OULUN AMMATTIKORKEAKOULU

Heidi Jussila

FAULT REPORT ID REPORTING AUTOMATION

FAULT REPORT ID REPORTING AUTOMATION

Heidi Jussila
Bachelor's Thesis
Spring 2023
Degree programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

---

Author(s): Heidi Jussila
Title of the thesis: Fault Report ID reporting automation
Thesis examiner(s): Andras Nemeth (Nokia), Lasse Haverinen (OUAS)
Term and year of thesis completion: Spring 2023 Pages: 33

The aim of this thesis was to find solution for automating fault report IDs reporting to test management tool used in Nokia. When reporting test results of failed test cases in testing, fault report IDs are needed to be reported. Fault reports are made when fault is found in testing. Each fault report gets unique ID in creation. Initially the fault report IDs were manually updated to the test management tool after test results were sent. Multiple tests run continuously generates lots of test results and manual update of fault report IDs gets challenging and takes lots of valuable time. Information of current fault reports was recorded locally by test line owners. Test line owners are responsible of test runs and updating the fault reports. In this thesis the goals were to get the fault report IDs reported automatically to the test management tool, save time from the manual update work, and create better way of recording current fault reports.

As a result, the fault report ID automation was implemented to Python programming language written test results reporting tool that was already used for test results reporting in the team. Fault report IDs were recorded in JSON, JavaScript object syntax, file. That file was decided to be stored in version control in the test results reporting tool repository. All the new information needed for the fault report ID test management tool reporting was stored in this file. Test line owners are responsible of updating all the fault reports in that file.

The developed solution is in use in the team, and it has been beneficial. Currently fault reports are reported automatically in time to the test management tool with test results. Test line owners have now one location where all current fault reports are listed and updating one file instead of manual update to the test management tool takes less time. When fault reports are listed in one file, everyone in the team has access to it.

In the future, planning would be to the direction where the file containing the fault reports would be in different location instead of in version control. Also, possibility for automatic updates for the fault report storing could be planned.

Keywords: Testing, Software Development, Python, GitLab, CI/CD

CONTENTS

1 INTRODUCTION

This thesis was done for Nokia. Nokia is a global company working in technology industry. Nokia has headquartered in Espoo Finland. Nokia Finland has around 6,500 employees. Nokia was founded in 1865 but the company started working in the field of telecommunications in 1980's. Nokia's current slogan is "We create technology that helps the work act together". Nokia became known worldwide for their expertise in mobile industry. Nowadays Nokia is one of the largest companies providing network equipment and solutions. [1]

The aim for this thesis was to research and develop a new feature for test results reporting. CI – team tests continuously radio software and test results are reported into different Nokia's internal test result management tools. When fault is found in testing, it needs inspection from test line owners and if found needed, fault report is created. In this thesis, test results reporting was to be improved to cover fault report IDs in fail case results reporting.

2 TARGET / OBJECTIVE

Target of this thesis was to develop a solution to automate fault report IDs reporting to test management tool, when reporting fail case test results.
When new fault is discovered in continuous testing, fault report is created. Every fault report has unique ID in it. Fail testcase results needs to be reported with fault report ID. With this information, objective was to end up with a solution to keep record with current fault reports and update results based on test result status and available fault reports.

Before this thesis work, all this work needed to be done manually. Each fault report ID had to be reported manually to the test management tool after each fail case test run. In wide CI environment plenty of test results are delivered daily by continuous testing. When the target of testing is to find bugs and errors from software builds, test results come out also with failure status and when multiple tests are running simultaneously and continuously, there is large amount of fail case test results. Manual reporting of fault report IDs takes valuable time from the test line owners. When new fault report is created, its ID must be updated to its corresponding test instance in the test management tool. When fault report is closed its ID must be removed from the test instance in the test management tool. One fault report can be related to more than one test instance. If no record is kept where the ID has been reported, it might be challenging to find all the test instances where the ID has been reported. Before new solution for fault report IDs reporting, all the information of current fault reports was held by test line owners locally. With the new way of fault report IDs reporting, better solution was wanted for this information to be available for other team members too.

3 THEORY

3.1 CI/CD

CI/CD is abbreviation for Continuous Integration and Continuous Delivery. These both are processes in test automation. CI/CD process automates steps needed in software development from new code commit to the phase to get the code for production. That includes build, testing and deployment. Precise CI/CD process is crucial part of modern software development. It will reduce the time needed for the processes to be done manually, make it more reliable without human intervention and provide better environment for faster code commits with small changes to be done more often. This provides more features to be deployed more often when the lifecycle of the software development process is faster. [2], [3]

3.1.1 Continuous Integration

Continuous Integration is a process where developers commit or merge their code to main source code branch early and often. In Continuous Integration, possible errors and bugs in code can be caught early in the development process. When Continuous Testing is implemented with CI, new commit or merge trigger a testing process where the changes are tested. [2], [3]

3.1.2 Continuous Delivery

Continuous Delivery is automating the deployment process. This process automatically deploys all new changes to production. Continuous Delivery release process is automated, but it is triggered manually. [2], [3].

3.1.3 Continuous Testing

Continuous testing is part of CI/CD process. Continuous testing process is triggered when new code is committed. In CI/CD, this is done automatically. This provides information about bugs and

errors in early phase of the software development cycle. When the code commits are done often and contain only one new feature, identifying the problem takes less effort.
There are different types of testing in CI. Unit Tests are testing that single code units are working as expected. Integration tests are verifying how different modules and services operate in an application together. Regression tests are needed to verify bug is not happening again, when bug found in previous testing is fixed. [2], [3]

When implementing CI/CD process, it is possible to only take in use Continuous Integration without Continuous Delivery. Continuous Delivery is not implemented in the CI team to where this thesis work was done. [2], [3]

3.2 GitLab

GitLab is a DevOps platform. In GitLab, it is possible to provide ideas to production in one platform. GitLab is offering source code management and CI/CD solutions. In GitLab webpage CI/CD fundamentals are listed as follows, a single source repository, frequent check-ins to main branch, automated builds, frequent iterations, stable testing environment, maximum visibility, and predictable deployments anytime. A single source repository is providing all the information needed for build. Frequent check-ins to main branch are for enabling focus for one branch and frequent commits for one feature at a time. Automated builds and self-testing for build to allow only code that passes tests proceed into the build. Frequent iterations, so possible conflicts in the code can be more traceable from small, frequent iterations. Stable testing environment to test in environment close to the production version. Maximum visibility providing easy access for developers to see the changes made to the code. Predictable deployments anytime, as developers should be fearless to commit new code as routine to reliable CI/CD environment. [4], [2]

3.3 Jenkins

Jenkins is an open-source Continuous Integration and Continuous Delivery tool. It is used to automate software delivery process. Jenkins provides automation for building, testing, releasing, and delivering software. Jenkins was developed to improve software delivery process and to provide tool for automating the process. Jenkins offers multiple plugins to integrate with multiple Continuous Integration and Continuous Delivery processes. Jenkins provides web interface which makes it

easy to use. Jenkins server, the Jenkins controller is running on private network. Remote source code repository is connected to the Jenkins controller and every new commit to repository triggers Continuous Integration process. [5]

The Jenkins controller is the original node that Jenkins is originally installed in. The Jenkins controller node can split continuous integration work to remote agents, see Figure 1. Launching Jenkins agents is possible in physical machines, virtual machines, Kubernetes clusters, and with Docker images. [6], [5]
Other Jenkins like CI/CD tools are example Buddy, Bamboo and Circle CI. [7], [8], [9]

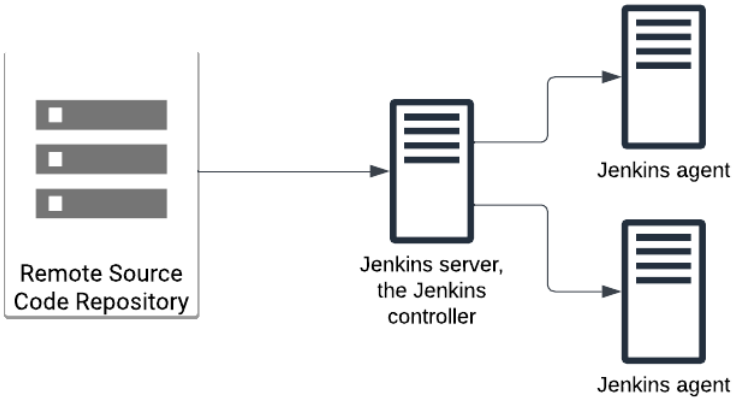

Figure 1 The Jenkins controller – agent architecture.

3.4 Test results reporting

The test results reporting in the team was automated by a tool written in Python. This tool is run after every automated test run in Jenkins and it sends the results to different Nokia's internal platforms containing information about the test and test results.

Every test generates output files in csv, xml, and JSON format. The test results reporting tool has importer modules for each file format. These modules parse each output file and parsed files are combined in main script to payload, a JSON type file, for test results reporting. All of payload or parts of payload will be sent to Nokia's internal use platforms that are used for collecting test results. For each result platform in the test results sending tool is made exporter module. Each exporter module is responsible of the results sending and communicating to one platform.

3.5 Fault reports

The purpose of continuous testing is to verify software and try to find bugs and errors. When test is run and it does not pass, test line owner who is responsible of the test, inspects the results. If it is discovered in inspection that the reason for test failure is from the tested product, new fault report needs to be created. Fault reports are created and managed in its own tool. The tool is made to keep track of the fault processing. The report contains information of the found fault and fault correction status. Each fault report gets its own unique ID. This ID needs to be reported with all failed test results that occur while testing the same product after the fault report is created. The fault report ID is not needed to be reported anymore, when the fault is corrected, and fault report is closed.

3.6 Python

Python is one of the most popular programming languages. It is general purpose language and is suitable for automation among other purposes for its versability. Python documentation, community and news can be found at Python webpage. [10], [11]

3.7 Pytest

Pytest is a test framework developed for Python testing. Pytest is easy to install via pip. Pip is a package installer that can be used to install Python JSpackages. Pytest is developed for testing Python applications and libraries. Pytest provides testing from small tests to complex functional testing. Pytest offers guide of how to get started with pytest and detailed examples of tests in pytest documentation page. [12], [13]

3.8 Jsonschema

Jsonschema is implementation for Python of JSON Schema. JSON Schema is a declarative language made for validating JSON documents. [14], [15]

4 DESIGN

Work started with research about the topics. Time for planning and familiarization was scheduled. Aim in this part of the thesis work was to get familiar with how reporting is implemented in automation in our team and to get to know the test management tool where the fault report IDs were to be updated.

4.1 Requirements specification

Task planning started with requirements specification after research about the topics. One specific requirement was defined for the tool when it was requested, fault report IDs had to be reported when fail test results are reported in test management tool. If no fault report IDs were available, the test cases with fail status need to be reported with status blocked and add comment of environmental problems with the results reporting (Figure 2). More requirements had to be resolved in our team point of view. For test line owners, who were going to use the tool, requirements were easy to use in a way that new person in a job can use the tool, and the tool should not deliver more work than what manual fault report ID update work took. Test line owners are responsible of keeping the CI test automation work going. This improvement in test results reporting was to make their work faster and upgrade the test results reporting in CI test automation. From developers' point of view, tool had to be maintainable and code readable. One requirement was that the tool needed to be implemented in test lines in this thesis timeframe. Final requirements introduced in Table 1.

Table 1. Requirements

Requirement	Description
Requirement 1	Fail test cases can only be reported with fail status if fault report ID is available.
Requirement 2	Fail test cases without fault report IDs available, must be reported with "Blocked" status and comment of "Environmental problems" added to the payload.

Requirement 3	Fault report IDs reporting in new solution must take less time compared to manual update to the test management tool.
Requirement 4	Automated solution for fault report IDs reporting is simple so that new hire in the team can use the solution.
Requirement 5	Maintainable and readable code.
Requirement 6	A working solution for fault report IDs reporting must be implemented in this thesis timeframe.

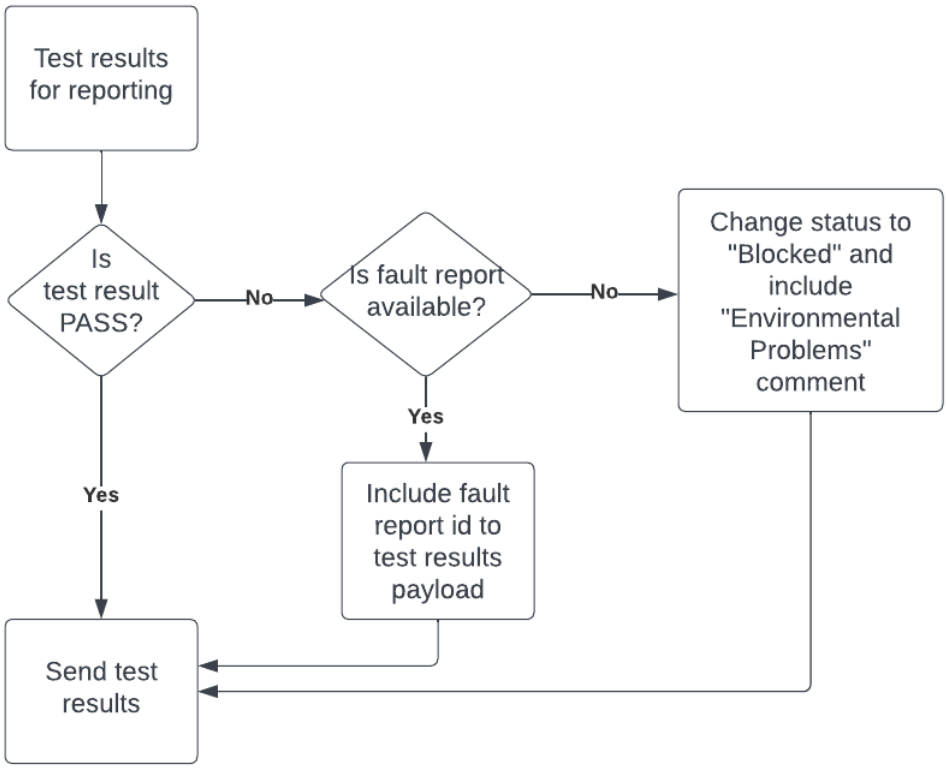

Figure 2 Test results reporting principle.

4.2 Planned options for implementation

When requirements were specified, implementation planning could start. Different solutions were planned and considered for implementation. All the new and affecting components needed to be considered in the planning process. The team was already using a Python tool made for test results

reporting and after possible solutions research, it became the best option for fault report IDs reporting to be as a new feature for that.

4.2.1 Manager tool option

In other option, instead of test results reporting tool, was to create separate tool to update test management tool results after test results were updated.
In this solution, after results would have been sent to the test management tool, the new tool written in Python could make a request to the test management tool and get results with given parameters. Results could be then exported to csv format and be edited easily locally with editor. Then the created tool could send the results back to the test management tool.

This solution would have had much more possibilities to be extended because all the information in the test management tool results could have been edited, not only the fault report IDs. It is also stated that the results with fail status without fault report IDs should not be sent to the test management tool. When results with failed status without fault report IDs are sent to test management tool, it will be noted by a tool following the correct results reporting and trigger a notification of an incorrect results reporting. This option needed much more work and time than the option for new feature for already existing test results reporting tool. Because the solution had to be implemented and deployed in test lines inside the team in this thesis work timeframe, this option was not considered to be planned further. It could not be implemented partly, because working solution could not be provided in a way that it would fill the requirements in the given timeframe.

4.3 Storing fault reports

One of the topics was to plan the component containing info about all the fault reports related to the test cases. The fault reports are created and managed in a specific tool. REST API resources are provided to the fault report tool, but the tool does not contain enough information needed for reporting. The most information about status of each fault report comes from the test line owners. It was clear from the start that the information about fault report IDs to be reported, had to come from the test line owners.

4.3.1 How to link fault reports and test results

The information needed to be able to report the fault report IDs had to be planned. It was clear that the fault report IDs needed to be in that component. The link between the test results in processing and the fault report IDs related to those test results had to be planned. All the fault report IDs had to be separated at least by radio type, software, and test type. First it was considered to only separate the fault report IDs based on radio type and software. Then more information was delivered by the test line owners, info about test type was needed in the separation too, because the fault report IDs could be different based on the testing type.

Test results contain information of the executed test details. These details are known as test path. This path contains all the information that was planned to be needed to separate the fault reports. This path is also used in the test results reporting tool as one parameter to navigate the test results to the right destination in the test management tool. Other identifications are also used but those are not covered in this thesis. This path was decided to be used as a link between the test results and the fault report IDs related to those results.

4.3.2 Fault reports storage options

One part of planning the fault report IDs storing, was to plan what was the information this component should contain. What was needed, was the fault report IDs and the path to where the fault report IDs were to be included. At the same time when planning the contents of this component, its format and location was planned. This information could be stored to a file or a database.

4.3.2.1.1 Database storage

Couple databases were considered for storing the data. It could be stored as a file or table. If the information was a table in database, it would be option to fetch only the needed data in the run to the test reporting tool. If the fault report ID data was in a file in database, it could be fetched from the database in the run and be handled in the test reporting tool.

The team has two internal use databases that were considered as an option to store the fault report ID data. One of these databases are mainly for test results and another one is for storing configurations. It was decided that this information will be stored in a file. The configuration database would be suitable for storing the file as it is a configuration file.

This option had good arguments to be considered as final solution, but it also had some downsides. The actors in the process to keep this file up to date were the test line owners. The configuration database was not fully implemented in use in the team and therefore it was not considered to be good option from the test line owners' point of view. Also, user interface form to update the file would have been needed to create to the database since it does not have any file editor to edit files. This would have taken much more time to implement.

4.3.2.1.2 Version control

After the database option for this file was abandoned, idea for storing the data to the fault report tool repository in GitLab was introduced. GitLab was described in chapter 3.2 This option was strongly requested by the test line owners.

If the file would be in GitLab, there was some problems to solve. It was important that this file is always up to date. A scenario where multiple different versions of this file would been in local branches wanted to be avoided. Therefore, it was decided that if this file will be stored in test reporting tool branch in GitLab, it should be edited only in GitLab web view.

GitLab has web interface and possibility to file edit in web view. This is beneficial since the file needs to be updated manually. GitLab is used in the team daily and the test line owners are familiar with it. The option to keep the file in GitLab also has some downsides. When using GitLab for storing and editing file, it is not used for what it was developed for. Surely, a new version of the file is created when it is updated and edited in GitLab web view. To be able to keep the file records of fault reports' current status reliable, it was decided that if this file is stored in GitLab, every new edit should trigger a new release of the test result reporting tool and the file will be included in the build of each release.

4.3.3 Data type

The file type was still not decided and both csv, comma separated value, file type and JSON, JavaScript Object Notation, file type were considered. Csv type file reduces the number of lines in the file but when not opened in csv supported editor, it is not clear for the user. The number of lines in a JSON file is more than in csv, but it is clearer to read in editor since JSON formatting is more often offered by any editor. The csv type file would have been possible in other solutions, but in GitLab solution, the JSON file was better.

Since the option of the GitLab was strongly requested by the test line owners, it was decided that the file containing the fault report IDs information was going to be stored to the test reporting tool repository in Gitlab in JSON type.

4.3.4 Updating principles

In the fault report ID reporting, the IDs need to be removed from the test management tool when the fault report is closed. This needed to be considered in the planning. The file containing the fault report IDs needed to have information if the fault report ID is to be inserted or removed. The test line owners also wanted to have sections for free comment to each fault report ID. It was also decided that the time of update of fault report ID in the file need to be recorded as well as the user who have been updating the file. With this information the process could go further.

The test reporting tool was studied and planned what parts of the code needed to be updated. The fault report IDs containing JSON file is included in the source distribution of the package since this is crucial for the test results reporting tool fault report ID reporting feature. This is a way to ensure that the file the test result reporting tool is using, is from trusted source.

5 RESULT

In the implementation of the thesis, fault report ID reporting automation, fault report IDs containing JSON file was created, test reporting tool was updated for fault report IDs processing and workflow for fault report IDs updating were implemented. The final implementation of the fault report ID reporting in the test results reporting tool is following the rules set in the requirements. Failed test results are allowed to be reported with status of failed only with fault report ID. Failed test results without fault report ID are reported with status blocked instead of failed, and comment of environmental problems is added to the test results.

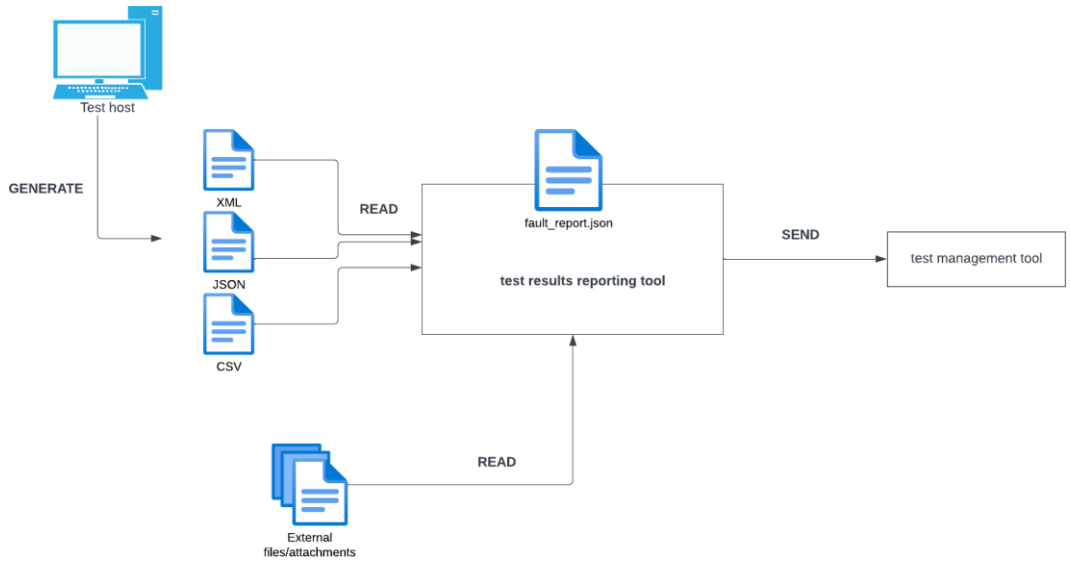

Figure 3 Results reporting process.

Fault report IDs automated reporting was implemented as a part of test results reporting tool. Test results reporting process is visualized in Figure 3. In the process, test host generates ouput files for each test run. Test results reporting tool reads and processes these files and sends results based on them to the test management tool. The fault_report.json file used in fault test reporting, is included inside the test results reporting tool.

When test fails for the first time, the tool sends the test results with status blocked instead of fail. The test line owner inspects the cause of the failure and if found needed, new fault report is created.

When fault report is created, test line owner updated the fault report JSON file and records new fault report for reporting. When test fails for the second time and the test results reporting tool is upgrated, the tool fetches fault report ID from the fault report JSON file under corresponding path and adds the ID to test results payload and test results with fail status are sent to the test management tool.

When fault report is closed and its ID is not needed to be reported anymore, the test line owner edits that fault report ID record in the fault report JSON file to be removed. When the test results tool is run again after the update, the tool gets the information of current reported fault report IDs from the test management tool. If in the fault management tool is reported the fault report ID that is now recored to be removed, it is removed from the list of fault report IDs gotten from the test management tool, and a new list without that fault report ID is sent to the test management tool with the test results.

5.1 Fault report ID JSON file

The fault report JSON file has all the current fault report IDs recorded. It is read by the test results reporting tool every time the tool is used for sending results to the test management tool. The fault report IDs containing file is divided to list of JSON objects. Each list object has value of reporting path and value of fault reports related to that reporting path. Fault reports are stored in list of objects. Each fault report object in list has keys and values of “faultreportId”, “comment”, “updatedOn”, “updatedBy” and “action”. Part of fault report ID JSON file is shown in Figure 4.

5.2 Fault report ID JSON file validation script

For the fault report IDs containing file, a validator was made to verify the file contents. It is written in Python, and it uses jsonschema for validation of the file. Jsonschema was introduced in chapter 3.8. In the jsonschema declaration the JSON file contents are defined. The schema covers all required properties, file structure and item types. In addition, because the crucial part of reporting, the “action” item is defined to have only two allowed values, “insert” and “remove”. This value gives information to the test results reporting tool which action to perform for the fault report ID.

The Gitlab pipeline runs the validator among other tests set to run in the pipeline. The validator is checking if the JSON document is valid and if it contains all mandatory fields with correct values. If the fault report JSON file does not meet the requirements stated in the schema, the validation pipeline will fail, and the validation job will give error messages of what is wrong in the file. If file has missing property, it will provide error message of missing required property and for action property, if item value is not “insert” or “remove”, message of value not being one of “insert” or “remove” is provided. In this case, the user must edit the file and commit again until the pipeline for the validator passes. Figure 5.

5.3 Fault reports updating workflow

The fault report ID containing JSON file is updated in GitLab WebView. User opens the file in GitLab editor and updates needed fields. If new fault report is created, user creates new JSON object under correct path. If already existing fault report needs to be updated to be removed from test management tool, user changes the action, “updatedOn” and “updatedBy” sections under the affected fault report. Part of fault report JSON file is pictured in Figure 4. User commits changes and this triggers Gitlab pipeline for testing.

Figure 4. Two example fault report objects pictured from fault report IDs containing JSON file. Title names differ from the actual file for confidentiality reasons.

If the pipeline job passes, user creates a merge request to the repository maintainer. Changes are merged to master and new release of the test results reporting tool is made. The new version of

the test results reporting tool is updated to the test line using the test results reporting tool. Figure 6.

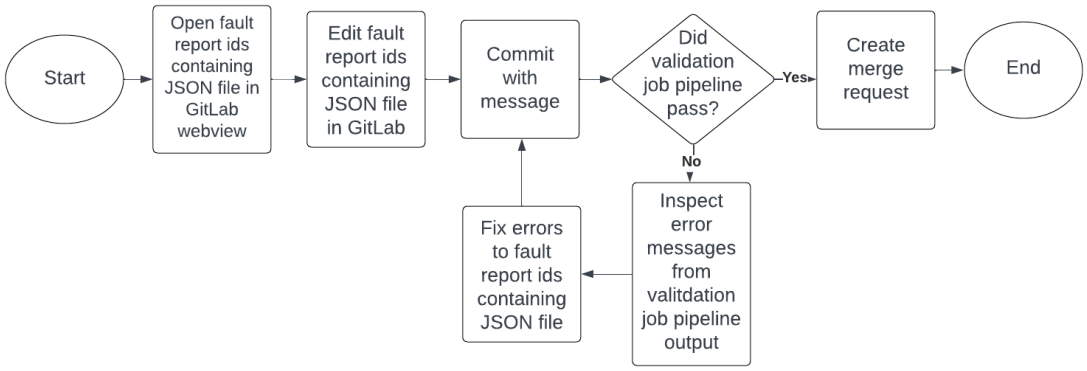

Figure 5 Updating fault report IDs containing file workflow.

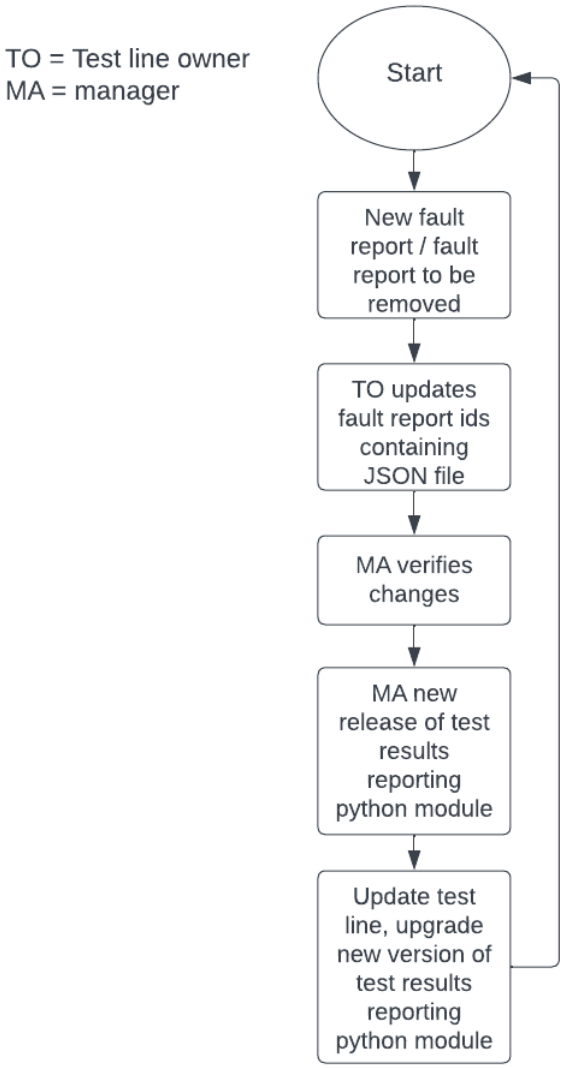

Figure 6 Fault report IDs reporting workflow.

5.4 Fault report handling class

New class was made for the module to handle the fault reports containing JSON file data. This class static methods are described in Table 2. This chapter describes the methods used in this class in more detailed level.

Method “update_test_instance_fault_reports” is responsible of creating list of fault report IDs that are going to be included in the payload for test results sending. From parameter “test_instance”, reported fault report IDs are parsed to a list. Value of the “test_instance” is from test management tool client module in the tool. That module makes different API requests in methods to the tool and needed values for reporting are returned. Parameter “fault_reports” is list of fault report object parsed from the fault report IDs containing JSON file. The value of parameter “fault_reports” is the output of “get_config_fault_reports” method. This list can be empty if no match of paths has found in “get_config_fault_reports”.

In method “update_test_instance_fault_reports”, list of found fault reports are divided to fault reports to insert and fault reports to remove by calling method “sort_fault_reports_by_action” with arguments of “insert” and “remove”. The method “sort_fault_reports_by_action” loops through every object in the list. If the action in the object “action” value is match to the action given as an argument, the fault report ID of that object is added to a list. The method returns a list of fault report IDs.

Fault report IDs are then compared to fault report IDs that are already reported in methods “sort_fault_reports_for_insert” and “sort_fault_reports_for_remove”. In “sort_fault_reports_for_insert”, list of not yet reported fault reports are returned. If no new fault report IDs for insert are found, empty list is returned. All the IDs in this list are added to the list of already reported fault report IDs in “update_test_instance_fault_reports” method. In “sort_fault_reports_for_remove”, all the matching IDs are returned in list and removed from already reported fault report IDs in “update_test_instance_fault_reports” method. These actions are done only if the methods “sort_fault_reports_for_remove” and “sort_fault_reports_for_insert” returned lists that were not empty. Based on these actions, Boolean values of “is_fault_reports_removed” and “is_fault_reports_inserted” are set. Method returns dict containing key – value pairs of “test_instance_fault_report_ids” that contains updated listing of fault report IDs to be sent to the test management tool with test results, Boolean values of “is_fault_reports_removed” and “is_fault_reports_inserted”.

Table 2. Fault report handling class static methods

Method definition	Method description
def get_faultreport_json(path: str):	Get faultreport.json from given path. Returns dict if file opened and read succesfully, otherwise returns None.
def get_config_fault_reports(CONFIG: dict, payload: dict) -> list:	Get fault reports with matching test path from config. Return list of found fault report objects.
def sort_fault_reports_by_action(fault_reports: list, action: str) -> list:	Sort fault reports to fault reports to remove and fault reports to insert. Return values are empty lists if no fault reports to remove or insert.
def update_test_instance_fault_reports(test_instance: dict, fault_reports: list) -> dict:	Returns test instance fault reports where config reported fault reports have been inserted and removed and info if fault reports have been removed or inserted in Boolean variables.
def sort_fault_reports_for_insert(fault_reports_action_insert: list, test_instance_fault_reports: list) -> list:	Find fault reports for insert. Compare lists, remove already reported fault reports from list to be added to test instance fault reports.
def sort_fault_reports_for_remove(fault_reports_action_remove: list, test_instance_fault_reports: list) -> list:	Find fault reports common to both lists and add to fault_reports_to_remove. Fault report cannot be removed if it is not in instance.

5.5 Workflow of fault report IDs reporting in the test results reporting tool

The test results reporting tool checks in the main script if the fault report IDs containing file is available and can be loaded. If the file cannot be loaded, info of not successful load is saved to a variable and all actions related to the fault report IDs is bypassed. If file is loaded correctly, info of successful load is saved to a variable. The loaded JSON data from the fault report IDs containing JSON file is merged to configuration module the tool is using in the reporting. When the part for reporting results to the fault management tool is reached, it is checked again if the JSON file is successfully loaded.

The test run generates output JSON file that has destination path to the test management tool for the reporting. The tool is trying to find match between the output JSON provided path and the path from the fault report IDs containing data from the configuration used in the tool. If match is found,

all fault report JSON objects under that path are saved to a variable. The tool handles one test case at a time and for each test case, the collected fault reports are sorted by their action.

The test results reporting tool makes request to the test management tool. As a response is returned data of the specified test report test instances current results from the management tool. This response contains information of current fault report IDs in the specified path in the test management tool. Main script makes method call to the fault report handling class method "update_test_instance_fault_reports" and passes current reported fault report IDs to the method as a parameter. Return values of updated fault report IDs and Boolean value of removed fault reports are saved to variables. Updated fault report IDs value can be empty and this means no fault report IDs are available.

When the results are ready to be send to the test management tool, the status of the test case is reviewed. If the status is fail and there is fault report IDs available, the updated listing of fault report IDs is added to the payload to be sent to the test management tool and with test case status failed. If the status is failed and there are no fault report IDs available, the status is changed to blocked and comment of environmental problems is added to the payload to be sent to the test management tool. In test results with pass status, fault report ID listing is only updated when there are fault report IDs removed.

6 TESTING THE IMPLEMENTAITON

Implementation of the feature in the test results reporting tool was tested with pytest. Pytest was introduced in chapter 3.7. Tests are written in separate file containing only the tests. Test are constructed in a way as it is advised in pytest documentation: arrange, act, assert, and clean up. First in the test the testing environment is prepared for the test. Needed values are imported for the test and set to the test variables. Then the test actions are performed, for example function calls. After actions, assertations of expected test output and actual output of test actions are made. Finally test cleans up itself so that it has no effect to other tests. Test files are structured to have tests of one Python script or class. All functions are tested with expectation of good workflow, in case of errors and possible uses of the test. Pytest mocker is used to mock functions in test if functions are not needed in test, or if function return value is needed to set without using the actual function [16]. All tests related to the new feature are introduced in Table 3.

Table 3. Pytest tests

Test definition	Test description
def test_get_config_fault_reports_pass():	Test get config fault reports expected passed.
def test_insert_sort_fault_reports_by_action():	Test sort fault reports by action when only fault reports with insert.
def test_remove_sort_fault_reports_by_action():	Test sort fault reports by action when only fault reports with remove.
def test_insert_and_remove_sort_fault_reports_by_action():	Test sort fault reports by action when fault reports with remove and insert action.
def test_pass_sort_fault_reports_for_insert():	Test sort fault reports for insert expected pass.
def test_sort_fault_reports_for_insert_already_reported():	Test sort fault reports for insert when fault report already reported.
def test_sort_fault_reports_for_insert_new_fault_report_and_already_reported():	Test sort fault reports for insert when fault report already reported and new fault report for report.

def test_sort_fault_reports_for_insert_no_duplicate():	Test sort fault reports for insert when both fault reports already reported. Both fault reports should be removed.
def test_sort_fault_reports_for_insert_empty_input():	Test sort fault reports for insert when empty input.
def test_sort_fault_reports_for_insert_to_empty_ti_fault reports():	Test sort fault reports for insert when test instance fault reports are empty list.
def test_sort_fault_reports_for_insert_empty_input_to_empty_ti_fault_reports():	Test sort fault reports for insert when empty input and test instance fault reports is empty list.
def test_pass_sort_fault_reports_for_remove():	Test sort fault reports for remove expected pass.
def test_sort_fault_reports_for_remove_not_in_ti_fault_reports():	Try to remove fault report that is not present in test instance fault reports.
def test_sort_fault_reports_for_remove_empty_input():	Try to remove empty input from test instance fault reports.
def test_sort_fault_reports_for_remove_empty_input_empty_ti():	Try to sort fault reports for remove when empty input is given, and test instance fault reports is empty.
def test_sort_fault_reports_for_remove_empty_ti():	Try to remove fault report from empty test instance fault reports.
def test_sort_fault_reports_for_remove_one_remove_one_not_present():	Try to remove fault report that is not present in test instance, and one that is in test instance fault reports.
def test_return_empty_sort_fault_reports_by_action():	Test sort fault reports by action remove when only fault reports with insert. Should return empty list.
def test_update_test_instance_fault_reports_pass():	Test update test instance fault reports expected pass case.
def test_insert_to_empty_ti_update_test_instance_fault_reports():	Test update test instance fault reports when test instance have no fault reports, and config has fault report with insert action

def test_remove_from_empty_update_test_instance_fault_reports():	Test update test instance fault reports when test instance has no fault reports and fault reports with action remove are given. Boolean values should not change.
def test_empty_input_empty_output_update_test_instance_fault_reports():	Test update test instance fault reports when test instance have no fault reports and fault reports from config is empty.
def test_empty_input_update_test_instance_fault_reports():	Test update test instance fault reports when test instance have fault reports and empty config fault reports is given. Boolean values should not change.
def test_remove_fault_report_not_present_in_test_instance_fault_reports():	Try to remove fault report that is not present in test instance fault reports. Boolean values should not change.
def test_no_duplicate_update_test_instance_fault_reports():	Test update test instance fault reports that no duplicate will happen when config fault report has action insert and fault report is already in test instance. Boolean values should not change.
def test_faultreport_return_nothing(mocker):	Test test results reporting functionality when no faultreport file is available. Functions not related to test are mocked.
def test_fail_with_faultreport_send_to_test_management_tool(mocker):	Test test results reporting for send to test management tool with fail test case and fault reports. Functions not related to test are mocked.
def test_fail_with_no_faultreport_send_to_test_management_tool(mocker):	Test that status changes to blocked with fail case and current reported fault report is removed. Functions not related to test are mocked.
def test_pass_with_fault_report_removed_send_to_test_management_tool(mocker):	Test that fault report is removed also with pass case. Only the one with remove is al-

	lowed to be removed and rest of the fault reports stay as reported. Functions not related to test are mocked.
def test_pass_report_no_fault_reports(mocker):	Test that no fault reports is reported in test instance or run with pass case even if fault report file has fault reports to insert.

Fault report IDs containing file is tested with jsonschema in a Python written validator script. A schema for JSON file is made and it contains structure, required keys and types of the expected JSON file. All possible errors are printed for user in case of the validation does not pass. These tests are run in GitLab pipeline stages after every commit to repository. Pytest testing and the validation script together have great test coverage of the new feature to the test results importing tool.

Goals set in the requirements were covered in the final implementation of the feature. Test results are only sent with fail status when fault report IDs are available. When no fault report IDs are available in fail result test case, status is changed to blocked and comment of environmental problems is added as stated in the requirements. Test line owners only need to update one file with fault reports to get the fault report IDs to the test management tool instead of manual update. This takes much less time to complete and the update of one file is more convenient when considering the amount of test cases needed to update one by one manually in test management tool.

7 SUMMARY

The aim of this thesis was to find a solution for automating fault report IDs reporting in case of fail test case. Among reporting, one goal was to make information of current fault reports more accessible inside team. One target was also to reduce work time used in current way of reporting fault report IDs.

As a result, current test results reporting tool was developed in a way, that it will report fault report IDs together with test results in fail cases. Fault report IDs are recorded in JSON file, where test line owner updates them. Single file, where all fault report IDs are collected, provides better information accessibility when all the fault report IDs are accessible for any team member in one location. Test line owners only must update fault report IDs to one location since all the information is stored in one file.

The goals set were to get fault report IDs reported to test owner tool with test results and provide working solution for use during this thesis work. Current solution is working as set in the requirements, and it has been taken to use in the team. By this time, the test line owners have found beneficial that in current solution all the fault report IDs are in one file, and that test management tool is not necessarily needed to use anymore since all the reporting is done with already used test results reporting tool, and changes are only needed to update in GitLab, a tool that was already in daily use.

8 CONCLUSION

For planning the work, implementation and deploying the solution, four months of time was reserved. Work was done in that time. I felt the planning phase the most challenging part of the work. Test management tool was completely new to me, and I did not understand much of fault reports or their reporting practises. Background research for those took long time and not much information dedicated to my problem was available in Nokia's internal sources. Tools and their use practises information had to be asked from test line owners. Also, not many requirements were set, fault report IDs had to be reported automatically to the test management tool. Other requirements seeking and planning and deciding good practices from wide possibilities caused challenges. When decision was made to implement fault report IDs reporting as a part of current results reporting tool, planning got easier. In my opinion, different options in different areas of the work were examined extensively, and the end users, i.e., the test line owners, were also listened to when making decisions. Implementation phase and deploying the solution proceeded fast compared to the planning phase. I am satisfied that the final solution is now in the use, and it is working as was hoped.

Current version of fault report reporting uses JSON file where the information of fault report IDs is recorded. This solution requires always new version of test results reporting tool when the fault report file is updated, so that each test line has latest version of the JSON file and thus the latest information of current fault reports. In the future, it would be beneficial to get the information in such location where test result reporting tool could load the latest version of the file during each run. This could be possible through the current database used inside the team if the database would be more widely used inside the team and the user interface in the database would be updated to more user friendly.

On the other hand, updating the file by the test results reporting tool based on its own information would be good direction for development, but this requires more information to the fault report so that assured link between test results and fault reports could be formed.

Current solution restricts the fault report IDs reporting to the test results reporting tool. Fault report IDs can be reported, when using the latest version of the test results reporting tool. If fault report IDs containing file would be in external data storage, the file could be used more widely. Sure, fault

report IDs are now firstly used for test results reporting and no other information of use inside the team for this information is known by now.

9 REFERENCES

[1] “Nokia Corporation,” nokia.com, [Online]. Available: https://www.nokia.com/. [Accessed 8 April 2023].

[2] “GitLab - What is CI/CD?,” GitLab B.V., [Online]. Available: https://about.gitlab.com/topics/ci-cd/#ci-cd-explained. [Accessed 6 April 2023].

[3] S. Pittet, “Continuous integration vs. delivery vs. deployment, Atlassian,” Atlassian, [Online]. Available: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment. [Accessed 8 April 2023].

[4] “The DevSecOps Platform | GitLab,” GitLab, [Online]. Available: https://about.gitlab.com/. [Accessed 8 April 2023].

[5] Jenkins, “Jenkins - automation server,” Jenkins.io, 6 April 2023. [Online]. Available: https://www.jenkins.io/.

[6] “Using Jenkins agents, documentation, Jenkins - automation server,” Jenkins.io. [Online]. Available: https://www.jenkins.io/doc/book/using/using-agents/. [Accessed 8 April 2023].

[7] Buddy, “Buddy: The DevOps Automation Platform,” Buddy.Works, [Online]. Available: https://buddy.works/. [Accessed 8 April 2023].

[8] Atlassian, “Bamboo Continuous Integration and Deployment Build Server,” Atlassian, [Online]. Available: https://www.atlassian.com/software/bamboo. [Accessed 8 April 2023].

[9] CircleCI, “Continuous Integration and Delivery - CircleCI,” CircleCI, [Online]. Available: https://circleci.com/. [Accessed 8 April 2023].

[10] Coursera, “What Is Python Used For? A Beginner’s Guide | Coursera,” Coursera, 14 November 2022. [Online]. Available: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python. [Accessed 8 April 2023].

[11] “Welcome to Python.org,” Python.org, [Online]. Available: https://www.python.org/. [Accessed 18 April 2023].

[12] “pytest: helps you write better programs - pytest documentation,” pytest, [Online]. Available: https://docs.pytest.org/en/7.2.x/. [Accessed 8 April 2023].

[13] “pip - PyPI,” pypi.org, [Online]. Available: https://pypi.org/project/pip/. [Accessed 21 April 2023].

[14] J. Berman, “jsonschema - PyPI,” pypi.org, [Online]. Available: https://pypi.org/project/jsonschema/. [Accessed 8 April 2023].

[15] OpenJS Foundation, “JSON Schema,” json-schema.org, [Online]. Available: https://json-schema.org/. [Accessed 8 April 2023].

[16] “pytest-mock documentation,” pytest, [Online]. Available: https://pytest-mock.readthedocs.io/en/latest/. [Accessed 8 April 2023].