TestResultWebApp

v. 0.1.3

Thomas Pollerspöck

18.10.2022

Contents

1 Introduction **1**

2 Description **2**

2.1 TestResultWebApp Architecture 2

2.2 Import Data 2

2.3 Data Visualization 3

2.3.1 Main menu 3

2.3.2 Dashboard View 3

2.3.3 DataTable View 7

2.3.4 Runtime View 11

2.3.5 Diff View 12

2.4 Developer guidance 14

2.4.1 How to run new TestResultWebApp instance 14

3 Appendix **17**

4 History **18**

Chapter 1

Introduction

TestResultWebApp is a web-based application which is developed and used at BOSCH since 2016 and was published as open source on Github in 2020.

TestResultWebApp is used for visualizing and tracking test execution results. It provides charts from an overview of the test result to the detail of all included test cases.

It also provides tools to control the quality of developing software version (under testing) by the a graphical comparison of test results from different test executions.

TestResultWebApp is highly modular written. Therefore it is also easy to extend it in order to add new graphics or evaluations of the test result data.

TestResultWebApp uses bootstrap, jquery, nodejs and mysql. For the charts is uses chartjs and D3.

Chapter 2

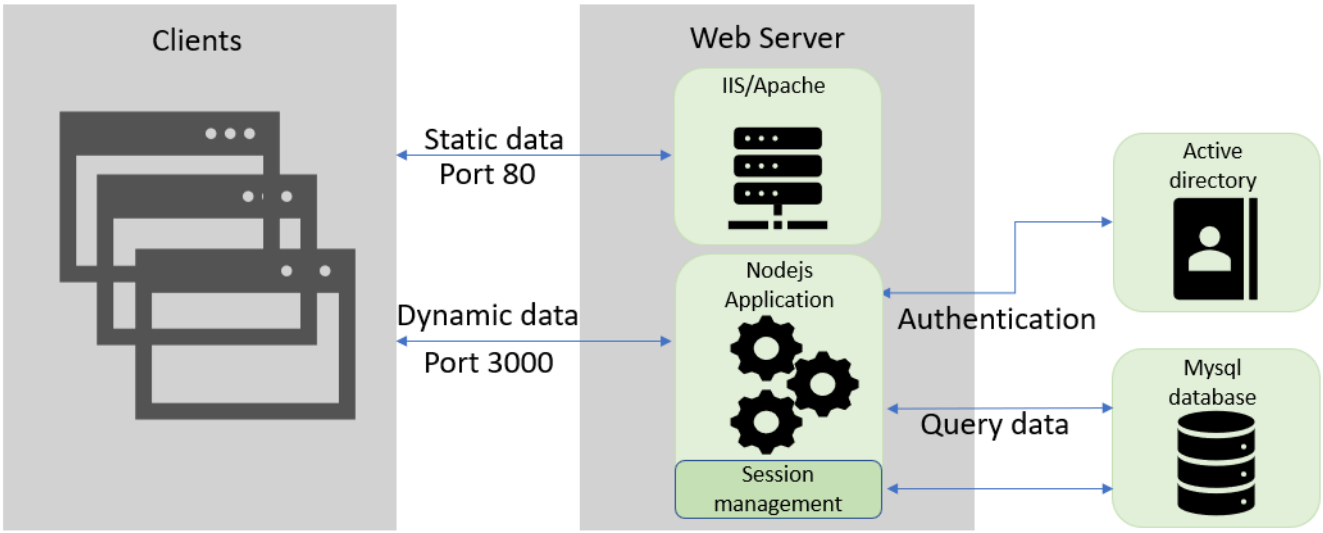
Description

2.1 TestResultWebApp Architecture

The TestResultWebApp application includes:

- Database: MySQL - which contains all test execution results. For details of all tables in the database, please refer the database model.
- Active Directory: for the authentication.
- Web server: which hosts the static data and runs the Node.js application for providing the dynamic data.
- Web client: is written in javascript which also use some libraries such as jQuery, bootstrap, Chart.js, D3 for data visualization.

Please refer below architecture for more detail:



2.2 Import Data

The data which will be visualized on WebApp comes directly from the database. For this the test execution result data must be transformed first into the defined database model then imported into the database.

The data base model is generic, so test result data can be any. Only a test result importer must transfrom and import the data.

We provide already the RobotResults2DB which helps to import the Robot Framework result file(s) *output.xml* to the database of the TestResultWebApp.

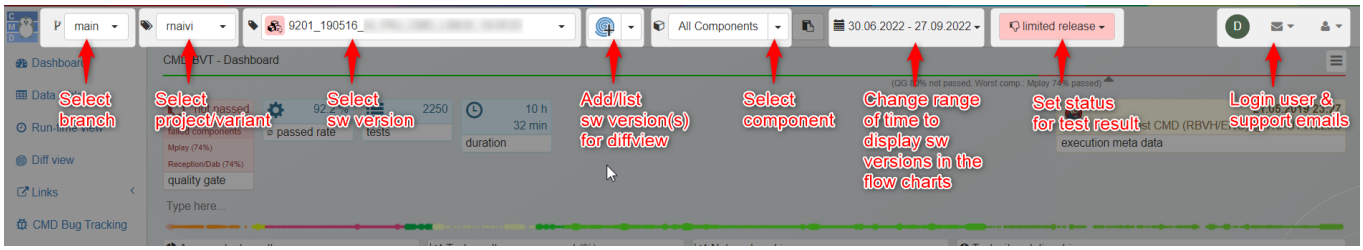
You will need to provide only the Robot Framework result file(s) and database's credential information, that RobotResults2DB will parse the test execution result data and interact with the provided database to import the data.

Please refer RobotResults2DB's usage and its document for more detail.

2.3 Data Visualization

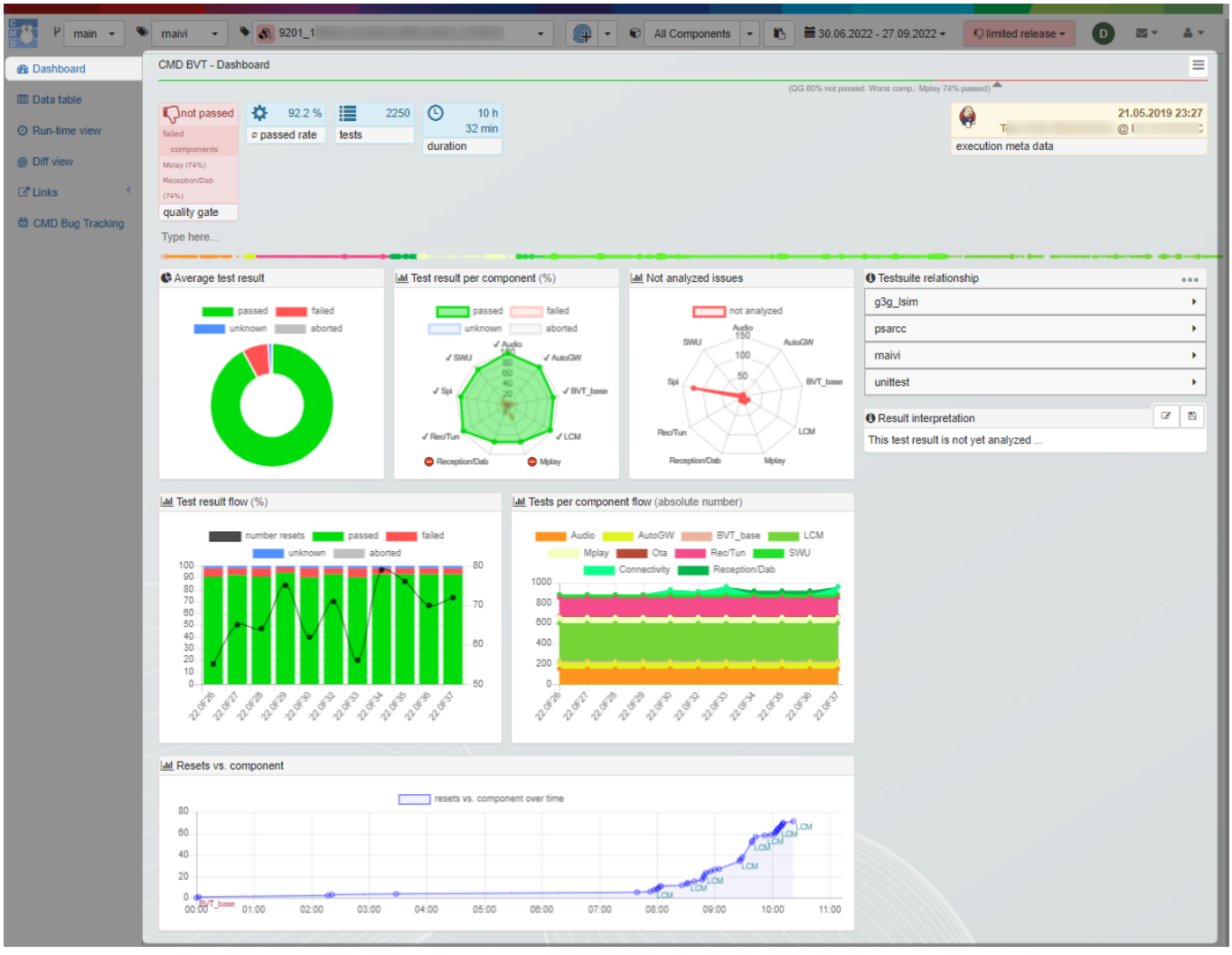
2.3.1 Main menu

The WebApp provides a main menu for selecting a specific **branch**, **project/variant** and **software version** to be displayed.



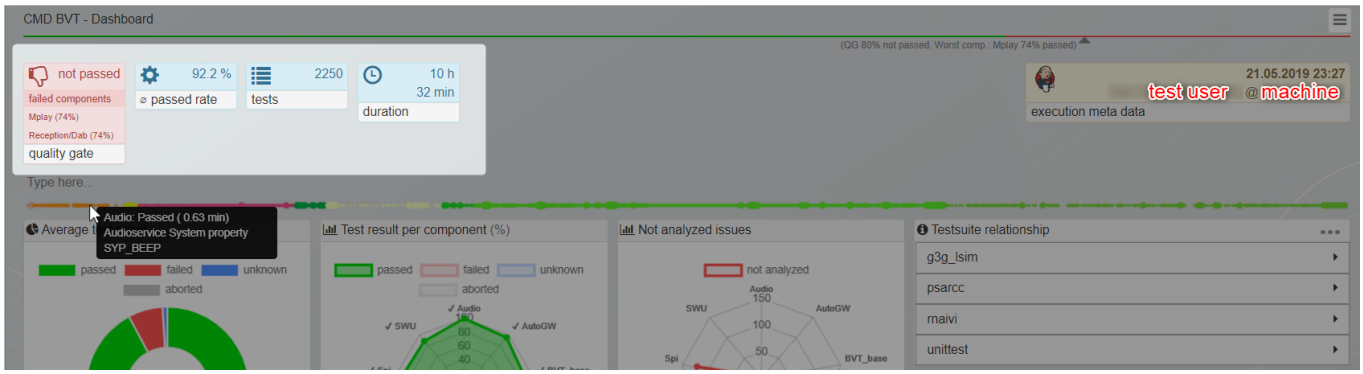
2.3.2 Dashboard View

The Dashboard view does not only provide an overview of test execution results but also shows the correlation betweeen components within the result and relationship with other test execution results.



Result overview

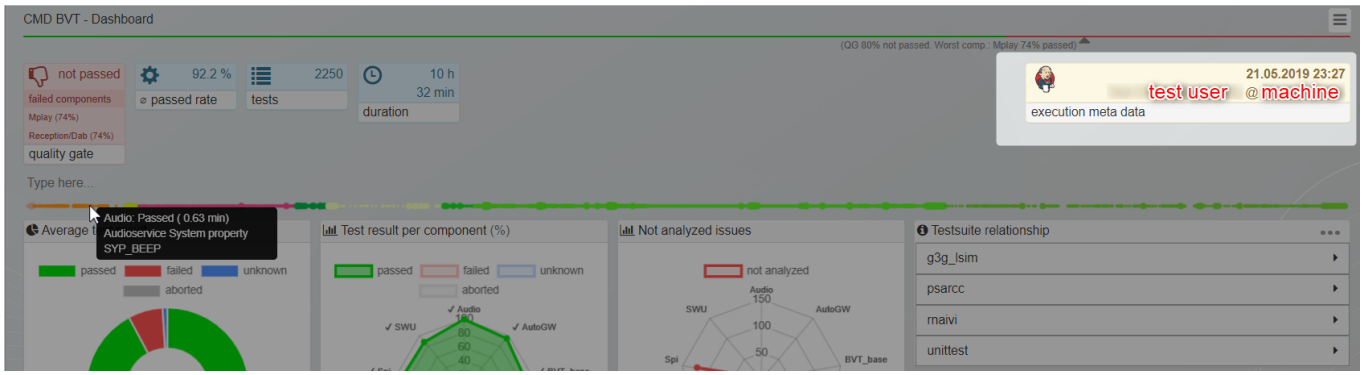
At the top left corner of the Dashboard view you find the test execution result statistics:



Which contains:

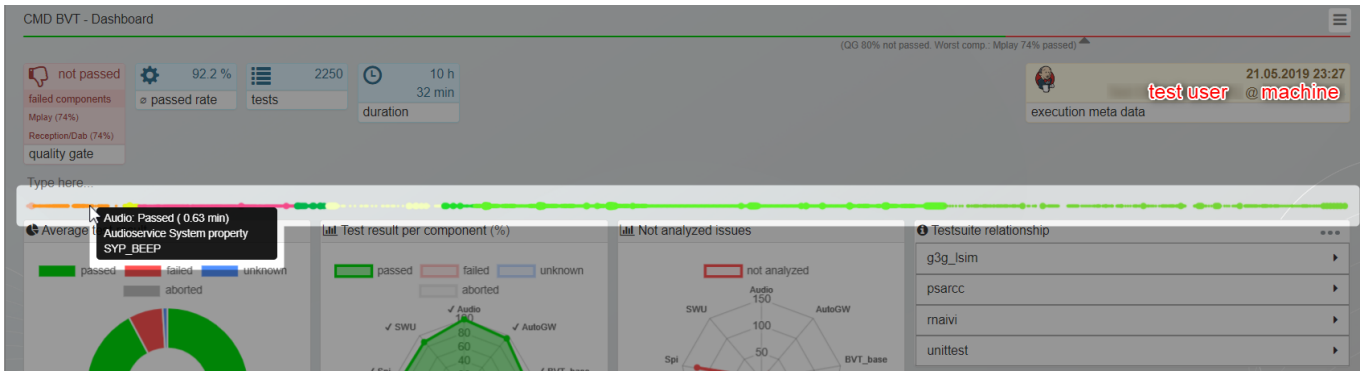
- Overrall status (you can define a qualitiy gate).
- Passed rate.
- Total number of executed test cases.
- Test execution duration.

On other right-hand side, there is information about the execution environment:



- Execution time (start of test execution)
- Test machine
- Test user
- Jenkins link (embedded URL in the Jenkins's icon)

Below them is the test execution result timeline:



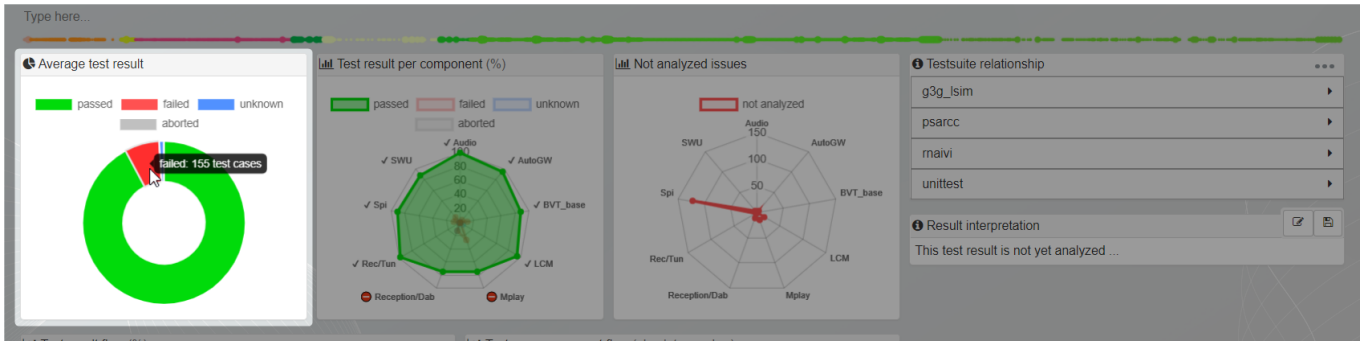
It provides:

- The timeline of the executed test cases which are grouped by components (different color). Left side is start of testing. Right side the end.
- How much time is consumed by the individual test case by the distance to the next test at the time line or the detail pop-up when hovering on the dot at the timeline.
- Test status result: A small dot for Passed status and a big dot for others.

Average test result

This chart will give you the detail of the test result with the percentage (number of test cases will be shown when moving the mouse over the pie chart) of each result status (Passed, Failed, Unknown or Aborted) of the execution.

So that, you can qualify this test execution result is good or not.

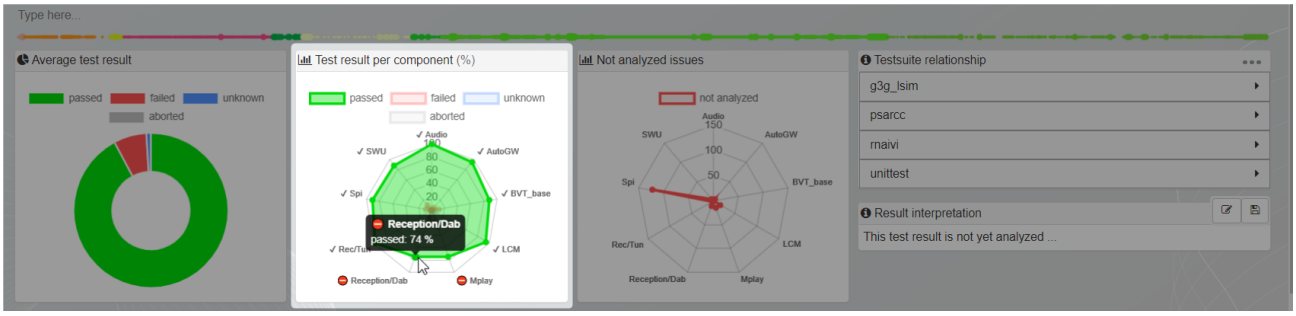


Component's correlation

The next charts will help you to get the correlation between components within the test result, so that you can know which component(s) impact(s) the test result.

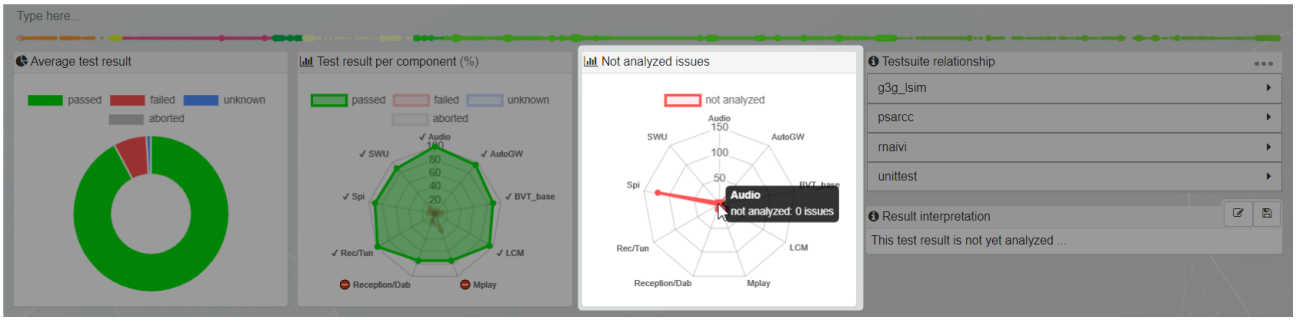
- **Test result per component** chart: provides a fast overview of test result percentages over all components. Here you can quickly see the quality of the system under test. Idially the spider chart should be fully green. This means that all component tests result in 100% Passed.

  You can also define a quality gate (default 80%) which results in a "minus" in front of the component name if the quality gate is not reached.



- **Not analyzed issues** chart: you can known how many test cases of components are issued without analysis.

  The Datatable view provides a process to set Failed test cases to "analyzed". Ideally this spider chart should have a red dot in the center. This means all Failed test cases are analyzed.



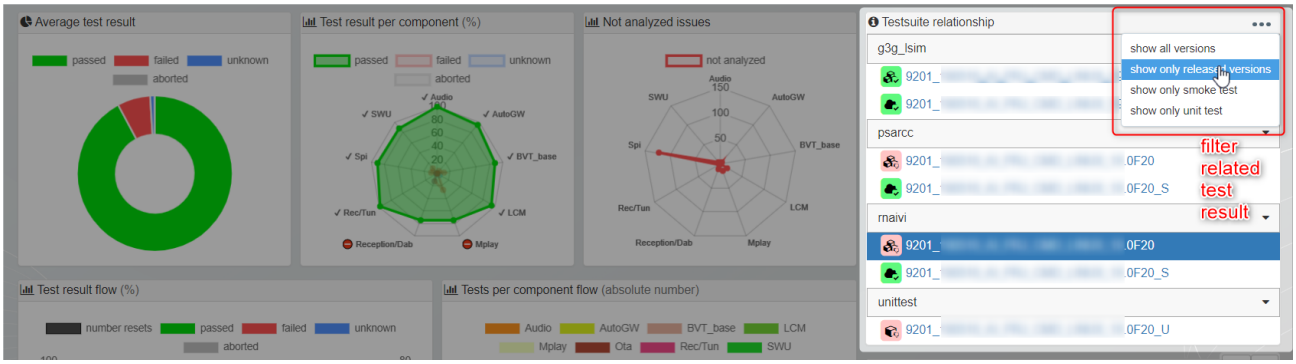
Relationships with other test execution results

- **Testsuite relationship**: will let you know all the related test results (grouped by project/variant) of the current selected version.

  This allows to quickly go to related test results to have a fast comparison about the quality of the selected version across all projects/variants.

For example: the selected version have been executed for 4 projects/variants: *g3g_lsim*, *psarcc*, *rnaivi* and *unittest*. Each variant (except the *unittest*) has 2 test results (one for the smoke test and one for the whole test execution result).

Then, all the related test execution results will be displayed as below:



There is also the context menu (**...**) that allows you to filter the related test results.

- **Test result flow** chart: provides the picture of quality change (percentages of each status) between versions.

  So that, you can understand the quality of testing software is being improved or vice versa.

- **Tests per component flow** chart: provides the change of number test cases per component between versions.

  You can aware the number of test cases per component and how many test cases are added or removed (per component or the whole test result) from those versions.

  This allows to quickly verify if all expected tests are executed, or if expected tests were not executed.

> **Notice**
>
> The versions which are displayed in **Test result flow** and **Tests per component flow** charts are the executed test results within the selected range of time in the main menu (default is **last 90 days**).

Result interpretation

You can give more information, provide an analysis, take notes, ... on the execution result by leaving comments in the **Result interpretation** section.

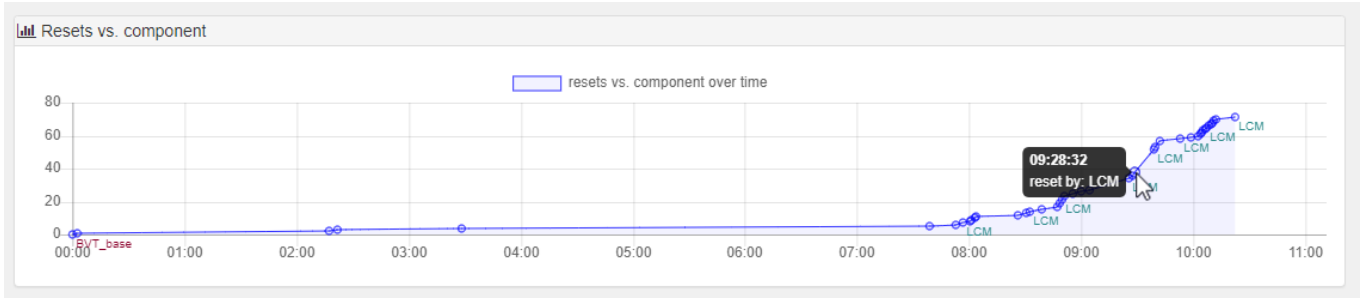
As soon as the Result interpretion is saved, that information will be updated to the database. So that, other users who browse to this test result can see the analysis/comment for reference.

Resets vs. component

The **Resets vs. component** chart will help you to know the interaction of component tests with the DUT (device under test) by providing the reset counter per component during the execution.

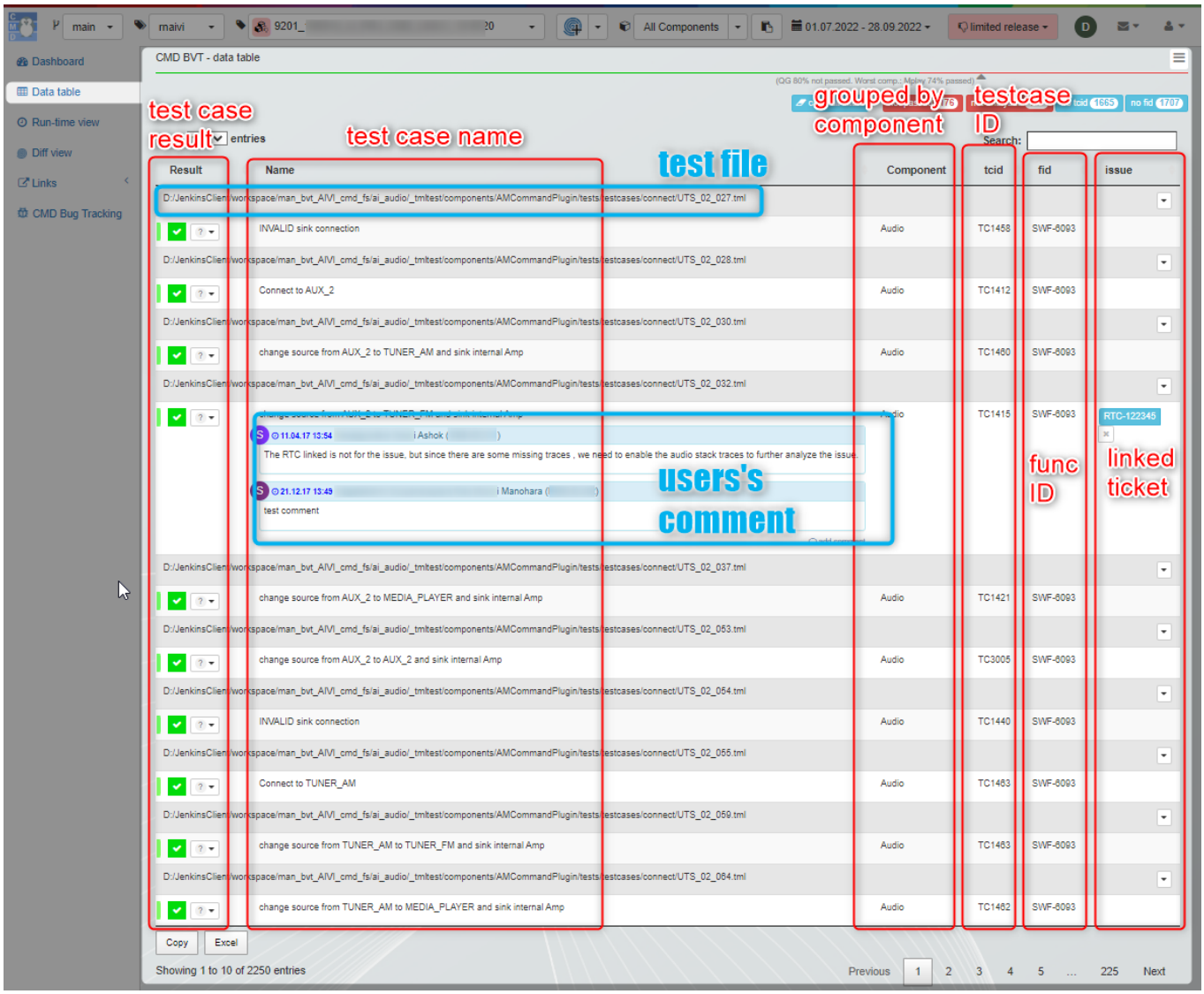
You can have awareness that:

- When the DUT has been reset.
- Which component tests were executed when a reset happened.
- How many reset has been done during each component and the whole test execution.



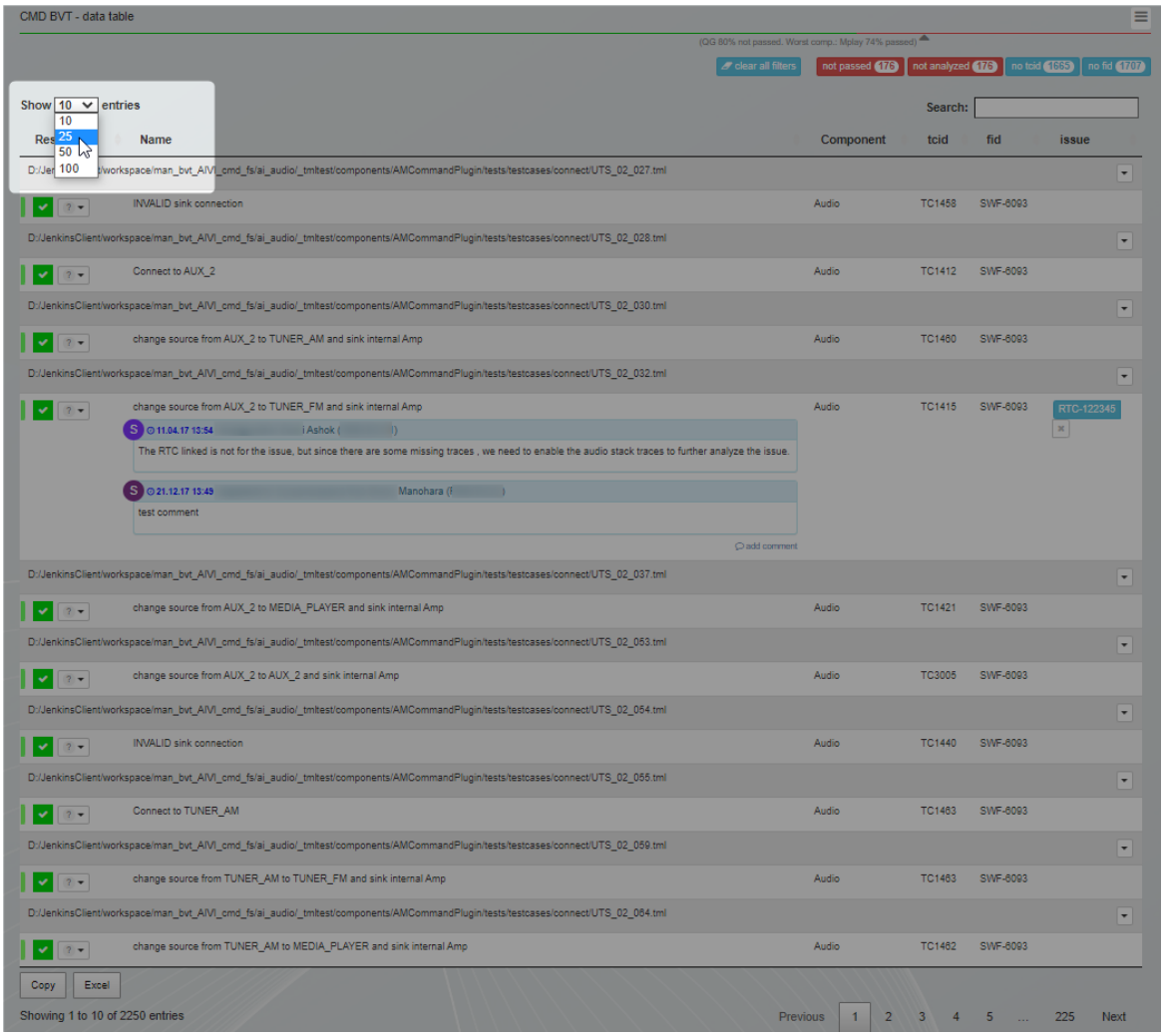
2.3.3 DataTable View

The Datatable view provides the summary table which contains all detail information of each test case (grouped by component) within the test execution.

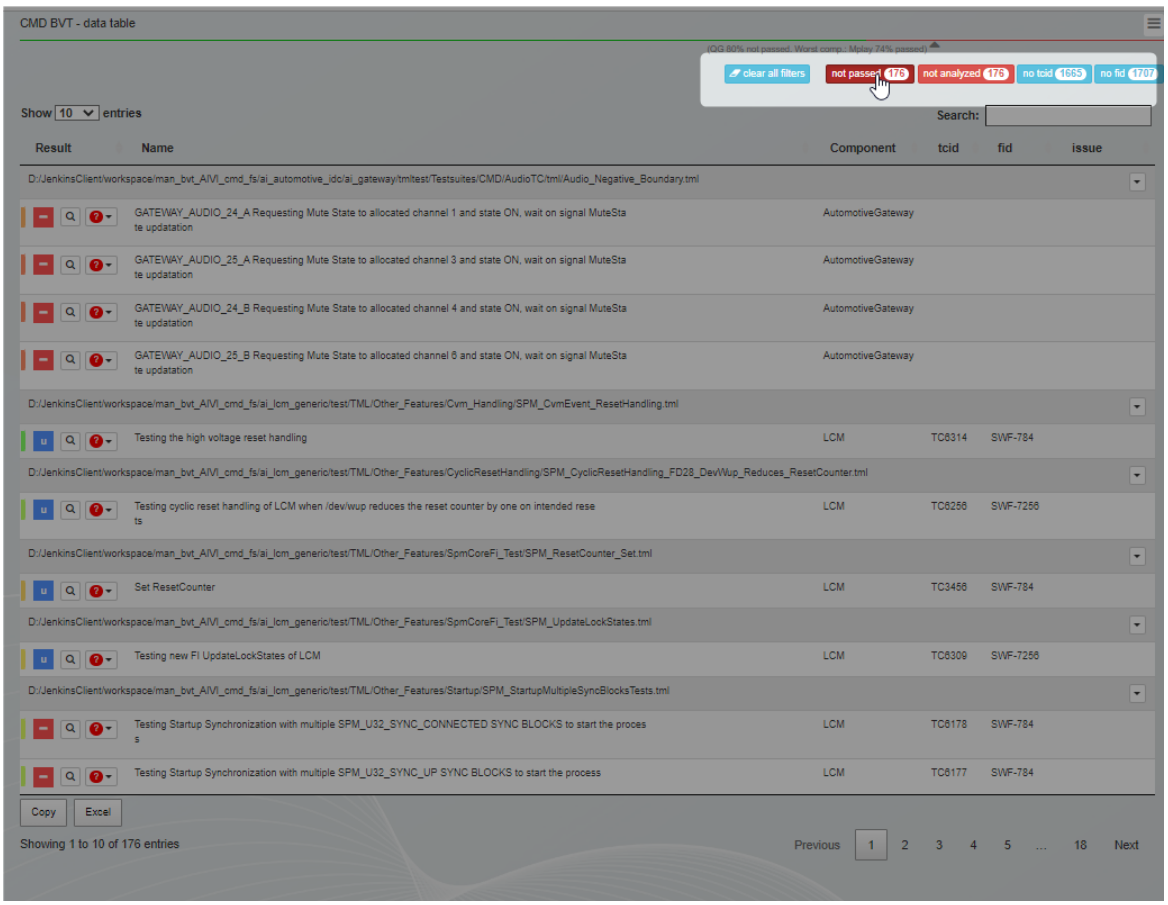


Besides, you can aslo:

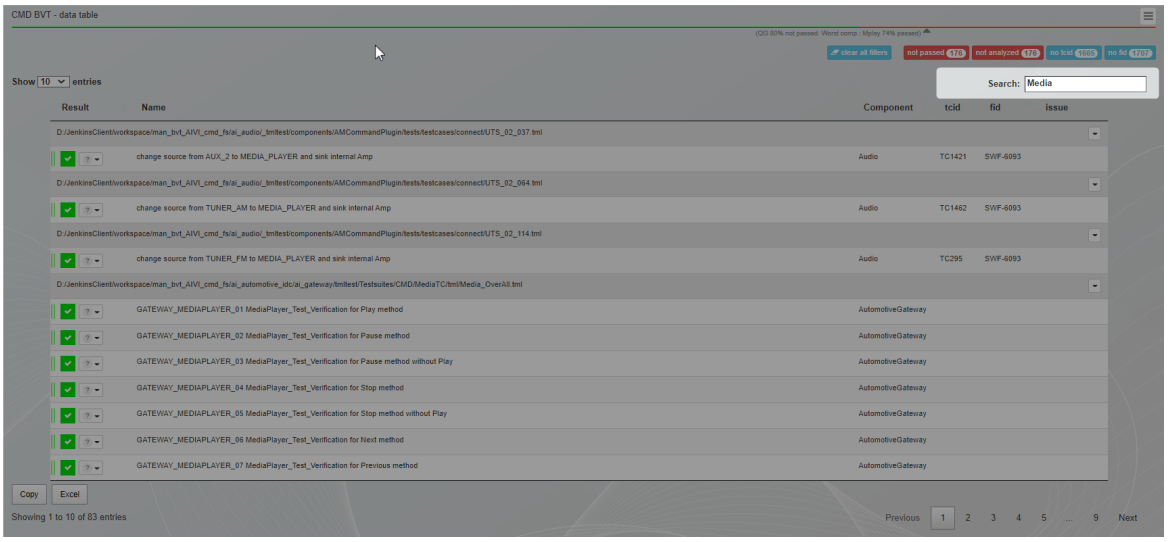
- determine how many test cases (entries) are displayed in the table page.



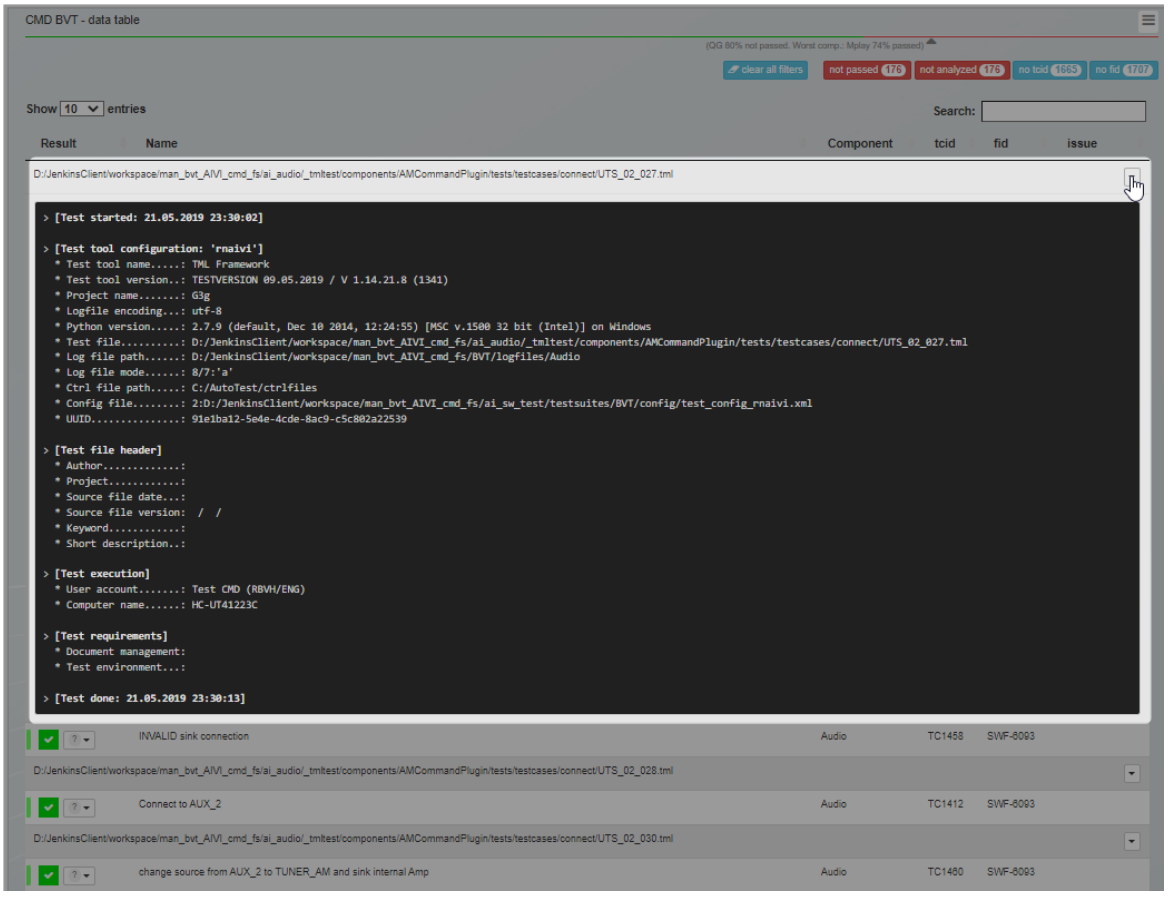
- apply filters to display such as not Passed test cases.



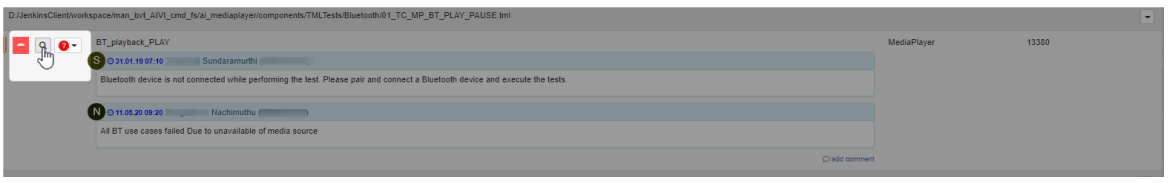
- search for the specific test case for reference.



- get more information about test environment, configurations.

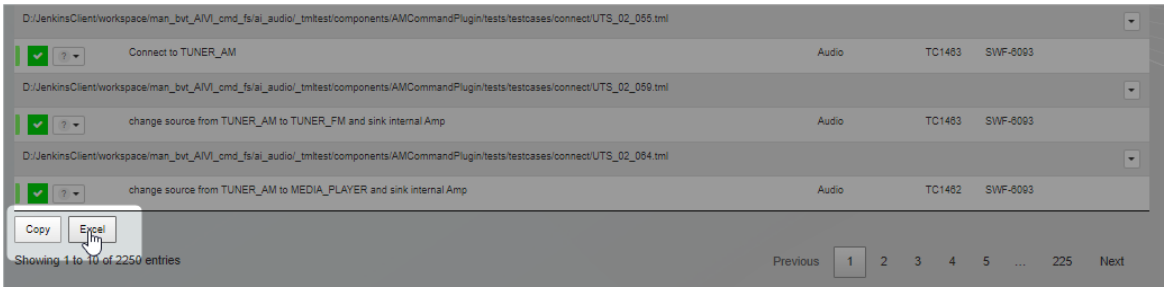


- get traceback information for not passed test case(s). An pop-up will be displayed which show all recorded traceback (error) information.



- give comment to the test case, link an issue ticket with the test case or watch the history of the user interaction on the test case.

- copy data table or export them to an excel file.

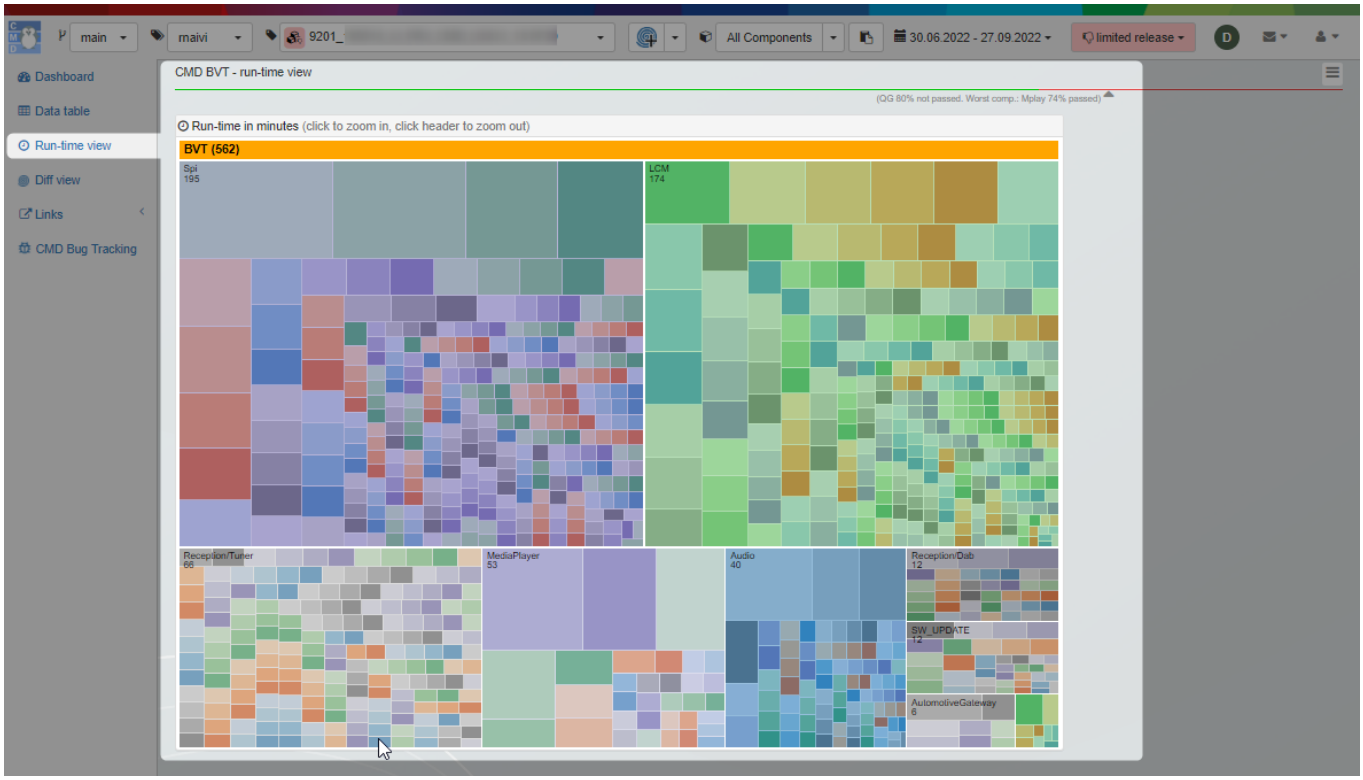


2.3.4 Runtime View

The runtime provides a treemap chart which helps you to know the runtime of components within test execution result or test cases within each component. The displayed number is always the runtime in minutes. The header shows the total runtime.

You can click on the component to go to detail runtime of all test cases within it and go back by clicking the component header.

With this view, you can understand the runtime of a your test suite then optimize the specific component or testcase if required.



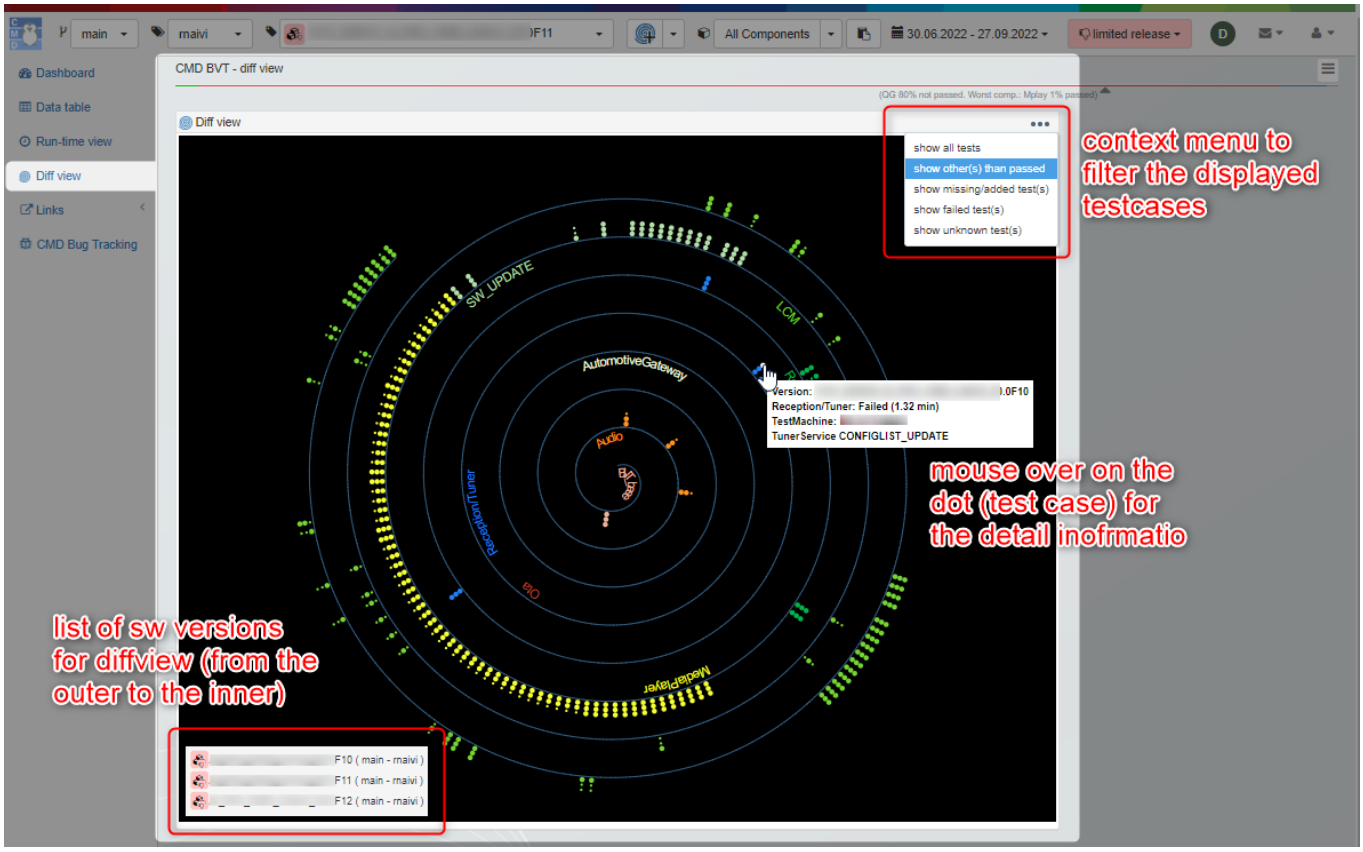
2.3.5 Diff View

The diffview contains a spiral chart which displays the differences between the software versions you want to compare. The center of the spiral is the start time of the test execution, the end of the sprial the end time. You find also the name of the component in the spiral. By default only changed or Failed test results are displayed.

So that you can:

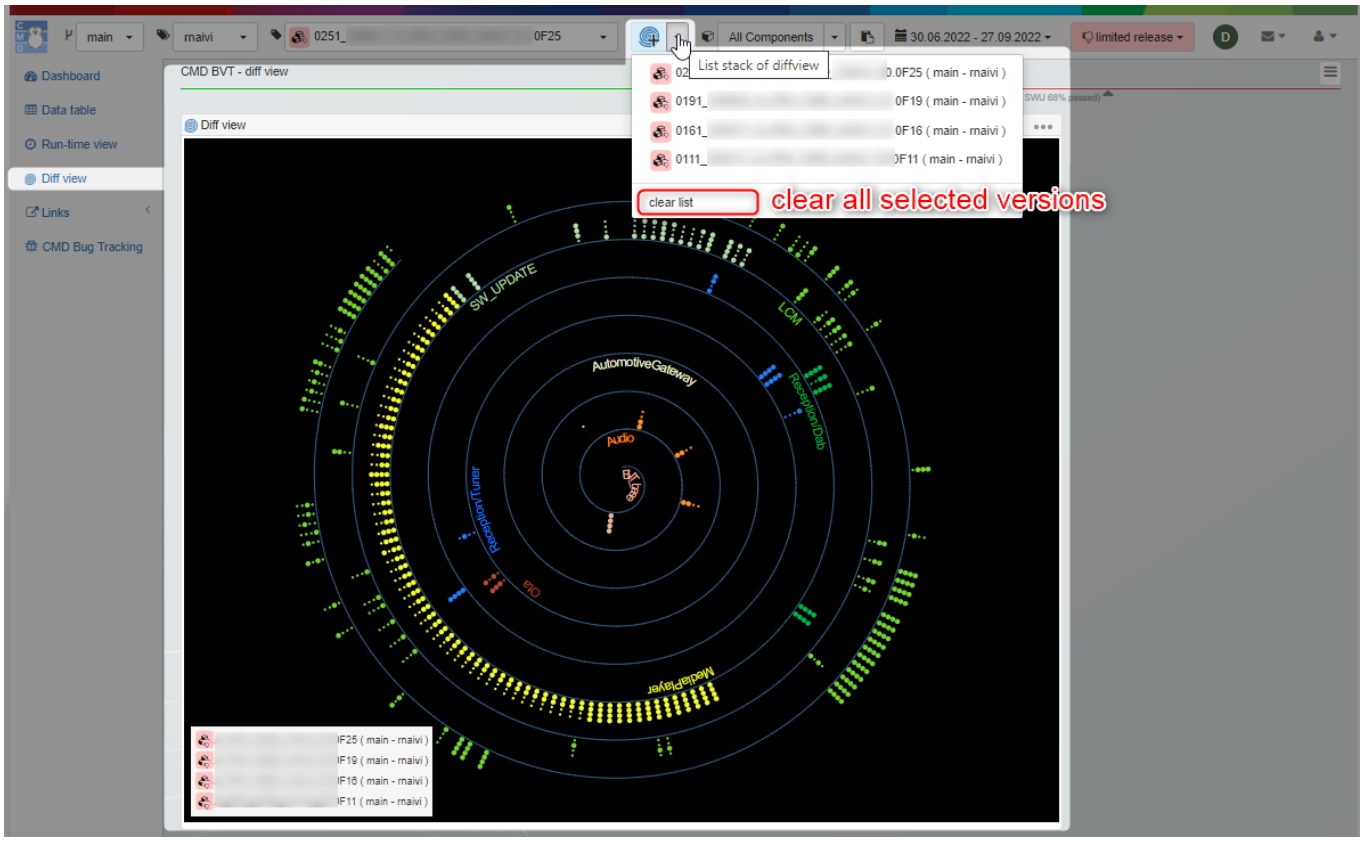
- see a test case result change (e.g from Passed to Failed).
- quickly find new or removed test case(s).
- recognize unstable test case(s)/component(s).

Be default, without adding the version for diffview, the current selected version and its around versions (the previous and the next version if existing) will be chosen for the diffview as below:



In case you want to add specific versions for the diffview, select the version from the version select box then click the add button  to add it to the list of versions to diff.

The dropdown button is used for viewing your selected versions. You can also clear your selection with **clear list** option.



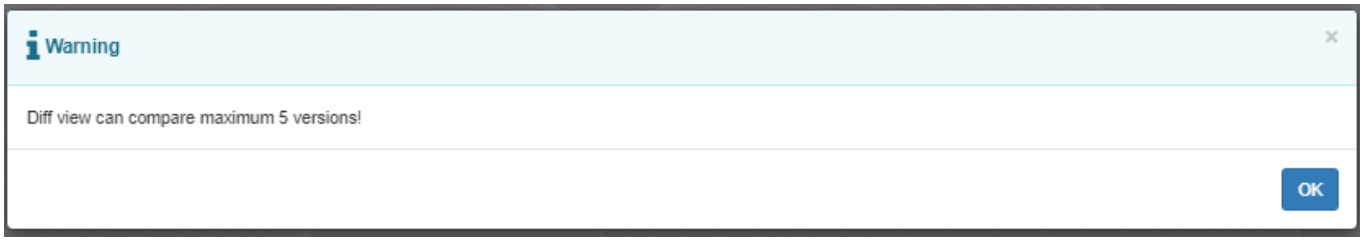
As soon as a new version is added for diffview, the spiral chart is updated immediately. Dependent on the number of executed test cases and browser rendering can need some time.

The dots which present the test cases along the spiral line (small dot is Passed and bigger one for other status) are interactable. It means that you can:

- movethe mouse over the dots to see the test case information.
- click on the failed test case (bigger dot) for the traceback information.

Notice:

You can only select maximum of **5** versions for diffview.
If the maximum of selected versions is reached and you click on the button to add more, a warning message will be displayed to prevent that action.



2.4 Developer guidance

Notice

In order to run up the TestResultWebApp, it requires some knowledge about:

- Web server: setup and run web server for web hosting.
- Nodejs platform and Express framework: adapt the sourcecode with your environemnt: domain, configurations, ...
- Mysql database: schema, tables, SQL scripting. We propose to use MySQL Workbench tool for working with Mysql database.

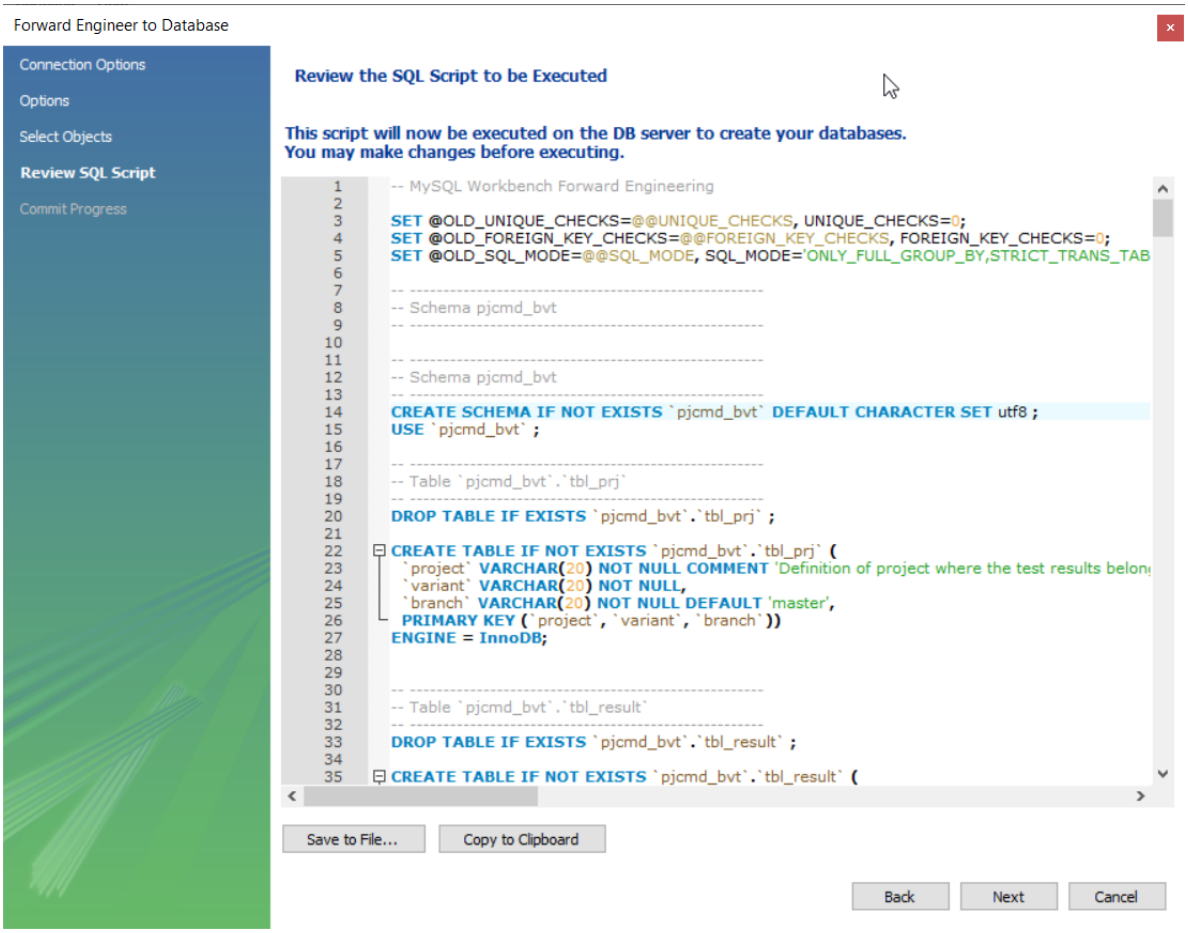
2.4.1 How to run new TestResultWebApp instance

1. Precondition:
   - Node.js shoulde be installed.
   - MySQL server should be installed.
   - A cloned package's resource from Github repo

     ```
     git clone https://github.com/test-fullautomation/testresultwebapp.git
     ```

   - The port which will run Node.js application (default is **3000**) is opened (enable) in firewall.
2. Setup mysql database:
   - create *user*, *password* and *schema* then update them as `global.mySQLOptions` in `webapp/web_server/lib/global.js` file.
   - create required tables in your database schema with MySQL Workbench.
     - open the datamodel which is located at `mysql_server/datamodel/test_result.mwb`
     - export all defined tables in datamodel to your schema, use **Foreward Enginneer to Database** feature of MySQL Workbench.

       ```
       EER Diagram > Database > Forward Engineer... > your schema
       ```



> **Notice**
>
> If you have created your schema with other name than the default name *pjcmd_bvt*, you should replace the schema name at the **Review SQL Script** step:
>
> * copy SQL script to text editor such as VsCode
> * replace *pjcmd_bvt* which your new schema name
> * paste SQL script back to the **Foreward Enginneer to Database** tool
>
> before moving to next step to execute SQL script.

– create all store procedures by loading all SQL scripts under `mysql_server/TMLdb_sproc/` folder then execute them.

> **Notice**
>
> If you have created your schema with other name than the default name *pjcmd_bvt*, you should replace the schema name before executing SQL scripts.

3. Import the initial data to the database with RobotResults2DB tool.
4. Adapt Web server:
   - `web_server/lib/global.js` : for domain, database's configuration, keys for authentication, ldap server for authentication, ...
   - `web_server/test.js` : for listening port of nodejs application.
5. Adapt Web client:
   - `web_client/dashboard/dist/js/common/global.js` : for domain.
   - `web_client/dashboard/dist/js/common/communication.js` : for listening port of Node.js application's API.
6. Start nodejs application by command:

```
node testdb.js
```

7. Start web server for hosting the static files:

   You can use any web server Apache, IIS or Nginx for hosting the static files under `web_client/dashboad/` folder when running as production.

   For development, you can use directly `express.static` which is supported by Express framework for hosting static files. The `web_server/test.js` file should be modified to add this setting.

```
'use strict';

var global = require('./lib/global');
var path = require('path');

...

//for local GUI tests deliver static HTML
//content from port 3000
app.use(express.static(path.join(__dirname, "../web_client/dashboad/")));

var session = require('express-session');
var MySQLStore = require('express-mysql-session')(session);
...
```

8. Now open your favourite browser, go to the domain of webapp and enjoy.

Chapter 3

Appendix

About this package:

Table 3.1: Package setup

Setup parameter	Value
Name	TestResultWebApp
Version	0.1.3
Date	18.10.2022
Description	Web based display of test results
Package URL	testresultwebapp
Author	Thomas Pollerspöck
Email	Thomas.Pollerspoeck@de.bosch.com
Language	Programming Language :: JavaScript
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 4

History

0.1.0	07/2022
Initial version	
0.1.1	09/2022
Update README file and package's document	
0.1.2	05/10/2022
Fix findings with package's document	

TestResultWebApp.pdf

Created at 18.10.2022 - 15:50:43

by GenPackageDoc v. 0.33.0