# **Katapult KT**

Katapult Support Team:

QAC related issues -> Kavita Mule

Polyspace related issues -> Vanshika Kalsaik

Phase 2 parameter related issues -> Ajinkya barve/Shivling Mane

Analyzer Personality File:

For all rule which will be analysis for QAC.

Compiler Personality File:

Message Personality File:

Message personality file use for showing the message while getting error or during execution.

Polyspace – Polyspace is a static code analysis tool for large-scale analysis by abstract interpretation to detect, or prove the absence of, certain run-time errors in source code for the C, C++, and Ada programming languages. **The tool also checks source code for adherence to appropriate code standards**.

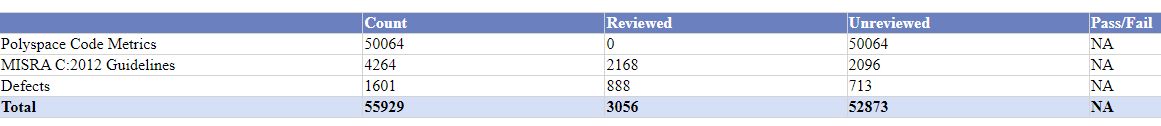
Bugfinder file

**Polyspace Bug Finder**

Report Author: auto.build auto.build is a user for katapult for interaction with git and katapault team

Polyspace Version(s): Polyspace Bug Finder 3.1 (R2019b Update 8) - R2019b is katapault polyspace version

Chapter 1. Polyspace Bug Finder Summary



In this chapter show the total violence and defect data

In unreviewed data this column shows how many data

**How to give justification:**

Chapter 5. Appendix 1 - Configuration Settings

* **Author** – author who trigger the polyspace Auto.build is user who read/ write data from katapault side
* **Bug-finder**- bug finder feature is on/off
* **Checkers**- which feature is enabled for polyspace (please ensure that your and customer feature should be same otherwise polyspace result will be different)
* **Code-metrics** – code metrics feature is on.
* **Compiler**- which compiler is use for compilation
* Dos-
* Target -

Polyspace developer.html

In developer polyspace is similar as bug-finder polyspace but one extra feature is Guidelines Violations show line by line.

How to give justification?

Polyspace runtime check-

***Difference b/w QAC and Polyspace***

**Polyspace** – **Code violation (Dynamic code check)**

* **Dynamic Code Analysis** is a method used to analyze an application during its execution. This Dynamic Code Analysis process is often broken up into these steps:

1. Preparing input data.
2. Running the program.
3. Gathering the necessary parameters.
4. Analyzing the output data.

* **Polyspace** examines the source code to determine where potential run-time errors such as arithmetic overflow, buffer overrun, division by zero, and others could occur
* Code standards or guidelines such as MISRA C attempt to address code quality, portability and reliability
* It annotates source code with a color-coding scheme to indicate the status of each element in the code:
  + **Green** indicates reliable code
  + **Red** indicates faulty code causing a run-time error
  + **Gray** indicates dead or unreachable code
  + **Orange** indicates unproven code
* **Defects** – **Compiler Warnings**

**QAC** – **Code violation (Static code check)**

**Static code analysis** is a method of debugging by examining source code before a program is run. It’s done by analyzing a set of code against a set (or multiple sets) of coding rules.

**QAC Report**

• Review the entirety of the code by project and section.

• Identify issues and deviations.

• Assign rule configurations (such as MISRA, AUTOSAR, and CERT).

• Verify how compliant the code is with coding standards and industry best practices.

• Measure overall code quality.

• Monitor developing trends with customizable reports.

In QAC report there will be 4 files QAC.html, QAC.json, Annotated report and QAC summary report

**QAC.html** – this file contains all summary of QAC

Rule Compliance –

**Number of file’s** - It show the number of files those pass through QAC

**Messages** – it shows the how many files break the QAC rules.

**Violation group** – there are 10 group based on priority LEVEL 0 is lowest priority

And level 9 is highest priority. We will consider only level 4 to level 9. Level 0 - level 3 can be ignore because it gives minor effect on QAC.

**Katapault**

* TD: technical debt
* ND: No data (When trigger are not happened)
* NC: Not configure
* NA: Not available

**CPD -** **(% of code duplication)**

CPD is a tool that finds out copy-pasted code in Source code. Duplicated source code blocks can harm the maintainability of software systems.

**UT & coverage – Unit test and coverage**

Path should be for UT & coverage

vc3\_ir2/Development/TestReports/BSW\_CDD\_PIL\_Reports/Reports/module wise xml file will be there/

*Reason* – Katapault read data from this location only for UT pass.

**CLOC** – **Percent Comments** **in code comment** loc (line of code)

In report show the percentage of comment in every file

**Tessy** **Code coverage (unit test /integration test)**

**Lizard** - **Complexity of most complex function**

Phase 2 parameter**:**

1. UT pass
2. % Coverage
3. SW Test pass %
4. % Requirement covered

***Why some module in pause mode?***

When modules are not in working then it put in pause mode.

Any module can be paused by only Delivery Manager.

**Full scope Katapault project:**

In this type of project run max file for code quality

**Example: BSW/Driver, BSW/GenData, CDD-wrapper/wrapper**

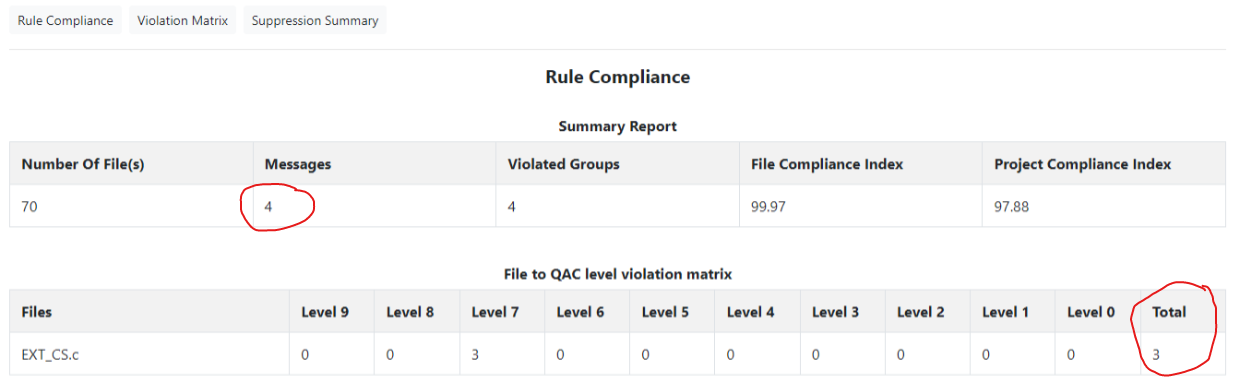
**Limited Scope Katapault project:**

In this type of project rum limited file for code quality.

**Example: CDD/wrapper**

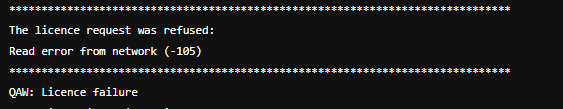
**Issues:**

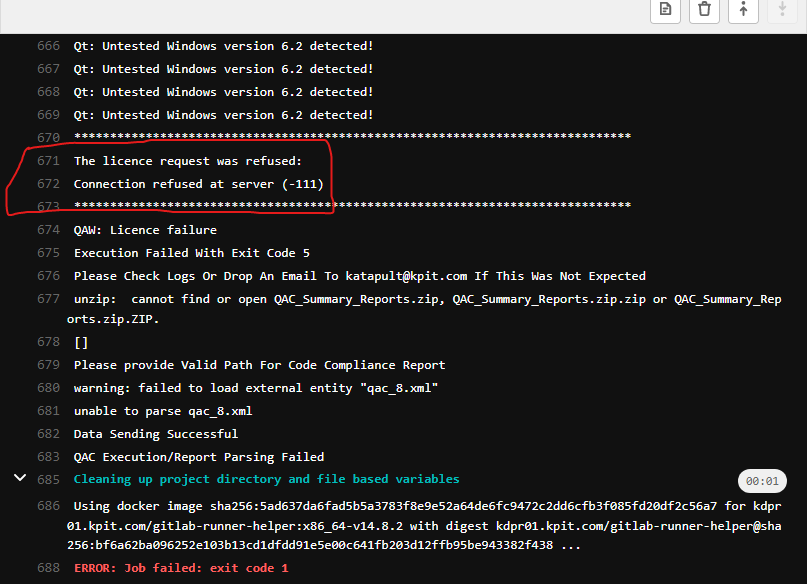
**1.**



This is QAC report error in this case we need to check QAC annotated file (EXT\_CS.c) and katapult dashboard.

**2. QAC license failure**





**Licence failure >>in this case first inform to katapult QAC member and write a mail for licence**

**3.**

1. When a product (electronic CIs and Physical CIs) is received from the customer, send the original product to the Project Configuration Manager, mentioning the Configuration ID, Product name, Version Number and Product type.
2. Graphical user interface, text

   Description automatically generated