

Understanding the Characteristics and the Role of Visual Issue Reports

Hiroki Kuramoto · Dong Wang ✉ · Masanari Kondo · Yutaro Kashiwa · Yasutaka Kamei · Naoyasu Ubayashi

Received: date / Accepted: date

Abstract Issue reports are a pivotal interface between developers and users for receiving information about bugs in their products. In practice, reproducing those bugs is challenging, since issue reports often contain incorrect information or lack sufficient information. Furthermore, the poor quality of issue reports would have the effect of delaying the entire bug-fixing process. To enhance bug comprehension and facilitate bug reproduction, GitHub Issue allows users to embed visuals such as images and videos to complement the textual description. Hence, we conduct an empirical study on 34 active GitHub repositories to quantitatively analyze the difference between visual issue reports and non-visual ones, and qualitatively analyze the characteristics of visuals and the usage of visuals in bug types. Our results show that visual issue reports have a significantly higher probability of reporting bugs. Visual reports also tend to receive the first comment and complete the conversation in a relatively shorter time. Visuals are frequently used to present the program behavior and the user interface, with the major purpose of introducing problems in reports. Additionally, we observe that visuals are commonly used to report GUI-related bugs, but they are rarely used to report configuration bugs in comparison to non-visual issue reports. To summarize, our work highlights the role of visual play in the bug-fixing process and lays the foundation for future research to support bug comprehension by exploiting visuals.

Keywords Visual Issue Reports, GitHub, Mining Software Repositories

✉ Corresponding author - Dong Wang
College of Intelligence and Computing, Tianjin University, China.
E-mail: d.wang@ait.kyushu-u.ac.jp

Hiroki Kuramoto, Masanari Kondo, Yasutaka Kamei, Naoyasu Ubayashi
Kyushu University, Japan
E-mail: kuramoto@posl.ait.kyushu-u.ac.jp, {d.wang, kondo, kamei, ubayashi}@ait.kyushu-u.ac.jp

Yutaro Kashiwa
Nara Institute of Science and Technology, Japan
E-mail: yutaro.kashiwa@is.naist.jp

1 Introduction

The question “*What makes a good issue report?*” has been studied for decades and is still the ultimate research question for many studies that aim to improve the quality of issue reports (Bettenburg et al., 2007; Herzig et al., 2013; Zimmermann et al., 2010). Issue reports (a.k.a. bug reports) often lack the necessary information for developers to reproduce bugs (Joorabchi et al., 2014). For example, Zimmermann et al. (2010) reported that stack traces and steps for reproducing a bug are considered to be helpful by developers. The developer interview conducted by Soltani et al. (2020) similarly stated that crash description, reproducing steps or test cases, stack traces, and user contents are the important elements in bug reports. Moreover, their empirical results showed that on average, over 70% of bug reports lack these elements. However, it is difficult for users to provide the needed information in practice, and the information is often insufficient or incorrect. This mismatch between what developers need and what reporters can provide would further delay the fixing process of bugs (Joorabchi et al., 2014). Specifically, numerous studies have demonstrated that the quality of issue reports impacts both the issue resolution time (Breu et al., 2010; Guo et al., 2010) and the issue resolution rate (Zimmermann et al., 2012; Zou et al., 2015).

To facilitate developers’ bug-reproduction work, GitHub Issue includes features designed for users to embed visual content, such as images and videos. For instance, since May 2021,¹ GitHub launched a new feature that allows users to share videos (e.g., mp4 files). By utilizing these videos, developers can create detailed bug reports by recording the symptoms, reproduction steps, and other important aspects of the issue. Visual content can help developers understand the nature of the bug and what users were doing when the bug occurred. Recent work also observed that developers are increasingly sharing visual content in social coding environments (e.g., Bugzilla and Stack Overflow) and developers regard visual content as an important element (Nayebi, 2020; Wang et al., 2023). Meanwhile, Taesiri et al. (2022) confirmed that utilizing the information from the gameplay videos is useful in identifying bugs. However, few studies empirically analyze the impact of these visuals on the bug-fixing process and the role that visuals play in understanding issue reports.

A comprehensive understanding of the visual content would provide valuable insights into facilitating OSS members in engaging in more efficient discussions and researchers in developing automated tools to extract useful knowledge (e.g., reproduction information) during the issue resolution process.

In this paper, we conducted an empirical study of 683,810 issue reports from 34 GitHub repositories to demystify the popularity of visual content in GitHub Issues, quantitatively analyze the difference between visual issue reports and non-visual ones, and qualitatively analyze the characteristics and the usage of the visual content. Four research questions are formulated to guide this study:

RQ1: To what extent are visual issue reports used for reporting bugs?

Motivation: Issue reports on GitHub are used for a variety of purposes, including bug reporting, questions, feature requests, and development management. Especially bug reporting requires precise and accurate information to reproduce unexpected be-

¹ <https://github.blog/2021-05-13-video-uploads-available-github>

havior (Joorabchi et al., 2014). We assume that developers may tend to report bugs more frequently using visuals, as visuals are likely to provide more comprehensive information. Therefore, in this RQ, we investigate the difference in the frequency of bug reporting between visual issue reports and non-visual ones.

Results: Our results suggest a greater preference for the use of visuals when reporting issues related to bugs. Specifically, compared to issue reports without visuals, the ones that contain images or videos have a significantly higher probability of reporting bugs, 44.6% and 71.1% (roughly 1.4–2.2 times) being identified.

RQ2: What is the relationship between visuals and the bug-fixing process?

Motivation: Existing work has extensively studied the effect of various elements on bug resolution (Soltani et al., 2020; Chen et al., 2021). However, the effect of visuals (images or videos), a popularly used non-textual element in modern society, is largely unknown. Thus, we would like to explore whether or not the presence of visuals also plays a role specifically in terms of three dimensions: report content, discussion process, and bug fixing.

Results: In terms of report content, compared to non-visual issue reports, the length of descriptions is relatively smaller in the visual ones. In terms of the discussion process, the statistical results confirm that the visual issue reports are likely to receive the first comments and complete the discussion conversation faster. In terms of bug fixing, it takes a relatively shorter time for issue reports with images to be closed.

RQ3: What are the contents and purposes of visuals used in the issue reports?

Motivation: Although the prior work reported that developers are increasingly using visuals in issue tracking systems (Nayebi, 2020), it remains unclear what characteristics these visuals have (e.g., contents and their purposes). Analyzing their characteristics will lay the foundation for understanding the role of visuals as complementary pieces of information when reporting bugs.

Results: Seven kinds of visual content and five kinds of purposes are classified from our manual analysis of 959 visuals. We observe that visuals are commonly used to depict the output of the program behavior and the user interface (for instance, 41.3% and 20.8% for images, respectively). **Results show that regardless of whether images or videos, visuals are commonly used to provide an example in a report, followed by helping to identify the issue cause.**

RQ4: What bug types are raised in visual issue reports?

Motivation: Several studies have generally categorized the bug types (Catolino et al., 2019; Nayrolles and Hamou-Lhadj, 2018). Considering the intuitive nature of visuals, we assume that visuals could be utilized in the specific bug types by developers. Answering this question would help developers gain a better understanding of the extent to which visuals are used in different bug types.

Results: Through a manual analysis of 1,124 issue reports (including the control group), results show that on the one hand visuals are much more common to be used in reporting GUI-related bugs, around 2–3 times. On the other hand, the second common bug type (i.e., Configuration bug) in non-visual reports is rarely being reported in visual ones (19.6% against 1.1%). It relatively takes less time for issue reports with images that report GUI-related and performance bugs to receive the first comment.

In summary, our contributions are three-fold: (I) we use quantitative analysis to shed light on the difference in effects between visual issue reports and non-visual

ones; (II) we use qualitative analysis to manually classify visual characteristics, which can be utilized for future large-scale studies using an automated classifier; (III) third, we highlight the role of visuals as an important element in understanding issue reports through qualitative analysis, complementing the knowledge regarding the good quality of a bug report. We provide a public replication package² to encourage future replication studies.

Paper Extension. This paper extends our previous study, as a short paper of an international conference (Kuramoto et al., 2022). For the studied dataset, we collected approximately twice as many issue reports from more active GitHub repositories. We newly proposed RQ1 to examine the bug rates of visual issue reports. In RQ2, we re-evaluated the relationship between visuals and the bug-fixing process by introducing additional metrics to gain a more comprehensive understanding, including the frequent words in the description, the last comment time, and the number of participants. RQ3 and RQ4 were completely new to this paper. We conducted a series of manual analyses to classify the characteristics (i.e., the contents and purposes) of visuals used in the issue reports, as well as the types of bugs raised in visual issue reports compared to non-visual ones. Additionally, we investigated the relationship between the characteristics and the bug types. Based on these new empirical results, we provide a more insightful discussion about the implications and challenges.

Paper Organization. The remainder of this paper is organized as follows. Section 2 describes the data preparation process. Sections 3 – 6 present the experiments that we conducted to address RQ1–RQ4 with their results, respectively. Section 7 discusses the main findings of our study with future work. Section 8 discloses threats to validity of our study. Section 9 situates this paper with respect to the related work on bug fixing time and non-textual information sharing in software development. Finally, we conclude the paper in Section 10.

2 Dataset Preparation

In this section, we first introduce the studied repositories, then describe the data collection and identification of visual reports, and finally present the dataset description.

Studied Repositories. To understand the characteristics of the issue reports that contain visual content, we mined GitHub, one of the most popular social coding platforms. In this study, we chose the repositories that actively maintain the issue tracker, employing the GitHub Search (Dabic et al., 2021). We assume that the active issue tracker is more likely to have visual issue reports and is representative to be investigated. GitHub Search can be queried through a web application that allows the selection of various combinations of criteria needed for a study. To make sure that the repositories perform the development actively, we collected those projects that contain more than 10,000 issue reports created during the last five years (since 2017), resulting in 108 repositories by the end of June 2022. In our preliminary study (Kuramoto et al., 2022), we observed that visual content started being attached to issue

² <https://doi.org/10.5281/zenodo.10565699>

Table 1 Statistical summary of studied dataset

Data Attributes	
Studied Period	2017 – 2022.06
# Studied GitHub Repositories	34
# Studied Issue Reports	481,603
# Issue Reports Contain Images	79,212 (17.10%)
# Issue Reports Contain Videos	9,615 (2.30%)
# Images per Issue Reports (1st Qu./Median/3rd Qu.)	1.0/1.0/2.0
# Videos per Issue Reports (1st Qu./Median/3rd Qu.)	1.0/1.0/1.0

reports around 2017. Hence, we decided to limit our dataset collection to the most recent five-year period. We also argue that the more recent the issue report, the more likely it is to have a visual attached to it. During our preliminary work, we also realized that one threat exists, where the pull requests could be wrongly counted as issue reports that were retrieved from the GitHub Search. Then, for each of the issue report candidates from 108 repositories, by checking whether the issue contains pull request key,³ we automatically distinguished them into issue reports (i.e., true positives) and pull requests (i.e., false positives). After we excluded the false positives, finally 34 repositories met the criteria, i.e., more than 10,000 issue reports were created since 2017.

Data Collection and Cleaning. For the remaining 34 selected repositories, we then retrieved all the issue reports and their metadata (e.g., title and description, created time, comment time), using `get_issue` function provided by PyGitHub⁴ that internally executes GitHub API v3. To ensure the quality of the studied issue reports, we did the following five filters. *First of all*, we filtered out those issue reports with open status since we can not calculate the resolution time of these issue reports. We were able to collect 683,810 closed issue reports. *Secondly*, we filtered out those issue reports that were written in languages other than English and therefore excluded 11,856 non-English issue reports. *Third*, we filtered out those issue reports that were resolved in a very short period, because developers sometimes report issues after treatment. Thus, we excluded another 5,541 issue reports that were resolved within 30 seconds. *Fourth*, we filtered out issue reports submitted by bots. To do so, we referred to the work of Golzadeh et al. (2022), which systematically compares the performance of the existing bot detection techniques, and we leveraged the combination of two bot detection techniques with the highest precision known as “bot” suffix and list of bots. “bot” suffix is a technique that relies on the presence of the string “bot” at the end of the author’s name, which has notably been used by other researches (Dey et al., 2020; Saadat et al., 2021). The list of bots refers to the technique that relies on a predefined list of ground-truth bots manually validated by Golzadeh et al. (2021). We excluded 158,506 issue reports submitted by bots. *Finally*, based on the tags provided by GitHub, we filtered out those invalid issue reports. We excluded 24,904 issue reports with tags that indicate invalid issues (i.e., “duplicated” and “invalid”).

³ <https://github.com/PyGithub/PyGithub/issues/2206>

⁴ <https://pygithub.readthedocs.io/en/latest/index.html>

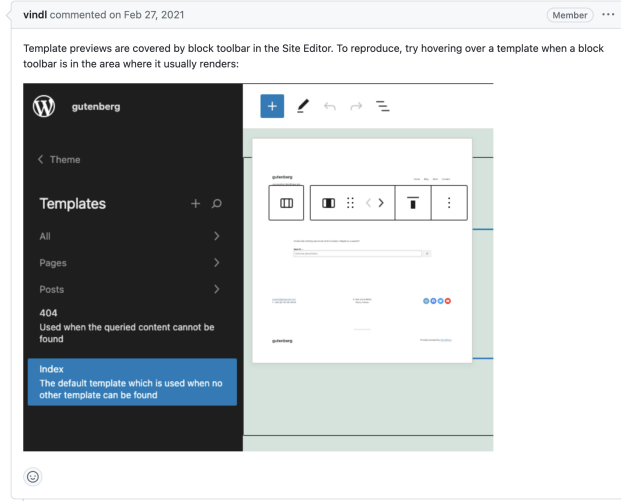


Fig. 1 A visual issue report containing an image (<https://github.com/wordpress/gutenberg/issues/29392>).

Note that tags may be changed to an arbitrary notation (e.g., “Type:duplicated” and “State:duplicated”). Hence, we further excluded issue reports in the same way notations containing strings that match those tags. 483,003 issue reports survived and were used in our subsequent analyses, as shown in Table 1.

Identification of Visual Issue Reports. To identify the visual contents (i.e., images and videos) that are embedded in the textual description of an issue report, we applied regular expressions (i.e., [https://\(user-images|cloud|camo|raw\)\.githubusercontent\.com/\[a-zA-Z0-9\-_\]/+\\[a-zA-Z0-9\\]+](https://(user-images|cloud|camo|raw)\.githubusercontent\.com/[a-zA-Z0-9\-_]/+\[a-zA-Z0-9\]+)) to search for hyperlinks that indicate visual contents. Specifically, for an image, we used a list of image file extension types (i.e., “png”, “PNG”, “jpg”, “JPG”, “jpeg”, and “JPEG”) to retrieve image-related links. While, for an animation (i.e., GIF) or a video, we used a list of file extension types (i.e., “gif”, “GIF”, “mp4”, “MP4”, “mov”, “MOV”, “webm”, and “WEBM”) to retrieve animation-related or video-related links. We noticed that an issue report could contain both an image and an animation/video. In our work, we consider videos and images as two distinct visual types being investigated. They may have different roles in the bug-reporting process, with videos often providing richer information. To eliminate confounding factors influenced by each other, we excluded these 1,400 issue reports from both 80,612 issue reports containing images and 11,015 issue reports containing animations/videos. In the end, as shown in Table 1, we were able to collect 125,355 images from 79,212 issue reports and 10,821 animations/videos from 9,615 issue reports. Figure 1 presents an issue report containing an image to display the problematic user interface. Any issue reports without image/video-related hyperlinks in the descriptions were classified as non-visual issue reports, resulting in 392,776 reports.

Popularity of Visual Issue Reports. We now briefly characterize the popularity of issue reports that contain visual content. Figure 2 depicts the trend of the studied issue reports per year during our time span, i.e., between 2017 and June 2022. The figure shows that the ratio of issue reports that have visual content (either images or [animations/videos](#)) out of all issue reports generally follows an upward trend. Specifically, the ratio of visuals reached 27.4% in 2021, compared to 14.9% in 2017. The relatively low ratio in 2022 compared to 2021 is reasonable, as our data collection period only covered half a year. Interestingly, we find that although GitHub officially launched the feature to share videos in May 2021, developers had already embedded animations with issue reports before that. During the manual inspection, we observe that these videos are related to GIF files.

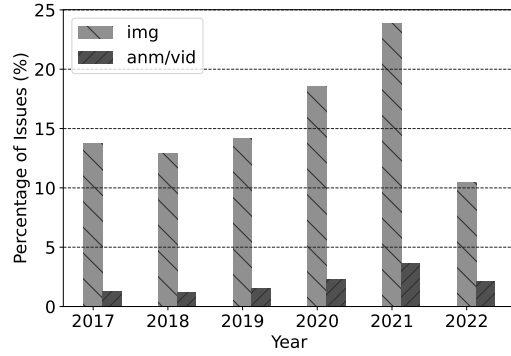


Fig. 2 The proportion of visual issue reports (i.e., images and [animations/videos](#)) per year.

3 RQ1: To what extent are visual issue reports used for reporting bugs?

GitHub issue reports are served for diverse purposes, such as bug reporting, questions, feature requests, and development management. Since reporting a bug requires more precise and accurate information than other purposes, developers could be more likely to use visual content due to its advantage of conveying comprehensive information. Hence, in this section, we first set out to explore how frequently visuals contribute to bug reporting. Below, we present our approach to analyzing the usage of visual contents in issue reports (Section 3.1), followed by the results (Section 3.2).

3.1 Approach

To address RQ1, we conduct a quantitative analysis to determine the extent to which [visual issue reports contribute to bug reporting](#). Similar to our exploratory study ([Kurahato et al., 2022](#)), we divided the visual contents into two kinds: Image (*Img*)

and [Animation/Video](#) (*Anm/Vid*), since we conjecture that the image and the [animation](#)/video play different roles. We used the 481,603 issue reports from the data collection, comprising 79,212 issue reports containing images and 9,615 issue reports containing [animations](#)/videos.

Bug/Non-bug Classification. We classify issue reports into bug-related issues and non-bug issues based on the tags provided by GitHub similar to the prior work ([Li et al., 2023](#)). To do so, we first collected 5,079 distinct tags from 481,603 issue reports. Then, the first author manually inspected these tags and observed that those tags including the following keywords “bug”, “crash”, “defect”, “regression”, and “unexpected behavior” are used to indicate bugs. To evaluate the accuracy of the keyword-based method, we randomly selected 30 instances from each keyword group and the first author manually validated a total of 150 instances to confirm whether or not they are real bug reports. The evaluation result suggested that the keyword-based method was robust enough to be applied, where all the validated instances were related to reporting bugs. Afterward, issue reports were automatically classified into bug/non-bug ones by the presence of the keywords in their tags. Note that non-bug tags (e.g., “not-bug”, “not a defect”), which are used in some specific projects, were also taken into account. [In addition, to reduce the threat posed by untagged issues, we filtered out 88,592 untagged issues from 481,603 issue reports.](#) Finally, the issue reports were classified into 138,458 bug-related ones and 254,553 non-bug reports. We then calculated the proportion percentage of bug reports in terms of visual (image and [animation](#)/video, respectively) and non-visual content.

We further propose the following hypothesis to test:




□ (*H1.1*) *Visual issue reports tend to describe bugs.*

For the *H1.1*, the rates of issue reports that are used for reporting bugs by each type (i.e., *None*, *Img*, and *Anm/Vid*) are statistically compared. First, we adopt the *Chi-squared Test* (Pearson, 1900), a multiple comparison test, to examine the difference among the three types. Afterward, we perform the two-proportions *Z-test* (Casella and Berger, 2021) as a post-hoc test, which is a statistical test used to determine whether two proportions are different from each other. In addition, Cohen’s *h* (Cohen, 2013) is applied to measure the effect size between two proportions ($h = 0.20$, small; $h = 0.50$, medium; $h = 0.80$, large).

3.2 Results

Visual issue reports are more likely to report bugs. Table 2 shows the rates of issue reports that contain at least one of the bug-related tags (*BugRate*) divided by three issue types. As we can see from the table, visual issue reports tend to report bugs more frequently when compared to non-visual ones. For example, the rates of *Img* and *Anm/Vid* are 44.6% and 71.1%, separately, whereas the rate of *None* is only 32.2%. [The statistical result of the Chi-squared test suggests that there is a significant difference among the three types of issue reports. The two-proportion Z-Test further confirms a significant difference between pairs, with \$p\$ -values \$< 0.001\$ for *Img* vs. *None* and *Anm/Vid* vs. *None*, respectively.](#) Additionally, the measured variances of

Table 2 Rate of the issue reports related to bug reporting.

<i>IssueType</i>	<i>#Issues</i>	<i>#Bugs</i>	<i>BugRate</i>	Chi-squared <i>p-value</i>	<i>p-value</i> (vs. <i>None</i>)	Cohen’s <i>h</i> (vs. <i>None</i>)
<i>None</i>	316,758	102,087	0.322 	***	-	-
<i>Img</i>	67,218	29,947	0.446 		***	0.256 (small)
<i>Anm/Vid</i>	9,035	6,424	0.711 		***	0.800 (large)

o $p \geq 0.05$; * $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$

Cohen’s *h* show that there exist a small effect (0.256) and a large effect (0.800) between *Img* vs. *None*, and *Anm/Vid* vs. *None*, respectively. These results suggest that the hypothesis: [\(H1.1\) Visual issue reports tend to describe bugs](#) is supported.

RQ1 Summary

Compared to issue reports that do not have visual content, the ones that contain images or videos have a significantly higher probability of reporting bugs, 44.6% and 71.1% (roughly 1.4–2.2 times) being identified, respectively. The result indicates that visual contents are more likely to be used for the purpose of reporting bugs.

4 RQ2: What is the relationship between visuals and the bug-fixing process?

Compared to plain texts in non-visual issue reports, images or videos could facilitate the communication-intensive process as a picture is worth a thousand words (Wang et al., 2023). In this section, we take a further look at how visual issue reports are different from non-visual issue reports by examining three dimensions. For the dataset, we utilize a total of 138,458 issue reports that are related to reporting bugs, which were identified in RQ1 (Section 3). Below, we present our approach to quantitatively analyze the difference between visual and non-visual issue reports (Section 4.1), followed by the results (Section 4.2).

4.1 Approach

To answer RQ2, we perform quantitative analysis to investigate the characteristic difference between the issue reports that have visual contents (i.e., images and [animations](#)/videos) and the ones that do not have any visual contents. Specifically, based on the available dataset attributes, we analyze the differences in the following three main dimensions that are studied commonly in the literature: the issue report itself (*Report*), the process of the issue discussion (*Discussion*), and the issue resolution (*Fix*). Table 3 shows the three studied dimensions with their defined metrics. Note that during the quantitative analysis, we compare the difference between the issue reports

not having visuals (*None*) and the issue reports having images (*Img*), and the difference between *None* and the issue reports having [animations or videos](#) (*Anm/Vid*), separately. Below we describe our approach in detail for each dimension along with the rationale of its selected metrics.

(I) *Report dimension*. Potter and Faulconer (Potter and Faulconer, 1975) showed that, in general, visual images are a more effective approach for describing what people want to communicate compared with text. Inspired by that, we assume that [animations/videos or images](#) can reduce the effort for reporting bugs. Hence, we introduce the below two metrics to understand how an issue report containing visual contents is different from the ones that only have text in the view of their report contents:

- *DescriptionLength*: is the number of words written in the description. Note that we only look at the description of an issue report, excluding its title. Meanwhile, URLs to attach images or [animations/videos](#), code blocks, and tables are not counted as words. We notice that issue reports are likely to adopt the issue templates that are provided by specific projects, which may introduce a threat. To relieve this, we leverage the method provided by Li et al. (2023) to remove the set of common template-related items (e.g., “describe the bug”, “library version”, etc.).
- *DescriptionWords*: refers to the frequently occurring keywords in the description. To do so, we calculate *TF-IDF* values to quantify the frequency of characteristic keywords for each type of issue (i.e., *Anm/Vid*, *Img*, and *None*), applying the common data pre-processing including tokenization, stop word removal, and stemming on the cleaned descriptions (removal of URLs, code block, tables, and templated-related items). We conjecture that characteristic keywords tend to differ among the issue types (i.e., *Anm/Vid*, *Img*, and *None*).

After measuring the above metrics, we further introduce a hypothesis to validate:

□ (H2.1) “An issue report having visual contents include fewer words.”

To examine the H2.1, we first elect the *Kruskal-Wallis test* (Kruskal and Wallis, 1952) as a non-parametric test for comparing the differences among multiple independent groups (three types of issue reports). Then, to statistically compare the difference between pairs, we apply the *Steel test* (Casella and Berger, 2021). *Steel test* performs the multiple comparisons, taking into account the number of comparisons to prevent increases in the family-wise error rate. The studied dataset does not follow normal

Table 3 Attributes collected from the issue reports

Dimension	Metrics	Description
Report	<i>Images</i>	Number of images in the description
	<i>Videos</i>	Number of videos in the description
	<i>DescriptionLength</i>	Number of words in the description
	<i>DescriptionWords</i>	Words in the description
Discussion	<i>FirstCommentTime</i>	Days until the first comment is made
	<i>Comments</i>	Number of comments in the issue report
	<i>LastCommentTime</i>	Days until the last made comments
	<i>Participants</i>	Number of people who wrote comments
Fix	<i>ClosedTime</i>	Days to close the issue

distributions and does not satisfy homoscedasticity, hence it is appropriate to apply the *Steel test*. We determined the sample size for each attribute based on the criteria, significance level = 0.05 and confidence interval = 95%, and defined the alternative hypothesis as that true relative contrast effect p -value is not equal to 1/2 (i.e., two-sided test). A p -value less than one-half means that the treatments tend to be smaller than the control class. Moreover, we apply Cliff's δ (Cliff, 1993), a non-parametric metric, to measure the effect size. Effect size is analyzed as follows: (1) $|\delta| < 0.147$ as Negligible, (2) $0.147 \leq |\delta| < 0.33$ as Small, (3) $0.33 \leq |\delta| < 0.474$ as Medium, or (4) $0.474 \leq |\delta|$ as Large (Romano et al., 2006).

(II) *Discussion dimension*. Joorabchi et al. (2014) cited that lack of proper communication between reporters and developers often ends up with reports in which the reported bugs are not able to be reproduced. In addition, many studies claimed that comments made to a bug contribute to improving bug-fixing activities (Giger et al., 2010; Panjer, 2007; Zhang et al., 2012). Visual issue reports might have the potential to increase the quality of communication between reporters and developers. Developers could quickly understand the reported bugs vividly, which may reduce the needless exchange of confirmation comments. In terms of *Discussion dimension*, we define the following three metrics to measure the difference:

- *FirstCommentTime*: is the duration (#days) from when the author submits an issue report until when the first comment is made.
- *Comments*: denotes the number of comments that are made in an issue report.
- *LastCommentTime*: refers to the duration (#days) from when the author submits an issue report until when the last comment is made.
- *Participants*: is the number of developers who participated in the issue discussion by making comments.

We remove the comments that are made by bots since our focus is on developers. We apply the same technique described in our data collection to remove bot commenters. Based on the above metrics, we propose three hypotheses:

- (H2.2) “An issue report having visual contents receives the quicker.”
- (H2.3) “An issue report having visual contents receives fewer comments.”
- (H2.4) “An issue report having visual contents takes a shorter time to complete the conversation.”
- (H2.5) “An issue report having visual contents are resolved with fewer participants”.

To validate these four hypotheses and measure their effect sizes, similarly, we invoke the *Kruskal Wallis test*, the *Steel test*, and Cliff's δ .

(III) *Fix dimension*. Zimmermann et al. (2010) reported that issue reports occasionally have missing or incorrect steps to reproduce bugs, which delays the entire bug-fixing process. Also, Ohira et al. (2012) showed that bug-fixing activities are delayed when the reporter and developer are different persons because this situation requires communication between the two. We assume that embedding visual content may mitigate this issue by facilitating their communication, and further shorten the resolution time. Thus, we raise the two metrics to understand the *Fix dimension*:

- *ClosedTime*: refers to the duration (#days) from when the author submits an issue report until when the issue is finally closed.

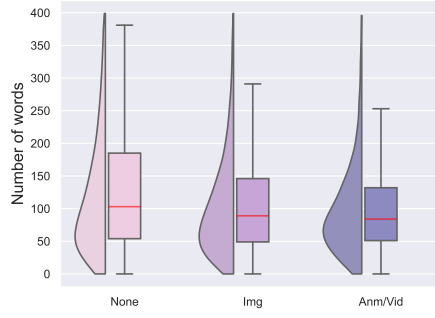


Fig. 3 Report dimension - Number of words in the description of an issue report.

Related to the *Fix dimension*, we propose the following hypothesis:

□ (H2.6) “An issue report having visual contents is closed quicker.”

To confirm this hypothesis, we consistently adopt the *Kruskal-Wallis test* and the *Steel test* to examine the significance of the statement, and we also use Cliff’s δ to measure the effect size.

4.2 Results

(I) Report Dimension

The description length of visual issue reports is shorter than that of those without visual content. Figure 3 shows the distribution of the number of words in the descriptions of issue reports and Table 4 shows the statistical values of *DescriptionLength* metric including the *p-values of the Kruskal-Wallis test and the Steel test*, and the *values of the effect sizes*. When compared to non-visual issue reports (i.e., *None*), as shown in the figure, we observe that visual issue reports (i.e., *Img* and *Anm/Vid*) relatively contain smaller numbers of words in the descriptions. Specifically, the median values are 89 and 84 for *Img* and *Anm/Vid*, respectively, whereas the median value is 103 for non-visual issue reports. Such results suggest that issues tend to be explained in fewer words when a visual is attached.

The *Kruskal-Wallis test* confirms a significant difference among the three groups in terms of *DescriptionLength*, i.e., *p-value* < 0.001. Furthermore, our *Steel test* shows statistically significant differences between visual issue reports and non-visual issue reports, with *p-values* < 0.01 for both *Img* vs. *None* and *Anm/Vid* vs. *None*. Regarding the effect size, small effects (0.224 and 0.310) are observed between *Img* and *None*, *Anm/Vid*, and *None*, respectively. These test results indicate that our proposed hypothesis, □ (H2.1) “An issue report having visual contents include fewer words.” is supported.

Visual issue reports are likely to describe the graphical user interface. Table 5 shows the top-20 characteristic words in each issue report kind, calculated from *TF-IDF* analysis. Note that we stemmed the words to convert them to the base words. First of all, we find that some characteristic words are likely to commonly occur in visual

Table 4 The medians of studied metrics and their significance.

Metrics	Category	Q1/Median/Q3	K-W's <i>p</i> -value	<i>p</i> -value (vs. <i>None</i>)	Cliff's δ (vs. <i>None</i>)
<i>DescriptionLength</i>	<i>None</i>	54/103/185	***	-	-
	<i>Img</i>	49/ 89/146		**	.224 (small)
	<i>Anm/Vid</i>	51/ 84/132		**	.310 (small)
<i>FirstCommentTime</i>	<i>None</i>	.026/.389/3.91	***	-	-
	<i>Img</i>	.001/.238/3.20		*	.232 (small)
	<i>Anm/Vid</i>	.007/.267/5.66		*	.190 (small)
<i>Comments</i>	<i>None</i>	1/2/5	***	-	-
	<i>Img</i>	1/2/5		<i>o</i>	.006 (negligible)
	<i>Anm/Vid</i>	1/2/4		<i>o</i>	.051 (negligible)
<i>LastCommentTime</i>	<i>None</i>	1.01/15.4/139.	***	-	-
	<i>Img</i>	.879/9.80/77.1		***	.356 (medium)
	<i>Anm/Vid</i>	.870/12.0/82.8		**	.160 (small)
<i>Participants</i>	<i>None</i>	1/1/2	***	-	-
	<i>Img</i>	1/1/2		<i>o</i>	.120 (negligible)
	<i>Anm/Vid</i>	1/1/2		<i>o</i>	.092 (negligible)
<i>ClosedTime</i>	<i>None</i>	1.71/15.0/109.	***	-	-
	<i>Img</i>	1.90/13.7/83.7		*	.037 (negligible)
	<i>Anm/Vid</i>	2.90/18.1/102.		<i>o</i>	.052 (negligible)

o $p \geq 0.05$; * $p < 0.05$; ** $p < 0.01$ *** $p < 0.001$;

issue reports (as highlighted in red boxes). Specifically, the word “when” is retrieved by relatively higher *TF-IDF* values, suggesting that issue reports utilized with visuals tend to describe the timing of specific operations (i.e., steps to reproduce). Second, the commonly used words in *animations/videos* (*Anm/Vid*) are related to the operations and elements of graphical user interface (GUI), such as “click”, “tab”, “block” and “button”, while the word “window” is prominent in image reports (*Img*) (as highlighted in blue boxes). In contrast, non-visual issue reports frequently use words such as “file” and “code” that are related to programming issues (as highlighted in green boxes). These findings suggest that non-visual issue reports are used to describe programming things while visual ones are used to mention visible things appearing on the screens. In all, the results of our *TF-IDF* analysis indicate that the characteristic keywords tend to differ between issue reports with/without visual content.

(II) Discussion Dimension

Visual issue reports tend to receive the s and complete the discussion conversations faster. Figure 4 shows the related results of the Discussion dimension, including *FirstCommentTime*, *Comments*, and *Participants*. Figure 4 (a) depicts the distribution of time that is taken to receive the s (*FirstCommentTime*) and Table 4 shows the statistical values of the *FirstCommentTime* metric. We observe that the duration of visual issue reports receiving the first comments is shorter compared to non-visual ones. Specifically, the median values are 0.238 and 0.267 days for *Img* and *Anm/Vid*,

Table 5 Top-20 words that commonly appear in visual and non-visual issue reports.

	<i>None</i>	<i>Img</i>	<i>Anm/Vid</i>
1	issu	reproduc	record
2	error	step	step
3	reproduc	issu	reproduc
4	use	when	when
5	step	shot	issu
6	when	use	click
7	expect	expect	behavior
8	ansibl	error	open
9	behavior	if	expect
10	if	os	select
11	test	screenshot	test
12	file	bug	os
13	code	behavior	tab
14	run	test	if
15	type	creat	descript
16	terraform	what	creat
17	bug	see	use
18	work	open	block
19	os	window	button
20	name	set	see

respectively, whereas the median value is 0.389 days for non-visual issue reports. For *Comments* (number of comments that are made in an issue report), as shown in Figure 4 (b), we find that the numbers do not greatly differ from the issue report kinds, with the same median values of two comments, but reveals that *Anm/Vid* tends to have slightly fewer comments. Figure 4 (c) obviously shows that visual issue reports tend to complete the conversation faster than non-visual ones. Specifically, the median values of *Img* and *Anm/Vid* are 9.8 and 12.0 days, separately, while the median value of *None* is 15.4 days. Figure 4 (d) presents the distribution of the number of participants (*Participants*). From the box plot, we observe that there is no difference between *Img*, *Anm/Vid*, and *None*.

The statistical tests on the one hand reveal that significant differences do exist concerning *FirstCommentTime* and *LastCommentTime* metrics for both *Img* vs. *None* and *Anm/Vid* vs. *None*, as shown in Table 4. On the other hand, with regard to *Comments* and *Participants* metrics, there are no significant differences (i.e., p -values > 0.05 for both *Img* vs. *None* and *Anm/Vid* vs. *None*). In addition, regarding *FirstCommentTime* and *LastCommentTime* metrics, small or medium effects are observed between *Img* and *None*, *Anm/Vid* and *None*, respectively. We now summarize the validation of the four proposed hypotheses:

- (H2.2) “An issue report having visual contents receives the first comment quicker.” is supported.
- (H2.3) “An issue report having visual contents receives fewer comments.” is not supported.
- (H2.4) “An issue report having visual contents takes a shorter time to complete the

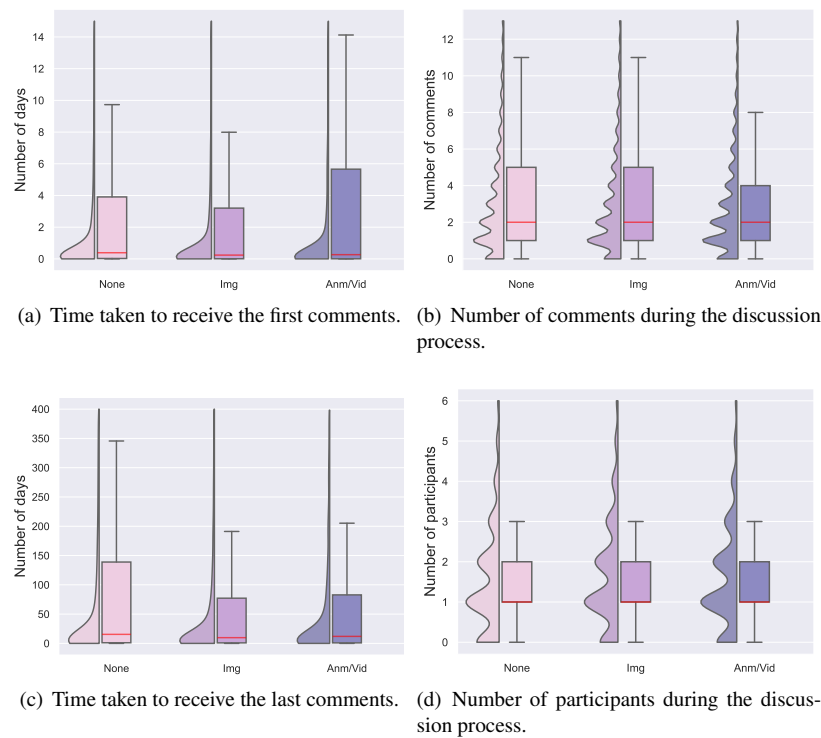


Fig. 4 Discussion dimension - *FirstCommentTime*, *Comments*, *LastCommentTime*, and *Participants*.

conversation.” is supported

□ (H2.5) “An issue report having visual contents are resolved with fewer participants.” is not supported.

(III) Fix Dimension

Time taken to close the issue reports with the image is relatively shorter. Figure 5 presents the distribution of time taken for issue reports to be closed (*ClosedTime*). As shown in the figure, we find that compared against non-visual issue reports (i.e., the median value is 15.0 days), it takes less time for issue reports with images (*Img*) to be closed (i.e., the median value is 13.7 days). On the other side, the median value of *Anm/Vid* is 18.1 days, which is longer than non-visual issue reports.

Furthermore, the statistical results of the *Kruskal-Wallis test* and the *Steel test* confirm significant differences between *Img* and *None*. The result suggests that our hypothesis □ (H3.1) “An issue report having visual contents is closed quicker.” is supported partially for *Img*. Yet only a negligible effect (0.037) is observed through effect size analysis.

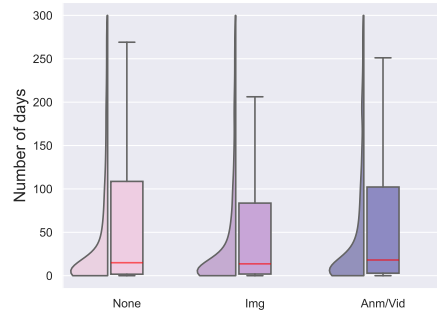


Fig. 5 Fix dimension - *ClosedTime*.

While our analyzed metrics provide insights into the relationship between visuals and the bug-fixing process, including factors such as description length, time taken to receive the first comments, completion of conversations, and issue closure, we cannot establish the causality between them.

RQ2 Summary

Our quantitative results show that compared to non-visual issue reports, the length of descriptions of the visual issue reports is significantly smaller, and visual ones tend to describe the graphical user interface more frequently. We also observe that the visual issue reports are likely to receive first comments and complete discussion conversations more quickly. Moreover, it takes a relatively shorter time for issue reports with images to be closed.

5 RQ3: What are the contents and purposes of visuals used in the issue reports?

The visual contents are diverse in the different issue report topics. Moreover, the purpose for attaching visuals (i.e., an image or an animations/video) may differ. In this section, we set out to better understand the usage patterns (contents and purposes) of visuals through a manually-intensive method. Below, we present our approach to qualitatively investigate the usage patterns (Section 5.1) and the related results (Section 5.2).

5.1 Approach

To answer RQ3, we conduct a content analysis, one of the most broadly used qualitative data analysis methods (Stemler, 2000), to manually investigate usage patterns of visual contents from the following two perspectives: “What type of visual content is attached (*VisualType*); for what purpose is visual content used (*VisualPurpose*)?”. We now describe the construction of the representative sample and manual coding of visual contents and their purposes.

(I) Representative Sample Construction. A total of 36,373 bug-related issue reports contain visuals in our collected dataset, i.e., 29,947 have images and 6,424 have [animations/videos](#) as shown in Section 3. Since coding all of these instances is impractical, we elect to code a sample of visual issue reports. Under the condition of a confidence level of 95% and a confidence interval of 5 (Krejcie and Morgan, 1970), we sampled 379 issue reports with images and 362 issue reports with [animations or videos](#). Note that the total number of visuals is greater than the number of visual issue reports because some visual issue reports have multiple visual contents. In other words, 559 images and 400 [animations/videos](#) from the representative issue reports were analyzed during our manual coding analysis.

(II) Manual Coding of Visual Contents. To the best of our knowledge, few studies empirically examined the visual contents embedded into issue reports, including [animations and videos](#). Hence, the first step is to construct a reliable taxonomy.

To discover as complete of a list of types of visual contents as possible, similar to the prior work (Zanaty et al., 2018; Xiao et al., 2021), we strive for *theoretical saturation* (Eisenhardt, 1989) to achieve analytical generalization. We set our saturation criterion to 40 visual issue reports (i.e., 20 with images and 20 with [animations or videos](#)). In other words, we continue to code randomly selected visual issue reports until no new codes have been discovered for 40 consecutive instances. Open coding was performed by the first three authors of this paper (the first author is a master’s student and the other two authors have several years of research experience with GitHub). To do so, three authors independently coded the samples in multiple rounds. When we classified the contents of visuals, we allowed multiple codes for each issue report as an issue report could contain more than an image or an [animation/video](#). After each round, an open discussion was held among the three coders to discuss each disagreed sample until a consensus was reached. Since open coding is an exploratory data analysis technique, it may be prone to errors. To mitigate errors, after completing an initial round of coding, we performed a second pass over all of the samples to correct miscoded entries and refine the definition of the devised taxonomy. We repeated the process for two rounds and coded 80 visual issue reports. To validate the understanding of the constructed taxonomy, we computed the inter-rater agreement using Krippendorff’s alpha $Kr\alpha$ (Bauer, 2007), which is commonly adopted in software engineering research (Blincoe et al., 2016; He et al., 2023). Krippendorff’s α can be interpreted as a measure of the degree of agreement achieved out of chance: the larger $Kr\alpha$ is, the better agreement is observed. As demonstrated by Krippendorff (2019), as a rule of thumb, it is common to find that $Kr\alpha \geq 0.667$ is the threshold needed to draw conclusions from the data. The iterative content analysis reached an adequate $Kr\alpha$ score, i.e., 0.682, suggesting that our taxonomy is reliable to use.

For the remaining 661 visual issue reports, three graduate students having more than five years of programming experience were assigned to complete the manual classification. To ensure that these students were equipped with sufficient knowledge, under the guidance of the first author who joined the aforementioned taxonomy construction, they read and re-read the description of the taxonomy by referring to the already classified codes until a consensus was reached among the three of them. They further conducted independent coding on another 40 samples (i.e., 20 having images and 20 having [animations or videos](#)) randomly selected from the 661 visual issue

reports. Similarly, we used Krippendorff's α to measure their inter-rater agreement. The scores returned 0.684, which were above the threshold (i.e., 0.667). Encouraged by this result, the rest of the 621 samples were divided into three sets, and the three graduate students independently coded the corresponding set.

(III) Manual Coding of Visual Purposes. Agrawal et al. (2022) established a coding taxonomy for purposes of visual issue reports in a case study of Jupyter Notebook project hosted on GitHub. The taxonomy introduces five general purposes that consist of eleven themes. We used this taxonomy as our initial coding schema because it is closely relevant to our study.

To test how well the existing taxonomy can be used to classify the purposes of our studied issue reports, the three authors (same as the manual coding of visual contents) independently categorized the purposes of visual contents in 40 samples (i.e., 20 having images and 20 having animations or videos). The agreement score of Krippendorff's α was 0.696, indicating that the three authors achieved a relatively high level of comprehensive understanding. After the classification, an open discussion was held among the three authors to resolve the disagreements and discuss the feasibility of the taxonomy. In the end, we adopted five purpose themes for visuals in our work as our focus is on the visuals that are embedded into the issue description. Note that one issue report can be annotated for multiple purposes, as an issue report can have multiple visuals. The definitions of the five purposes and corresponding examples are described as follows:

1. *Introducing problem in report*: refers to the instance, in which the visual content is used to illustrate the problem introduced in the issue description. Example: <https://github.com/rancher/rancher/issues/17310>.
2. *Help identifying the issue cause*: refers to the instance, in which the visual content is used to show additional information about the issue (e.g., error message, additional symptoms, behavior in another environment, etc.) to support the identification of its cause. Example: <https://github.com/microsoft/vscode/issues/105145>.
3. *Illustrating expected behavior*: refers to the instance, in which the visual content is used to illustrate the expected behavior of the system without a bug. Sometimes, the reporters used such visual content to compare the current behavior and the expected behavior. Example: the second visual content in <https://github.com/godotengine/godot/issues/54688>.
4. *Providing instructions or workarounds*: refers to the instance, in which the visual content is used to provide guidance on how to handle a problem, either as instructions or alternative solutions to assist other participants in resolving the issue. Example: the second visual content in <https://github.com/elastic/kibana/issues/40015>.
5. *Social purposes*: refers to the instance, in which the visual content is used to express feelings and facilitate casual conversations. Example: the third visual content in <https://github.com/cleverraven/cataclysm-dda/issues/43973>.

For the remaining 701 visual issue reports, we assigned three graduate students who participated in the coding experiment of visual contents to manually classify them. Similarly, three coders independently coded another 40 samples (i.e., 20 having

images and 20 having animations or videos) to validate whether they had sufficient knowledge to complete the classification task. The agreement score of Krippendorff's α reached 0.720. After the validation, the remaining 661 samples were divided into three sets, and the three graduate students independently coded the corresponding set.

5.2 Results

Visual contents are more frequent to be used for displaying the program behavior or the layout of the screen, with the purpose of providing examples. Seven kinds of the visual content of issue reports are classified from our manual coding. The description of each kind is presented as follows, along with the representative example links:

1. **Code**: refers to the visual content that captures programming codes that are executed during the occurrence of the bug. Example: <https://github.com/microsoft/vscode/issues/117675>.
2. **Configuration**: refers to the visual content that captures information about the configuration (component arrangement, option settings, etc.) related to the bug. Example: <https://github.com/godotengine/godot/issues/17773>.
3. **Diagram**: refers to the visual content that captures a user-created diagram/chart or includes annotations artificially created or edited by developers. Example: <https://github.com/elastic/kibana/issues/88670>.
4. **Logs**: refers to the visual content that captures a log history or a successful/error message displayed in the console or pop-ups. Example: <https://github.com/eclipse/che/issues/19097>.
5. **Output**: refers to the visual content that captures the output or behavior from a program or an algorithm, regardless of its correctness. Example: <https://github.com/godotengine/godot/issues/48772>.
6. **Performance**: refers to the visual content that captures the performance behavior of the program when it becomes slow or freezes, as well as information about CPU and memory usage. Example: <https://github.com/gatsbyjs/gatsby/issues/25942>.
7. **User Interface**: refers to the visual content that displays the application or web page interface. It includes issues related to layout or UI elements, such as unresponsive buttons or changes not being reflected in the display. Example: <https://github.com/wordpress/gutenberg/issues/29886>.
8. **Others**: refers to the visual content that does not fit into any of the above categories. Example: the first image on <https://github.com/gatsbyjs/gatsby/issues/29213>.

Figure 6 shows the distribution of kinds of visual contents and Figure 7 presents the frequency of their purposes. As we can see from Figure 6, the most frequent visual content is *Output*, accounting for 41.3% of images and 62.0% of *animations/videos*. This result suggests that developers tend to show the output, the behavior of the program, or the algorithms, along with the textual contents in the issue report. The second common kind is *User Interface*, with 20.8% of images and 25.0% of *animations/videos* of instances being classified, followed by *Logs* (15.9% and 7.0% of

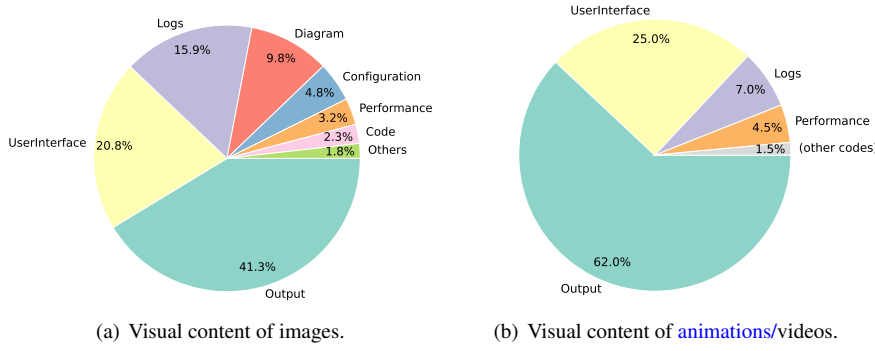
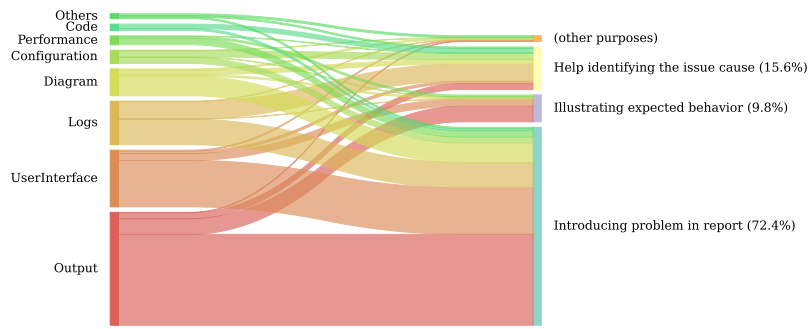


Fig. 6 Distribution of kinds of visual content. Note that “(other codes)” include Diagram (1.3%) and Code (0.25%).

images and [animations/videos](#), respectively). On the other hand, in terms of images, *Configuration*, *Performance*, and *Code* are relatively less frequent, accounting for 4.8%, 3.2%, and 2.3%, separately. In addition, we observe that the kinds of visual content of [animations/videos](#) are less diversified when compared to those of images. As shown in Figure 7, we observe that the visual attached in an issue report is more likely to introduce the problem in a report, i.e., accounting for 72.4% and 83.7% of images and animations/videos, respectively. The second common purpose is to help identify the issue cause, with 15.6% of images and 11.0% of animations or videos of instances being coded, followed by the purpose of illustrating expected behavior (9.8% and 5.3% of images and animations/videos, respectively). Providing instructions or workarounds and social purposes are the least common, with only 1.8% and 0.4% of images being classified, separately. Furthermore, we investigate the relationship between visual contents and purposes by drawing a parallel category diagram. We find that most of the images and animations/videos are consistently used to introduce problems regardless of their content kinds.

Representative Examples. Now we show two representative examples to illustrate the categories of the content and purpose of the visual. Figure 8(a) shows a visual that is used to introduce a problem in report from our manual analysis. We classify the image content into the user interface since the menu list and the buttons are clearly displayed. As shown in the issue body, the text details an unexpected behavior (i.e., the input shows empty) in the synced machine view, and the textual contents match the information that is conveyed from the attached visual. In this instance, although as an example, the image is considered essential since it is tricky for a developer to imagine the vision of the unexpected interface. Figure 8 (b) presents a visual that is used to help identifying the issue cause. The image content records lines of log output generated from the test code, hence we classify this content into logs. As shown in the description, a submitter met an NSE exception problem on the output console, but concrete exception information was not provided. Thus, a visual was attached to show the exception logs (e.g., error: ‘No value present’) from the desktop, serving as additional information.



(a) Purposes of images.

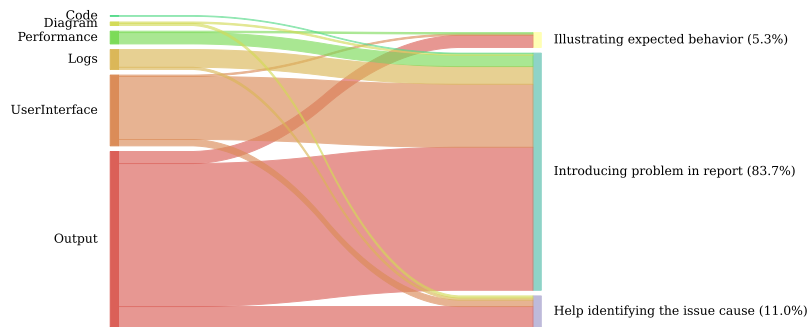
(b) Purposes of [animations/videos](#).

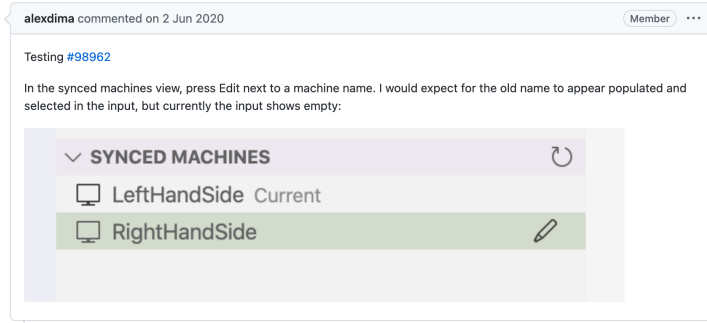
Fig. 7 Relationships between visual contents and their purposes. Note that “(other purposes)” include “Providing instructions or workarounds” (1.8%) and “Social purposes” (0.4%).

RQ3 Summary

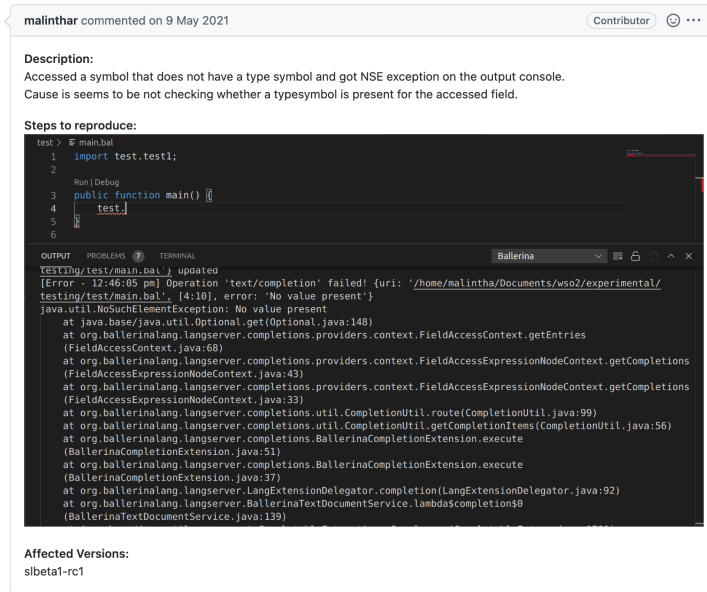
Seven content kinds and five purpose kinds of visuals are identified. Results show that visual contents more frequently depict the output of the program behavior (41.3% and 62.0% for images and [animations/videos](#), respectively) and the user interface (20.8% and 25.0% for images and [animations/videos](#), respectively). We observe that the majority of visuals are served as introducing problem in report to make the textual contents intuitive.

6 RQ4: What bug types are raised in visual issue reports?

We conjecture that visuals are likely to be used in reporting specific bugs. Hence in this section, we further take a deeper insight into what types of bugs visual issue reports frequently raise and whether visuals play a role in fixing different types of



(a) Visual issue report that introduces problem in report.



(b) Visual issue report that helps identifying the issue cases.

Fig. 8 Representative examples. (a): <https://github.com/microsoft/vscode/issues/99078>, (b): <https://github.com/ballerina-platform/ballerina-lang/issues/30450>.

bugs. Below, we present our approach to analyzing the bug types of issue reports (Section 6.1) and the related results (Section 6.2).

6.1 Approach

To answer RQ4, we perform a manual analysis on the representative samples (379 issue reports with images and 362 issue reports with [animations](#) or [videos](#)) used in RQ3 to investigate the content reported in issue reports. Our focus lies on comparing the differences between visual issue reports and non-visual ones in order to shed light

on “For what bug are visuals used (*BugType*)?”. We now describe the manual coding approach for identifying bug types.

Manual Coding of Bug Types. A group of studies has established the coding schema of bug types (Catolino et al., 2019; Nayrolles and Hamou-Lhadj, 2018; Lal and Sureka, 2012). Taking into account our study’s focus, we refer to the coding schema proposed by (Catolino et al., 2019), since this schema targets the same social coding platform *GitHub* and bug types are more fine-grained to be classified. The coding schema consists of the following nine categories:

1. *Configuration bug*: refers to bugs related to building configuration files, often caused by outdated or faulty external libraries or incorrect directory/file paths in XML or manifest artifacts.
2. *Network bug*: refers to bugs related to connection or server problems, including network issues, unexpected server shutdowns, or improper usage of communication protocols in the source code.
3. *Database-related bug*: refers to bugs related to the connection between the main application and a database, specifically issues with failed queries or connections.
4. *GUI-related bug*: refers to bugs occurring within the graphical user interface (GUI) of an application, encompassing stylistic errors in screen layouts, element colors, padding, text box appearance, buttons, as well as unexpected failures resulting in unusual error messages.
5. *Performance bug*: refers to bugs reporting performance problems such as memory overuse, energy leaks, and infinite loops.
6. *Permission/deprecation issue*: refers to bugs stemming from deprecated method calls or APIs, as well as problems related to unused API permissions.
7. *Security bug*: refers to bugs related to vulnerabilities and security problems, including the need to reload certain parameters and remove unused permissions that could impact system reliability.
8. *Program anomaly bug*: refers to bugs introduced by developers during code enhancements, specifically concerning exceptions, return values, and unexpected crashes resulting from logical issues in the program (excluding SQL statement errors). Note that SQL statement issues are classified as database-related bugs.
9. *Test code-related bug*: refers to bugs appearing in test code, commonly reporting problems with running, fixing, or updating test cases, intermittent tests, or difficulties in identifying localized bugs.

The first author and three students manually coded 741 visual issue reports into the above nine categories. The titles, descriptions, comments, and commits of the issue reports were relied upon to classify the types of bugs. To validate the comprehensive understanding of the coding schema among coders, four coders first independently classified 40 randomly sampled issue reports, returning a *Krα* score of 0.719. Encouraged by this satisfying agreement score, the remaining issue reports were separated into four sets and the four coders manually classified each set.

Control Group. In addition, we construct a control group to fairly compare the popularity difference in terms of bug types that are raised in issue reports with visuals and those without visuals. To do so, we randomly selected 383 representative samples, with a confidence level of 95% and a confidence interval of 5, from 102,087

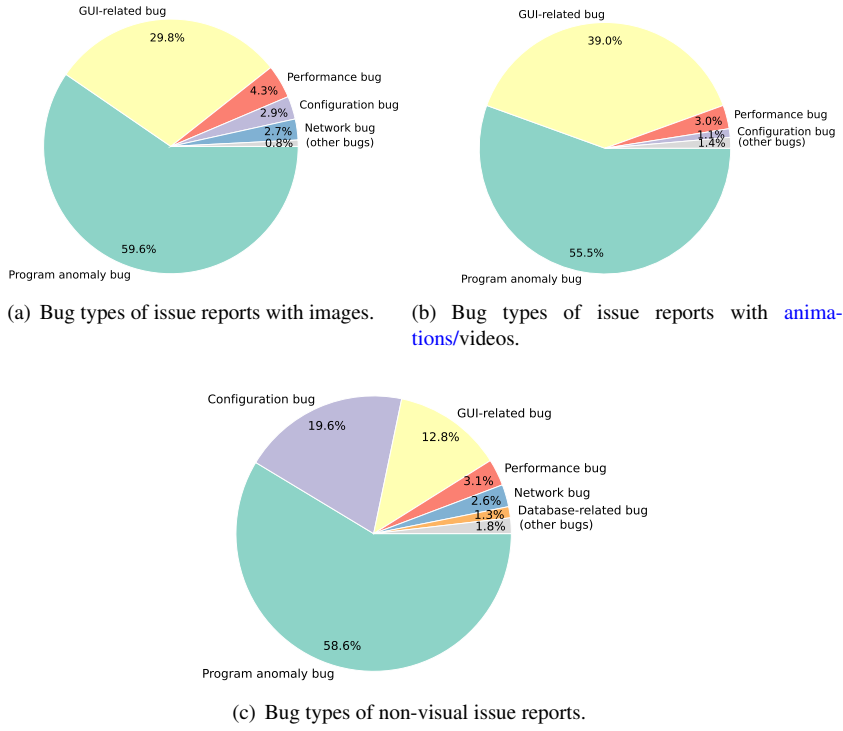


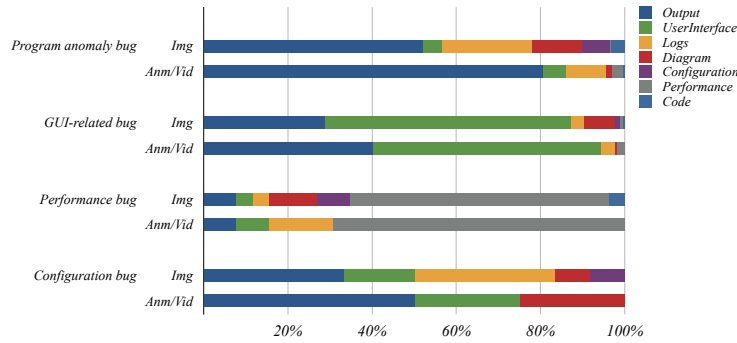
Fig. 9 Distribution of bug types of each issue report kind. (a)“(other bugs)”: Database-related bug (0.53%), and Test code-related bug (0.27%), (b)“(other bugs)”: Network bug (0.83%), Database-related bug (0.28%), and Test-code-related bug (0.28%), (c)“(other bugs)”: Permission/deprecation bug (1.3%) and Test code-related bug (0.44%).

issue reports without visuals. Since the measured agreement was already sufficient, the first author then manually classified these representative samples.

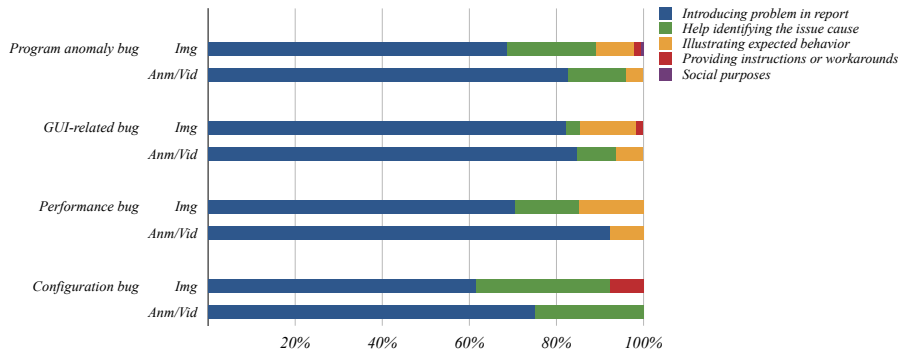
After we classified the bug types of sampled issue reports with visuals and the control group, we further analyzed the effect of these visuals in fixing different bug types. To reduce the threat caused by the small amount of data, we only focus on the relatively frequent issue types that are reported in both visual and non-visual ones, with the number of identified instances being greater than 10 (3%). We selected two interesting metrics from the *Discussion* and *Fix* dimensions as described in Section 4.1, namely *FirstCommentTime* and *ClosedTime*. Similarly, to measure the statistical significance and the effect size, we adopt the *Kruskal Wallis test*, the *Steel test*, and Cliff’s δ .

6.2 Results

Visual issue reports more commonly describe GUI-related bugs, whereas configuration bugs are reported less frequently compared to non-visual issue reports. Figure 9



(a) Contents of visuals in each type of bug report.

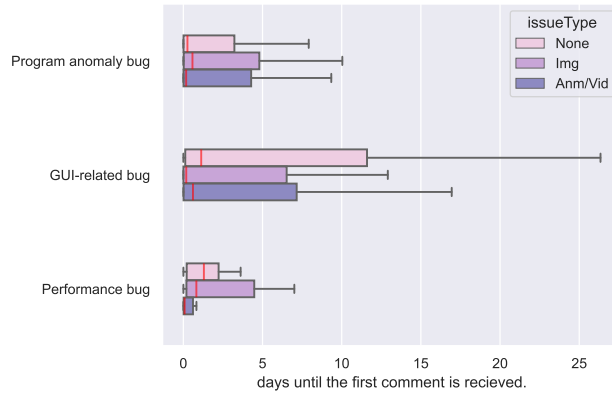


(b) Purposes of visuals in each type of bug report.

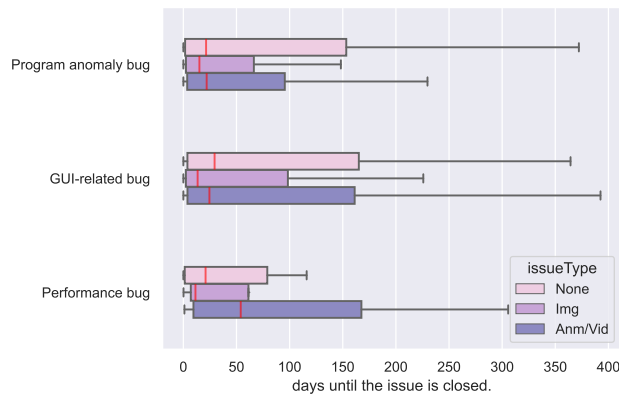
Fig. 10 Relation between visual types/purposes and the bug type.

shows the distribution of bug types in the three kinds of issue reports. Firstly, we observe that the most frequently reported issues, irrespective of the type of issue report, are related to program anomaly bugs (59.6%, 55.5%, and 58.6% being identified for images, [animations](#)/videos, and non-visual reports, separately). Second, the pie charts show that the proportion of GUI-related issues is much higher in the reports with images or [animations](#)/videos (29.8% and 39.0%, respectively) when compared to non-visual reports (12.8%). Third, we find that the configuration bug is rarely reported with images or [animations](#)/videos, only with 2.9% and 1.1% of issue reports being identified, while the configuration bug ranked as the second common kind is frequently reported in the non-visual reports, accounting for 19.6%.

The major kinds of visual contents are likely to vary depending on the bug types, while introducing problems in the report is the most frequent purpose regardless of bug types. We further explore the relationship between the characteristics of visuals



(a) FirstCommentTime.



(b) ClosedTime.

Fig. 11 Metrics FirstCommentTime and ClosedTime for the frequent bug types across three kinds of issue reports.

(their contents and purposes) and the bug types. Figure 10 (a) and Figure 10 (b) present the related results. In terms of the relationship between visual contents and bug types, as shown in Figure 10 (a), we observe that program anomaly, GUI-related, and performance bugs tend to be embedded with dominant kinds of visual contents, respectively. Specifically, *Output*, *User Interface*, *Performance* are commonly used to report the three aforementioned bug types separately. Their frequencies for either images or animations/videos are all over 50%. However, for configuration bugs, no specific kind of visual content dominance is observed. In terms of the relationship between visual purposes and bug types, as shown in Figure 10 (b), the results indicate that regardless of the bug types, the most common purpose is to introduce problems in reports. At the same time, the visual used to help identify the issue cause is relatively more frequent when reporting configuration bugs.

Table 6 The statistical results of the metrics *FirstCommentTime* and *ClosedTime*.

Metrics	BugType	Pairs	K-W's <i>p</i> -value	<i>p</i> -value	Cliff's δ
<i>FirstCommentTime</i>	Program anomaly bug	Img vs. None	<i>o</i>	<i>o</i>	.524 (large)
		Ann/Vid vs. None	<i>o</i>	<i>o</i>	.207 (small)
	GUI related bug	Img vs. None	<i>o</i>	*	.601 (large)
		Ann/Vid vs. None	<i>o</i>	<i>o</i>	.386 (medium)
	Performance bug	Img vs. None	<i>o</i>	<i>o</i>	.198 (small)
		Ann/Vid vs. None	<i>o</i>	<i>o</i>	.348 (medium)
<i>ClosedTime</i>	Program anomaly bug	Img vs. None	<i>o</i>	<i>o</i>	.057 (negligible)
		Ann/Vid vs. None	<i>o</i>	<i>o</i>	.004 (negligible)
	GUI related bug	Img vs. None	<i>o</i>	<i>o</i>	.133 (negligible)
		Ann/Vid vs. None	<i>o</i>	<i>o</i>	.012 (negligible)
	Performance bug	Img vs. None	<i>o</i>	<i>o</i>	.042 (negligible)
		Ann/Vid vs. None	<i>o</i>	<i>o</i>	.227 (small)

o $p \geq 0.05$; * $p < 0.05$; ** $p < 0.01$;

Bug-fixing process may differ from the bug types across issue report kinds. Figure 11 presents the results of the metrics *FirstCommentTime* and *ClosedTime* for the relatively frequent bug types (Program anomaly bug, GUI-related bug, and Performance bug) and Table 6 shows their statistical results. We observe that issue reports with images or animations/videos that raise GUI-related and performance bugs are likely to receive the first comment within a shorter period, while reports with images that raise program anomaly bugs tend to take a longer time. Although no significant difference is drawn, the effect size analysis suggests that there indeed exists a relationship, varying from small to large size. One possible reason is that our sample size is not large enough (Sullivan and Feinn, 2012). In terms of the *ClosedTime*, from the figure, we notice that the taken time for issue reports with images is constantly shorter across the three studied bug types but there is no significant difference and magnitude effect size. On the other side, when raising performance bugs, issue reports with animations/videos significantly take a longer time to be closed, with a small effect size being examined.

RQ4 Summary

Results show that visuals are more frequently used to report GUI-related bugs (2–3 times) but less commonly to report configuration bugs compared to non-visual issue reports. It relatively takes less time for issue reports with images that report GUI-related and performance bugs to receive the first comment.

7 Discussion

In this section, we discuss the main findings of our empirical results and provide potential future works.

Potential to make a good issue report. Due to their intuitive nature, it is not surprising that visuals have become a popular choice for electronic developer communication, such as Q&A forums and issue tracker systems (Nayebi, 2020; Agrawal et al., 2022; Wang et al., 2023). Our large-scale empirical study on 34 projects fur-

ther confirmed this upward trend for both images and animations/videos, as shown in Figure 2. Compared to the animations/videos, images are more frequently embedded with issue reports, almost reaching one-fourth of the reports in the past two years. Possible reasons for this could be that the GitHub feature for uploading videos was newly launched in 2021, and uploading images requires less effort. Prior works demonstrated that several important elements (e.g., examples, user contents, and reproducing steps) are largely missing in the issue reports (Soltani et al., 2020). The increasing use of visuals to report issues indicates that developers are becoming more aware of the information needed to facilitate issue resolution tasks. Furthermore, the results of our manual analysis of visual characteristics in RQ3 indicate that visuals are frequently utilized to depict outputs and user interfaces (41.3% and 20.8% for images, and 62.0% and 25.0% for animations/videos). This result suggests that developers tend to effectively leverage visuals to convey information that is difficult to describe in text-based content. In particular, the animations/videos support a dynamic reproduction of the issues, which can further enhance the context comprehension to introduce problem and help to identify the cause (83.7% and 11.0% being identified, respectively, in Figure 7). These empirical findings imply that visuals may aid to construct a good issue report with richer information beyond the text.

On the other hand, we found some cases where visuals may bring over or unnecessary information. For instance, in this report,⁵ three images are attached, however, the last two images describe the similar contents as the first image and could be replaced by the text in the description. Hence, to make a good issue report, we also suggest that reporters judiciously use visuals to supplement the information when reporting issues. Another direction for researchers is to analyze the extent to which the information presented in visuals aligns with the textual description.

Association between the visual issue reports and the bug reporting. Our RQ1 results show that compared to issue reports that do not have any visuals, the ones that contain images or animations/videos significantly have a higher probability of raising bugs, 44.6% and 77.1% being identified (as shown in Table 2). This finding suggests that developers tend to attach visuals to describe bug-related topics. Complementarily, our RQ4 results show that visuals are more frequent to be attached to the program anomaly and GUI-related bugs (as shown in Figure 9) but less commonly attached to configuration bugs. Based on these findings, we suggest that developers should prioritize visual issue reports, especially those related to animations/videos. Additionally, GitHub maintainers could consider introducing a new issue label to indicate the inclusion of visuals, as these reports are more likely to be associated with bugs.

Role of visuals in the issue resolution process. Our empirical results (RQ2) statistically confirm that compared to issue reports that do not contain visuals, visual reports are likely to shorten the issue discussion process. For instance, it significantly takes a relatively shorter time for visual issue reports to receive the first comment (*FirstCommentTime*) and complete the conversation (*LastCommentTime*) as shown in Table 4. This finding suggests that visuals could attract the developers' attention and initiate the discussion earlier. However, in terms of the closed time, we do not observe a substantial significant difference from both results shown in Figure 5 and

⁵ <https://github.com/magento2/magento2/issues/34476>

Figure 11. Therefore, developers should not expect that attaching visuals to the issue reports will result in faster resolution times. One potential reason is that visuals may not be effective during all phases of the bug-fixing process, such as the bug localization phase. The next logical step for researchers would take into account other confounding factors and see the role of visuals in the statistical models. Another possible step is to conduct a developer survey to understand how they perceive the impact of visuals on bug fixing. In addition, we overlook the visuals in the issue discussions, hence, in future work, we will further examine their effect and characteristics.

Usage of the characteristics of visuals. We suggest researchers utilize the taxonomy of characteristics of visuals to further understand the features of issue reports (such as the reporters' experience). For example, as shown in Figure 10, we shed light on the relationship between the characteristics of the visuals and the bug types. The results offer potential recommendations for the content of visuals that are commonly shared in specific bug reports. Another possible future direction for researchers includes a classifier proposal that can automatically classify the kinds of visual contents and their purposes, which could support the larger-scale study which relies on statistical models to examine the effect of visuals and would also be beneficial for developing automated tools to sort out the priority between visual issue reports.

We outline the implications, challenges and future directions for practitioners and researchers from our discussions below.

Implications:

- Developers tend to become more aware of the information needed to facilitate issue resolution tasks.
- Visuals have the potential to construct a good issue report with richer information beyond the text.
- Reporters should use visuals judiciously to supplement information when reporting issues, in order to avoid providing excessive or unnecessary information.
- GitHub maintainers/Developers should prioritize visual issue reports as they are highly associated with bug reporting, especially those related to animations/videos.
- Visuals may initiate the discussion earlier, but developers should not expect that attaching visuals will result in faster resolution times.

Challenges/Future directions:

- In future work, we plan to analyze the extent to which the information presented in visuals aligns with the textual description.
- Researchers would take into account other confounding factors and examine the role of visuals in the issue resolution process.
- A developer survey would be beneficial to understand how they perceive the impact of visuals on bug fixing.
- To examine the role of visuals and their characteristics in the issue discussion would be the next step.
- A classifier could be developed to classify the characteristics of visuals, in order to support larger-scale studies and automated tasks.

8 Threats to Validity

In this section, we now discuss the threats to the validity of our empirical study.

External Validity is concerning our ability to generalize based on our results. We only conduct an empirical study on 34 large and active GitHub repositories (i.e., containing more than 10,000 issue reports) to investigate the role of the visual contents on issue reports. However, these repositories demonstrate diversity in terms of languages and domains. Specifically, they cover over twelve languages including the popular ones (e.g., Java, Python, C/C++, JavaScript, and Go), and five domains (i.e., application software, software tools, libraries and frameworks, system software, and documentation) based on the work of Borges et al. (2016). Furthermore, we believe that the large-scale size of the studied issue reports (i.e., 481,603) does provide valuable insights into the role of the visual issue reports. Nonetheless, it is important to acknowledge that these observations may not be applicable to smaller or medium-sized GitHub repositories. Conducting replication studies in the future can help enhance the validity of the generalizations that can be made.

Construct Validity denotes the degree to which our measurements capture what we aim to study. Three potential threats are concluded. The first threat could occur during our data collection of visual issue reports. To identify the visuals, we rely on a list of image file extension types (i.e., “png”, “PNG”, “jpg”, “JPG”, “jpeg”, and “JPEG”) and a list of video file extension types (i.e., “gif”, “GIF”, “mp4”, “MP4”, “mov”, “MOV”, “webm” and “WEBM”). It is possible that some images or videos may not be retrieved by any type in these lists. However, we are confident that we cover the most commonly used file extension types. In addition, our work is limited to focusing on the visuals that were attached to the descriptions of the issue reports. We omitted those visuals in the issue comments, hence future work is encouraged to systematically investigate their characteristics and their relationship with the bug resolution process. The second threat exists in the measurement of bug rate in RQ1. In this analysis, we rely on the tags to determine whether or not the issue report is related to bugs. In our RQ1 analysis, we rely on the tags to determine whether or not the issue report is related to bugs. Although we performed a sanity check to examine the precision of this method similar to the work of (Li et al., 2023), there could be issues incorrectly identified as non-bugs due to the liberal nature of GitHub tags. To further mitigate the threat, we filtered out those issues that were not tagged and then calculated the bug rates. Nonetheless, as an exploratory study, our results do shed light on the practice of bug reporting between visual issue reports and non-visual issue reports. The third threat is concerning the metric selection in RQ2 to analyze the characteristic difference between visual issue reports and non-visual ones. We focus on seven metrics from three dimensions, and we make sure that these metrics are representative to be analyzed based on the literature review.

Internal Validity refers to the approximate truth about inferences regarding cause-effect or causal relationships. Two main internal threats are summarized. The first

threat is concerning the choice of the statistical test selection in our quantitative analysis (RQ2 and RQ4). Different statistical tests may introduce the threat of the significance measurement. However, we believe that the Steel test is an appropriate test to fit our data (e.g., data does not follow normal distributions, and does not satisfy homoscedasticity) and is broadly adopted in the prior work. The second threat exists in our qualitative analysis (RQ3 and RQ4), where we manually classified the contents and purposes of visuals and the bug types of issue reports. We rely on manually coded data, which may be miscoded due to the subjective nature of understanding the coding schema. To relieve this threat, we properly apply an open coding approach by three authors, and disagreements are solved until these authors reach a consensus.

9 Related Work

In this section, we compare and contrast our study to prior research in two dimensions: first, we consider the work that analyzes bug comprehension and its effect on fixing time; then, we introduce the literature that is related to non-textual information sharing in software development.

9.1 Bug Comprehension & Fixing Time

Fixing bugs is a crucial activity in the software development process and maintenance, with 80% of the total cost of a software project being spent (Planning, 2002; Weiss et al., 2007). However, bug reports often lack the necessary information for developers to reproduce bugs (Joorabchi et al., 2014). Prior researchers have widely studied the association between technical and non-technical factors and fixing time, including the role of the significant information elements. Soltani et al. (2020) found that crash reproducing steps, stack traces, fix suggestions, and user contents, have a statistically significant impact on bug resolution time. Through a study on 10 large-scale open-source systems, Chen et al. (2021) stated that reporters may not attach accurate or sufficient logs, which extended the bug resolution time. Li et al. (2023) empirically studied the bug report templates on GitHub and reported that bug reports with templates have shorter resolution times and higher comment coverage. At the same time, a plethora of automated models to aid bug comprehension and shorten the fixing time have been proposed. To name a few, Moran et al. (2015) introduced assistant systems FUSION for reporters, which auto-complete reproduction steps based on user-provided information and information extracted via a combination of static and dynamic program analyses. Song et al. (2022) proposed an interactive GUI tool for easy reporting of bugs, allowing users to easily report reproduction steps by selecting candidate screenshots of the application. Fazzini et al. (2023) introduced a system called EBug, and EBug is capable of automatically suggesting potential future steps using predictive models trained on realistic app usage. *To facilitate the needs of bug reproduction work from the developer viewpoint, GitHub allows developers to submit issue reports embedded with visuals (especially since 2021, video sharing is permitted). Our study takes a first step towards empirically understanding the usage of visuals and highlights the role of visuals plays in supporting bug comprehension.*

9.2 Non-Textual Information Sharing in Software Development

During contemporary software development, developers nowadays commonly share non-textual information (e.g., links, code snippets, and visuals) to encourage knowledge exchange, and support their collaboration and communication needs. For instance, the value of link sharing has been widely explored. [Ye et al. \(2017\)](#) used the URLs shared in Stack Overflow to generate a knowledge network, in order to enable more effective knowledge sharing during the community. [Hata et al. \(2019\)](#) investigated the characteristics of 9.6 million links in source code comments and the results show that links are prevalent, frequently referring to software homepages and specifications. During code review, [Wang et al. \(2021\)](#) observed seven intentions behind sharing links, and their developer survey results suggested that link sharing is more useful than plain text during code review. Regarding the usage of code snippets, [Fu et al. \(2022\)](#) observed that code implementation is the most common purpose and highlighted the role in code reviews.

In addition to the images and code snippets, developers are increasingly making use of visuals (i.e., screenshots and screen recordings) as a means to report issues. For instance, [Nayebi \(2020\)](#) reported that there was a steady increase in sharing images over the past five years in Stack Overflow and Bugzilla. Their developer survey suggested that shared images are meaningful and provide complementary information. [Johnson et al. \(2022\)](#) reported that bug reports with cosmetic and navigation failures have a higher proportion of cases in which the information is reported using image-based modalities as compared to output and crash failures. [Agrawal et al. \(2022\)](#) found that more than a quarter of the issues in the Jupyter Notebook project included visual content. Apart from the above empirical studies that aim to understand the popularity and the characteristics of the issues, a group of studies has explored the usage of visuals in facilitating a developer’s automatic task. [Wang et al. \(2019\)](#) proposed SETU which combines information from the screenshots and the textual descriptions to detect duplicate crowd-testing reports, outperforming the existing state-of-the-art. [Cooper et al. \(2021\)](#) introduced TANGO to aid developers in determining whether video-based bug reports depict the same bug, by leveraging tailored computer vision techniques, optical character recognition, and text retrieval. [Feng et al. \(2023\)](#) developed GIFdroid and CAPdroid, which are capable of extracting user actions (i.e., steps to reproduce) from submitted videos. *Our study expands upon the work of [Nayebi \(2020\)](#) and [Agrawal et al. \(2022\)](#), we conducted a large-scale empirical study on 34 GitHub repositories to statistically analyze the characteristic difference between visual issue reports and non-visual reports. For instance, one interesting result shows that visual issue reports are more likely to be associated with bugs.*

10 Conclusion

GitHub Issue includes features for users to embed visuals (i.e., images and videos) to facilitate developers’ bug-reproduction tasks. To address the ultimate research question “*What makes a good issue report?*”, we conducted an empirical study using 34 active GitHub repositories to quantitatively analyze the difference between visual is-

sue reports and non-visual ones, and qualitatively analyze the characteristics of the visuals and their usage in raising bugs. The quantitative analysis importantly shows that visual issue reports are more likely to describe the bug-related issue (almost 1.5–2.5 times). Seven visual content kinds and five purpose kinds are classified from our qualitative analysis. The results show that visuals are frequently used to describe the program behavior and the user interface, with the primary purpose of introducing problems in reports. Furthermore, visuals are commonly used in raising GUI-related bugs while less frequently used in raising configuration bugs (the second common bug type in non-visual reports). Our study highlights the role that visuals play in GitHub issues, the next step is to further investigate the causality relationship between visuals and closed time by considering various confounding factors and to understand how developers perceive the impact of visuals on bug fixing through a survey. Other directions also include studying the usage of visuals that are attached in the issue conversation process, and the extension of the usage of visuals for reproducing bugs to downstream tasks.

Acknowledgment

This research was partially supported by JSPS KAKENHI Japan (Grant Numbers: JP21H04877, JP21K17725, JP22K17874, JP22K18630, JP23K16864) and Yasutaka Kamei is supported by Inamori Research Institute for Science, Kyoto, Japan (InaRIS Fellowship).

Data Availability Statements

The replication package that supports the findings of this study is available publicly.⁶

Declarations

Funding and/or Conflicts of interests/Competing interests

All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

Agrawal V, Lin YH, Cheng J (2022) Understanding the characteristics of visual contents in open source issue discussions: A case study of jupyter notebook. In: Proceedings of the the 26th International Conference on Evaluation and Assessment in Software Engineering 2022, pp 249–254

⁶ <https://doi.org/10.5281/zenodo.10565699>

- Bauer MW (2007) Content analysis. an introduction to its methodology – by klaus krippendorff from words to numbers. narrative, data and social science – by roberto franzi. *The British Journal of Sociology* 58(2):329–331, DOI https://doi.org/10.1111/j.1468-4446.2007.00153_10.x
- Bettenburg N, Just S, Schröter A, Weiß C, Premraj R, Zimmermann T (2007) Quality of bug reports in eclipse. In: *Proceedings of the 2014 Workshop on Eclipse Technology eXchange*, pp 21–25
- Blincoe K, Sheoran J, Goggins S, Petakovic E, Damian D (2016) Understanding the popular users: Following, affiliation influence and leadership on github. *Information and Software Technology* 70:30–39
- Borges H, Hora A, Valente MT (2016) Understanding the factors that impact the popularity of github repositories. In: *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp 334–344
- Breu S, Premraj R, Sillito J, Zimmermann T (2010) Information needs in bug reports: improving cooperation between developers and users. In: *Proceedings of the 14th Conference on Computer Supported Cooperative Work*, pp 301–310
- Casella G, Berger RL (2021) *Statistical inference*. Cengage Learning
- Catolino G, Palomba F, Zaidman A, Ferrucci F (2019) Not all bugs are the same: Understanding, characterizing, and classifying bug types. *Journal of Systems and Software* 152:165–181
- Chen AR, Chen THP, Wang S (2021) Demystifying the challenges and benefits of analyzing user-reported logs in bug reports. *Empirical Software Engineering* 26(1):1–30
- Cliff N (1993) Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114:494–509
- Cohen J (2013) *Statistical power analysis for the behavioral sciences*. Routledge
- Cooper N, Bernal-Cárdenas C, Chaparro O, Moran K, Poshyvanyk D (2021) It takes two to TANGO: combining visual and textual information for detecting duplicate video-based bug reports. In: *Proceedings of the 43rd International Conference on Software Engineering*, pp 957–969
- Dabic O, Aghajani E, Bavota G (2021) Sampling projects in github for MSR studies. In: *Proceedings of the 18th International Conference on Mining Software Repositories*, pp 560–564
- Dey T, Mousavi S, Ponce E, Fry T, Vasilescu B, Filippova A, Mockus A (2020) Detecting and characterizing bots that commit code. In: *Proceedings of the 17th international conference on mining software repositories*, pp 209–219
- Eisenhardt KM (1989) Building theories from case study research. *Academy of management review*
- Fazzini M, Moran K, Bernal-Cárdenas C, Wendland T, Orso A, Poshyvanyk D (2023) Enhancing mobile app bug reporting via real-time understanding of reproduction steps. *IEEE Trans Softw Eng* 49(3):1246–1272
- Feng S, Xie M, Xue Y, Chen C (2023) Read it, don't watch it: Captioning bug recordings automatically
- Fu L, Liang P, Zhang B (2022) Understanding code snippets in code reviews: A preliminary study of the openstack community. In: *Proceedings of the IEEE/ACM 30th International Conference on Program Comprehension (ICPC)*, pp 152–156

- Giger E, Pinzger M, Gall H (2010) Predicting the fix time of bugs. In: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, pp 52–56
- Golzadeh M, Decan A, Legay D, Mens T (2021) A ground-truth dataset and classification model for detecting bots in github issue and pr comments. *Journal of Systems and Software* 175:110911
- Golzadeh M, Decan A, Chidambaram N (2022) On the accuracy of bot detection techniques. In: Proceedings of the IEEE/ACM 4th International Workshop on Bots in Software Engineering (BotSE), pp 1–5
- Guo PJ, Zimmermann T, Nagappan N, Murphy B (2010) Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In: Proceedings of the 32nd International Conference on Software Engineering, pp 495–504
- Hata H, Treude C, Kula RG, Ishio T (2019) 9.6 Million Links in Source Code Comments: Purpose, Evolution, and Decay. In: Proceedings of the 41st International Conference on Software Engineering, p 1211–1221
- He R, He H, Zhang Y, Zhou M (2023) Automating dependency updates in practice: An exploratory study on github dependabot. *IEEE Transactions on Software Engineering*
- Herzig K, Just S, Zeller A (2013) It's not a bug, it's a feature: how misclassification impacts bug prediction. In: Proceedings of the 35th International Conference on Software Engineering, pp 392–401
- Johnson J, Mahmud J, Wendland T, Moran K, Rubin J, Fazzini M (2022) An empirical investigation into the reproduction of bug reports for android apps. In: Proceedings of the IEEE 29th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 321–322
- Joorabchi ME, MirzaAghaei M, Mesbah A (2014) Works for me! characterizing non-reproducible bug reports. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp 62–71
- Krejcie RV, Morgan DW (1970) Determining sample size for research activities. *Educational and Psychological Measurement* 30(3):607–610
- Krippendorff K (2019) *Content Analysis: An Introduction to Its Methodology*. SAGE Publications, Inc.
- Kruskal WH, Wallis WA (1952) Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association* 47(260):583–621
- Kuramoto H, Kondo M, Kashiwa Y, Ishimoto Y, Shindo K, Kamei Y, Ubayashi N (2022) 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 IEEE/ACM 30th International Conference on Program Comprehension, IEEE, pp 511–515
- Lal S, Sureka A (2012) Comparison of seven bug report types: A case-study of google chrome browser project. In: Proceedings of the 19th Asia-Pacific Software Engineering Conference, APSEC, vol 1, pp 517–526
- Li H, Yan M, Sun W, Liu X, Wu Y (2023) A first look at bug report templates on github. *Journal of Systems and Software* 202:111709
- Moran K, Linares-Vázquez M, Bernal-Cárdenas C, Poshyvanyk D (2015) Auto-completing bug reports for android applications. In: Proceedings of the 10th Joint

- Meeting on Foundations of Software Engineering, p 673–686
- Nayebi M (2020) 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*, Association for Computing Machinery, pp 49–52
- Nayrolles M, Hamou-Lhadj A (2018) Towards a classification of bugs to facilitate software maintainability tasks. In: *Proceedings of the 1st International Workshop on Software Qualities and Their Dependencies*, p 25–32
- Ohira M, Hassan AE, Osawa N, Matsumoto K (2012) The impact of bug management patterns on bug fixing: A case study of eclipse projects. In: *Proceedings of the 28th International Conference on Software Maintenance*, pp 264–273
- Panjer LD (2007) Predicting eclipse bug lifetimes. In: *Proceedings of the 4th International Workshop on Mining Software Repositories*, p 29
- Pearson K (1900) X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50(302):157–175
- Planning S (2002) The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology* p 1
- Potter MC, Faulconer BA (1975) Time to understand pictures and words. *Nature* 253(5491):437–438
- Romano J, Kromrey JD, Coraggio J, Skowronek J, Devine L (2006) Exploring methods for evaluating group differences on the nsse and other surveys: Are the t-test and cohen’s d indices the most appropriate choices? In: *Proceedings of the Annual Meeting of the Southern Association for Institutional Research*, pp 1–51
- Saadat S, Colmenares N, Sukthankar G (2021) Do bots modify the workflow of github teams? In: *Proceedings of the IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*, IEEE, pp 1–5
- Soltani M, Hermans F, Bäck T (2020) The significance of bug report elements. *Empirical Software Engineering* 25(6):5255–5294
- Song Y, Mahmud J, Zhou Y, Chaparro O, Moran K, Marcus A, Poshyvanyk D (2022) Toward interactive bug reporting for (android app) end-users. In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Association for Computing Machinery, p 344–356
- Stemler S (2000) An overview of content analysis. *Practical assessment, research, and evaluation* 7(1):17
- Sullivan GM, Feinn R (2012) Using effect size—or why the p value is not enough. *Journal of graduate medical education* 4(3):279–282
- Taesiri MR, Macklon F, Bezemer CP (2022) Clip meets gamephysics: Towards bug identification in gameplay videos using zero-shot transfer learning. *arXiv preprint arXiv:220311096*
- Wang D, Xiao T, Thongtanunam P, Kula RG, Matsumoto K (2021) Understanding shared links and their intentions to meet information needs in modern code review. *Empir Softw Eng* 26(5):96

- Wang D, Xiao T, Treude C, Kula RG, Hata H, Kamei Y (2023) Understanding the role of images on stack overflow. arXiv preprint arXiv:230315684
- Wang J, Li M, Wang S, Menzies T, Wang Q (2019) Images don't lie: Duplicate crowdtesting reports detection with screenshot information. *Information and Software Technology* 110:139–155
- Weiss C, Premraj R, Zimmermann T, Zeller A (2007) How long will it take to fix this bug? In: *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*, pp 1–1
- Xiao T, Wang D, McIntosh S, Hata H, Kula RG, Ishio T, Matsumoto K (2021) Characterizing and mitigating self-admitted technical debt in build systems. *IEEE Transactions on Software Engineering*
- Ye D, Xing Z, Kapre N (2017) The structure and dynamics of knowledge network in domain-specific q&a sites: A case study of stack overflow. *Empirical Softw Engg* p 375–406
- Zanaty F, Hirao T, McIntosh S, Ihara A, Matsumoto K (2018) An empirical study of design discussions in code review. In: *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp 1–10
- Zhang F, Khomh F, Zou Y, Hassan AE (2012) An empirical study on factors impacting bug fixing time. In: *Proceedings of the 19th Working Conference on Reverse Engineering*, pp 225–234
- Zimmermann T, Premraj R, Bettenburg N, Just S, Schröter A, Weiss C (2010) What makes a good bug report? *IEEE Transactions on Software Engineering* 36(5):618–643
- Zimmermann T, Nagappan N, Guo PJ, Murphy B (2012) Characterizing and predicting which bugs get reopened. In: *Proceedings of the 34th International Conference on Software Engineering*, pp 1074–1083
- Zou W, Xia X, Zhang W, Chen Z, Lo D (2015) An empirical study of bug fixing rate. In: *Proceedings of the 39th Annual Computer Software and Applications Conference*, pp 254–263