ASTES

Advances in Science, Technology and Engineering Systems Journal
Vol. 9, No. 5, 1-8 (2024)
www.astesj.com
Special Issue on Computing, Engineering and Multidisciplinary Sciences

ASTES Journal
ISSN: 2415-6698

Web Application Interface Data Collector for Issue Reporting

Diego Costa*, Gabriel Matos, Anderson Lins, Leon Barroso, Carlos Aguiar, Erick Bezerra

SIDIA Institute of Science and Technology, Manaus, Brazil

ARTICLE INFO

Article history:
Received: 24 April, 2024
Revised: 01 August, 2024
Accepted: 03 August, 2024
Online: 14 September, 2024

Keywords:
Bug Reporting
Software Management
Web Application
Browser API

ABSTRACT

Insufficient information is often pointed out as one of the main problems with bug reports as most bugs are reported manually, they lack detailed information describing steps to reproduce the unexpected behavior, leading to increased time and effort for developers to reproduce and fix bugs. Current bug reporting systems lack support for self-hosted systems that cannot access third-party cloud environments or Application Programming Interfaces due to confidentiality concerns. To address this, we propose Watson, a Typescript framework with a minimalist User Interface developed in Vue.js. The objectives are to minimize the user's effort to report bugs, simplify the bug reporting process, and provide relevant information for developers to solve it. Watson was designed to capture user's interactions, network logs, screen recording, and seamlessly integration with issue tracker systems in self-hosted systems that cannot share their data to external Application Programming Interfaces or cloud services. Watson also can be installed via Node Package Manager and integrated into most JavaScript or TypeScript web projects. To evaluate Watson, we developed an Angular-based application along with two usage scenarios. First, the users experimented the application without using Watson and once they found a bug, they reported it manually on GitLab. Later, they used the same application, but this time, whenever they detect another bug, they reported it through Watson User Interface. Watson, as stated by the experiment participants and the evidences, is useful and helpful for development teams to report issues and provide relevant information for tracking bugs. The identification of bug root causes was almost three times more effective with Watson than manual reporting.

1. Introduction

This paper is an extension of work originally presented at the 2023 IEEE 30th Annual Software Technology Conference (STC) [1]. Bug reports play a crucial role in software maintenance, enabling developers to prioritize, reproduce, identify, and resolve defects [2], [3]. Detailed information is expected from the reports, such as the unexpected behavior, the steps to reproduce it, logs, or screenshots, and others, so developers may recreate it to find a solution [2, 4, 5].

Insufficient information is often pointed out as one of the main problems with bug reports, generally most bugs are reported manually by end-users or testers, the reports lack of details and sufficient information describing the steps to reproduce the unexpected behavior to allow the developers to find a solution [2, 6].

The most common way to report bugs is through issue tracking systems, but in the majority cases, there is not any standard, which causes misinformation for the development team due the unclear or insufficient data [7]. Steps to reproduce the bug, stack trace errors, test case scenarios, logs, and images are factors that impact the quality of the bug reports [6, 8, 9, 10, 11, 12].

In this paper, we introduce Watson, a framework developed in Typescript[1] with an User Interface (UI) developed in Vue.js[2]. The objective is to save time and effort for the person who is going to report the bug, and standardize bug reporting by collecting fundamental information that will aid the developers to reproduce the undesired behavior, as such as: user interaction on the page, network requests, and screen video. The main points of Watson are that it is a framework that can be installed via Node Package Manager (NPM), it can be integrated into most Javascript[3] or Typescript web projects, and it is designed for self-hosted systems that cannot share its data to external Application Programming Interfaces (APIs) or cloud services.

*Corresponding Author: Diego Costa, SIDIA Institute of Science and Technology, Manaus, Brazil, diego.costa@sidia.com
[1]https://www.typescriptlang.org/
[2]https://vuejs.org/
[3]https://developer.mozilla.org/en-US/docs/Web/JavaScript

Watson was evaluated empirically, using a test application developed in Angular[4] and inviting developers and testers to use it with Watson. The goal was to evaluate the participant's perception of Watson usefulness in comparison with manually reporting a bug on GitLab[5]. Experienced web developers an testers judged Watson as useful and Watson's features to collect information proved helpful in identifying the root cause of bugs reported.

The rest of this work is organized as follows: Section 2 provides details about the problem and related work. Section 3 provides details about the proposed software. Section 4 presents the results of the use of Watson. Section 5 concludes by discussing the main points found.

2. Related Works

Our research focused on bug reporting tools that provided relevant and effective information for software maintenance, mainly for web applications, but due to the lack of recent works, we expanded the search for Android applications and approaches that are emerging, such as machine learning.

The authors of [13] propose an automated bug reporting system which serves as a foundation for testing frameworks in web applications, generating failure reports based on the test cases. The generated reports consist of the number of test cases executed, the number of failed, passed and skipped tests, and the time it took to perform the tests. When the report is ready, it is checked if it is a duplicated bug report, then it is mailed to the development team. The tool was used by the authors to simulate regression test cases, resulting in an 8% reduction in test execution time. Additionally, it summarized the bug report, reducing human effort and time spent filtering duplicated bug reports.

In the work [14], they created a tool, Euler, that automatically analyzes the written description of a bug report, evaluates the quality of reproduction steps, and provides feedback to users about ambiguous or missing steps. Neural sequence labeling combined with discourse patterns and dependency parsing identifies sentences and individual steps to reproduce. It matches these steps to program state and Graphical User Interface (GUI) in a graph-based execution model. An empirical evaluation was conducted to determine the accuracy of Euler in identifying and assessing the quality of reproduction steps for bug reports. The results indicated that Euler correctly identified 98% of the existing steps to reproduce and 58% of the missing ones, and 73% of its quality annotations being accurate.

The work [2] presents Bee, an open-source tool that can be integrated with GitHub[6] and automatically analyzes user-written bug reports using machine learning textual classification. It offers insights into the system's observed comportment, expected comportment, and reproduction steps for the reported bugs. As result they achieved 87% recall, indicating the ability to correctly detect and classify the described sentences.

In order to generate better bug reports for Android, CrashScope [15] was created. It works by collecting system version and hardware information, the application state, the user entry text description to reproduce the bug, the GUI events, and app's stack trace error. The purpose was to assess the reliability and comprehensibility of reports generated by CrashScope relative to five current tools. To evaluate this work, they used 8 real world open source applications bug reports extracted from their corresponding issue trackers. They invited 16 users to reproduce 4 bugs reported using CrashScope and 4 bugs reported manually. It was discovered that reports produced by CrashScope were equally reproducible compared to those from other tools, although it yielded more comprehensible and beneficial reports for developers.

In [16], the author presented a chatbot that designed for Android, combines dynamic software analysis, natural language processing, and automated report quality assessment to assist users in writing better descriptions and receiving issue reports. By inviting 18 end-users to identify 12 bugs across 6 Android apps, we found that Burt provides more accurate and complete reproduction steps than Itrac, a template-based bug reporting system used by another 18 users.

The work [17] proposes an automated tool for integrating user feedback into the testing process. In order to achieve this, they collected datasets of mobile application issues reviews along with user feedbacks to train a machine learning algorithm that would be capable to link the user's feedback to stack traces with the objective to relate a feedback that might describe the cause of a failure to a bug. By following this process, they concluded user feedback is highly promising to integrate into the testing process of mobile apps as it complements the capabilities of testing tools identifying bugs that are not reveled in this phase of software development, facilitating the diagnosis of bugs or crashes.

In the study [18], the authors explored automating Android bug replaying using Large Language Models (LLMs). Motivated by the success of these models, they proposed AdbGPT, a method for reproducing bugs from bug reports via prompt engineering. By following this approach, they demonstrated 81.3% effectiveness and efficiency to reproduce bugs from users' reports, and in terms of average time to reproduce bug reports, the AdbGPT outperformed the average time of experiment participants.

Other studies on bug reporting tools for web applications [19, 20] and commercial tools like Usersnap[7], BugHerd[8], and Bird Eats Bugs[9] utilize browser extensions or embedded scripts to capture Document Object Model (DOM) events, stack trace errors, and screenshots. Data is sent to cloud services but lacks integration with self-hosted systems that restrict sending its private data to external APIs. This issue is addressed by Watson, a framework installable via NPM for JavaScript or TypeScript web projects. It provides flexibility for development teams to integrate and utilize self-hosted systems, enabling configuration of Watson to use entirely their own systems while collecting crash report data without external environments.

[4] https://angularjs.org/
[5] https://about.gitlab.com/
[6] https://github.com
[7] https://usersnap.com/
[8] https://bugherd.com/
[9] https://birdeatsbug.com/

3. Watson Framework

Watson was developed in Typescript, offering an API to help collect important information. Watson collects information such as DOM events related to the user's interaction, when a user interacts with the application, Watson interceptors will collect the web page events, such as mouse clicks, network requests and screen video to capture what the user sees on the page. This information will be passed to the reporter class that implements the interface WatsonReporter to attach the information to a bug report with a description provided by the user and the Watson collected data, then it will send to the chosen issue tracking system.

As presented in Figure 1, Watson acts in the native browser API, injecting a collection of interceptors that will intercept data during web application usage. When the user initiates the recording process through the UI, Watson begins collecting data. Upon completion of the recording, the gathered information can be transmitted to an issue tracking system based on the preconfigured reporter implementation. This may involve utilizing a built-in reporter such as GitLabReporter or implementing a customized reporter to facilitate integration with alternative issue trackers or systems.

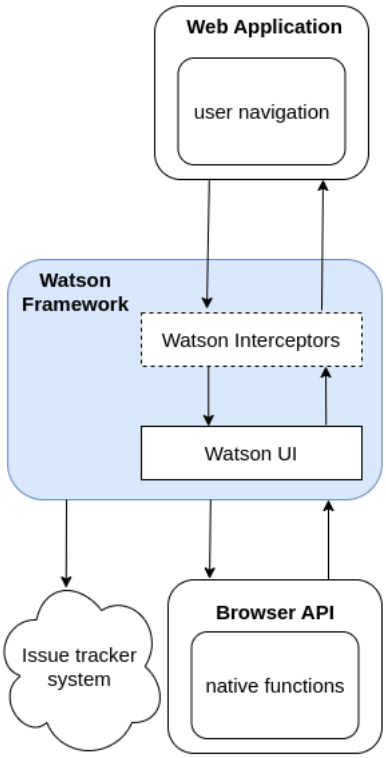
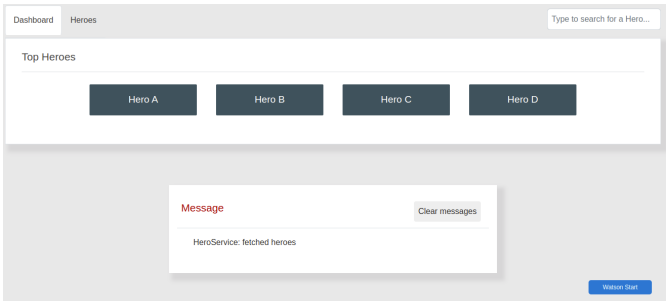
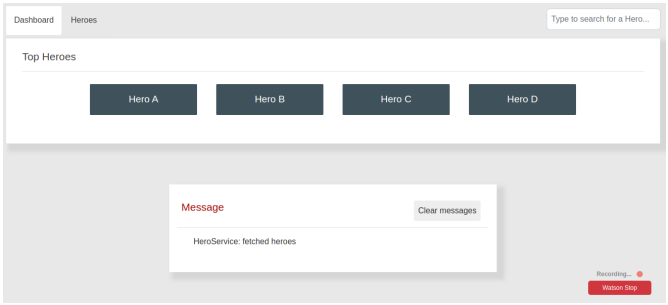


Figure 1: Watson Architecture

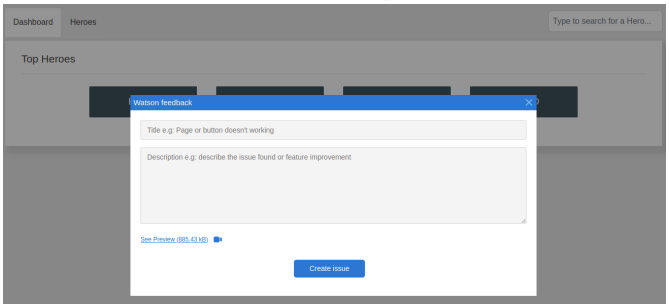
The UI workflow used to report bugs with Watson is shown in Figure 2. Starting with the Watson UI start button (Figure 2a), users can proceed as normal in their testing scenario or regular application usage. Once Watson is running, it attaches interceptors to the browser's native API, listens for native events, collects data, and redirects event parameters to the original event calls as proxies, allowing it to intercept and save the user interaction with the web page.



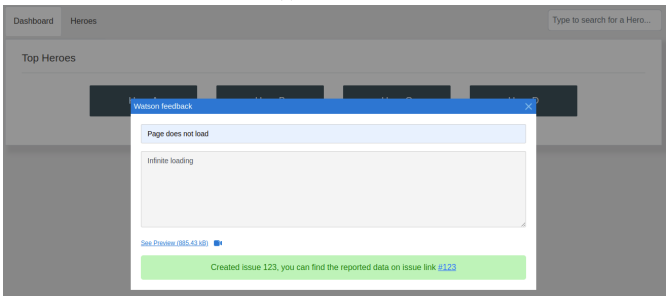
(a) Watson start



(b) Watson stop



(c) Create issue



(d) Issue created

Figure 2: Watson UI

After the data is intercepted, it gets transferred to the initial web browser function API to perform its regular actions. To stop the data collection, the user can use the stop button on Watson UI (Figure 2b), then, as shown in Figure 2c, a dialog message opens for users to provide additional description about the bug and a title for the issue, which will be sent to an external issue tracking system such as GitHub[10], Jira[11] or another one which the developers may have

[10]https://github.com
[11]https://www.atlassian.com/software/jira

previously integrated Watson to it. As in this case, it was integrated with GitLab, it returned the issue link in Figure 2d.

3.1. Implementation details

Figure 3 presents Watson's class diagram. It is composed of four components: **Watson**, **WatsonConfig**, **WatsonReporter**, and **Watson interceptors**.

Watson can be installed on a web project with NPM. After installation, the Watson instance is a singleton and must be initialized at application start-up, initializing and attaching its injectors to the browser's native API, functioning as proxies. The Watson framework must be initialized and configured by adding the code **Watson.getInstance(config)** on application startup code. The config paramter is object of type **WatsonConfig** that contains which interceptors should be active and the reporter implementation of **WatsonReporter** interface that has the responsibility to connect Watson to an external issue tracking system. This enables reporting of collected information alongside user description and issue title.

3.1.1. Watson

The Watson class connects all parts of the system and ensures the data capture functions properly. It manages the start and stop of data capture, stores the collected data, controls the event interceptors activating and deactivating the capture and also uses the reporter instance to send the collected data to issue tracker. With the config parameter, it enables which event interceptors that will be attached to native API and the event types that should be captured, and also defines the WatsonReporter instance to send the collected data.

3.1.2. WatsonConfig

The configuration interface determines which event interceptors should be activated. Currently, there are 3 event interceptors to capture user interaction data: network interceptor, DOM events interceptor, and the screen recorder. By default, all interceptors are enabled, but the development team can define which of them are active according to their requirements. For example, to disable screen recorder by setting **screenRecorder** option to false. Additionally, the implementation instance of the WatsonReporter interface must be specified in the config. This can utilize either a built-in implementation such as GitlabReporter or a custom report class that implements the interface.

3.1.3. WatsonReporter

This is the interface implementation required for reporting to external issue tracking system. A method must be implemented to send the collected data to an external issue tracking system like Jira, GitLab, or others. The development team can write its code to connect to the external system and implement this interface to send data. For this experiment, Figure 3 presents the GitLabReporter class designed to send the collected data to GitLab.

To implement this interface, the development team must first obtain access to the required issue tracking system API and follow its documentation to consume its API and have information about it such as API key, authentication, and available endpoints. Following this step, the developers can implement a class with this interface containing the methods to communicate with the issue tracking system and receive the data collected by Watson to finally transmit them.

The WatsonReporter interface has a method called **reportData**, accepting two parameters. The first parameter consists of the issue's basic information, including its title and description, while the second one is the Watson collected data. This function is responsible for generating issues within the issue tracker and returning an object containing the issue's ID and corresponding link, or null in case of failure.

3.1.4. Watson Interceptors

Watson interceptors are built-in functions that will be attached on native API, in order to listen network requests and DOM events

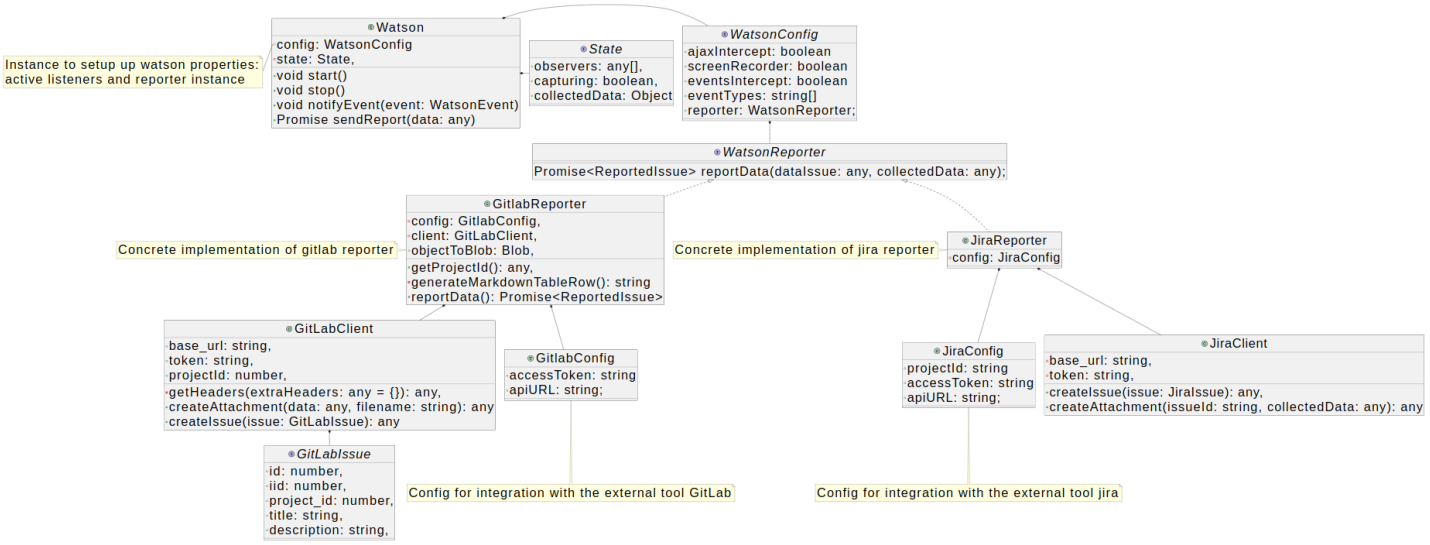


Figure 3: Watson Class Diagram

specified in the event types list, according to active event interceptors specified in the Watsonconfig when Watson is instantiated. These functions will be added to a list of observables at Watson states. Watson has three types of interceptors:

- **Video recording**: Watson utilizes the MediaStream API[12] to capture the user's screen activity. By clicking the start button within the Watson UI, the user initiates the recording process; while stopping it requires clicking the stop button. Besides bug reporting, this can be used to record test scenarios. The recorded video will be attached in the issue report to be sent along with the logs.
- **Network requests**: As represented in Figure 4, Watson injects the ajax interceptor to the XMLHttpRequests API[13] for collecting vital data about network requests. As soon the application is in use, it sends Hypertext Transfer Protocol (HTTP) requests based on user interactions. The native API is proxied, allowing the interceptor to gather information about the network requests such as headers, start and end time, status codes, sent and received data, and request duration. All collected information is stored and used to generate a JSON log file, which will be attached on the issue report.

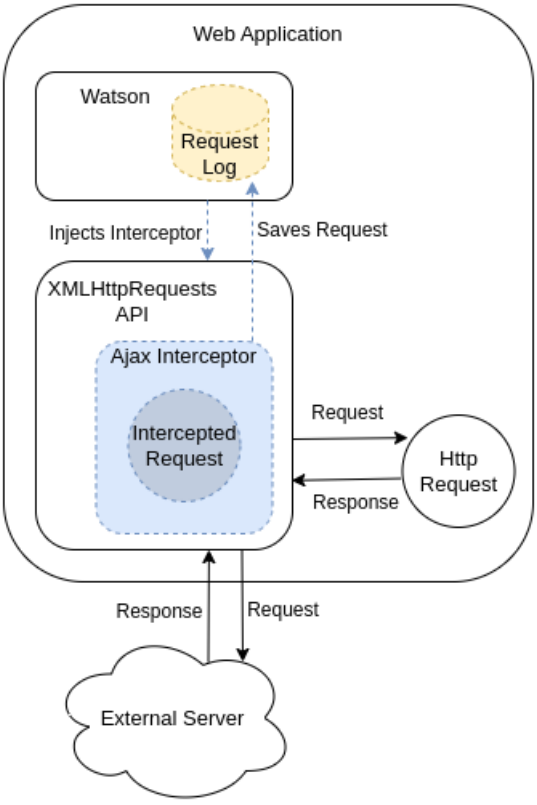


Figure 4: Watson - Network Interceptor

- **Capturing DOM events**: In order to collect the user's interaction with the web application, such as mouse and keyboard actions, Watson injects its interceptor into the native API[14] of AddEventListener, gathering data on events such as clicks, double clicks, mouse enter, mouse leave along with element details including HyperText Markup Language (HTML) tag name, node xpath, text content, as shown be seen in Figure 5.

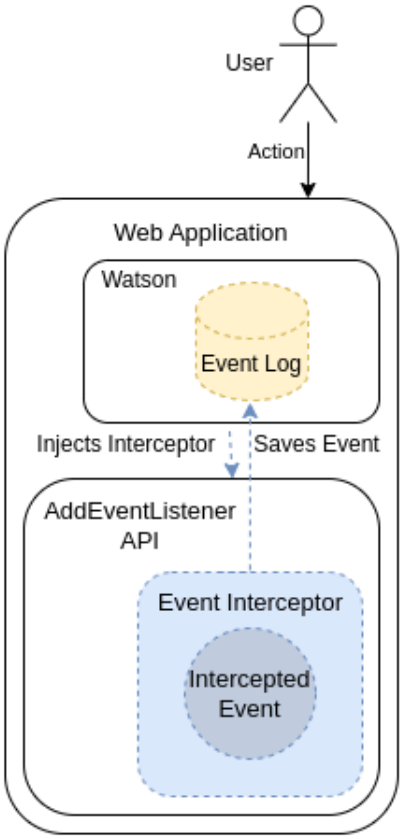


Figure 5: Watson - Event Interceptor

3.1.5. *GitlabReporter*

It is a concrete implementation of interface WatsonReporter, it can create an issue on Gitlab through its Web API and upload the collected information to the created issue.

The components of the reporting system: GitLabClient, GitLabConfig, and GitLabIssue. Each component is detailed below:

GitlabConfig serves as the interface to represent the minimum information necessary to identify the project and access it through the Gitlab API. It requires the server API url, as it may be a self-hosted server, and a valid access token registered to a Gitlab bot authorized to create issues within the project. These information are necessary during the configuring of the GitlabReporter as a WatsonReporter instance at application launch, all those configuration must be provided and setting up by the development team.

GitlabClient is responsible to use the Gitlab Web API use the GitlabConfig information. It is able to create issues and upload attachments to an issue.

[12]https://developer.mozilla.org/en-US/docs/Web/API/MediaStream
[13]https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest
[14]https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

GitlabIssue Is a representation of represents a created issue on Gitlab. It contains basic issue details like issue ID, project ID, title and description.

4. Test Case

A web application was developed with Angular to evaluate the Watson framework. Watson was set up through NPM, configured and incorporated into the application accordingly. As previously mentioned, a GitLab reporter class that implemented the interface for Watson reporter was created to integrate and report the issue to Gitlab.

As in the works of [3, 14, 15, 16, 21], an empirical experiment was conducted in two parts to evaluate if the users enjoyed Watson's functionalities and if these would be relevant for debugging and solving issues.

As described in Table 1, an online questionnaire was created for the users to answer after completing the tests. This questionnaire aimed to assess user perceptions of Watson's features and effectiveness in resolving bugs relative to manual reporting. Quantitative and qualitative questions were included, with participants able to respond on a scale of 0 to 5, providing feedback on areas for improvement and suggestions.

Table 1: Questionnaire

1	Considering the manual method of creating an issue, how error-prone do you think this method is?
2	Considering the Watson method of creating an issue, how error-prone do you think this method is?
3	What would be the main reasons to use the manual method?
4	What would be the main reasons to use Watson?
5	Considering the method using Watson, how much would the network log be relevant to debug?
6	Considering the method using Watson, how much would DOM events be relevant to debug?
7	Considering the method using Watson, how much would the video be relevant to debug?
8	How relevant would Watson be to your project's issue report?
9	If you use or know other issue trackers. How much value would Watson add as a library compared to other issue trackers?
10	Do you have any suggestions for improvement? If yes, which one?

Six web developers and six testers were invited to try the application, discover, and report bugs. Some bugs were found in the web application, including issues with the network connection to database, visual elements, and behavior. The participants were not told about which bugs were incorporated into the application, and Watson was installed and configured by us.

In the fist part of the experiment, the users tried the web application without Watson installed, and once they noticed a bug, they manually reported it on GitLab, describing and attaching anything they judged necessary to solve the bug. In the second part, they tried the application with Watson installed. As in the first part, they explored the system until they found a bug, but this time, with Watson available in the application, they used Watson's UI to report the bug.

Upon completion of testing the application across both scenarios, participants responded to an online survey assessing their satisfaction with Watson's functionalities and the relevance of data gathered by Watson towards addressing reported bugs.

The purpose of the first and second questions was to evaluate how error-prone manual report and the Watson report were. For these two questions, the lower value is better because it indicates less error-prone than a higher value. For Watson method, the average grade was 2.22, and the manual method was 3.11.

The third and fourth questions were about the motives to use one method over another. The distribution about the main reasons the participants considered using the manual method was three votes for duplicate issue report prevention, two votes for organization, one vote for quickly generating evidences, one vote for using a text editor, and one vote for less-error prone, indicating that the participants noticed the prevention of duplicate issues more. For Watson, the results were of 5 votes for taking less time to report an issue, two votes for more bug evidences and two votes for less-error prone, indicating that the participant's preferred Watson because they took less time to report an issue.

The fifth, sixth, and seventh questions are about the relevance of Watson features: network logs, DOM events, and screen records collected by Watson. The average rates for these features were 4.11, 5 and 4.78, respectively.

The eighth question asks about the relevance of Watson for the participant's project issue report. The average rate for it was 4.89. The ninth question asks to compare Watson's reports to other tools if the participant used another one. The average rate for it was 4.22.

The tenth question asks for suggestions. The suggestions the participants cited were to add a text editor for the Watson's description field, an option to download the logs and generated video, an option to choose which project they would like to report the issue and label it, and mainly the history search to avoid duplicate issues before sending the new issue.

To evaluate the efficiency of Watson compared to manual reporting, all the bug reports were evaluated and analyzed for its potential to identify the root cause of the issue. This evaluation involved utilizing videos and user descriptions to recreate the bug scenario, as well as examining the collected stack traces to trace the faulty functionality.

Based on evidence provided by participants through manual bug reporting, it was found the bug root cause on 18.75% of the manually reported issues. In contrast, using Watson's reports containing stack traces and video, the number of issues that could be determined the bug root cause increased to 63.63%.

The recorded video helped to reproduce all of the bug reported with Watson. Additionally, the event and network logs were used to investigate the application's behavior during the occurrence of these bugs.

5. Conclusion

This work presented Watson, a software program developed to standardize the bug report process. Its main purposes are to reduce the

effort to collect bug evidences and provide relevant information for developers to solve it.

Experienced web developers and testers tried Watson and perceived it as useful and relevant for crash reports. Watson scored an average rate of 4.89 out of 5 from the participants questionnaire when asked about its impact on issue reporting in their web projects.

Watson's features for collecting event and network stack traces proved useful in identifying the root cause of bugs in 63.63% of cases when using Watson, against to only 18.75% of the issues reported manually.

This study focused on Watson's ability to gather necessary data for bug reports and make easier for user to report bugs with evidences. Unlike from many other reporting tools, Watson operates independently of any browser plugins and does not require sending data to external servers.

The results of the analysis provided by this article are limited and defined by the test case performed. To interpret the knowledge obtained as a general approach, additional use cases in different projects are needed. Therefore, our proposal suits the need to use a bug reporting tool without using a third-party cloud, which is a requirement for scenarios where highly confidential projects are developed.

As points of improvement mentioned by the participants, most of them were related to the user experience, such as the text editor, options to download collected information, and the recorded screen. It was also suggested to Watson block duplicated bugs.

The main achievement of this project was creating a framework using TypeScript, which can be easily incorporated into web applications and various issue tracking systems. This enables development teams to conveniently obtain standardized bug reports and user feedback. Projects can benefit from this solution as it allows them to bypass third-party servers and incorporate a reporting tool within their self-hosted system, especially when handling sensitive data.

As future work, besides the already mentioned user experience suggestions, we plan to develop a mechanism to prevent duplicate bug reports using Watson and machine learning techniques.

Acknowledgment This work is the result of the R&D project *Projeto de Engenharia de Software e Ciência de Dados aplicados ao Desenvolvimento de Sistemas*, performed by Sidia Instituto de Ciência e Tecnologia in partnership with Samsung Eletrônica da Amazônia Ltda., using resources from Federal Law No. 8.387/1991, and its disclosure and publicity are under the provisions of Article 39 of Decree No. 10.521/2020. Rodrigo José Borba Fernandes, Isabelle Maria Farias de Lima Teixeira, and Thiago Cruz Ferraz former members.

References

[1] G. Matos, D. Costa, A. Lins, E. Bezerra, L. Barroso, C. Aguiar, T. Ferraz, I. Teixeira, "Watson: Web Application Interface Data Collector for Feedback Reporting," in 2023 IEEE 30th Annual Software Technology Conference (STC), 3–6, 2023, doi:10.1109/STC58598.2023.00007.

[2] Y. Song, O. Chaparro, "Bee: A tool for structuring and analyzing bug reports," in Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 1551–1555, 2020, doi:10.1145/3368089.3417928.

[3] K. Moran, "Enhancing android application bug reporting," in Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, 1045–1047, 2015, doi:10.1145/2786805.2807557.

[4] M. Erfani Joorabchi, M. Mirzaaghaei, A. Mesbah, "Works for me! characterizing non-reproducible bug reports," in Proceedings of the 11th working conference on mining software repositories, 62–71, 2014, doi:10.1145/2597073.2597098.

[5] M. Soltani, F. Hermans, T. Bäck, "The significance of bug report elements," Empirical Software Engineering, **25**, 5255–5294, 2020, doi:10.1007/s10664-020-09882-z.

[6] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, C. Weiss, "What makes a good bug report?" IEEE Transactions on Software Engineering, **36**(5), 618–643, 2010, doi:10.1109/TSE.2010.63.

[7] D. Huo, T. Ding, C. McMillan, M. Gethers, "An empirical study of the effects of expert knowledge on bug reports," in 2014 IEEE International Conference on Software Maintenance and Evolution, 1–10, IEEE, 2014, doi:10.1109/ICSME.2014.22.

[8] J. Wang, M. Li, S. Wang, T. Menzies, Q. Wang, "Images don't lie: Duplicate crowdtesting reports detection with screenshot information," Information and Software Technology, **110**, 139–155, 2019, doi:10.1016/j.infsof.2019.03.003.

[9] N. Cooper, C. Bernal-Cárdenas, O. Chaparro, K. Moran, D. Poshyvanyk, "It Takes Two to Tango: Combining Visual and Textual Information for Detecting Duplicate Video-Based Bug Reports," in 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 957–969, 2021, doi:10.1109/ICSE43902.2021.00091.

[10] M. Bheree, J. Anvik, "Identifying and Detecting Inaccurate Stack Traces in Bug Reports," in 2024 7th International Conference on Software and System Engineering (ICoSSE), 9–14, IEEE Computer Society, Los Alamitos, CA, USA, 2024, doi:10.1109/ICoSSE62619.2024.00010.

[11] Y. Noyori, H. Washizaki, Y. Fukazawa, K. Ooshima, H. Kanuka, S. Nojiri, "Deep learning and gradient-based extraction of bug report features related to bug fixing time," Frontiers in Computer Science, **5**, 1032440, 2023, doi:10.3389/fcomp.2023.1032440.

[12] R. Krasniqi, H. Do, "A multi-model framework for semantically enhancing detection of quality-related bug report descriptions," Empirical Software Engineering, **28**(2), 42, 2023, doi:10.1007/s10664-022-10280-w.

[13] Y. Sharma, A. Dagur, R. Chaturvedi, et al., "Automated bug reporting system in web applications," in 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI), 1484–1488, IEEE, 2018, doi:10.1109/ICOEI.2018.8553850.

[14] O. Chaparro, C. Bernal-Cárdenas, J. Lu, K. Moran, A. Marcus, M. Di Penta, D. Poshyvanyk, V. Ng, "Assessing the quality of the steps to reproduce in bug reports," in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 86–96, 2019, doi:10.1145/3338906.3338947.

[15] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, D. Poshyvanyk, "Automatically discovering, reporting and reproducing android application crashes," in 2016 IEEE international conference on software testing, verification and validation (icst), 33–44, IEEE, 2016, doi:10.1109/ICST.2016.34.

[16] Y. Song, J. Mahmud, Y. Zhou, O. Chaparro, K. Moran, A. Marcus, D. Poshyvanyk, "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, 344–356, 2022, doi:10.1145/3540250.3549131.

[17] G. Grano, A. Ciurumelea, S. Panichella, F. Palomba, H. C. Gall, "Exploring the integration of user feedback in automated testing of Android applications," in 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), 72–83, 2018, doi:10.1109/SANER.2018.8330198.

[18] S. Feng, C. Chen, "Prompting Is All You Need: Automated Android Bug Replay with Large Language Models," in Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, 1–13, 2024, doi:10.1145/3597503.3608137.

[19] B. Burg, R. Bailey, A. J. Ko, M. D. Ernst, "Interactive record/replay for web application debugging," in Proceedings of the 26th annual ACM symposium on User interface software and technology, 473–484, 2013, doi:10.1145/2501988.2502050.

[20] J. Hibschman, H. Zhang, "Unravel: Rapid web application reverse engineering via interaction recording, source tracing, and library detection," in Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology, 270–279, 2015, doi:10.1145/2807442.2807468.

[21] J. Johnson, J. Mahmud, T. Wendland, K. Moran, J. Rubin, M. Fazzini, "An Empirical Investigation into the Reproduction of Bug Reports for Android Apps," in 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 321–322, 2022, doi:10.1109/SANER53432.2022.00048.