0.39$ ,  $df = 19.51$ ,  $p\text{-value} > 0.06$ ), and more than 60 seconds (Welch Two Sample t-test,  $t = -1.103$ ,  $df = 19.85$ ,  $p\text{-value} > 0.06$ ). The differences, however, were not statistically significant, see Figure 51 (c).

Also, the difference between the average number of back-and-forth for bug reports with videos showing actual results and the ones that did not, was not statistically significant (Welch Two Sample t-test,  $t = -0.33573$ ,  $df = 21.926$ ,  $p\text{-value} = 0.7403$ ), see Figure 51 (d).

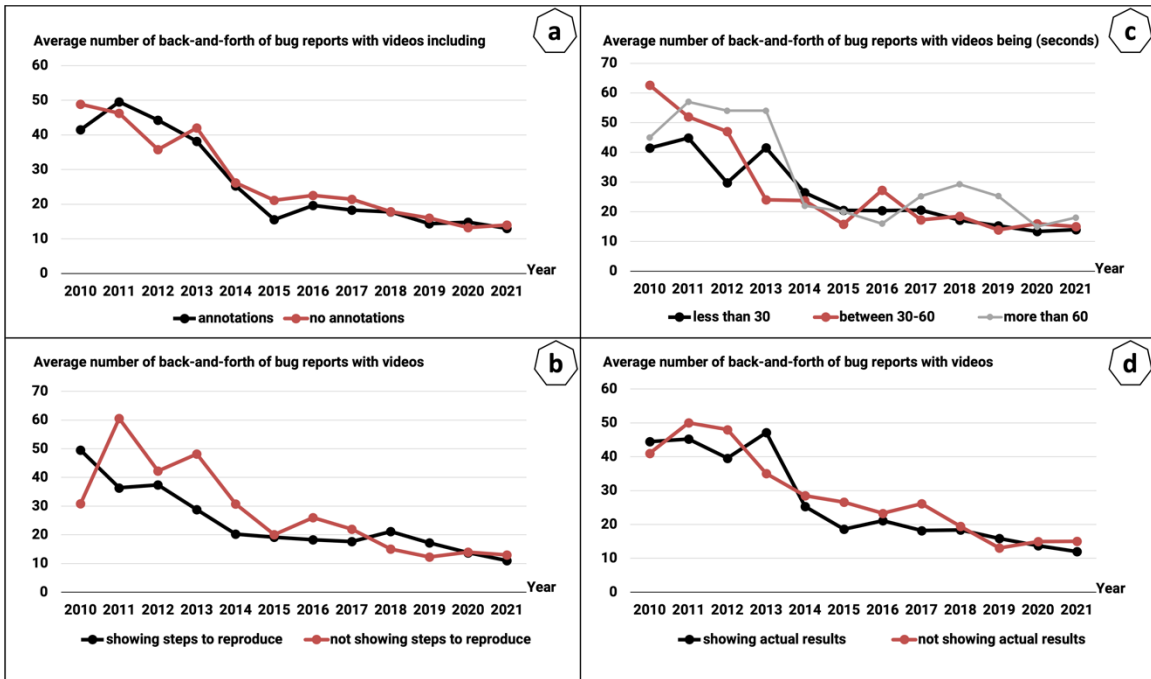


Figure 51. Comparing the average number of back-and-forth in bug reports with videos with different video characteristics.

**Observation 9.** The difference between the average number of back-and-forth in bug reports with videos that included annotations or showed steps to reproduce or actual results versus those that did not, was not statistically significant. The difference, similarly, was not statistically significant for the bug reports with videos shorter than 30 seconds versus the ones with the length of between 30-60 seconds or than 60 seconds.

### 5.2.3.3.1 Priority

I examined the potential role of the priority that is assigned to bug reports with videos. If the typical priority of bug reports with videos with different characteristics were low compared to those without, it might explain why the number of back-and-forth is not significantly affected when the reporters applied various video characteristics. To examine this, the bug reports with videos were grouped into high priority (P1, P2, P3) and low priority (P4, P5,

not set). Then the number of back-and-forth for each group was compared, for videos with certain video characteristics and the ones without.

The difference was not statistically significant (Welch Two Sample t-test,  $t = -1.8$ ,  $df = 8.2$ ,  $p\text{-value} > 0.06$ ) between the average number of back-and-forth for the high priority bug reports with video with annotation and high priority bug reports with video without annotation. The difference was also not statistically significant when examining only the last three years (Welch Two Sample t-test,  $t = 3.61$ ,  $df = 10.7$ ,  $p\text{-value} > 0.06$ ).

Similar results were achieved when examining steps to reproduce and actual results. The differences were not statistically significant between the average number of back-and-forth for high priority bug reports with videos with steps to reproduce and those without steps to reproduce (Welch Two Sample t-test,  $t = -1.93$ ,  $df = 5.36$ ,  $p\text{-value} > 0.06$ ); as well as for videos with actual results and without actual results (Welch Two Sample t-test,  $t = 13.9$ ,  $df = 22$ ,  $p\text{-value} > 0.06$ ).

For the three video length groups, the difference between average number of back-and-forth for high priority bug reports with videos being less than 30 seconds and those between 30-60 seconds was not statistically significant (Welch Two Sample t-test,  $t = 0.33$ ,  $df = 13.60$ ,  $p\text{-value} > 0.06$ ). The results were similar when comparing high priority bug reports with videos shorter than 30 seconds and between 30 and 60 seconds (Welch Two Sample t-test,  $t = 3.76$ ,  $df = 7.73$ ,  $p\text{-value} > 0.06$ ).

Putting these findings together with the earlier results, it appears that including various characteristics in videos did not have a correlation with the average number of back-and-forth between bug reporters and developers, even when only high priority bug reports were considered.

#### **5.2.3.3.2 Severity**

Next, the potential role of the severity that is assigned to bug reports was examined. It could be that bug reports with video with various characteristics have a different severity than those without, and this may impact the back-and-forth because of the implied importance of

fixing severe bugs. The severity levels were grouped into three severity groups: high severity (group 1: blocker, critical, and major), low severity (group 2: normal, trivial, minor, Not set), and request for enhancement (group 3: enhancement). Then the mean of the average number of back-and-forth was compared for group 1 with and without video characteristics. The results showed that there were no statistically significant difference between the average number of back-and-forth for the high severity bug reports with video with and without annotation (Welch Two Sample t-test,  $t = 0.82$ ,  $df = 2.24$ ,  $p\text{-value} > 0.06$ ), steps to reproduce (Welch Two Sample t-test,  $t = -0.65$ ,  $df = 3.19$ ,  $p\text{-value} > 0.06$ ), and actual results (Welch Two Sample t-test,  $t = -1.11$ ,  $df = 2.17$ ,  $p\text{-value} > 0.06$ ).

Similarly, there was no statistically significant difference between the average number of back-and-forth for the high severity bug reports with video shorter than 30 seconds and between 30-60 seconds (Welch Two Sample t-test,  $t = -0.39$ ,  $df = 19.51$ ,  $p\text{-value} > 0.06$ ), and shorter than 30 seconds and more than 60 seconds (Welch Two Sample t-test,  $t = -1.103$ ,  $df = 19.85$ ,  $p\text{-value} > 0.06$ ). The results suggest that including different video characteristics, regardless of bug reports' severity, did not have a correlation with the number of back-and-forth.

### **5.2.3.3.3 Generalized Linear Model**

To study the effect of including various video characteristics on the number of back-and-forth associated with bug reports, I built a Generalized Linear Model. The resulting best GLR model consisted of 20 factors out of all factors mentioned in the methodology including time to resolution, priority, whether the bug was confirmed, component, readability, product, operating system, version, number of unique participants, presence of actual results in the bug description, presence of steps to reproduce in the bug description, presence of actual results in video, presence of steps to reproduce in the video, annotation in the video, length of the video, count, video submitted initially, video submitted voluntarily later, video submitted by request later, and voiceover. The predicted value is the average number of back-and-forth. The McFadden Adjusted ( $R^2$ ) of the model is 0.64, which means that this set of factors together is able to explain 64% of the variability. Table 36 shows the results. The



same model was built for shorter time windows (2010-2021, 2013-2021, 2016-2021, and 2019-2021) and similar results persisted. Again, the model cannot be used to interpret or explain the effect of different video characteristics on the number of back-and-forth. Further study will need to discover what other factors besides the twenty we studied may be causing the difference.

Table 36. Generalized Linear Model predicting the impact on the average number of back-and-forth by different video characteristics included in the video.

|                 |  | <b>2010-2021</b> | <b>2013-2021</b> | <b>2016-2021</b> | <b>2019-2021</b> |
|-----------------|--|------------------|------------------|------------------|------------------|
| <b>Estimate</b> | <b>McFadden's Adjusted R<sup>2</sup></b> | 0.64             | 0.59             | 0.58             | 0.61             |
|                 | <b>Annotation</b>                        | -4.481e-03       | -4.488e-02       | 2.978e-02        | -2.078e-02       |
|                 | <b>Length</b>                            | 6.565e-04        | 6.539e-04        | 7.157e-04        | -2.216e-05       |
|                 | <b>Steps to reproduce</b>                | 8.499e-02        | 9.760e-02        | 9.453e-02        | 6.310e-02        |
|                 | <b>Actual results</b>                    | 5.284e-02        | 2.650e-02        | 8.010e-02        | -2.329e-02       |
|                 | <b>Video submitted initially</b>         | -7.389e-01       | -6.826e-01       | 5.965e-02        | -3.735e-01       |
|                 | <b>Video submitted voluntarily later</b> | -7.150e-01       | -6.236e-01       | 7.207e-02        | -2.111e-01       |
|                 | <b>Video submitted by request later</b>  | -4.737e-01       | -3.844e-01       | 3.621e-01        | 1.378e-01        |
|                 | <b>Time to resolution</b>                | 1.103e-04        | 2.364e-04        | 1.247e-04        | -3.909e-04       |
|                 | <b>Whether the bug was confirmed</b>     | 5.725e-02        | 9.578e-02        | 3.327e-01        | 3.093e-01        |
|                 | <b>Readability</b>                       | 3.289e-04        | 3.337e-04        | 4.300e-04        | 5.546e-04        |

|   |           |           |           |            |
|---|-----------|-----------|-----------|------------|
| <b>Number of unique participants</b>                            | 8.213e-02 | 9.206e-02 | 1.039e-01 | 1.119e-01  |
| <b>Presence of steps to reproduce in bug report description</b> | 2.082e-01 | 2.411e-01 | 2.126e-01 | 1.443e-01  |
| <b>Presence of actual results in bug report description</b>     | 4.450e-02 | 8.885e-02 | 7.754e-02 | 3.496e-01  |
| <b>Voiceover</b>  | 3.711e-01 | 3.761e-01 | 4.465e-01 | -4.639e-01 |
| <b>P-value</b>  | < 0.06    | < 0.06    | < 0.06    | < 0.06     |

#### 5.2.4 Impact of video being perceived as helpful by developers on Bug Report Resolution Process

I performed three additional analyses to understand whether video appeared to be perceived as helpful or not helpful might impact the resolution process of bug reports. For example, if the video seemed to be perceived as not helpful by developers, it might explain why the bug report takes longer to resolve.

Figure 52 shows the trend for the average number of days to resolve bug reports with video appeared to be perceived as helpful and not helpful. Note that videos that were perceived as not helpful took longer to resolve than those that were perceived as helpful (comparing the mean, the difference was statistically significant (Welch Two Sample t-test,  $t = -5.08$ ,  $df = 12.82$ ,  $p\text{-value} < 0.06$ ). The difference in how much longer, however, did get lower over time, with the average over the three most recent years being 105 days of difference (Welch Two Sample t-test,  $t = -3.88$ ,  $df = 3.31$ ,  $p\text{-value} < 0.06$ ).

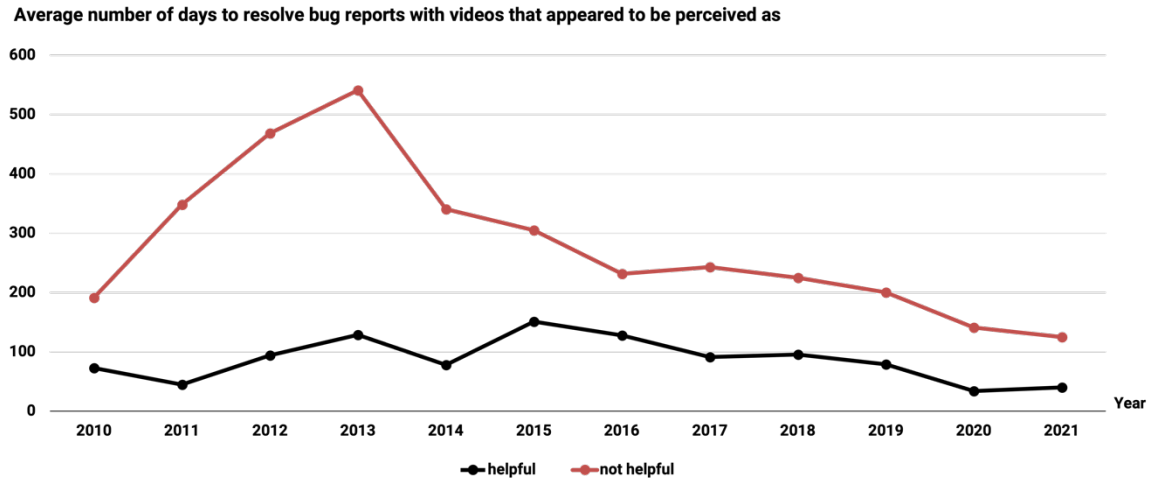


Figure 52. The average number of days to resolve bug reports with video that appeared to be perceived as helpful versus not helpful.

Another desirable outcome, besides a reduced time to resolution, is that a higher percentage of bug reports with videos that appeared to be perceived as helpful by developers was resolved with a patch (i.e., resolution status of FIXED). Figure 53 shows that across all 11 years, compared to videos that were seemed to be perceived as not helpful, a higher percentage of videos that were perceived as helpful were resolved with a resolution status of FIXED. This difference was statistically significant (Welch Two Sample t-test,  $t = 6.6958$ ,  $df = 21.998$ ,  $p\text{-value} < 0.06$ ) and remains so when examining just the last three years (Welch Two Sample t-test,  $t = 6.15$ ,  $df = 4$ ,  $p\text{-value} < 0.06$ ). Note that only in 2015 the results were the opposite, but the difference was not statistically significant between videos that seemed to be perceived as helpful and not helpful.

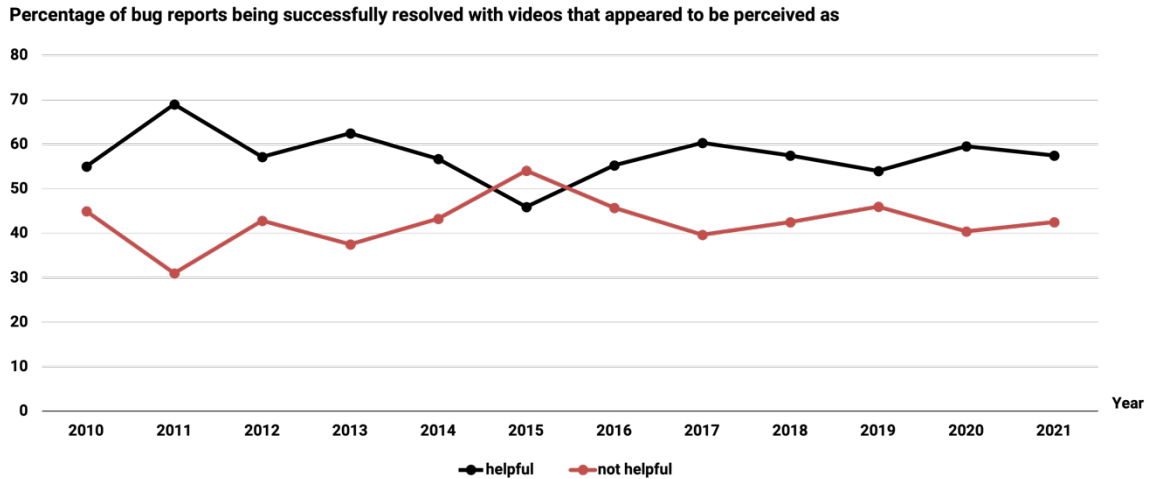


Figure 53. Percentage of bug reports with videos appeared to be perceived as helpful versus not helpful that were successfully resolved with a patch.

Another way in which videos that were perceived as helpful by developers could possibly have a positive impact on the resolution process is if they reduced the back-and-forth, that is if their presence led to fewer requests from developers to reporters for clarification or additional information. Figure 54 shows the trend for the average number of back-and-forth for videos that appeared to be perceived as helpful and not helpful. The difference was not statistically significant (Welch Two Sample t-test,  $t = -0.078$ ,  $df = 21.55$ ,  $p\text{-value} > 0.06$ ). The difference in the number of back-and-forth gets higher over time, however, still is not statistically significant (Welch Two Sample t-test,  $t = -2.1866$ ,  $df = 3.3056$ ,  $p\text{-value} > 0.06$ ).

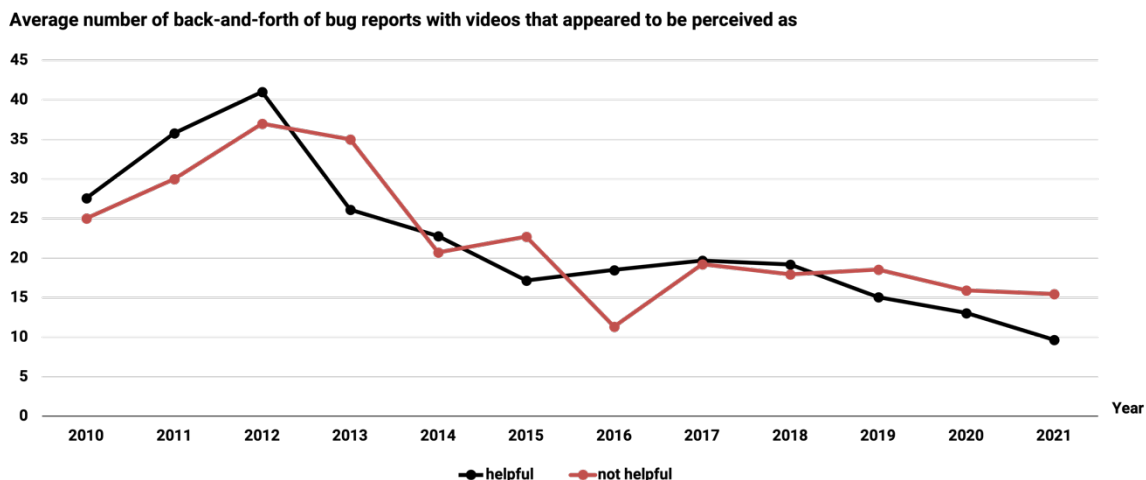


Figure 54. The average number of back-and-forth in bug reports with videos perceived as helpful versus not helpful.

**Observation 10.** Overall, bug reports with videos that appeared to be perceived as helpful had a: (1) higher chance of getting resolved as FIXED (statistically significant), (2) less amount of time to resolve (statistically significant), and (3) less back-and-forth (not statistically significant) than the ones that were perceived as not helpful.

### 5.2.5 Survey

First, I analyzed the results from the first question of the survey: how developers characterize the content of the video attachments in relation to the corresponding textual descriptions.

Several of the answers to this open-ended question mentioned that the videos are useful to look at, for instance: *"I would say video content is "energy saving", because it takes much less time and brain energy to understand and see the issue."* Another respondent mentioned that *"It depends on the quality of the text description. With a good text description and attached exception stack trace or pictures, a video attachment is useful to look at. With poor text description a video attachment is uniquely enlightening."* One respondent wished many

more reporters include videos because even *“Information in “video with bad scenario” is mostly useful as source of additional details which may be crucial for proper reproducing and fixing but reporters sometimes are not aware what might be important and skip some details in issue description.”*

Not everybody agreed, however. One respondent stated *“In most of the cases, a developer first needs to reproduce the problem. To do this, they need to re-create the user’s environment on their machine. In the domain of IDE developers, the most helpful thing in this case is a code snippet or a sample project together with the steps one needs to perform. It’s tiresome to copy code/files from a video, so they must be present as text. As to steps, they can be easily inferred from a video, though it’s still better to have them written down.”* Another respondent believed that video is useful for some subsystems *“the ones where there is a lot of different UI controls, different ways to do one thing.”*

In the second question, the developers shared the kinds of content they consider most and least useful. The following are the properties that they found desirable for the video bugs:

- *“Important things should be emphasized either with extra mouse gestures and the pauses or by voice”,*
- *“It should show steps reproducing the issue and the result”,*
- *“It should not be too long not too short”,*
- *It should be “Full screen (or several screens if any) video with keyboard (KeyCastr for example) and mouse capturing”,*
- *“Sometimes it is useful to have two videos, one with correct application behavior and one with wrong behavior”,*
- *“The actions should not be very fast”,*
- *It “should show in the overlay user actions (keys being pressed, mouse gestures, etc). Possibly repeat the same actions across different version of the software”.*

In response to what kind of content they consider least useful, the developers mentioned *“without steps to reproduce”, “showing only a cropped view of the bug or small part of the IDE”, “attaching the video as a .gif file since it is not possible to stop the playing”, and “not*

*accompanied with bug description (e.g., “if there is a search query that doesn’t work, it’s better to post a query, not show it in the video”).* One developer said *“The least helpful situation is when a report consists only of a video. It makes it harder to start working on a problem.”* Another developer mentioned *“Sometimes users record a video with their own voice describing what they do. This makes no sense and is hard to consume. Description or scenario is always better to see in text format.”* That might be the reason why bug reporters do not make the effort to either directly narrate while a video is being recorded, or overlay audio at a later time (results from manual labeling showed that the percentage of videos with a voiceover was only 2.87%). Bug reporters know what is important: showing the bug itself. Lastly, developers answered whether the inclusion of video attachments: (1) speeds up the process of bug reports being resolved; (2) leads to a higher percentage of bugs being resolved as fixed; and (3) causes less back and forth between bug reporters and developers? They had to choose from definitely not, probably not, about the same, probably so, and definitely so. Through this question, I wanted to understand what developers think about the impact of including videos on the overall bug report resolution process. The results of the survey showed that nearly all of them consider video attachments very useful in understanding bugs more easily and resolving them quicker, by selecting definitely so and probably for all three questions, except two developers. One of them rather preferred reporters to attach a stack trace or a picture, and another one explained that *“Video should be only used as a tool when a textual description doesn’t help to understand the problem for some reason. And even then, the content of the video has to be converted to text when the problem becomes clear.”*

### **5.3 Discussion**

In the introduction to this chapter, I highlighted the prevailing beliefs surrounding the inclusion of videos in bug reports: that videos offer an opportunity to share context-rich bug information with developers [36], [39], [42] and that, in doing so, they help developers in understanding users' interactions, see the actual behavior of the system in response to some input, and internalize what may have contributed to the manifestation of the bug [35], [141], [142], [161].

This chapter asks a complementary question: what sets videos that are more effective apart from videos that are less effective, in terms of their characteristics? With the clear increase in video attachments over the past 11 years (see Section 4.2.1), it is particularly important to examine if various characteristics of videos attached to bug reports appeared to be perceived as helpful by developers and provide tangible benefits for bug reporters. In this context, the characteristics of videos were assessed from two dimensions: (1) How different characteristics of video appear to be perceived by developers? (2) Do different characteristics of video have observable effects on the bug report resolution process (i.e., in reducing time to resolution, leading to an actual fix, or reducing back-and-forth)?

Using manual labeling, different video characteristics were recognized, out of which only four had enough instances in the dataset to allow the statistical analysis to be valid. Each characteristic was assessed, and the results were presented. While one could think that including different video characteristics might seem to be perceived as helpful to developers and positively impactful in the bug resolution process, the results showed that not all of them do. Table 37 summarizes the results.

**Annotation.** Across 11 years of study, compared with the videos that included annotation, a statistically significant higher percentage of videos without annotation appeared to be perceived as helpful by developers (Table 37 (2)); bug reports with videos without annotation had a lower average number of days to resolve; a statistically significant higher percentage of the bug reports with videos without annotation were resolved with a patch; and on average for the bug reports with videos without annotation, no statistically significant difference existed in the average number of back-and-forth (Table 37 (6, 7, 8, 9)). Examining the high severity and high priority bug reports did not alter these outcomes. Taken together, it is difficult to argue that including annotations in videos are not beneficial. Especially since the results of the survey revealed that developers considered annotations such as *“mouse gestures”* or *“mouse capturing”* as the most useful characteristics of video bugs. A possible reason can be that the videos that were studied across the corpus misused annotations and caused confusion for developers, thus affecting the results adversely. To truly sort this out, an in-depth interview is needed with developers and assignees, in which the videos and bug reports are discussed in more details.



**Length.** Comparing bug reports with videos of different length, the videos that were less than 30 seconds appeared to have a statistically significant higher percentage of being perceived as helpful by developers than the ones that are longer (between 3-60 and more than 60), see Table 37 (3). The difference in percentage of being perceived as helpful by developers between videos with lengths of 30-60 seconds and more than 60 seconds, however, was not statistically significant. Moreover, for the three video length groups, the differences in the average number of days to resolve were not statistically significant; a statistically significant higher percentage of bug reports with videos shorter than 30 seconds were resolved with a patch; and no statistically significant difference existed in the average number of back-and-forth (Table 37 (6, 7, 8, 9)). The results also showed that within the class of high priority or severity having a video shorter than 30 seconds did not improve the outcomes. On the whole, it appears that across the corpus of videos that was studied, the videos that were shorter in length were more in favor. This finding was somehow confirmed by the results of the survey where a respondent mentioned the video “should not be too long not too short”.

**Steps to reproduce.** In terms of including the steps to reproduce the bug in the video, the results showed that there was not a statistically significant difference between the percentage of videos being perceived as helpful with videos showing steps to reproduce and the ones that did not (Table 37 (4)). Moreover, with respect to the average number of days to resolve, the percentage of bug reports being successfully resolved, and the average number of back-and-forth, no statistically significant difference existed between the bug reports with videos showing steps to reproduce and the ones that do not show them (Table 37 (6, 7, 8, 9)). Similar results were also achieved when high priority and high severity bug reports were studied. These findings in a way challenge the results of the survey where steps to reproduce was mentioned as one of the most useful video characteristics by a respondent; and videos without steps to reproduce was stated to be the least useful by another respondent. Moreover, one developer also mentioned that steps to reproduce are better to be written down in the bug description itself.

**Actual results.** The results for showing actual results in the videos showed that a statistically significant higher percentage of videos showing actual results appeared to be perceived as helpful by developers than the ones that did not. In terms of the bug resolution process, bug

reports with videos showing actual results took a statistically significant lower number of days to resolve and a statistically significant higher chance of getting successfully resolved (Table 37 (5)). The difference of average number of back-and-forth, however, was not statistically significant between bug reports with videos showing actual results and the ones that did not (Table 37 (6, 7, 8, 9)). Finally, for bug reports of high priority and bug reports with high severity the results remained the same. Overall, it seems that showing what went wrong in videos is effective, similar to including actual results in bug descriptions (see Section 2.2.1).

Table 37. Observations derived from the analyses of video characteristics.

| <b>Observations</b> |  |
|---------------------|--|
| <b>1</b>            | Across all 11 years on average a statistically significant higher percentage of videos appeared to be perceived as helpful by developers.  |
| <b>2</b>            | The difference between the percentage of videos with and without annotation that appeared to be perceived as helpful by developers was statistically significant. Over the last three years, the difference became smaller and no longer statistically significant.  |
| <b>3</b>            | The videos with a length of less than 30 seconds appeared to have a statistically significant higher percentage of being perceived as helpful by developers than the ones that are of length of between 3-60 and more than 60. The difference in percentage of being perceived as helpful by developers between videos with lengths of 30-60 seconds and more than 60 seconds, however, was not statistically significant. |
| <b>4</b>            | The difference between the percentage of videos being perceived as helpful with videos showing steps to reproduce versus those that did not, was not statistically significant.  |
| <b>5</b>            | There was a statistically significant higher percentage of videos showing actual results that appeared to be perceived as helpful by developers than those that did  |

|    |   |
|----|---|
|    | not. Moreover, the difference remained statistically significant when examining just the last three years.  |
| 6  | Across 11 years it took on average a statistically significant lower number of days to resolve the bug reports with videos showing actual results or including annotation. For the three video length groups or steps to reproduce, the differences in the average number of days to resolve were not statistically significant.  |
| 7  | The impact of including certain video characteristics such as showing actual results in videos on bug report's time to resolution is minimal.   |
| 8  | Overall, bug reports with videos that included annotation, showed actual results, or were shorter than 30 seconds had a statistically significant higher chance of getting successfully resolved with a patch. However, no statistically significant difference existed between the bug reports with videos showing steps to reproduce and the ones that do not show them.  |
| 9  | The difference between the average number of back-and-forth in bug reports with videos that included annotations or showed steps to reproduce or actual results versus those that did not, was not statistically significant. The difference, similarly, was not statistically significant for the bug reports with videos shorter than 30 seconds versus the ones with the length of between 30-60 seconds or than 60 seconds. |
| 10 | Overall, bug reports with videos that appeared to be perceived as helpful had a: (1) higher chance of getting resolved as FIXED (statistically significant), (2) less amount of time to resolve (statistically significant), and (3) less back-and-forth (not statistically significant) than the ones that were perceived as not helpful.  |

Taken together, from the perspective of perhaps providing motivation for reporters to produce and submit videos with specific video characteristics along with their bug reports, the findings suggest that the only tangible incentive exists for two of the characteristics

(length and actual results). For annotation and steps to reproduce, no tangible incentive seems to exist.

The findings suggest that on average a higher percentage of videos attached to bug reports appeared to be perceived as helpful than not helpful by developers, see Table 37 (1). The results are reported on averages, which means that some videos do contribute to beneficial effects and others detract. The results therefore do not imply that including annotation or showing steps to reproduce in video is always ineffective. Especially when the results are contrasted with the results of the survey where developers considered including annotation ("*mouse gestures*", "*keyboard and mouse capturing*") and showing steps to reproduce in the videos as helpful. The results of the survey also uncovered the fact that some of the video characteristics, which were removed from further analyses because a low number of videos included them, are not actually helpful in the eyes of developers. For example, one could assume a direct narration or voiceover while a video is being recorded that shows a bug would be ideal for developers. A respondent stated, however: "*I prefer videos where everything is clear without explanation by voice*".

Regarding length and actual results, the results provide motivation for creating videos that are on the shorter side (less than 30 seconds) and focus on what went wrong (showing actual results). Since such videos appeared to have a higher chance of being perceived as helpful by developers and the bug reports with videos that appeared to be perceived as helpful took a statistically significant smaller number of days to resolve than those that appeared to be perceived as not helpful. Also, a statistically significant higher percentage of videos that seemed to be perceived as helpful were resolved with a resolution status of FIXED (Table 37 (10)).

The results lay the groundwork for research that can be performed next. First, an important direction for research is to repeat the study beyond Mozilla. This study of the videos of Mozilla bug reports represents a first, small step into whether different characteristics of videos might provide clues as to useful differences. It might be that the outcomes would be different for certain domains (e.g., video games, mobile apps), for different bug trackers (e.g.,

Jira<sup>67</sup>), or for other ecosystems (e.g., Apache<sup>68</sup>). Understanding whether such differences exist and, if so, why, would enable important practices to transcend across communities.

Second, an important piece of research is to perform a field study with developers to study how they process various video characteristics. Such a study can uncover factors that help certain videos be more effective than others. Rather than doing so based on data already captured, as in the above, this kind of study would do so based on observing developers at work and talking with them about their experiences and perceptions, offering an important complementary perspective.

Third, it is worthwhile to further delve into the potential differences between bug reports with more than one video. The results showed that around 10% of bug reports contained a second video. A close look at the differences between bug reports with one and two (and more) videos in terms of various video characteristics they include, may unearth valuable insight. Informally I looked at 50 bug reports (randomly selected) and observed that some second videos were submitted because the bug reports were difficult to understand, and thus an additional video was requested by the developer (e.g., bug id 1346961<sup>69</sup>). Moreover, some of first videos represented only the actual results whereas the second videos showed the expected results (e.g., bug id 1324918<sup>70</sup>).

Fourth, it may be valuable to invest in tool support for bug reporters to create videos that focus on video characteristics that seemed to be perceived as helpful by developers and have positive impacts on the overall bug resolution process. Such tool support could actively guide a reporter through the steps they should take, ensure that all of the necessary information is there, and automatically submit the bug report on behalf of the reporter.

---

<sup>67</sup> [www.atlassian.com/software/jira](http://www.atlassian.com/software/jira)

<sup>68</sup> [www.apache.org](http://www.apache.org)

<sup>69</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1346961](https://bugzilla.mozilla.org/show_bug.cgi?id=1346961)

<sup>70</sup> [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1324918](https://bugzilla.mozilla.org/show_bug.cgi?id=1324918)

## 5.4 Threats to Validity

While the study was planned to avoid introducing bias, it is possible that adapted strategies may not have been effective to eliminate the possible effects of random noise. This section reviews the threats to the validity of this study.

The dataset used for the study contained bug reports from projects that are all related to Mozilla Firefox. As a result, the findings may not be generalizable to different projects and domains. That said, sampling bug reports from a number of Mozilla projects, rather than just one, means that the dataset includes software in multiple languages and across multiple operating systems. Further study is needed, however, to assess whether similar results can be obtained for other projects.

Another threat is that the training and testing data were labeled manually, which could have introduced bias or mistakes due to the lack of domain expertise. To address this concern, two researchers individually labeled a significant portion of the data. Because of the high inter-rater reliability that resulted, I assume that the risk of individual bias is minimized.

The choice to only include resolved bug reports may represent a threat to internal validity if the characteristic of video of these bug reports differs significantly from those of unresolved bug reports. The results may also be affected if any of the unresolved bugs are reopened in the future. Fortunately, the Mozilla bug report repository is of large size and has a long history, meaning that what used to be unresolved bug reports have moved on to become resolved. As such, I believe this threat is fairly minimal.

Finally, since all survey respondents were related to IntelliJ, the survey responses may not be representative across OSS and commercial projects as the characteristics of developers in other OSS projects may be different from those included in this study. This risk can be reduced by repeating the survey across other constituencies of developers.

## 5.5 Conclusions

This chapter contributes a first empirical study that examines what sets videos that are more effective apart from videos that are less effective, in terms of their characteristics. This study specifically examines how developers appear to perceive different characteristics of video

and whether they may have observable effects on the bug report resolution process (i.e., in reducing time to resolution, leading to an actual fix, or reducing back-and-forth).

The primary finding is that, on the whole, the inclusion of some of the video characteristics does appear to have a positive impact (length and actual results): a statistically significant higher percentage of videos that are less than 30 seconds or showing actual results appeared to be perceived as helpful by developers, as compared to videos that are not. In terms of the bug resolution process, bug reports with videos that are less than 30 seconds or show actual results had a statistically significant higher chance of getting successfully resolved; and bug reports with videos showing actual results took a statistically significant lower number of days to resolve. Further study is needed for annotation and steps to reproduce. As the results are reported on averages, which means that some videos with annotation and steps to reproduce may contribute to beneficial effects. Especially when the findings are contrasted with the results of the survey, the developers considered those video characteristics as helpful.

Another finding is that the bug reports with videos that appeared to be perceived as helpful took a statistically significant smaller number of days to resolve than those that seemed to be perceived as not helpful. Also, a statistically significant higher percentage of videos that appeared to be perceived as helpful were resolved with a resolution status of FIXED.

I believe that taking a close look at the video characteristics from systems other than Mozilla can provide further insight into what sets videos that are more effective apart from videos that are less effective in terms of their characteristics. Moreover, it is worthwhile to perform a field study of how real-world developers process videos being submitted, to complement this deep dive and to understand more about what are and are not effective video characteristics from their perspective and to identify what challenges they encounter in using the videos.





## 6 CHAPTER 6: CONCLUSION

Bug reports collect relevant information about the bugs that users experience while using the software. The information provided in bug reports helps developers diagnose and resolve software bugs. A bug report typically contains a detailed description of the bug, and occasionally includes attachments such as stack traces to point to the location of bug or videos to provide further context.

Unfortunately, bug reports differ in their quality; they are either sufficient and clear to understand the essence of a bug report and know what to do with them (good or actionable bug report), as they are not (bad or non-actionable).

It is not surprising that developers are slowed down by poorly written bug reports, as it takes longer to determine what is wrong and seek meaningful clarifications or crucial extra information [13], [14]. That leads to non-reproduced bugs [7], unfixed bugs [16], and extra effort to triage bugs [7], [15]–[17].

Many studies have proposed potential solutions to solve problems related to bug reports. On the one hand, some work focused on aiding developers analyzing newly submitted bug reports, for instance by predicting their severity or priority (e.g., [22]–[24]), predicting if the bug report is likely to be resolved in a given time frame [25], or by predicting if the bug report represents a duplicate of one or more previous bug reports (e.g., [26]–[28]).

On the other hand, some studies aimed at helping those who submit bug reports before they are submitted, for instance by predicting the quality of individual fields in a bug report (e.g., steps to reproduce, bug report title) [30]–[34], or by giving an overall sense of bug report quality to the reporter together with generic suggestions for improvements [14], [32].

This dissertation complemented this increasingly growing research surrounding bug reports by contributing to a study of several aspects of the bug reporting process that can potentially be improved in order to support those who report bugs and those who process them. The study was conducted in three parts:

## **6.1 Study 1 – Predicting Actionable versus Non-Actionable Bug Reports**

The dissertation first revisited an older problem, that of bug report quality prediction. It particularly looked at the overall quality of bug reports by implementing a classifier which categorizes bug reports as actionable or non-actionable. The classifier was trained on 1,423 bug reports collected from across all Mozilla Firefox projects and achieved results that were significantly improved over the state of art to a precision of 94%, recall of 89%, and F-measure of 0.91. The primary cause of this improved performance appeared to be a combination of needing a binary classifier only, using a bag-of-words approach rather than higher-level features, hyper-parameter optimization, and a large training dataset.

Second, it performed detailed analyses of whether additional features (alone or together), such as readability of bug description or the role of bug submitter, can further improve the results. Results contradicted the long-held beliefs on the impact of those features; with a sufficiently strong base model, the impact of including those features disappears.

Third, the dissertation performed cross-project prediction to assess the portability of the best resulting model. The results were stronger than any to date in terms of portability and provide a good starting point for further exploration.

Fourth, the dissertation presented the results from a survey conducted among developers to get an understanding as to how they feel the quality of bug reports, with the results showing that they believe the quality over the years stayed the same with a small percentage saying it has become better.

The findings can serve as a basis for developing tools that assist reporters in improving their bug reports. Such tools could intervene before a non-actionable bug report is submitted and help reporters to turn their bug report into an actionable one. That, in turn, could benefit developers as well, since a higher percentage of bug reports submitted through bug reporters could be more easily understood and resolved.

## **6.2 Study 2 – An Analysis of Video Submissions in Bug Reporting**

The dissertation contributed a first empirical study that examined whether including videos in bug reports leads to effects that may benefit the reporters in terms of time to resolution,

resolution with a patch aiming to fix the reported bug, and the amount of back-and-forth. Through statistical analyses on 2,814,599 bug reports and their metadata from five different systems (Mozilla, Android, LibreOffice, IntelliJ, and Minecraft), the results suggested that:

- Including videos incurs on average a longer time to resolution. The difference has shrunk considerably, however, and over the last six years the impact has become minimal. This holds true when videos are submitted as part of the initial bug report submission and when they are submitted at a later time.
- While for some systems a higher percentage of bug reports ends up being fixed if a video is included, for other systems the opposite occurs in fewer bug reports being fixed if videos are included.
- On average bug reports with videos incur a statistically significant higher average back-and-forth than bug reports without. For bug reports that are submitted later, even if only the back-and-forth after the submitted video is counted, a statistically significant difference remains in the back-and-forth still being higher as compared to bug reports without videos.

The dissertation also performed a deep dive into a select set of potential factors for Mozilla bug reports. Results suggested that bug reports with videos submitted by developers are resolved at a higher rate with less average number of back-and-forth.

The results overall suggested that the inclusion of videos does not appear to have an incentive for bug reporters. Including video hardly impacted the time to resolution, the percentage of bug reports being fixed for most of the systems, and the average amount of back-and-forth (which was actually higher for bug reports that include video than those that do not). The findings opened up a number of important research directions, discussed in Section 6.4.

### **6.3 Study 3 – A Content-Based Analysis of Video Submissions in Bug Reporting**

The last part of the dissertation refined the analysis from the second study (which only looked at the presence of videos) with a focus on the content of videos. It offered a first empirical exploration as to what distinguishes more effective videos from less effective videos, in terms of their content. It first examined the detailed characteristics of 1,045 videos from Mozilla bug reports, such as whether the video shows steps to reproduce, whether the video creator emphasizes a bug through mouse movements, and whether the video contains a voice over. Then it studied how developers appear to perceive videos with different characteristics and whether certain characteristics may exhibit a tangible benefit for the bug reporter by measuring the bug report resolution process (i.e., average time to resolution, percentage of bug reports being resolved with a patch, or average number of back-and-forth).

The main finding was that, on average a higher percentage of videos attached to bug reports appeared to be perceived as helpful than not helpful by developers. Moreover, bug reports with videos that appeared to be perceived as helpful took a statistically significant smaller number of days to resolve than those that seemed to be perceived as not helpful, and a statistically significant higher percentage of videos that appeared to be perceived as helpful were resolved with a resolution status of FIXED.

From the perspective of maybe providing motivation for reporters to produce and submit videos with specific video characteristics along with their bug reports, results suggested that a possible tangible incentive exists for two of the characteristics (length and actual results): a statistically significant higher percentage of videos that are less than 30 seconds or showing actual results appeared to be perceived as helpful by developers, as compared to videos that are longer or do not show actual results. Also, bug reports with videos that are less than 30 seconds or show actual results had a statistically significant higher chance of getting successfully resolved; and bug reports with videos showing actual results took a statistically significant lower number of days to resolve.

The dissertation also conducted a small survey among IntelliJ developers to put the results in context. The developers had various experiences with and perceptions of different characteristics of the video attachments. On the whole, they considered the videos useful to look at, especially the ones with characteristics such as video showing steps to reproduce and actual results, or extra mouse gestures. The developers also shared what they consider least useful, such as when a report consists only of a video and is not accompanied with a textual description, or without steps to reproduce.

The findings can serve as basis to develop tools that help bug reporters create videos with characteristics that seemed to be perceived as helpful by developers and had a positive impact on the overall bug resolution process. Especially since the presence of video attachments on average had minimum impact on the overall bug report resolution process (Study 2), such tool support could actively guide bug reporters in creating videos with certain characteristics that appeared to be beneficial to both developers and bug reporters.

## **6.4 Contributions**

Overall, the main contributions of this dissertation are:

- A classifier that categorizes bug reports as actionable or non-actionable, achieving results that significantly improve over the state-of-the-art.
- New findings about the absence of improvement in performance of the classifier with the inclusion of auxiliary features (e.g., length of bug description or experience of bug reporter) that were shown in prior research to have a positive impact on the predictive capability of the resulting models.
- Analyses of portability of the classifier which resulted in a better predictive capability than the best models to date.
- New findings from a survey conducted among Mozilla developers about the quality of newly submitted bug reports.
- A dataset of 1,423 Mozilla bug reports, manually labeled as actionable or non-actionable.

- A first empirical study that examines whether including videos in bug reports could have potential benefits for the reporters in terms of time to resolution, resolution with a patch aiming to fix the reported bug, and the amount of back-and-forth.
- A custom web scraping tool that downloads all available metadata of bug reports from different bug tracking systems, including Bugzilla, YouTrack, and Google Issue Tracker.
- A dataset of 2,814,599 bug reports and their metadata from five different systems (Mozilla, Android, LibreOffice, IntelliJ, and Minecraft), from 2010 to 2021.
- Deep analyses of Mozilla bug reports to explore the potential impact of the role of video submitter, type of bug, and bug report severity on potential incentives to the reporters in terms of overall bug resolution process.
- A first deep empirical study which studies the content of videos attached to bug reports to understand whether or not developers perceived the corresponding videos as helpful, as well as whether various characteristics of the videos have observable effects for the reporters in terms of time to resolution, percentage of being fixed with a patch, and amount of back-and-forth.
- A dataset of 1,045 videos and their associated bug reports from Mozilla, manually labeled based on the contents of videos and how developers reacted publicly to them.
- New findings from a survey conducted among IntelliJ developers to understand developers' experiences with and perceptions of different characteristics of the video attachments.

## **6.5 Future work**

The findings from the dissertation suggest several avenues of future investigation. The most immediate future work is to develop tools that assist reporters in improving their bug reports before they are eventually submitted.

- In terms of tool support for the quality prediction of newly-submitted bug reports, it is important to explore ways to guide reporters in turning non-actionable bug reports

into actionable ones. For example, one could develop an interactive bug reporting system that: (1) allows an end-user to report the bug through conversation, (2) automatically predicts the overall quality of the report, and (3) provides meaningful feedback and recommendations to reporters about the bug report so it can be improved by the reporters before it is eventually submitted. Such a system can employ a chatbot to guide reporters through the conversation to find and report the necessary information for developers to proceed and resolve the bug. The tool can also have a database of already-submitted similar and actionable bug reports to propose a template or provide feedback and recommendation to reporters about their bug report. For example, the chatbot can tell the reporter: "Your report is not actionable because of X and Y, let me pull up three reports that are actionable as an example for you."

- In terms of tool support for reporters to create effective videos, one could implement a screen recording tool which, once a video is created, automatically generates a list of characteristics that are missing in the video but beneficial to both developers and bug reporters (e.g., the importance of inclusion of actual results or the acceptable length of an effective video). Such a tool can also have a built-in video editor to allow reporters improve their videos by adding annotations and audio recordings.

Additionally, each study in the dissertation laid the groundwork for research that can be performed next:

- Study 1: A future direction is to explore whether the results of the prediction model can be further improved. Another future work is to dive deeper into the properties of bug reports that the machine learning classifiers pick up on to distinguish actionable versus non-actionable bug reports. It is also worthwhile to explore whether it is possible for the portability of the classifier to be further improved.
- Study 2: A particularly interesting future direction is to weave the results of Study 1 and 2 and explore if video-based bug reports are more actionable. Another possible future direction is to understand whether the non-benefit of including videos in bug reports, as seen in the results, is actually because of another factor such as the

difficulty level of the bug. An interesting future work is to repeat the study for systems in additional domains that are highly visual such as video games. Moreover, there is an opportunity to perform a field study with developers to study how they process videos as part of bug report resolution.

- Study 3: An important future work is to perform a field study with developers to uncover factors that help certain videos be more effective than others. It is also worthwhile to further delve into the potential differences between bug reports with more than one video. Another possible future work is to repeat the study beyond Mozilla.



## Bibliography

- [1] I. Ahmed, N. Mohan, and C. Jensen, "The Impact of Automatic Crash Reports on Bug Triaging and Development in Mozilla," in *Proceedings of The International Symposium on Open Collaboration*, 2014, pp. 1–8.
- [2] J. Zhang, X. Wang, D. Hao, B. Xie, L. Zhang, and H. Mei, "A survey on bug-report analysis," *Sci. CHINA Inf. Sci.*, vol. 58, no. 2, p. 021101(24)-021101(24), Jan. 2015, doi: 10.1007/s11432-014-5241-2.
- [3] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *2009 IEEE 31st International Conference on Software Engineering*, IEEE, 2009, pp. 298–308.
- [4] A. J. Ko and P. K. Chilana, "How power users help and hinder open bug reporting," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010, pp. 1665–1674.
- [5] D. Vyas, T. Fritz, and D. Shepherd, "Bug reproduction: A collaborative practice within software maintenance activities," in *COOP 2014-Proceedings of the 11th International Conference on the Design of Cooperative Systems, 27-30 May 2014, Nice (France)*, Springer, 2014, pp. 189–207.
- [6] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, "Auto-completing bug reports for android applications," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 673–686.
- [7] M. Erfani Joorabchi, M. Mirzaaghaei, and A. Mesbah, "Works for me! characterizing non-reproducible bug reports," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 62–71.
- [8] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, "AR-miner: mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 767–778.
- [9] M. E. Joorabchi, A. Mesbah, and P. Kruchten, "Real challenges in mobile app development," in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, IEEE, 2013, pp. 15–24.
- [10] S. Just, R. Premraj, and T. Zimmermann, "Towards the next generation of bug tracking systems," in *2008 IEEE symposium on visual languages and human-centric computing*, IEEE, 2008, pp. 82–85.
- [11] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *J. Syst. Softw.*, vol. 117, pp. 166–184, 2016.
- [12] A. G. Koru and J. Tian, "Defect handling in medium and large open source projects," *IEEE Softw.*, vol. 21, no. 4, pp. 54–61, 2004.
- [13] E. I. Laukkanen and M. V. Mantyla, "Survey reproduction of defect reporting in industrial software development," in *2011 International Symposium on Empirical Software Engineering and Measurement*, IEEE, 2011, pp. 197–206.
- [14] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What Makes a Good Bug Report?," *IEEE Trans. Softw. Eng.*, vol. 36, no. 5, pp. 618–643, Sep. 2010, doi: 10.1109/TSE.2010.63.
- [15] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, *Information Needs in Bug Reports: Improving Cooperation Between Developers and Users*.

- [16] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *2012 34th International Conference on Software Engineering (ICSE)*, IEEE, 2012, pp. 1074–1083.
- [17] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering-Volume 1*, 2010, pp. 495–504.
- [18] M. El Mezouar, F. Zhang, and Y. Zou, "Are tweets useful in the bug fixing process? An empirical study on Firefox and Chrome," *Empir. Softw. Eng.*, vol. 23, no. 3, pp. 1704–1742, 2018.
- [19] N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann, "Quality of Bug Reports in Eclipse," in *Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology eXchange*, in eclipse '07. New York, NY, USA: ACM, 2007, pp. 21–25. doi: 10.1145/1328279.1328284.
- [20] M. R. Karim, A. Ihara, X. Yang, H. Iida, and K. Matsumoto, "Understanding key features of high-impact bug reports," in *2017 8th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, IEEE, 2017, pp. 53–58.
- [21] P. Bhattacharya, L. Ulanova, I. Neamtii, and S. C. Koduru, "An empirical analysis of bug reports and bug fixing in open source android apps," in *2013 17th European Conference on Software Maintenance and Reengineering*, IEEE, 2013, pp. 133–143.
- [22] G. Yang, T. Zhang, and B. Lee, "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports," in *2014 IEEE 38th Annual Computer Software and Applications Conference*, IEEE, 2014, pp. 97–106.
- [23] M. Alenezi and S. Banitaan, "Bug reports prioritization: Which features and classifier to use?," in *2013 12th International Conference on Machine Learning and Applications*, IEEE, 2013, pp. 112–116.
- [24] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empir. Softw. Eng.*, vol. 20, no. 5, pp. 1354–1383, 2015.
- [25] P. Hooimeijer and W. Weimer, "Modeling Bug Report Quality," in *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, in ASE '07. New York, NY, USA: ACM, 2007, pp. 34–43. doi: 10.1145/1321631.1321639.
- [26] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, IEEE, 2012, pp. 70–79.
- [27] B. S. Neysiani and S. M. Babamir, "Automatic Duplicate Bug Report Detection using Information Retrieval-based versus Machine Learning-based Approaches," in *IEEE 6th International Conference on Web Research (ICWR)*, 2020.
- [28] J. Deshmukh, S. Podder, S. Sengupta, N. Dubash, and others, "Towards accurate duplicate bug retrieval using deep learning techniques," in *2017 IEEE International conference on software maintenance and evolution (ICSME)*, IEEE, 2017, pp. 115–124.
- [29] L. L. Wu, B. Xie, G. E. Kaiser, and R. Passonneau, "BUGMINER: Software Reliability Analysis Via Data Mining of Bug Reports," 2011, doi: 10.7916/D8W95JCN.
- [30] A. J. Ko, B. A. Myers, and D. H. Chau, "A linguistic analysis of how people describe software problems," in *Visual Languages and Human-Centric Computing (VL/HCC'06)*, IEEE, 2006, pp. 127–134.

- [31] A. Sureka and K. V. Indukuri, "Linguistic analysis of bug report titles with respect to the dimension of bug importance," in *Proceedings of the Third Annual ACM Bangalore Conference*, 2010, pp. 1–6.
- [32] P. Schuegerl, J. Rilling, and P. Charland, "Enriching SE ontologies with bug report quality," in *Proc. 4th International Workshop on Semantic Web Enabled Software Engineering (SWESE'08)*, 2008.
- [33] O. Chaparro *et al.*, "Detecting missing information in bug descriptions," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 396–407.
- [34] O. Chaparro *et al.*, "Assessing the quality of the steps to reproduce in bug reports," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 86–96.
- [35] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What Makes a Good Bug Report?," *IEEE Trans. Softw. Eng.*, vol. 36, no. 5, pp. 618–643, Sep. 2010, doi: 10.1109/TSE.2010.63.
- [36] D. Lin, C.-P. Bezemer, and A. E. Hassan, "Identifying gameplay videos that exhibit bugs in computer games," *Empir. Softw. Eng.*, vol. 24, no. 6, pp. 4006–4033, 2019.
- [37] L. B. Jacob, T. C. Kohwalter, A. F. Machado, E. W. Clua, and D. De Oliveira, "A non-intrusive approach for 2d platform game design analysis based on provenance data extracted from game streaming," in *2014 Brazilian Symposium on Computer Games and Digital Entertainment*, IEEE, 2014, pp. 41–50.
- [38] Z. Luo, M. Guzdial, N. Liao, and M. Riedl, "Player experience extraction from gameplay video," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2018.
- [39] C. Lewis, J. Whitehead, and N. Wardrip-Fruin, "What went wrong: a taxonomy of video game bugs," in *Proceedings of the fifth international conference on the foundations of digital games*, 2010, pp. 108–115.
- [40] S. Varvaressos, K. Lavoie, S. Gaboury, and S. Hallé, "Automated bug finding in video games: A case study for runtime monitoring," *Comput. Entertain. CIE*, vol. 15, no. 1, pp. 1–28, 2017.
- [41] S. Ariyurek, A. Betin-Can, and E. Surer, "Automated video game testing using synthetic and human-like agents," *IEEE Trans. Games*, 2019.
- [42] M. Sjöblom and J. Hamari, "Why do people watch others play video games? An empirical study on the motivations of Twitch users," *Comput. Hum. Behav.*, vol. 75, pp. 985–996, 2017.
- [43] P. Krieter and A. Breiter, "Analyzing mobile application usage: generating log files from mobile screen recordings," in *Proceedings of the 20th international conference on human-computer interaction with mobile devices and services*, 2018, pp. 1–10.
- [44] N. Cooper, C. Bernal-Cárdenas, O. Chaparro, K. Moran, and D. Poshyvanyk, "It Takes Two to Tango: Combining Visual and Textual Information for Detecting Duplicate Video-Based Bug Reports," *ArXiv Prepr. ArXiv210109194*, 2021.
- [45] G. Hu, L. Zhu, and J. Yang, "AppFlow: using machine learning to synthesize robust, reusable UI tests," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 269–282.

- [46] C. Bernal-Cárdenas, N. Cooper, K. Moran, O. Chaparro, A. Marcus, and D. Poshyvanyk, “Translating video recordings of mobile app usages into replayable scenarios,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 309–321.
- [47] S. Feng and C. Chen, “GIFdroid: Automated Replay of Visual Bug Reports for Android Apps,” *ArXiv Prepr. ArXiv211204128*, 2021.
- [48] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, “‘Not my bug!’ and other reasons for software bug report reassignments,” in *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, 2011, pp. 395–404.
- [49] J. Zhou, H. Zhang, and D. Lo, “Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports,” in *2012 34th International Conference on Software Engineering (ICSE)*, IEEE, 2012, pp. 14–24.
- [50] A. T. Nguyen, T. T. Nguyen, J. Al-Kofahi, H. V. Nguyen, and T. N. Nguyen, “A topic-based approach for narrowing the search space of buggy files from a bug report,” in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, IEEE, 2011, pp. 263–272.
- [51] “Bugzilla Helper.” <https://www-archive.mozilla.org/events/dev-day2001/community-testing/bugzilla-helper> (accessed Oct. 03, 2022).
- [52] “Bug Writing Guidelines.” <https://bugzilla.mozilla.org/page.cgi?id=bug-writing.html> (accessed Oct. 11, 2022).
- [53] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, “How long will it take to fix this bug?,” in *Fourth International Workshop on Mining Software Repositories (MSR’07: ICSE Workshops 2007)*, IEEE, 2007, pp. 1–1.
- [54] “Life Cycle of a Bug.” <https://www.bugzilla.org/docs/2.18/html/lifecycle.html> (accessed Oct. 05, 2022).
- [55] S. Davies and M. Roper, “What’s in a Bug Report?,” in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, in ESEM ’14. New York, NY, USA: ACM, 2014, p. 26:1-26:10. doi: 10.1145/2652524.2652541.
- [56] P. Flach, *Machine learning: the art and science of algorithms that make sense of data*. Cambridge university press, 2012.
- [57] “Multinomial Naive Bayes classifier,” *scikit-learn*. [https://scikit-learn/stable/modules/naive\\_bayes.html](https://scikit-learn/stable/modules/naive_bayes.html) (accessed Sep. 29, 2022).
- [58] D. D. Lewis, “Naive (Bayes) at forty: The independence assumption in information retrieval,” in *European conference on machine learning*, Springer, 1998, pp. 4–15.
- [59] “Support Vector Machine,” *scikit-learn*. <https://scikit-learn/stable/modules/generated/sklearn.svm.SVC.html> (accessed Sep. 29, 2022).
- [60] T. Joachims, “Text categorization with support vector machines: Learning with many relevant features,” in *European conference on machine learning*, Springer, 1998, pp. 137–142.
- [61] “Decision Trees,” *scikit-learn*. <https://scikit-learn/stable/modules/tree.html> (accessed Sep. 29, 2022).
- [62] X.-E. Pantazi, D. Moshou, and D. Bochtis, *Intelligent Data Mining and Fusion Systems in Agriculture*. Academic Press, 2019.
- [63] “Random Forest.” 2022. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

- [64] N. Japkowicz and M. Shah, *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.
- [65] P. Probst, A.-L. Boulesteix, and B. Bischl, “Tunability: Importance of hyperparameters of machine learning algorithms,” *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 1934–1965, 2019.
- [66] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *J. Mach. Learn. Res.*, vol. 13, no. Feb, pp. 281–305, 2012.
- [67] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*, vol. 112. Springer, 2013.
- [68] F. Rahman and P. Devanbu, “How, and why, process metrics are better,” in *Software Engineering (ICSE), 2013 35th International Conference on*, IEEE, 2013, pp. 432–441.
- [69] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, “Cross-project defect prediction: a large scale experiment on data vs. domain vs. process,” in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 91–100.
- [70] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, “Local vs. global models for effort estimation and defect prediction,” in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, IEEE, 2011, pp. 343–351.
- [71] R. Feldman and J. Sanger, *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge university press, 2007.
- [72] G. Williams, *Data mining with Rattle and R: The art of excavating data for knowledge discovery*. Springer Science & Business Media, 2011.
- [73] J. Ramos and others, “Using tf-idf to determine word relevance in document queries,” in *Proceedings of the first instructional conference on machine learning*, New Jersey, USA, 2003, pp. 133–142.
- [74] J. Demšar, “Statistical comparisons of classifiers over multiple data sets,” *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.
- [75] B. Boehm and V. R. Basili, “Software defect reduction top 10 list,” *Softw. Eng. Barry W Boehms Lifetime Contrib. Softw. Dev. Manag. Res.*, vol. 34, no. 1, pp. 426–431, 2007.
- [76] N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann, “What makes a good bug report?,” in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008, pp. 308–318.
- [77] M. Soltani, F. Hermans, and T. Bäck, “The significance of bug report elements,” *Empir. Softw. Eng.*, vol. 25, no. 6, pp. 5255–5294, 2020.
- [78] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, “Duplicate bug reports considered harmful... really?,” in *2008 IEEE International Conference on Software Maintenance*, IEEE, 2008, pp. 337–345.
- [79] G. Antoniol, M. Di Penta, H. Gall, and M. Pinzger, “Towards the integration of versioning systems, bug reports and source code meta-models,” *Electron. Notes Theor. Comput. Sci.*, vol. 127, no. 3, pp. 87–99, 2005.
- [80] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*, vol. 39. Cambridge University Press Cambridge, 2008.
- [81] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk, “Crashscope: A practical tool for automated testing of android applications,” in *2017*

- IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, IEEE, 2017, pp. 15–18.
- [82] A. Gromova, I. Itkin, S. Pavlov, and A. Korovayev, “Raising the Quality of Bug Reports by Predicting Software Defect Indicators,” in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, IEEE, 2019, pp. 198–204.
- [83] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: synthetic minority over-sampling technique,” *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.
- [84] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, “Categorizing bugs with social networks: A case study on four open source software communities,” in *2013 35th International Conference on Software Engineering (ICSE)*, May 2013, pp. 1032–1041. doi: 10.1109/ICSE.2013.6606653.
- [85] S. Zaman, B. Adams, and A. E. Hassan, “Security versus performance bugs: a case study on firefox,” in *Proceedings of the 8th working conference on mining software repositories*, 2011, pp. 93–102.
- [86] H. Kuramoto *et al.*, “Do visual issue reports help developers fix bugs?: a preliminary study of using videos and images to report issues on GitHub,” in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, Virtual Event: ACM, May 2022, pp. 511–515. doi: 10.1145/3524610.3527882.
- [87] J. Xie, M. Zhou, and A. Mockus, “Impact of triage: a study of mozilla and gnome,” in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, IEEE, 2013, pp. 247–250.
- [88] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, “Is It a Bug or an Enhancement?: A Text-based Approach to Classify Change Requests,” in *Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, in CASCON ’08. New York, NY, USA: ACM, 2008, p. 23:304-23:318.
- [89] T. Menzies and A. Marcus, “Automated severity assessment of software defect reports,” in *2008 IEEE International Conference on Software Maintenance*, IEEE, 2008, pp. 346–355.
- [90] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, “Predicting the severity of a reported bug,” in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, IEEE, 2010, pp. 1–10.
- [91] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, “Comparing mining algorithms for predicting the severity of a reported bug,” in *2011 15th European Conference on Software Maintenance and Reengineering*, IEEE, 2011, pp. 249–258.
- [92] C.-Z. Yang, C.-C. Hou, W.-C. Kao, and X. Chen, “An empirical study on improving severity prediction of defect reports using feature selection,” in *2012 19th Asia-Pacific Software Engineering Conference*, IEEE, 2012, pp. 240–249.
- [93] Y. Tian, N. Ali, D. Lo, and A. E. Hassan, “On the unreliability of bug severity data,” *Empir. Softw. Eng.*, vol. 21, no. 6, pp. 2298–2323, 2016.
- [94] Y. Tan, S. Xu, Z. Wang, T. Zhang, Z. Xu, and X. Luo, “Bug severity prediction using question-and-answer pairs from Stack Overflow,” *J. Syst. Softw.*, vol. 165, p. 110567, 2020.
- [95] S. E. Robertson and S. Walker, “Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval,” in *SIGIR’94*, Springer, 1994, pp. 232–241.

- [96] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Adv. Neural Inf. Process. Syst.*, vol. 26, 2013.
- [97] Y. Tian, D. Lo, and C. Sun, "Drone: Predicting priority of reported bugs by multi-factor analysis," in *2013 IEEE International Conference on Software Maintenance*, IEEE, 2013, pp. 200–209.
- [98] W. Abdelmoez, M. Kholief, and F. M. Elsalmy, "Bug fix-time prediction model using naïve Bayes classifier," in *2012 22nd International Conference on Computer Theory and Applications (ICCTA)*, IEEE, 2012, pp. 167–172.
- [99] Q. Umer, H. Liu, and I. Illahi, "CNN-based automatic prioritization of bug reports," *IEEE Trans. Reliab.*, vol. 69, no. 4, pp. 1341–1354, 2019.
- [100] G. Murphy and D. Cubranic, "Automatic bug triage using text categorization," in *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, Citeseer, 2004.
- [101] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 361–370.
- [102] T. Zhang and B. Lee, "A hybrid bug triage algorithm for developer recommendation," in *Proceedings of the 28th annual ACM symposium on applied computing*, 2013, pp. 1088–1094.
- [103] A. Yadav, S. K. Singh, and J. S. Suri, "Ranking of software developers based on expertise score for bug triaging," *Inf. Softw. Technol.*, vol. 112, pp. 1–17, 2019.
- [104] X. Peng, P. Zhou, J. Liu, and X. Chen, "Improving Bug Triage with Relevant Search.," in *SEKE*, 2017, pp. 123–128.
- [105] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *29th International Conference on Software Engineering (ICSE'07)*, IEEE, 2007, pp. 499–510.
- [106] K. Aggarwal, F. Timbers, T. Rutgers, A. Hindle, E. Stroulia, and R. Greiner, "Detecting duplicate bug reports with software engineering domain knowledge," *J. Softw. Evol. Process*, vol. 29, no. 3, p. e1821, 2017.
- [107] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," in *2013 10th Working Conference on Mining Software Repositories (MSR)*, IEEE, 2013, pp. 183–192.
- [108] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, 2010, pp. 45–54.
- [109] J. Zhou and H. Zhang, "Learning to rank duplicate bug reports," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 852–861.
- [110] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, IEEE, 2011, pp. 253–262.
- [111] M. Nayebi, "Eye of the mind: image processing for social coding," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, 2020, pp. 49–52.
- [112] "In-app bug reporting and user feedback for mobile apps | BugClipper." [Online]. Available: <https://bugclipper.com/>

- [113] “Bug and crash reporting for iOS and Android | Bugsee.” [Online]. Available: <https://www.bugsee.com/>
- [114] “TestFairy - Mobile testing.” [Online]. Available: <https://www.testfairy.com/>
- [115] “Bird Eats Bug: Report and fix bugs up to 50% faster.” [Online]. Available: <https://birdeatsbug.com/>
- [116] “The Bug Squasher - Powerful Issue Tracker.” [Online]. Available: <https://thebugsquasher.com/>
- [117] “Experience Faster In-App Bug Reporting for Mobile Apps | Instabug.” [Online]. Available: <https://instabug.com/product/bug-reporting>
- [118] “ScreenCast Videos for Online Learning | Outklip Video Recorder and Editor.” [Online]. Available: <https://outklip.com/>
- [119] “The most efficient bug reporting tool for your project. Ubertesters.” [Online]. Available: <https://ubertesters.com/bug-reporting-tools/>
- [120] C.-C. Lin, H. Li, X. Zhou, and X. Wang, “Screenmilk: How to Milk Your Android Screen for Secrets,” in *NDSS*, 2014.
- [121] L. Bao, J. Li, Z. Xing, X. Wang, and B. Zhou, “scvRipper: video scraping tool for modeling developers’ behavior using interaction data,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, IEEE, 2015, pp. 673–676.
- [122] C. Frisson, S. Malacria, G. Bailly, and T. Dutoit, “Inspectorwidget: A system to analyze users behaviors in their applications,” in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 2016, pp. 1548–1554.
- [123] H. Hu, H. Zhang, J. Xuan, and W. Sun, “Effective bug triage based on historical bug-fix information,” in *2014 IEEE 25th International Symposium on Software Reliability Engineering*, IEEE, 2014, pp. 122–132.
- [124] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen, “Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement,” *Sociol. Methods Res.*, vol. 42, no. 3, pp. 294–320, 2013.
- [125] T. Zimmermann and N. Nagappan, “Predicting defects using network analysis on dependency graphs,” in *Software Engineering, 2008. ICSE’08. ACM/IEEE 30th International Conference on*, IEEE, 2008, pp. 531–540.
- [126] J. Noll, S. Beecham, and D. Seichter, “A qualitative study of open source software development: The open EMR project,” in *2011 International Symposium on Empirical Software Engineering and Measurement*, IEEE, 2011, pp. 30–39.
- [127] M. L. McHugh, “Interrater reliability: the kappa statistic,” *Biochem. Medica*, vol. 22, no. 3, pp. 276–282, 2012.
- [128] J. R. Landis and G. G. Koch, “An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers,” *Biometrics*, pp. 363–374, 1977.
- [129] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
- [130] J. P. Kincaid, R. P. Fishburne Jr, R. L. Rogers, and B. S. Chissom, “Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel,” Naval Technical Training Command Millington TN Research Branch, 1975.
- [131] L. Passos *et al.*, “A study of feature scattering in the linux kernel,” *IEEE Trans. Softw. Eng.*, 2018.



- [132] F. Medeiros *et al.*, “Discipline matters: Refactoring of preprocessor directives in the #ifdef hell,” *IEEE Trans. Softw. Eng.*, vol. 44, no. 5, pp. 453–469, 2017.
- [133] W. Fu, T. Menzies, and X. Shen, “Tuning for software analytics: Is it really necessary?,” *Inf. Softw. Technol.*, vol. 76, pp. 135–146, 2016.
- [134] P. B. Miranda and R. B. Prudêncio, “Active testing for SVM parameter selection,” in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2013, pp. 1–8.
- [135] L. C. Padierna, M. Carpio, A. Rojas, H. Puga, R. Baltazar, and H. Fraire, “Hyperparameter tuning for support vector machines by estimation of distribution algorithms,” in *Nature-Inspired Design of Hybrid Intelligent Systems*, Springer, 2017, pp. 787–800.
- [136] D. Anguita, S. Ridella, F. Riviuccio, and R. Zunino, “Automatic hyperparameter tuning for support vector machines,” in *International Conference on Artificial Neural Networks*, Springer, 2002, pp. 1345–1350.
- [137] A. Altmann, L. Tološi, O. Sander, and T. Lengauer, “Permutation importance: a corrected feature importance measure,” *Bioinformatics*, vol. 26, no. 10, pp. 1340–1347, 2010.
- [138] P. Bhattacharya and I. Neamtiu, “Bug-fix time prediction models: can we do better?,” in *Proceedings of the 8th Working Conference on Mining Software Repositories*, 2011, pp. 207–210.
- [139] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, “Extracting Structural Information from Bug Reports,” in *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, in MSR ’08. New York, NY, USA: ACM, 2008, pp. 27–30. doi: 10.1145/1370750.1370757.
- [140] “Mr. Tappy - Mobile UX research made easy.,” *Mr Tappy*. <https://www.mrtappy.com/> (accessed Nov. 06, 2022).
- [141] K. Mao, M. Harman, and Y. Jia, “Crowd intelligence enhances automated mobile testing,” in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2017, pp. 16–26.
- [142] R. Schusteritsch, C. Y. Wei, and M. LaRosa, “Towards the perfect infrastructure for usability testing on mobile devices,” in *CHI’07 extended abstracts on Human factors in computing systems*, 2007, pp. 1839–1844.
- [143] G. Catolino, F. Palomba, A. Zaidman, and F. Ferrucci, “Not all bugs are the same: Understanding, characterizing, and classifying bug types,” *J. Syst. Softw.*, vol. 152, pp. 165–181, 2019.
- [144] W. Poncin, A. Serebrenik, and M. Van Den Brand, “Process mining software repositories,” in *2011 15th European conference on software maintenance and reengineering*, IEEE, 2011, pp. 5–14.
- [145] M. Gupta and A. Sureka, “Nirikshan: Mining bug report history for discovering process maps, inefficiencies and inconsistencies,” in *Proceedings of the 7th India Software Engineering Conference*, 2014, pp. 1–10.
- [146] Y. Lee, S. Lee, C.-G. Lee, I. Yeom, and H. Woo, “Continual prediction of bug-fix time using deep learning-based activity stream embedding,” *IEEE Access*, vol. 8, pp. 10503–10515, 2020.

- [147] R. Sepahvand, R. Akbari, and S. Hashemi, "Predicting the bug fixing time using word embedding and deep long short term memories," *IET Softw.*, vol. 14, no. 3, pp. 203–212, 2020.
- [148] M. Habayeb, S. S. Murtaza, A. Miranskyy, and A. B. Bener, "On the use of hidden markov model to predict the time to fix bugs," *IEEE Trans. Softw. Eng.*, vol. 44, no. 12, pp. 1224–1244, 2017.
- [149] D. Thissen, L. Steinberg, and D. Kuang, "Quick and easy implementation of the Benjamini-Hochberg procedure for controlling the false positive rate in multiple comparisons," *J. Educ. Behav. Stat.*, vol. 27, no. 1, pp. 77–83, 2002.
- [150] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken, *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge, 2013.
- [151] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan, "An empirical study on factors impacting bug fixing time," in *2012 19th Working conference on reverse engineering*, IEEE, 2012, pp. 225–234.
- [152] L. D. Panjer, "Predicting eclipse bug lifetimes," in *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*, IEEE, 2007, pp. 29–29.
- [153] R. Kikas, M. Dumas, and D. Pfahl, "Using dynamic and contextual features to predict issue lifetime in GitHub projects," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, IEEE, 2016, pp. 291–302.
- [154] W. H. A. Al-Zubaidi, H. K. Dam, A. Ghose, and X. Li, "Multi-objective search-based approach to estimate issue resolution time," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2017, pp. 53–62.
- [155] P. Cohen, S. G. West, and L. S. Aiken, *Applied multiple regression/correlation analysis for the behavioral sciences*. Psychology press, 2014.
- [156] S. Hu, "Akaike information criterion," *Cent. Res. Sci. Comput.*, vol. 93, 2007.
- [157] I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles, "Towards a simplification of the bug report form in eclipse," in *Proceedings of the 2008 international working conference on Mining software repositories*, 2008, pp. 145–148.
- [158] D. A. Hensher and P. R. Stopher, *Behavioural travel modelling*. Routledge, 2021.
- [159] H. B. Mann, "Nonparametric tests against trend," *Econom. J. Econom. Soc.*, pp. 245–259, 1945.
- [160] F. Thung, "Automatic prediction of bug fixing effort measured by code churn size," in *Proceedings of the 5th international workshop on software mining*, 2016, pp. 18–23.
- [161] "Mr. Tappy - Mobile UX research made easy." [Online]. Available: <https://www.mrtappy.com/>