International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 06 | June 2020 www.irjet.net p-ISSN: 2395-0072

Designing an Optimized and Reliable Algorithm for the Automation of Problem Report State Flow

Prathamesh Sanjay Zingade[1], Saba Fareen N S[2]

[1]*Department of Electronics and Communication Engineering, RV College of Engineering Banglore, Karnataka, India*
[2]*Assistant Professor, Department of Electronics and Communication Engineering, RV College of Engineering Banglore, Karnataka, India*

Abstract - *Software designing, and maintenance is one of the major concerns in IT companies. Problem Report (PR) is the process being followed to track data and the problems associated with it while the product is being released. PR gets attached to the failed testcases of the scripts being written for testing. Once PR get attached it needs to undergo various states until the problem is being fixed. Thus Automating the problem report state flow helps the organisation to speed up the process and thereby achieving better efficiency. This work reviews the approaches being used for automating the PR state flow which in turn acts as scheduler.*

Key Words: **Problem Report (PR), Automation tool, python, Application Program Interface (API), CRON.**

1. INTRODUCTION

This Finding the issues and fixing them is the most expensive activity in software development [1]. Considering all the size, time, cost, complexity, and pressures - predicting and tracking quality becomes a major challenge in software development projects. Products like Routers and switches need logical software performance which would be developed by software developers. Before proceeding to the deployment of these software logics into hardware, testing the scripts plays an important role. Testing and debugging software is an important part of software companies used for make sure the correctness of reliability and functionality of software systems; but software testing is also a resource intensive activity which accounts more than half of total software development costs [2] and even more for ensuring the safety of critical software systems. It is essential to keep track of scripts being tested.

Various techniques are being used to track the debug history. One of the technique is using Git. Git is one of the distributed version system which helps to track the logs, changes in data and the defects which can be extracted easily [3]. As thousands of scripts will be tested it becomes difficult to handle the logs using open source software. Thus debugs and the problems associated with the scripts are handled using one of the tracking system developed in the organisation called as PR.

Organizing problem reports being discovered would provide tracking details and the history of all the problems and thus it is being treated as one of the most efficient way of dealing with software issues. It becomes essential to keep track of logs related to the exchange of information between testers and developers along with constant monitoring of issues until the resolution is being fixed and validated.

Most organizations maintain a database which could be local to the project or team or specific section of organization. All details regarding defects found when testing fails, till the failed script passes such activities are recorded as reports in those databases in a predefined format. The database has the facilities to keep track of the problem reports. It provides major options such as:

- Allows to Access the details of the problems or issues being attached to the failed scripts which has to be resolved
- Helps to Create, query, edit and check thousands of PR being available on the database
- Helps in Fetching the details of logs, owners of PR, scopes etc.
- Allows to Track the state of PR.

To provide all these options and facilities the database being maintained, the state of PR can be monitored by fetching the data from it. Scripts written by test engineers needs to be tested in some kind of automation tool which provides user friendly interface to register, run and validate the test cases.

The scripts running automation tool used is the next-gen test execution harness designed to cater to all workflow. It provides user interface to submit and run the scripts. It comprises an advanced registry for organizing scripts, a flexible resource set management system and a powerful scheduler that manages the queueing and execution of scripts on the various resource sets. It supports multiple automation frameworks and offers standard services such as scheduling, monitoring, result viewing, result analysis, troubleshooting of automation jobs, and provides the grid view of script profile search to show the pass percentage of the script profile.

The scripts running tool is used to submit the scripts written by various test engineers and check if it runs or not. The automation comes into consideration when the scripts

fail. And tracking begins when PR gets attached to the failed scripts.

2. PROBLEM REPORT

A Problem report is a database recorded in the PR system, represented by a number, and can correspond to: a bug, a feature request, tracking data for releases, a problem condition-such as a build break. One such example of a PR id is: PR: 1234, PR: 8976-2 (PR id with scope=2.)

PR captures data related to: who found the condition (submitter), the issue (what is the bug, enhancement request or condition being tracked), the release in which this condition needs to be fixed (scopes) and who shall be responsible for fixing this problem/condition (developer.) Once the developer gets assigned a particular PR the information about the fix for the condition (how, when, where it would get fixed) and any other communication or relevant data related to the PR (including any workarounds) would also get captured.

Scope - A single problem can exist in multiple releases, and the problem needs to be fixed in the releases being affected. A scope is a set of fields in a PR that correspond to data that might be unique to a particular release of a product. A PR can have multiple scopes, one for each release affected by the PR. Each scope would have its own State field; hence PRs would become multi-state when scopes were being introduced in the PR system. A PR scope is represented as PR-scope, for example: 12345-3.

The PR gets created by various teams depending upon the issues. These PR gets attached to the failed testcases which indicates that Testcase is failed and hence the code needs to be fixed. This tracking of code fix would be carried out using the PR state flow. Once the PR is attached it would undergo various states until the code gets fixed and validated. Once the testcases which was failed gets passed the PR can be closed, which indicates that the problem is being fixed and state of PR can be changed to closed state.

3. PROBLEM REPORT STATES

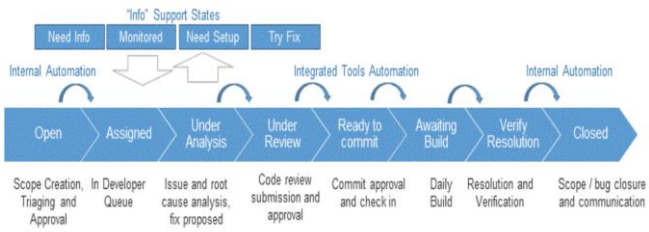

Fig-1: PR states

As shown in Fig 1 PR undergoes into different states. The details of each state and respective roles being responsible to handle the PR is given in Fig 2.

The Major state to be considered for automation is "Verify resolution state", whenever the state of PR changes to verify resolution state it indicates that the problem is being fixed and can be validated. This is where automation comes into play where automatically all the scripts image details have to be fetched and the it has to be re-run and checked whether the problem is fixed or not.

STATE	DESCRIPTION	RESPONSIBLE ROLE
Open	• When created, PR scopes are initially in this state • The issue in the PR is being triaged, but usually has not yet been assigned to a particular developer to analyse and address the issue.	Manager
Assigned	• The PR/scope has been assigned to a developer to analyse the issue and try to find a solution.	Developer
Under Analysis	• The PR/scope is being actively analysed by the developer (or writer) assigned to the PR.	Developer
Under-Review	• The assigned developer analysed the issue, determined a code change is required to fix it in the branch or version associated with the scope.	Reviewer
ready-to-commit	• Code reviews for the fix have been approved and a code fix is ready to be committed to the branch associated with the scope.	Developer
awaiting-build	• A code fix has been committed for the scope, and is awaiting the daily or other build in which the fix will be available to test.	Developer
verify-resolution	• Problems are resolved and the scope's resolution is ready to be validated.	Submitter
Info support States	• The responsible person working on the PR scope in the current state needs more information from another PR stakeholder in order to continue with their work.	Various

Fig-2: PR states and responsible role

The verify resolution state indicates that: In most of the cases, scopes must undergo a transition through this state when it needs to get resolved and ready to be validated. This applies whether or not the issue is being fixed, as described below. Resolution, validation, or acceptance is usually expected to be done by the submitter of the issue. A resolution must be recorded in the scope (in Resolution and Resolution-Reason fields) before or at the time when the scope enters into this state. If the issue was resolved with the status being recorded as fixed, the submitter of the PR/scope needs to perform activities which are required to re-verify that the issue is indeed fixed in the Planned-Release for the scope.

Otherwise, for a non-fixed resolution, the submitter confirms whether they accept or reject the resolution. It also includes that when the PR is resolved as a duplicate or mistaken PR, the issue would not be reproducible, or it would be decided that the issue will not be fixed in that release for a particular business reason.

In current work system, If the problem is fixed, the state of PR is manually set to 'resolution-accepted', and the system automatically makes a transition of its State to 'closed' which indicates that the problem is fixed. Alternatively, if problem is not fixed and resolution is not accepted then state is set to 'resolution-not- acceptable' to disagree with the resolution which indicates that the issue is not fixed and PR has to be opened and assigned back to the developer to make changes and provide new solution.

4. SYSTEM DEVELOPMENT

In most of the organizations, Problems or bugs related to software are fixed and the fix is tested continuously on daily basis. Fixing the problems requires attention of the developer. Once the problem is fixed by the developer human resource is required to test the solution and provide the feedback hence, to reduce the human interaction and provide an automation various designs were considered. As mentioned in [4] an automotive domain score card can be implemented to be used in small organization. As explained in [3] open source Git can be used to track and update the details which less confidential. As PR database and scripts running tool interactions are of major concern thus the final design would consist of an automation tool which would be developed using python as a scripting language.

5. METHODOLOGY

As shown in Fig 3 Python script would be developed in each step having a particular definition. The details of the steps being followed are as stated below:

1. Getting PR list: As discussed PR list can be fetched from different ways depending upon the requirement. Final design is made which will fetch the PR list from the given build. The steps being followed are detailed below-

- Branch would be considered as a starting point of the code. PR would be fixed on daily basis which are then stored in the daily folders with date on which it was addressed (example: 20201115 in the format of YYYYMMDD), these date folders are called as Target builds. All the PR fixed in a day would be stored on the file PRLIST.txt in their respective target build folders.
- The file would have software and Linux PR amongst which only software PR is considered. These are software PR. This file is read using 'cat' command.
- Once PR the is fetched it would be considered for the further processing

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper.

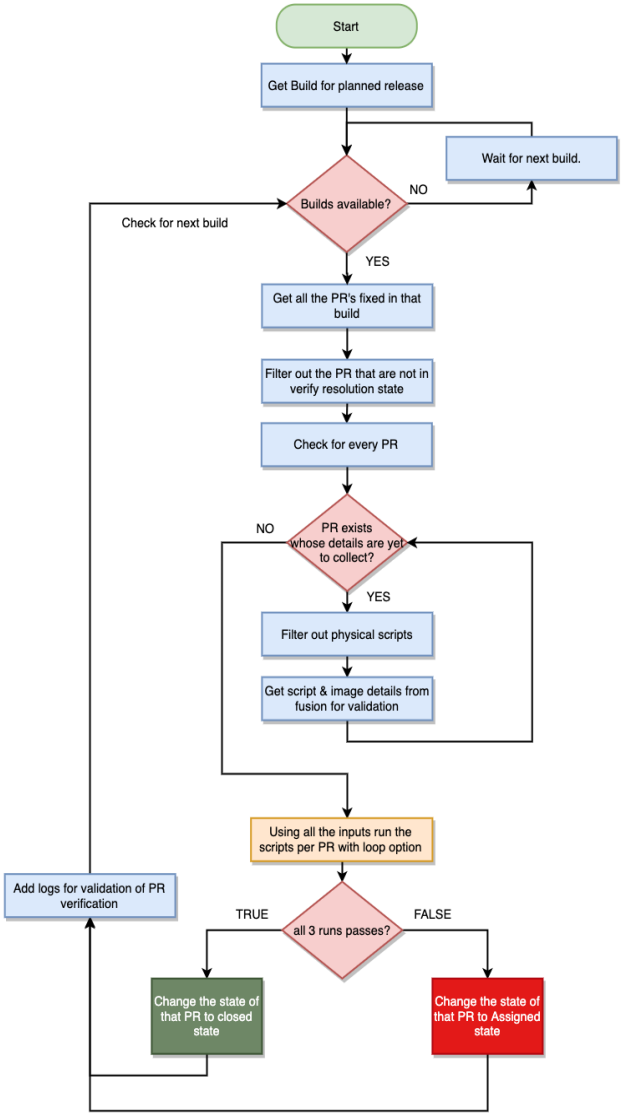

Fig-3: Tasks flowchart

2. Filtering PR list: Once the PR is fetched and software PR is considered, there are few conditions applied before giving the final PR list which is considered for the final process. This is done as follows-

- PR associated in different fields, among those PR, details of PR which are associated with the 'scripts running tool' is fetched. This is done by calling the query problem report API. This will give filtered list of PR which are associated with it
- Once 'scripts running tool' associated PR is filtered, the state of these PR is checked. PR can be validated only when the issue is fixed hence state of PR has to be 'Verify resolution'. Filtration would be carried out again if the PR are not in the 'Verify resolution state'.
- Details of Planned release and Target is checked which has to be matched with the build from which PR list is taken as explained in step 1.

- This will be the final list of PR which would be used for the next process.

3. Collecting inputs: This is the part of the tool where all the data related to respective PR is fetched which in turn is used to submit to run and then to validate the testcases. To fetch the details of the script, script exec id of that PR is fetched using query PR API. Script exec summary API will give all the details for given script exec id.

Scripts can be virtual based or physical based, as part of the testing only virtual based PR is considered, hence details are collected only if the script is virtual. This data is also fetched from the same get summary API. If it is virtual then inputs like- script profile, image, failed test case etc. are collected.

4. Getting image: Image serves as a major part while submitting run job. This has to be accurate. Following steps are followed to get the actual image of scripts

- Firstly all the images are checked from the ship folder of that build
- Among those only vmdk and iso (indicates virtual images) images are considered and rest is filtered out because only virtual scripts are considered.
- Using get script exec summary API, the previous image associated with the script is taken and stored.
- The string is compared to check if that substring image is present in ship folder images.
- If it exists then that image itself would be considered for the process.
- Dictionary is updated which would have script exec id: PR, Profile, Image, failed testcases.

5. Submitting scripts run job per PR: Dictionary items are flipped such that details is per PR having PR as key. It would have all the details such as scripts, testcase failed, images associated with that PR. submission is done in the tool to run the scripts per every PR i.e. For one PR one submission is done hence number of submissions would be equal to the number of PR. Loop option is enabled so that scripts would run and validation would be carried out 2 times.
6. Checking submission state: State of submission is polled every 5 minutes and checked if the submission is completed or not. This is done to ensure that the further process begins only after completion of run.
7. Validating failed testcases: All the scripts would run with some testcases. Hence validation depends on the test case, thus it is essential to check those test cases that are getting passed in the run. This is done as follows:

- Test case result is compared and checked if it Passes or Fails. Once the scripts run, each testcase is compared with the stored testcase.
- If the name matches then the result is checked. If it gets passes through all the loop runs which in turn indicates that the issue is fixed and PR can be closed. Else it indicates that PR has an issue and needs to be assigned back to the developer.

6. IMPLEMENTATION

Methods and functions are written and called in the main function as considering the logic explained in methodology. Each function call is implemented with an idea right from getting PR list till the submission of scripts and then the failed test cases are validated, informing the owner.

This helps the engineers to automatically check the result of submission instead of wasting time to submit all the scripts and waiting for the result. Each function call takes the various inputs depending upon the requirement. The logging module is also implemented to keep track of the run. Most of the functions fetch the data from various API and commands which would run and give the output. The python script can be run using CRON, as this job needs to be run every 24 hours. The final implementation is done till the owner of the PR gets informed about the submission details and result.

7. RESULTS

Fig-4: Notification result

Logger output: The data being tracked, and the code gets logged into the logger file. The logger file consists of step by step details of results being obtained. As the implementation is done in a way wherein it has to fetch many details and take many decisions based on the data, it is important to log all the data. The sequence of steps being followed are as given below.

- PR list is taken from the build
- PR associated with submission tool is checked and others are filtered out
- All virtual images are collected as logged
- PR state is checked, and filtration would be done
- Script exec details are fetched, and image details are updated
- Scripts are submitted to run per PR
- Failed testcases are validated and notification is sent

Final mail is sent to the owner of PR saying that PR is validated, testcases are passed and state of PR can be changed to closed state. The sample format is shown in the Fig 4 The mail consists of the details regarding owner of PR, PR number, submission id and the result. A link to the submission would be provided to the owner in order to

validate if the testing is done or not, depending on which the state of PR can be changed to closed state.

8. CONCLUSION

The major concern with IT industries was to track the problems, issues generated while running the scripts, as the PR gets attached in order to maintain the track. Whenever the problem gets solved by the developers the test engineers have to test those scripts again. To provide automation in this field and to reduce the work of engineers a detailed review of designs being followed so far was carried out and based these ideas a novel approach is suggested for the development of automation of PR state flow.

Python based tool was developed to fetch the details and to have the decisions depending upon which submission was done. Testcases are validated automatically and owner is informed about the result. The details of implementation and the results obtained are also reviewed in this paper.

Such automations are deployed in the tool which will invoke the Cron every day and automatically does the job of validation, this in turn helps the engineers working in almost all the teams not worry about the scripts which they have fixed as it is now handled by the tool "Automation of Problem Report state flow".

REFERENCES

[1] D. A. Gromova et al. "Raising the Quality of Bug Reports by Predicting Software Defect Indicators". In: 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C). 2019, pp. 198–204.

[2] A. Hazeyama and M. Hanawa. "A problem report management system for software maintenance". In: IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028). Vol. 1. 1999, 1019–1024 vol.1.

[3] P. S. Kochhar et al. "Understanding the Role of Reporting in Work Item Tracking Systems for Software Development: An Industrial Case Study". In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2018, pp. 605–614.

[4] R. Malhotra and L. Bahl. "A defect tracking tool for open source software". In: 2017 2nd International Conference for Convergence in Technology (I2CT). 2017, pp. 901–905.

[5] R. Malhotra et al. "Defect Collection and Reporting System for Git based Open Source Software". In: 2014 International Conference on Data Mining and Intelligent Computing (ICDMIC). 2014, pp. 1–7.

[6] T. Merten et al. "Software Feature Request Detection in Issue Tracking Systems". In: 2016 IEEE 24th International Requirements Engineering Conference (RE). 2016, pp. 166–175.

[7] R. Rana, T. Lagercrantz, and M. Staron. "Building an Effective Software Issues Scorecard: An Action Research Report from the Automotive Domain". In: 2018 IEEE International Conference on Software Architecture Companion (ICSA-C). 2018, pp. 136–143.

[8] Huaiqing Wang and Chen Wang. "Open Source software adoption: A status report". In: Software, IEEE 18 (Apr. 2001), pp. 90–95. doi: 10.1109/52.9147