Connecting Issue Tracking Systems and Continuous Integration / Continuous Delivery Platforms for Improving Log Analysis: A Tool Support

Oskar Picus[a] and Camelia Şerban[b]
Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1 M. Kogălniceanu, Cluj-Napoca, Romania

Keywords: Software Maintenance, Log Analysis, Pipeline Failure, Issue Tracking Systems.

Abstract: As the software industry embraces more and more DevOps practices, issue tracking systems and Continuous Integration / Continuous Delivery tools have become of utmost importance. However, as software projects' complexity increases, so does the amount of logs that are generated. As such, in case of a pipeline failure, finding its root cause by manually inspecting the resulting logs proves to be difficult and time-consuming. Research is limited on connecting these two types of systems and few or none of the proposals implementing this connectivity fully leverage the power of issue tracking or automatically running pipelines, among other features of these tools. Furthermore, none of the approaches accomplish automated log analysis of pipeline failures. Aiming to overcome this gap, in this paper, we propose an issue tracking system which connects to GitHub Actions to automatically analyse the logs of pipeline failures and generates an issue report containing its findings. Our contribution is two-folded: firstly, it introduces a tool for automatically analysing logs of pipeline failures; secondly, it makes advancements into facilitating the software maintenance process. The source code of the tool is available at https://github.com/bugsby-project, while its demonstration video can be found at https://figshare.com/s/47088a5a3bcb019acf41.

1 INTRODUCTION

An issue tracking system (ITS) is a software tool capable of managing issues such as defects or feature requests for a software project, typically used by the respective project's development team and other stakeholders. In time, ITSs have taken on various social aspects related to the software development process (Ortu et al., 2016), becoming, in this manner, knowledge repositories for projects. As such, ITSs are now essential tools to the software development life cycle.

Continuous Integration / Continuous Delivery (CI/CD) is a practice of automating various stages of the software development process. CI ensures that newly written code is well integrated with existing one, by triggering a script which picks the latest code from the repository, builds it and runs various levels of automated tests. CD refers to the ability of deploying new code automatically after a successful CI by running a script which builds the necessary artifacts and deploys them to a test or production environment (Singh et al., 2019). The series of steps and processes necessary for enabling CI/CD in an application are collectively called a *pipeline*. Each pipeline is conceptually split into *stages* (in Jenkins (Jenkins, 2023a) nomenclature) or *steps* (if using GitHub Actions' (GitHub, 2023) terminology). Throughout this paper, we will use the term *step* to refer to this.

At the moment, ITSs and CI/CD tools are isolated from each other. Within ITSs, issues are raised by a participant of the software project in question, who then needs to manually fill in the issue details, a process both time consuming and error prone. In addition, whenever running the CI/CD scripts fails, current tools such as Jenkins or GitHub Actions only send an email to the project participants with no details on the reason for failure. Instead, they are being redirected to the `.log` file generated after running the script.

However, analysing logs proves to be a great challenge, especially now that applications generate gigabytes of data per hour (Zhu et al., 2019). Despite the fact that current CI/CD tools highlight the pipeline

[a] https://orcid.org/0000-0001-9209-9180
[b] https://orcid.org/0000-0002-5741-2597

Picus, O. and Şerban, C.
Connecting Issue Tracking Systems and Continuous Integration / Continuous Delivery Platforms for Improving Log Analysis: A Tool Support.
DOI: 10.5220/0012626700003687

In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2024)*, pages 379-386
ISBN: 978-989-758-696-5; ISSN: 2184-4895

step that has failed, thus restricting the amount of logs to analyse, examining the failed step's logs is still laborious. As such, manually inspecting logs to find the root cause of the problem proves to be impractical and remains a software engineering challenge to address.

To overcome the above mentioned limitations, we propose a new component integrated with our previously developed ITS, Bugsby (Picus and Serban, 2022), which implements various Natural Language Processing models to automatically analyse an issue report, a subset of them being used in this newly implemented component. As a CI/CD pipeline fails, this new component automatically fills an issue report containing details extracted from the pipeline's logs, thus, directly presenting software developers the failure's root cause. With this feature in mind, we believe our tool contributes to automating the software maintenance process.

Therefore, the contribution of this paper is twofolded. Firstly, it proposes a new tool which combines the capabilities of ITSs and CI/CD tools for achieving automatic log analysis of pipeline failures. Secondly, it presents an integration with GitHub Actions for the purpose of improving the workflow of software developers and facilitating the software maintenance process.

The rest of the paper is organised as follows. Section 2 presents related works on connecting ITS and CI/CD tools. Section 3 describes Bugsby's implementation, alongside its limitations, with Section 4 detailing its preliminary evaluation. Section 5 concludes the paper, also highlighting planned future work.

2 RELATED WORK

One could argue if ITSs and CI/CD tools should even be connected in the first place, but we believe this connectivity would bring benefits in areas such as bug triage or issue resolution.

However, research on this subject is limited. When researching CI/CD tools, the academia focuses on comparing various tools in terms of viability, performance and other metrics (Singh et al., 2019) and proposing further actions to automate the CI/CD process (Nogueira et al., 2018).

Park et al. (Park and Choi, 2022) have proposed in their work an issue tracking system based on Dev Ops whose main feature is that the ITS collects information from the CI/CD tool and links the commit message to the corresponding issue. Nevertheless, we believe the connectivity between these two systems is not fully leveraged, as it is only used to link an already raised ticket to a pipeline run. In addition, it is not clear how their proposed system behaves in case of a failure, which CI/CD tool they have used in their implementation and how the commit message is processed in order to match it to an existing issue.

Jenkins's behaviour, including that in case of a pipeline failure, can be configured using plugins. However, plugins which connect Jenkins to ITSs such as Jira (Jenkins, 2023b) are limited. They are only used to connect a CI/CD pipeline run to an issue, its only benefit being visibility. In addition, plugins which analyse the logs after a CI/CD failure such as (Jenkins, 2022) use a knowledge base composed of regular expressions that the user has to manually add in; disadvantages of this approach include difficulty in defining regular expressions for more complex and multi-line logs and inability of the plugin to recognise patterns in the messages *out of the box*.

Concerning GitHub Actions, its behaviour can be similarly configured using plugins. For connecting with Jira, multiple plugins are available[1]; however, they are only used for accessing Jira's features, such as creating issues or changing the status of an issue, in the GitHub Actions environment, with little to no new functionality added.

In relation to existing approaches, our proposed tool makes valuable contributions, connecting an ITS and a CI/CD tool in an innovative way by enabling automatic log analysis for pipeline failures, becoming, in this manner, a potential tool for reducing the time and cost of maintaining software systems.

3 TOOL OVERVIEW

To improve the process of inspecting logs of failed CI/CD pipeline runs, we have extended our previously developed tool, Bugsby (Picus and Serban, 2022), by integrating it with GitHub Actions, in order to leverage its CI/CD capabilities. Using log analysis, our tool will construct for each failed CI/CD pipeline run a prefilled issue, containing details regarding the root cause of the failure, aimed at improving visibility and minimising developers' time in terms of debugging and resolving issues.

3.1 Choosing the CI/CD Tool

We first had to decide which CI/CD tool Bugsby should connect to. For this, we have chosen GitHub Actions for three reasons. Firstly, GitHub has become in time the industry leader when it comes to code

[1] https://github.com/search?q=jira&type=marketplace

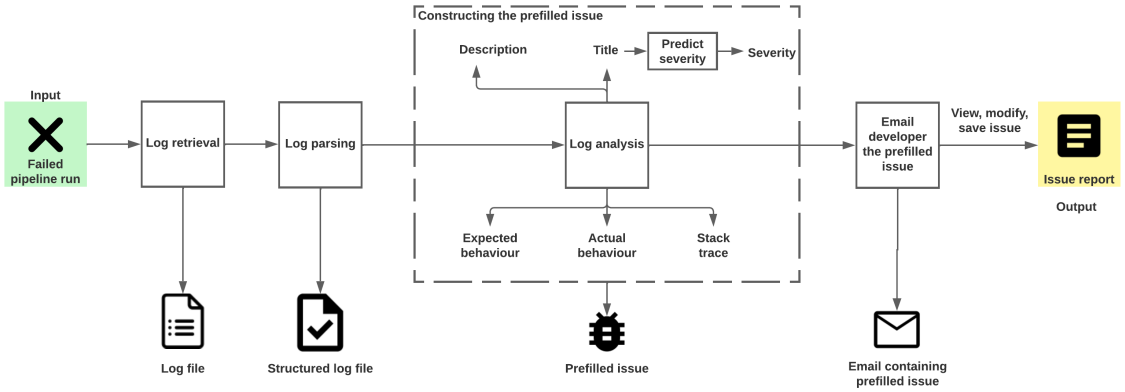

Figure 1: Bugsby's workflow.

hosting platforms, being used by more than 71 million users. Secondly, by making its REST API available, developers are able to quickly integrate their applications with GitHub. Lastly, GitHub Actions is now considered one of the most dominant CI/CD services, since its November 2019 public release (Decan et al., 2022).

Another CI/CD tool that we have considered was Jenkins. However, since Jenkins is self hosted, setting it up and configuring it can be challenging. In addition, compared to GitHub Actions, Jenkins has a steeper learning curve and a less intuitive interface.

3.2 Project Requirements

For a project to have its pipeline runs' logs analysed by Bugsby, it first needs to meet a series of requirements. Firstly, it has to be a Gradle (Gradle Inc., 2023) project and have its source code hosted on GitHub, either as a public or a private repository.

Secondly, in order to be able to connect to GitHub Actions, when creating a new project, Bugsby users will need to provide the GitHub repository name, repository owner and a token generated by GitHub used for authentication. In addition, as required by GitHub Actions, the repository needs to contain in the `.github/workflows` folder one or more YAML files, which should describe the process for enabling CI/CD in the respective application.

3.3 Implementation

Bugsby's workflow, detailed in the following, is presented in Fig. 1, while its architecture, composed of three main units, the *Data Layer component*, the *AI (Artificial Intelligence) component* and the *Web component*, is illustrated in Fig. 2. The *WorkflowRunJob* is a newly implemented module responsible for communicating directly with GitHub Actions, while the *Gradle Log Parser* parses and analyses a `.log` file in order to construct a prefilled issue. Modules such as *Controller* or *Service* were extended in order to facilitate the proposed workflow, for example, by creating new HTTP endpoints for the *Data Layer* and *AI components*.

3.3.1 Log Retrieval

Due to the fact that the GitHub API does not notify third parties about various events such as failed pipeline runs, the *Data Layer component* constantly queries for them at a configurable fixed interval (by default, every three minutes) via the *WorkflowRunJob*. For each failed pipeline run, it will also retrieve from the API the corresponding `.log` file, which will be sent to the *AI component* for analysis.

3.3.2 Log Parsing

To be able to analyse a `.log` file, it first needs to be parsed. Given that the logs for pipeline runs are unstructured data, being a combination of free text written by developers and predefined messages, specific to the technologies used in the project, log parsing techniques capable of evolving with the system were needed. Considering this, for log parsing we have decided to use *Drain* (He et al., 2017), which is one of the best performing log parsers, having high efficiency and accuracy (He et al., 2016) (Zhu et al., 2019).

As input, Drain requires a log file and optionally, a log format and generates as output a structured log file and an event template file. Taking into consideration that GitHub Actions logs generally follow the format of `<Date> <Content>`, we are providing Drain with that. Using data extracted from the resulting structured log file, the *AI component* will prefill an issue report.

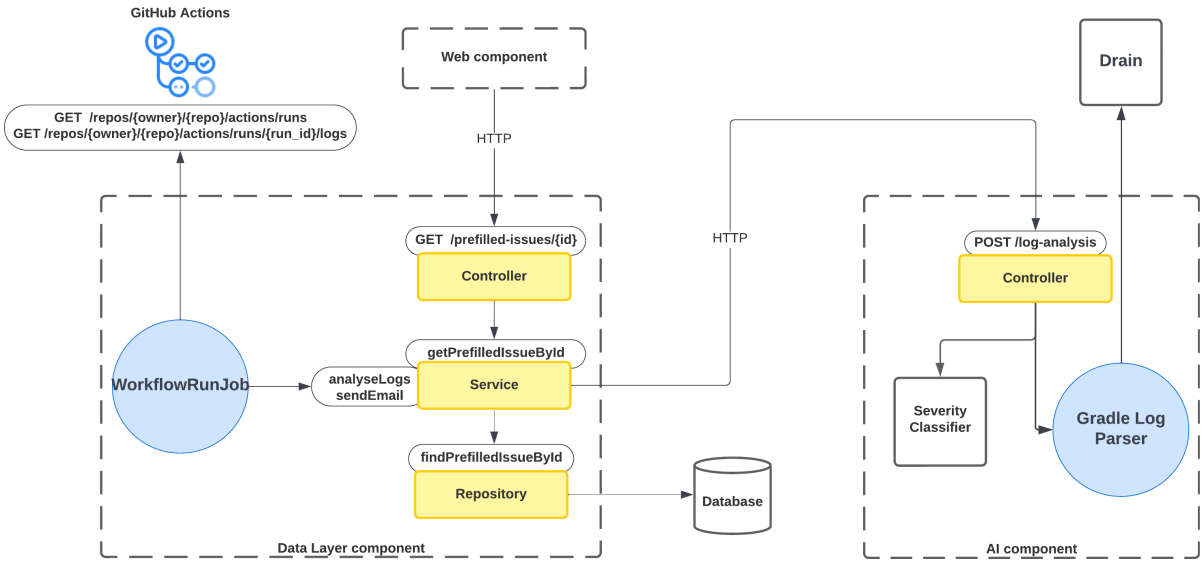

Figure 2: Bugsby's architecture, with newly added modules encircled in blue and in yellow, existing ones that were extended.

3.3.3 Constructing the Prefilled Issue

In Bugsby, an issue report contains the following fields: title, description, severity (a value between *TRIVIAL*, *MINOR*, *MAJOR*, *CRITICAL* and *BLOCKER*) and optionally, a description of the expected behaviour, of the actual behaviour and a stack trace. For the title of the prefilled issue, Bugsby will use the message of the commit that triggered the pipeline run. For the description and the expected behaviour field, our tool will extract the name of the failed Gradle task and construct a message containing details regarding the task's status. The severity is computed based on the title of the issue, using the severity classifier developed at (Picus and Serban, 2022). The classifier, developed using a Naive Bayes model, is implemented within the *AI component* and computes the most probable severity level of an issue, a value in the set of $\{severe, non-severe\}$. The actual behaviour field carries the most information regarding the pipeline run's failure; its value contains the name of the failed Gradle task, alongside the failure reason as logged by Gradle. Moreover, using

```
(?m)\^.*?Exception.*(?:\\n+\^.*at .*)+
```

as a regular expression, the *AI component* will try to find a Java Exception stack trace to populate the stack trace field.

3.3.4 Email Prefilled Issue

The *AI component* will return the prefilled issue to the *Data Layer component*, which will then email it to all project participants. The email contains the tool's and GitHub Actions' logos, the commit message that triggered the pipeline run, a link to the pipeline run on GitHub and a link to the prefilled issue, served by the *Web component*. Fig. 3 showcases the contents of the email sent by the tool, while an example of how the prefilled issue is displayed is illustrated in Fig. 4.

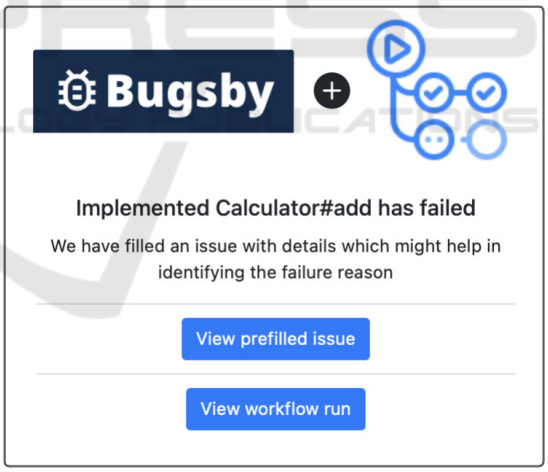

Figure 3: Example of an email sent by Bugsby after a pipeline failure.

3.3.5 View, Modify, Save Prefilled Issue

Developers are able to view the prefilled issue, modify it, add additional details and save it only if they would like to. This flexibility ensures that the issue report only contains information that is valuable to the project participants and that flooding of issues in the system is avoided.

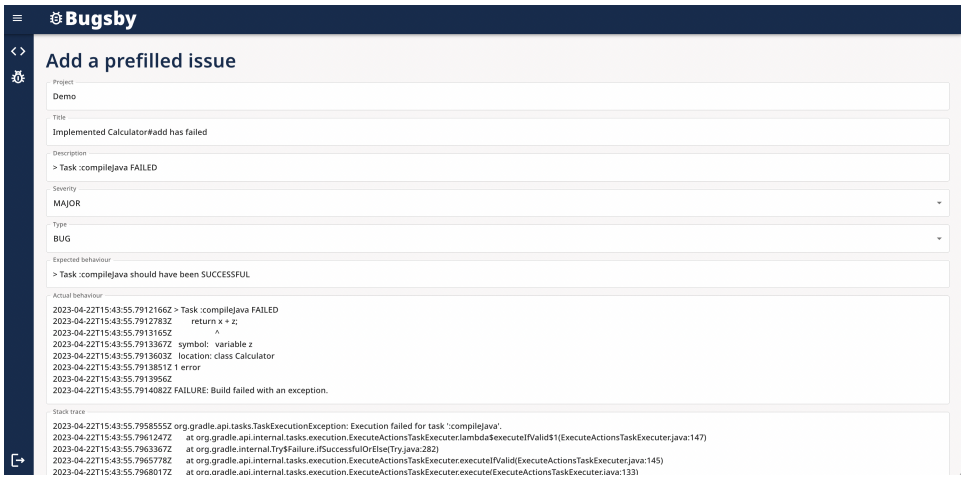

Figure 4: Example of a prefilled issue.

3.3.6 Statistics

In addition, for better visibility and to help project stakeholders make informed decisions that may affect the project's course, Bugsby offers a series of charts containing data extracted from the prefilled issues. One of them displays a statistic concerning the motives behind each pipeline failure i.e. the Gradle task that has failed.

3.4 Envisioned Users

Given the technical details that are present in the generated issue reports, we believe that software developers are the envisioned users for our tool; by automatically analysing the logs for pipeline failures, our tool reduces the time spent debugging and eases the software maintenance process. In addition, we also consider that our implementation is important to other project stakeholders and software engineering practitioners, as it offers visibility into the problems that a software project recurrently faces. Through its generated charts, Bugsby can be used to extract valuable information about a project, such as the types of challenges developers face while working on it.

3.5 Limitations

Two main limitations have been identified in our implementation. Firstly, due to GitHub's API limitations, Bugsby is not notified directly of new failed pipeline runs; as such, it needs to constantly call the API for new runs to analyse, a practice not that efficient. Since GitHub's code is not open-source, we are not able to implement this notification feature and therefore, we cannot overcome this limitation.

Secondly, our log analysis performs best on Gradle projects. As a consequence, we believe that we are reducing the tool's applicability and limiting its opportunities of analysing logs from different technologies. To overcome this limitation, we would need to implement new modules responsible for parsing and analysing logs for each technology we plan to support.

3.6 Running the Tool Locally

The source code of the tool is available at https://github.com/bugsby-project and it is split into three different repositories, which reflects its architecture from Fig. 2:

- **bugsby**, corresponding to the *Data Layer component*, written in Java and using Gradle,
- **bugsby_ai**, representing the *AI component*, written in Python,
- **bugsby_web_ts**, which contains the source code for the *Web component*, written in React and TypeScript.

Each repository contains a `Dockerfile` (Docker Inc., 2023b), increasing the tool's portability by ensuring that each component runs consistently across different environments. Since the application consists of multiple Docker containers, Docker Compose (Docker Inc., 2023a) was used for defining and configuring the services and networks needed by Bugsby. These definitions are present in the `docker-compose.yml` file, which is included in the **bugsby** repository. By using Docker Compose, one can start the application locally just by using the command `docker compose up`.

In addition, for ensuring a better code quality, each repository is using GitHub Actions for its CI/CD

pipelines, which define how each repository should be built and tested.

4 TOOL EVALUATION

For a preliminary evaluation of our tool, we have decided to conduct a usability test. In the following, we are presenting the elements and the results of our usability test, with Fig. 5 illustrating the evaluation process.

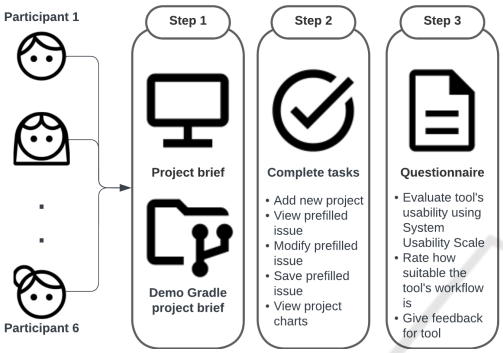

Figure 5: Evaluation setting.

4.1 Scope

The application under test is Bugsby, specifically its new component which integrates with GitHub Actions. We have decided not to test the components that were already developed, since they have been evaluated in our previous work.

4.2 Purpose

We have identified two goals that should be met by our usability test. Firstly, we wanted to understand if the proposed workflow is suitable for software developers, since they are the main envisioned users for our tool. Secondly, we will analyse how the users navigate the different menus of the application and if they generally find the tool usable.

4.3 Location

The usability test was held throughout the month of August 2023. Meetings with the participants were held in an online format; the participants had to share their screen and freely use the application to complete the given tasks, thus having complete control over their interaction with the tool.

4.4 Sessions

Each session began by presenting to the participants the context of the tool and its intended use. Each participant was provided with a Gradle project to work on, containing only a Java class with a single method, which was then added as a repository in their GitHub account. Afterwards, the participants were asked to complete the given tasks, with no interaction between the moderator and the participant. At the end of the session, the participants were asked to complete a questionnaire which contains all of the questions from the System Usability Scale (SUS) (Brooke, 1996), alongside other open questions intended to meet our goals detailed in Section 4.2. We have decided to use SUS because it is easy to use by study participants and it produces a score which can be compared to other tools.

4.5 Participants

Six participants were invited to our usability study. At the time of the study, each participant was aged 23 to 26, had at least one year of experience within the software industry and reported that the main technologies that they use include Java and Gradle, making the participants a subset of our envisioned users. Two of them were junior software developers, three of them, mid software developers and one of them was a senior. This diversity allowed us to receive feedback from people with a wide range of seniorities.

4.6 Tasks

The following are the tasks study participants were asked to complete in this order:

- add a new project, providing its GitHub credentials,
- view the prefilled issue generated after the participant made an error in the provided project,
- modify the prefilled issue,
- save the prefilled issue,
- view project charts.

These five tasks were chosen in such a way that when combined, they represent a real-life use case for our tool.

4.7 Metrics

When evaluating our tool, we have decided to compute the success rate and the SUS score, as defined in (Brooke, 1996). Given that we are measuring only

Table 1: Success rate for each task.

	Add a new project, providing its GitHub credentials	View the prefilled issue generated after making an error	Modify the prefilled issue	Save the prefilled issue	View project charts
Participant 1	Failure	Success	Success	Success	Success
Participant 2	Failure	Success	Success	Success	Success
Participant 3	Success	Success	Success	Success	Success
Participant 4	Success	Success	Success	Success	Success
Participant 5	Failure	Success	Success	Success	Success
Participant 6	Failure	Success	Success	Success	Success
Success rate	**33%**	**100%**	**100%**	**100%**	**100%**

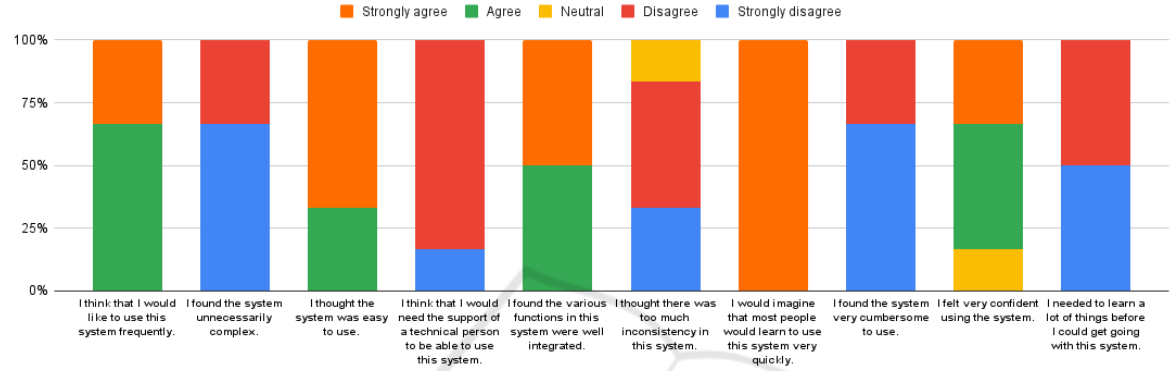

Figure 6: Responses to SUS questionnaire.

complete successes i.e. we disregard partial successes when completing tasks and we consider them as failures, we are computing the success rate as

$$Success\,rate = \frac{\#successfully\,completed\,tasks}{\#attempts}$$

, where # denotes the cardinality of the set.

4.8 Results

Table 1 presents the success rate for each individual task mentioned in Section 4.6. The results are promising, with each task having a success rate of 100%, with the exception of the first one, which has a success rate of 33%. Participants remarked that there were not enough instructions on how to add a GitHub project in Bugsby and therefore, we plan to address this as future work.

Regarding our first goal, all participants have agreed that the proposed workflow integrates well with a developer's daily activities and that it has the potential of bringing improvements in areas such as pipeline failure detection and resolution.

While completing the second task, the participants were asked to introduce an error of their choice in the source code of the project. Three of them chose to introduce a compilation error, two of them wrote unit tests that intentionally failed and one of them specified a dependency for the project that does not exist. In the given questionnaire, the participants admitted that the resulting prefilled issue had the potential of helping them resolve the error that they have introduced.

For achieving our second goal, we have computed the SUS score for our tool based on the participants' answers to the questionnaire they had to complete at the end of their session. Their responses were collected into Fig. 6. With a SUS score of 87.08, Bugsby receives a grade A+, based on the Sauro-Lewis curved grading scale (Lewis and Sauro, 2018), indicating that the tool is highly usable. Concerning our tool's navigation, participants were able to successfully switch between its different screens.

Overall, the evaluation results show that our tool can potentially be useful and adopted in real projects.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we have designed and implemented a new component integrated with our previously developed tool, Bugsby, aimed at bringing more visibility into the challenges software projects face and assisting software developers with investigating pipeline failures. Our proof-of-concept implementation is, to the best of our knowledge, the first one to propose an integration between ITSs and CI/CD tools focused

on automatically discovering pipeline failure reasons from the resulting logs.

Future work would be intended at making our tool a more comprehensive one, by integrating it with more CI/CD tools such as Jenkins. In addition, we plan to address the limitations described in Section 3.5 which we believe are hindering Bugsby's potential of becoming a widely used tool within the industry. Furthermore, we intend to facilitate the process of adding a new project, since we have identified issues to it during the usability test that we have conducted and extend our evaluation, by assessing the utility of our tool based on the time it saves developers into resolving a task. As a mean of comparison, this should involve setting a controlled group consisting of developers that do not use the tool and resolve their tasks as before.

REFERENCES

Brooke, J. (1996). Sus: a "quick and dirty'usability. *Usability evaluation in industry*, 189(3):189–194.

Decan, A., Mens, T., Mazrae, P. R., and Golzadeh, M. (2022). On the use of github actions in software development repositories. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 235–245. IEEE.

Docker Inc. (2023a). Docker Compose overview — Docker Docs. https://docs.docker.com/compose/. Last accessed in December 2023.

Docker Inc. (2023b). Docker Docs. https://docs.docker.com/. Last accessed in December 2023.

GitHub (2023). Features GitHub Actions GitHub. https://github.com/features/actions. Last accessed in April 2023.

Gradle Inc. (2023). Gradle Build Tool. https://gradle.org/.

He, P., Zhu, J., He, S., Li, J., and Lyu, M. R. (2016). An evaluation study on log parsing and its use in log mining. In *2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 654–661. IEEE.

He, P., Zhu, J., Zheng, Z., and Lyu, M. R. (2017). Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*, pages 33–40. IEEE.

Jenkins (2022). Build Failure Analyzer. https://plugins.jenkins.io/build-failure-analyzer/.

Jenkins (2023a). Jenkins. https://www.jenkins.io/. Last accessed in April 2023.

Jenkins (2023b). Jira — Jenkins plugin. https://plugins.jenkins.io/jira/. Last accessed in April 2023.

Lewis, J. R. and Sauro, J. (2018). Item benchmarks for the system usability scale. *Journal of Usability Studies*, 13(3).

Nogueira, A. F., Ribeiro, J. C., Zenha-Rela, M. A., and Craske, A. (2018). Improving la redoute's ci/cd pipeline and devops processes by applying machine learning techniques. In *2018 11th international conference on the quality of information and communications technology (QUATIC)*, pages 282–286. IEEE.

Ortu, M., Murgia, A., Destefanis, G., Tourani, P., Tonelli, R., Marchesi, M., and Adams, B. (2016). The emotional side of software developers in jira. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 480–483. IEEE.

Park, S.-H. and Choi, J.-H. (2022). A study on the implementation of issue tracking system based on devops. *Journal of the Korea Society of Computer and Information*, 27(1):91–96.

Picus, O. and Serban, C. (2022). Bugsby: a tool support for bug triage automation. In *Proceedings of the 2nd ACM International Workshop on AI and Software Testing/Analysis*, pages 17–20.

Singh, C., Gaba, N. S., Kaur, M., and Kaur, B. (2019). Comparison of different ci/cd tools integrated with cloud platform. In *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 7–12. IEEE.

Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., and Lyu, M. R. (2019). Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 121–130. IEEE.