Static code analysis in Robot Framework

Rules for Robocop

Tatu Alatalo

Bachelor’s thesis
May 2022
Information and Communications Technology

Description

Alatalo, Tatu

Static code analysis in Robot Framework, Rules for Robocop

Jyväskylä: JAMK University of Applied Sciences, May 2022, 42 pages.

Technology, Communication and Transport. Degree Programme in Information and Communications Technology. Bachelor's thesis.

Permission for web publication: Yes

Language of publication: English

Abstract

Static code analysis is becoming part of test automation as it can help to make the code easier to read, review and maintain. Having good practises while making code also makes the code to age slower, reduce bugs during implementation and increase reusability.

Task was commissioned by Qvantel Oy for the test automation team from a customer program as they were looking for a static code analysis tool called Robocop for Robot Framework. Objectives were to find out if Robocop is a suitable static code analysis tool for the test automation team. Secondly, how much work is needed to edit test suites and resource files if Robocop is taken into use. Finally, it should be found out what is the current coding standard in the test automation team.

Task was done using applied research strategy and blended research methods including quantitative, qualitative and action research. Three files were chosen during the task to test with Robocop, two test suites and one resource file. All three were ran through Robocop and all the unique issues that were raised by Robocop were noted and then discussed with the test automation team. During this "rules for Robocop" - meeting it was agreed how those issues should be handled and a configure file for Robocop was created from the meeting notes. It was also noticed that two custom rules are needed and those were created before running all three files again with the configure file that included modified built-in and custom rules.

Results of those both execution measurements were compared, and the results were that the issue amount was a lot less with the modified rules than with the original built-in rules. Even though there were less issues raised by Robocop and those are effortless to fix, it was also noted that there are excessive number of issues to be handled that takes quite a lot of time. Robocop tool was taken into use in the test automation team and created configure file was also started using as a coding standard. A user guide for how to install and use Robocop was also created as a side product of the task. In the future it is possible to integrate Robocop with a continuous integration pipeline so it can automatically review pull requests for issues.

Keywords/tags (subjects)

Automation, Static code analysis, Software development, Software testing, Test automation, Testing, Python, Robot Framework, Robocop, Quality assurance

Miscellaneous (Confidential information)

jamk

Kuvailulehti

Alatalo, Tatu

Staattinen koodianalysointi Robot Frameworkissa, Säännöt Robocopille

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2022, 42 sivua.

Tekniikan ala. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: englanti

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Staattisesta koodianalysoinnista on tulossa osa testiautomaatiota ja se voi tehdä koodista helpompaa lukea, katselmoida sekä ylläpitää. Hyvät käytänteet kirjoittaessa koodia myös hidastavat koodin vanhenemista, vähentävät vikoja sekä lisäävät koodin uudelleenkäytettävyyttä.

Työn tilaaja Qvantel Oy oli kiinnostunut staattisesta koodianalysointi-työkalusta nimeltä Robocop Robot Frameworkille, joka mahdollisesti voitaisiin ottaa käyttöön asiakasohjelman testiautomaatiotiimissä. Työn tavoitteena oli ottaa selvää, onko Robocop sopiva staattinen koodianalysointi-työkalu testiautomaatiotiimille, paljonko työtä tarvitaan testikokoelmien sekä resurssitiedostojen muokkaamiseen, jos Robocop otetaan käyttöön sekä ottamaan selvää mikä on testiautomaatiotiimin tämänhetkinen koodistandardi.

Työ toteutettiin käyttäen sovellettua tutkimusstrategiaa sekä monimenetelmällistä tutkimusta, joka koostui määrällisestä, laadullisesta ja kehittämistutkimusmenetelmästä. Kolme tiedostoa valittiin testattavaksi Robocopin kanssa, joista kaksi oli testisarjoja ja yksi resurssitiedosto. Kaikki kolme tiedostoa ajettiin Robocopin läpi ja yksittäiset ongelmat kerättiin yhteen. Näistä keskusteltiin testiautomaatiotiimin kanssa ”säännöt Robocopille” nimisessä tapaamisessa, jossa päätettiin kuinka nämä esiin nostetut ongelmat tullaan käsittelemään. Tapaamisen lopputuloksena päätöksistä luotiin konfiguraatiotiedosto, jossa muokattiin Robocopin sisäänrakennettuja sääntöjä. Huomattiin myös, että sen lisäksi tarvittiin luoda kaksi yksilöllistä sääntöä. Lopuksi nämä tiedostot ajettiin uudestaan Robocopin kanssa, joka käytti luotua konfiguraatiotiedostoa sisältäen muokatut sisäänrakennetut ja yksilölliset säännöt.

Molempien mittauskertojen tuloksia verrattiin keskenään ja tulokset osoittivat, että ongelmia oli paljon vähemmän mukautettujen ja yksilöllisten sääntöjen kanssa kuin alkuperäisten sisäänrakennettujen sääntöjen kanssa. Vaikka Robocopin ilmoittamia ongelmia oli vähemmän ja ne voivat olla suhteellisen helppoja korjata menee kuitenkin tiimin kaikkien testisarjojen ja resurssitiedostojen muokkaamiseen paljon aikaa. Robocop otettiin käyttöön ja konfiguraatiotiedostoa alettiin käyttämään koodistandardina. Sivutuotteena syntyi myös käyttöohje Robocopin käyttöönottoa varten. Tulevaisuudessa Robocop on mahdollista integroida automatisoituun julkaisuputkeen, jossa se voi automaattisesti katselmoida sääntörikkomuksia vetopyynnöstä.

Avainsanat (asiasanat)

Automaatio, Laadunvarmistus, Ohjelmistotuotanto, Ohjelmistotestaus, Python, Robot Framework, Robocop, Staattinen koodianalysointi, Testiautomaatio, Testaus

Muut tiedot (salassa pidettävät liitteet)

Contents

Figures

1 Introduction

1.1 Research

Coding standards and static code analysis are a key part of software development as that makes code easier to read, review and maintain by people on all levels. Having good practises while making code makes the aging of the code slower, reduces bugs during implementation and increases reusability. As static code analysis tools have been developed for test automation frameworks, practises have been also taken in use in test automation.

The subject for this thesis was commissioned by Qvantel Oy as the test automation team in the customer program are looking for a possibility to take in use a static code analysis tool called Robocop for Robot Framework. There is also a personal interest in this subject as these kinds of practises may bring the test automation in general more in line with coding in software development. It is also noticed that there are none or few existing research about this subject, at least regarding static code analysis in Robot Framework. It should also be acknowledged that the writer of this thesis is also working as a test automation engineer in the same test automation team.

Benefits of taking in use a tool for checking test suites and resource files would be that it gives more time to focus on the test or a keyword itself instead of using time to verify if there is a correct number of empty lines between test cases or keywords. Test automation team has a coding standard, but it is not fully documented. It was also discussed what would be the impact for the existing files if static code analysis tool with rules would be taken in use.

1.2 Qvantel Oy

Qvantel is an IT service company that offers customised business support systems for telecommunication service providers. Currently there are 501-1000 people working in Qvantel and 672 are in LinkedIn. (Qvantel LinkedIn 2022.)

Qvantel is operating in 23 different countries and its headquarters is in Finland. Customers that use Qvantel services are Very Mobile, DNA, XL Axiata, European Telco Group, Millicon Group, Altan Redes, MasMovil and WindTre. (Qvantel 2022.)

In Techopedia's article named "Business support system" it is listed that there are eight key BSS areas in business support systems, which are: product management, customer management, revenue management, customer order management, customer data management, billing and rating, business-to-business (B2B) and business-to-consumer (B2C) services. BSS is defined as follows: "Business support system (BSS) helps telecommunication service providers support and extend operations to enhance business services." (Business support system 2012.) In short, Techopedia's article states most of the areas that telecom provider needs to run everyday business.

1.3 Research questions

This research aims to answer into following research questions

- Is Robocop suitable static code analysis tool for the test automation team in customer program?
- How much work is needed to edit test suites and resource files if Robocop is taken in to use?
- What is the coding standard in the test automation team?

1.4 Research strategy and methods

How to answer these research questions is then required to choose a suitable research strategy and a method or methods. Hirsjärvi, Remes and Sajavaara (2013, 132) defines term research strategy as a set of methodological solutions and term research method should be distinguished as a narrower concept from it (Hirsjärvi, Remes & Sajavaara 2013, 132).

Hirsjärvi et al. paraphrases Robson (1995) to describe the main difference between a research strategy and a research method with an example analogy called "crossing the river". Research strategy is related to the decision should the river be crossed by swimming, walking across the bridge, flying or sailing over it and research method applies to the type between a boat, a bridge or a plane. (Ibid.)

They also freely reference Robson (1995) and compares main features between applied research and experimental research as a research strategy. In applied research the aim is to solve practical problems and predict impact where in experimental research the aim is to gather information and find the cause. Also, applied research is used to improve and test programs and services alike where in experimental research it is used to improve and test theories. Applied research is done as field research where in experimental research is done as laboratory research. It is also noted that in applied research the research itself is commissioned by an external organisation, such as a company, where the researcher itself is an expert in the researched field. (Hirsjärvi, Remes & Sajavaara 2013, 133).

Suitable research strategy for this research is then applied research as in this research there is a problem to be solved, predict impact, and the aim is to improve a process. Research is also done out there in the field and is commissioned by a private company where the researcher and writer of this thesis also works as a test automation engineer.

Qualitative research aims to get understanding of the subject and what it is about by asking questions. These questions give understanding of the subject to the researcher, and it raises more questions about the researched subject. These answers give a full structure, factors and connections related to the subject. Qualitative research results are words and sentences when again quantitative research uses numbers. (Kananen 2015, 34-35)

Quantitative research uses numbers to get results from the research and it is used when there are multiple units to be researched. Data analyse methods are used to get different statistics and distributions from the data what that then represents the structure of the subject in a summary. Quantitative research cannot be used if the subject is not known enough for the researcher. (Kananen 2015, 38)

Blended research, also known as mixed or multi-research, does not have its own methods but is a mixture of both qualitative and quantitative research methods. One of these is called action research where the aim is to get a change as a result to the problem through action. Ways to solve the problem is to improve a product, processes or the organization itself. Companies do improvements themselves too, but it should not be compared to action research as it requires research approach to the problem. (Kananen 2015, 39-40).

Kananen (2015, 43) states that the research itself is regularly involved in the change process making, or intervention. He also expresses (2015, 45) that the change should also be proved, and this is done by comparing before and after measurements. To achieve the change, and intervention is done between the measurements. Analysis can be done with anything that is quantitative. (Kananen 2015, 43-45).

This thesis will be using blended research that is a mix of qualitative, quantitative and action research as there is no one method that covers all the defined research questions. Research will be using action research to compare the results of Robocop, a qualitative method to find out the current coding standard and a quantitative method to handle the Robocop results.

2 Theoretical basis

2.1 Software testing

IBM's article "What is software testing?" gives excellent information what is software testing and there it is described that software testing commonly as the process of verifying and evaluating that an application or a software does what it is fit for use. The benefits of it are that it prevents bugs, improves performance, and reduces the development costs. (What is software testing? N.d.)

Same article also informs that since after the second world war the main method of the software testing was debugging of the software code but in the 1980s development teams started testing applications in a real-world setting. That created a quality assurance process that has been part of the software development life cycle since. IBM's article quotes Yaroshko (N.d.) from his post to the uTest developer site that in the 1990s quality assurance started to take shape and started covering the whole software development cycle. It started the process of planning, designing, creating and execution of test cases and that led to development of tools for managing testing processes and test automation. (What is software testing? N.d.)

Back then software testing was separated from the rest of the development but nowadays development teams use continuous testing where testing is executed while the development of the software is ongoing. This helps to detect bugs earlier in the process where they are easier and cheaper to fix. (Ibid.)

IBM’s article states that there are multiple different software test types such as acceptance testing where it is verified that the whole system is working as intended. Integration testing where it is ensured that the software components or functions are operating together. Unit testing where the smallest testable components of an application are performing as they should. Functional testing where testing is executed by emulating specific business scenarios based on functional requirements. Software performance is tested under load in a performance testing and regression testing is used to test that the new features do not break any previously working functionality. (Ibid.)

Perfecto’s article “Different types of testing in software” adds that there are also more types of testing such as non-functional testing that is used to verify that the software is ready according to non-functional parameters such as performance, accessibility and UX. Accessibility testing where it is ensured that the application is working correctly for users with disabilities. Black box testing where software code and paths are invisible. End to end testing where it is tested that the workflow of the software works from the beginning to the end. Security testing where software is tested against security vulnerabilities. Smoke testing is used to validate the stability of the software and is used on the initial software build to verify that critical functions of the application are working as expected. (Different types of testing in software 2021.)

2.2 Test automation

Software testing can be divided into two categories, manual and automated testing. In manual testing a human tester tests the Software Under Testing (SUT) with multiple input conditions and then compares the results against expected results. If results differ then tester comments and documents the quality of the software. Manual testing can be either scripted or exploratory. In scripted testing tests follow a strict test case and follow it step by step. In exploratory testing the tester tests by discovering and learning just by interacting with the application. (Kumar & Mishra 2016, 9)

Kumar & Mishra (2016) articulates that in automated testing automated tools are used to perform tests on the SUT. This requires writing of code or scripts that will identify bugs by using specified input conditions and comparing the results against expected results. This way automated testing is a suitable approach to replace time consuming manual testing, have consistent coverage, to avoid human errors, and speed up the testing process. (Ibid.)

However, Martin (2017) emphasizes that automated testing does not replace manual testing but is used to assist. Automation is great for setting up the environments and taking them down, data entries, form filling, varying data inputs in a repetitive process, backend testing, repetitive and boring tasks that are prone to human errors, tasks that have high reuse value across many work-flows, non-functional test types e.g., performance testing. (Martin 2017.)

Hamilton (2022) advises that some test cases are not suitable for automation. Those are newly designed test cases that are executed for the first time manually, test cases that are frequently changing and test cases that are executed on that purpose basis. (Hamilton 2022.)

In EuroSTAR Huddle article user Anmol (2021) declares that automated testing can be divided into three categories: (1) automation testing based on the types of testing such as functional and non-functional testing; (2) automation testing based on the different phases of testing such as unit testing, API testing and UI testing; and (3) automation testing based on the types of tests such as smoke testing, integration testing, regression testing, security testing, performance testing and acceptance testing. (Anmol 2021.)

Automated testing process is following 5 steps (see Figure 1). In the first step the test tool is being selected and usually it highly depends on the technology that the software is made of. After that the scope of the automation is defined, usually focusing on areas that have a large amount of data, have common functionalities, and have complex test cases. Planning, design, and development comes next, and it is where the test script is planned and implemented. After that, the test is executed by the tester. Finally, the automated test is kept maintained especially when new functionalities are added to the application. (Hamilton 2022.)

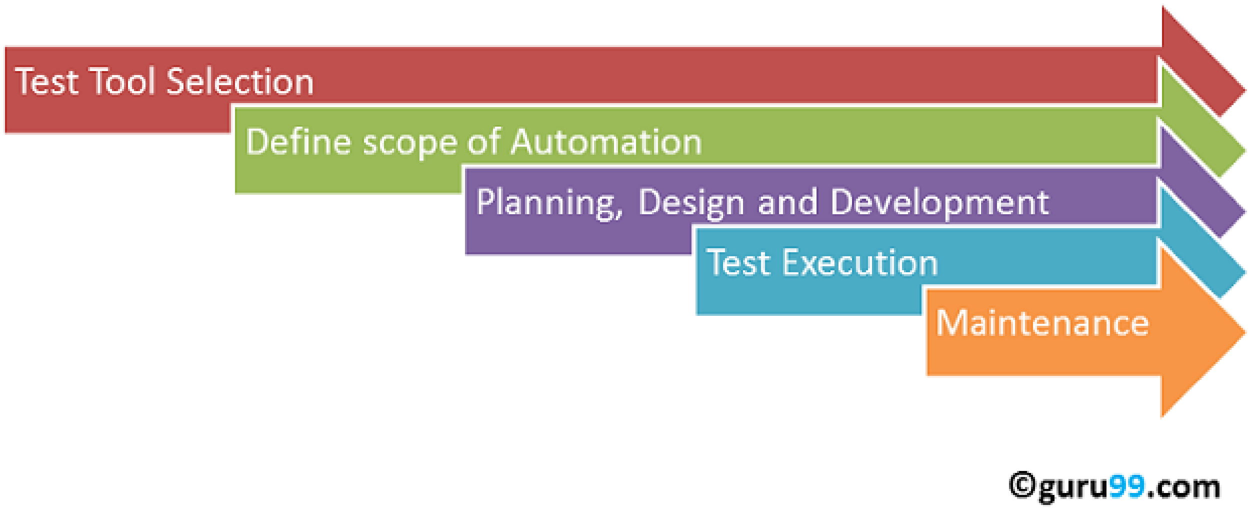


Figure 1. Test Automation Process (Hamilton 2022)

2.3 Static code analysis

Bellairs (2020) states that static code analysis is a debugging method where source code is examined automatically or manually before a program is run. Analysing is done against a set or various sets of coding rules. Static code analysis is the same as static analysis or source code analysis but not to be confused with dynamic analysis where code is examined while executing the program. (Bellairs 2020.)

On the gains of static code analysis Bellairs (2020) comments that it contends with weaknesses of the source code that would lead to vulnerabilities and complies the code with coding standards. Static code analysis is also often faster than dynamic code analysis since a program is not executed and therefore it identifies defects before a program has been run. (Ibid.)

Static code analyser tools allow analysing source code automatically instead of manually which is more effective. Bellairs (2020) mentions that manual code reviews have tendency to human errors and automated tools do not. Although automated tools check with specific rules that are self-made or defined by standards. (Ibid.)

Static code analysis also improves the quality of the code and automates code quality maintenance. Luminousmen’s article “Python static analysis tools” also states that there are seven different types of static analysis, which are: code styling analysis, security linting, error detection, UML diagram detection, complexity analysis, comment styling analysis and unused code detection. Most of the IDE (an integrated development environment) programs use specific tools automatically depending on the IDE and the coding language. For example, PyCharm uses PEP8 style guide for python code. (Python static analysis tools 2021.)

2.4 Robot Framework

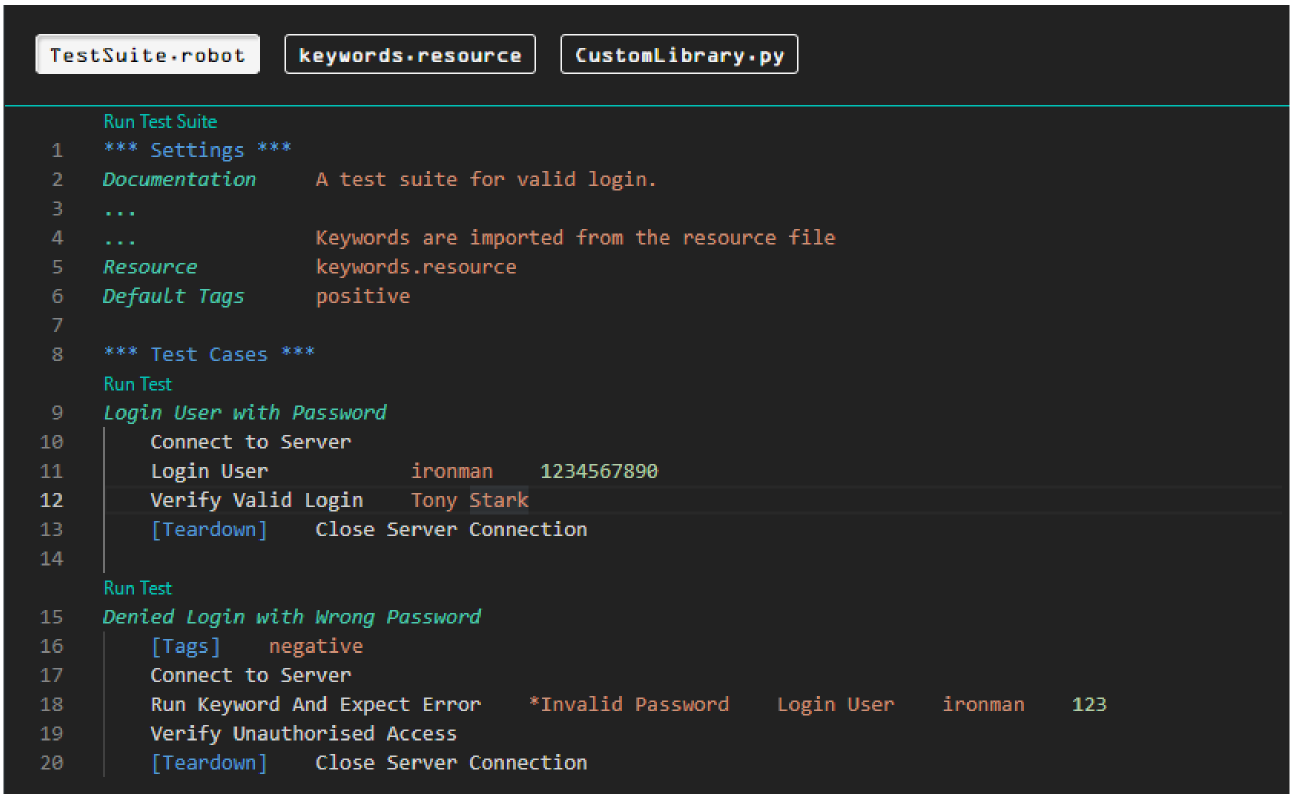
Robot Framework is a keyword-driven automation framework for test automation and robot process automation (RPA). It’s based on Python but also supports Java (Jython) and .NET (IronPython) if the Robot Framework version is 4.1.3 or older. Robot Framework has built in libraries, and it also supports user and community made libraries, such as SeleniumLibrary which is a web testing library. (Robot Framework User guide 2021.)

Prototype for Robot Framework was done by Pekka Klärck when he was doing his master’s thesis in 2004 and the first version was developed in 2005 when he was working in a customer project in Nokia Networks. Robot Framework was open sourced in 2008 when 2.0 version came out and its copyrights were owned by Nokia Network who also did the sponsoring. In 2015 direct Nokia sponsoring ended and Robot Framework Foundation has been doing the sponsoring and development since then. (Klärck 2016.)

According to Klärck (2016), Robot Framework has become the de facto standard test automation framework in Finland and companies that use it include like Nokia, Kone, Metso, Vaisala, Finnish Centre for Pensions and ZEF (Klärck 2016). Robot Frameworks homepage also lists other companies (not just from Finland) like Finnish Tax Administration, ABB, Axon, Cisco, and Finnair along with others (Robot Framework Users). Although, Vaisala, Finnish Centre for Pensions and ZEF are not listed there.

Keyword-driven testing makes test suites structured and readable. Keywords can be divided into three types: (1) Higher-level keywords that are used to test a certain part of the SUT; (2) lower-level keywords that keep test cases minimal and succinct; and (3) technical keywords that are used to access to the SUT and run the tests. (Robot Framework – Test automation the smart way N.d.)

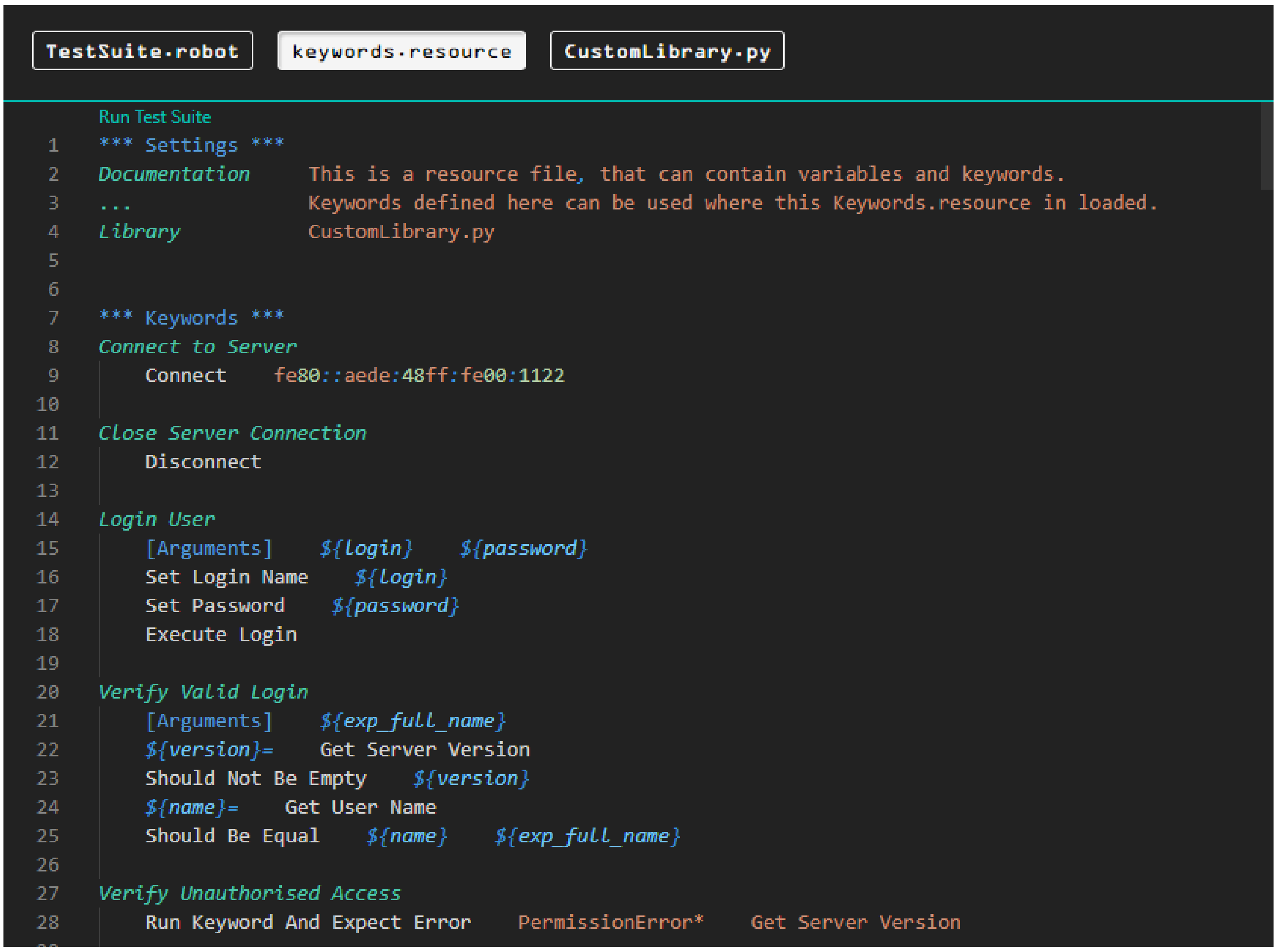
Robot Framework homepage contains an example of a robot test suite where Robot Framework syntax can be seen (see Figure 2). In the test suite there are two test cases where it is tested that the user can login to the website with a correct password and another case where the user tries to login with an invalid password that is expected to fail. (Robot Framework – Simple example N.d.)



```
*** Settings ***
Documentation     A test suite for valid login.
...
...               Keywords are imported from the resource file
Resource          keywords.resource
Default Tags      positive

*** Test Cases ***
Login User with Password
    Connect to Server
    Login User            ironman     1234567890
    Verify Valid Login    Tony Stark
    [Teardown]    Close Server Connection

Denied Login with Wrong Password
    [Tags]    negative
    Connect to Server
    Run Keyword And Expect Error    *Invalid Password    Login User    ironman    123
    Verify Unauthorised Access
    [Teardown]    Close Server Connection
```

Figure 2. Example test suite (Robot Framework - Simple example N.d.)

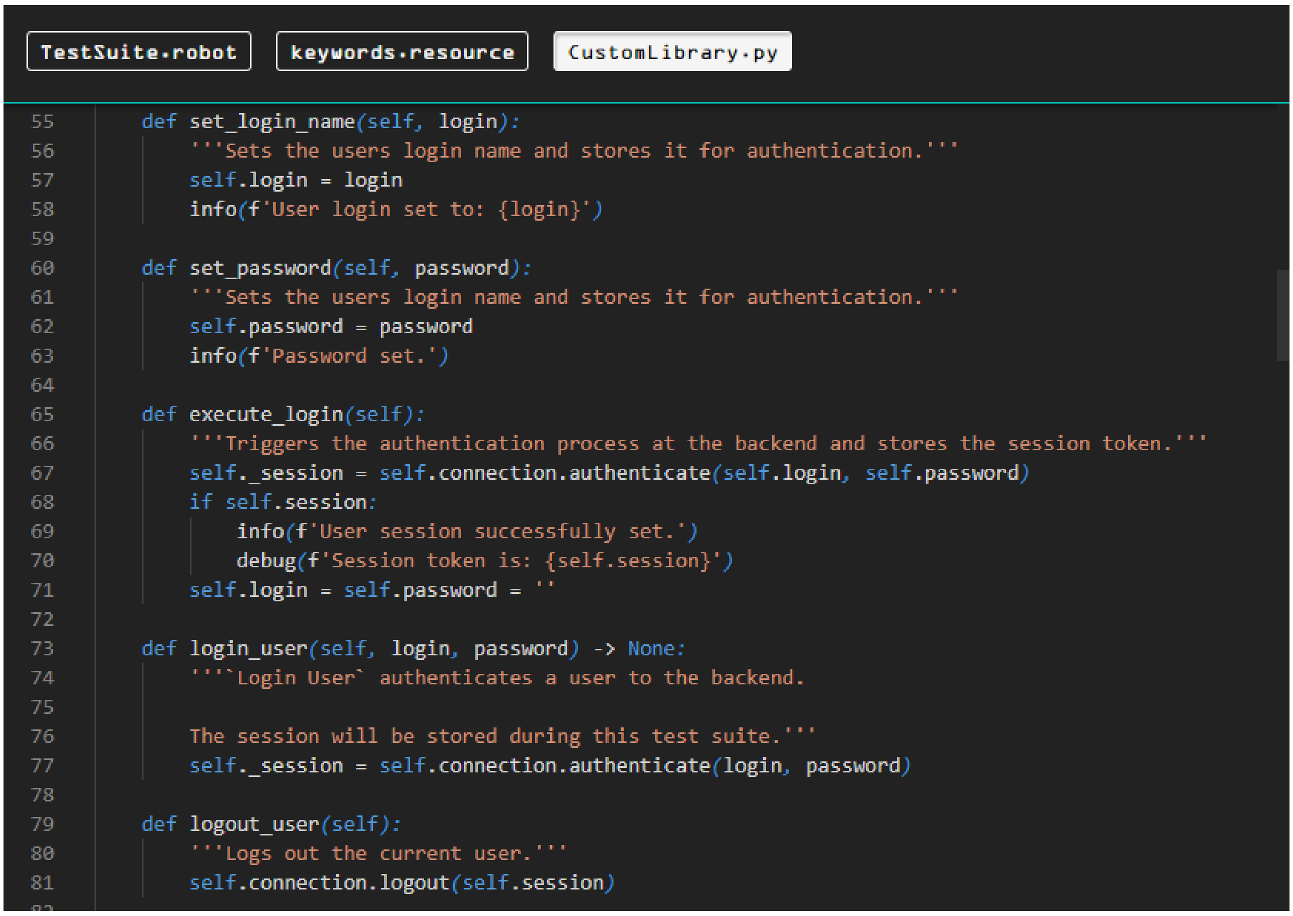
Test cases use keywords that are defined in a resource file. Here are technical and higher-level keywords such as “Connect to Server” and “Login User” (see Figure 3). Keywords also take in variables as an argument e.g., login name and password. (Ibid.)



```
*** Settings ***
Documentation     This is a resource file, that can contain variables and keywords.
...               Keywords defined here can be used where this Keywords.resource in loaded.
Library           CustomLibrary.py


*** Keywords ***
Connect to Server
    Connect    fe80::aede:48ff:fe00:1122

Close Server Connection
    Disconnect

Login User
    [Arguments]    ${login}    ${password}
    Set Login Name    ${login}
    Set Password    ${password}
    Execute Login

Verify Valid Login
    [Arguments]    ${exp_full_name}
    ${version}=    Get Server Version
    Should Not Be Empty    ${version}
    ${name}=    Get User Name
    Should Be Equal    ${name}    ${exp_full_name}

Verify Unauthorised Access
    Run Keyword And Expect Error    PermissionError*    Get Server Version
```

Figure 3. Example resource file (Robot Framework - Simple example N.d.)

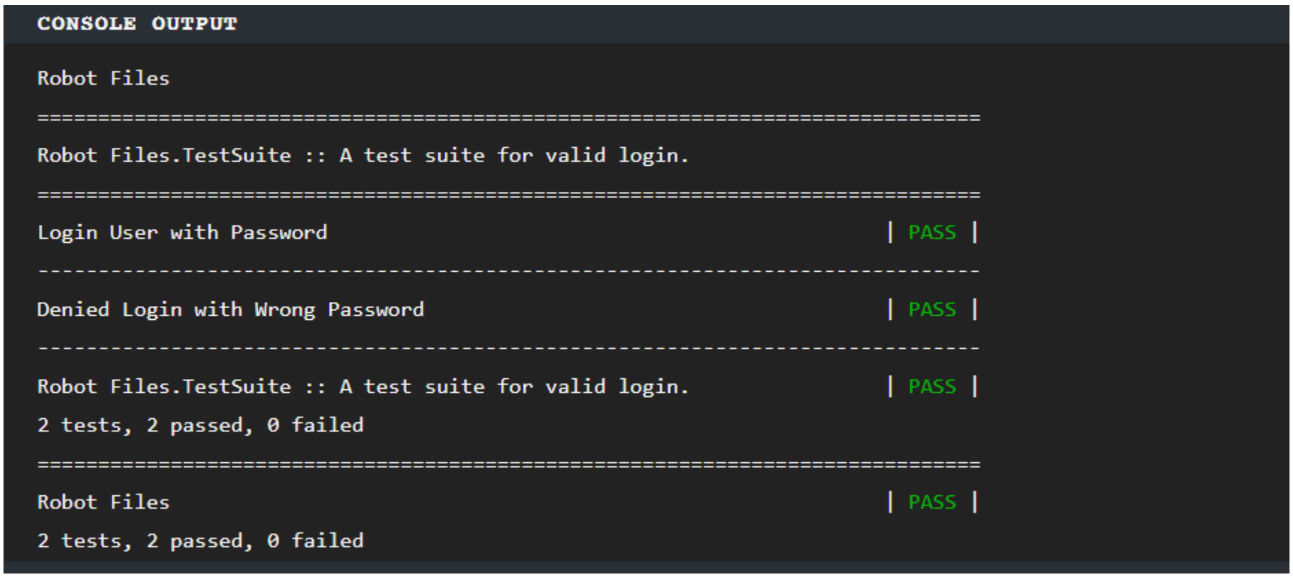
CustomLibrary.py contains lower-level keywords that are used in higher-level keywords (see Figure 4). In this example those are python-based keywords that are defined for more complex and specialized functionality. For example, “Login User” higher-level keyword uses the “Execute Login” lower-level keyword that is defined in this file. (Ibid.)



```python
    def set_login_name(self, login):
        '''Sets the users login name and stores it for authentication.'''
        self.login = login
        info(f'User login set to: {login}')

    def set_password(self, password):
        '''Sets the users login name and stores it for authentication.'''
        self.password = password
        info(f'Password set.')

    def execute_login(self):
        '''Triggers the authentication process at the backend and stores the session token.'''
        self._session = self.connection.authenticate(self.login, self.password)
        if self.session:
            info(f'User session successfully set.')
            debug(f'Session token is: {self.session}')
        self.login = self.password = ''

    def login_user(self, login, password) -> None:
        '''`Login User` authenticates a user to the backend.

        The session will be stored during this test suite.'''
        self._session = self.connection.authenticate(login, password)

    def logout_user(self):
        '''Logs out the current user.'''
        self.connection.logout(self.session)
```

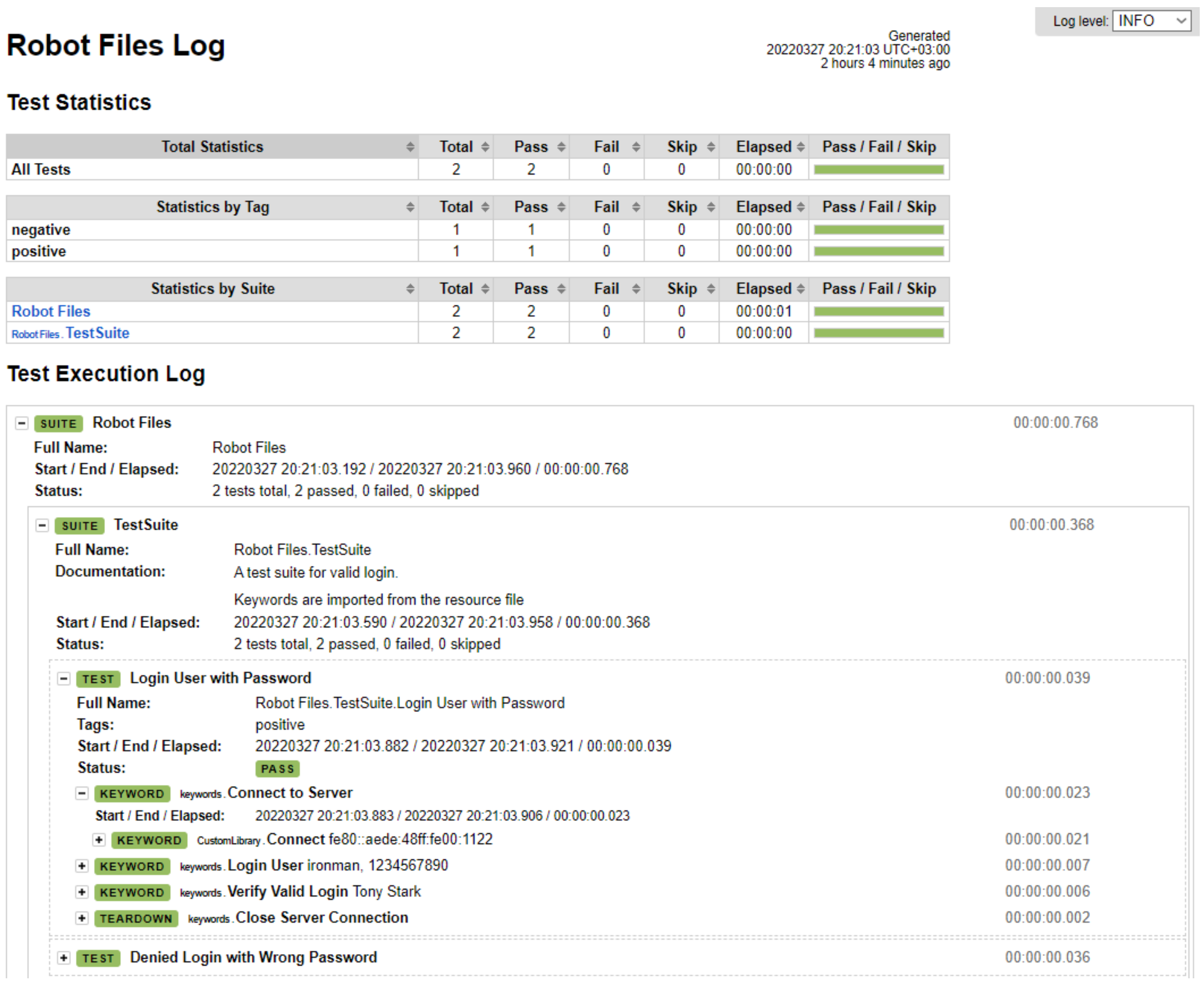
Figure 4. Lower-level keywords in CustomLibrary.py (Robot Framework - Simple example N.d.)

When executing the example test suite Robot Framework prints progress to the console (see Figure 5) where can be seen if the test case is PASS or FAIL. Finally, when the test suite is completely executed the results can be viewed from the log (see Figure 6) where results can be checked through keyword level. (Ibid.)



```
CONSOLE OUTPUT

Robot Files
==============================================================================
Robot Files.TestSuite :: A test suite for valid login.
==============================================================================
Login User with Password                                              | PASS |
------------------------------------------------------------------------------
Denied Login with Wrong Password                                      | PASS |
------------------------------------------------------------------------------
Robot Files.TestSuite :: A test suite for valid login.                | PASS |
2 tests, 2 passed, 0 failed
==============================================================================
Robot Files                                                           | PASS |
2 tests, 2 passed, 0 failed
```

Figure 5. Console output (Robot Framework - Simple example N.d.)



Robot Files Log

Generated
20220327 20:21:03 UTC+03:00
2 hours 4 minutes ago

Log level: INFO

Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	2	2	0	0	00:00:00	🟩

Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
negative	1	1	0	0	00:00:00	🟩
positive	1	1	0	0	00:00:00	🟩

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Robot Files	2	2	0	0	00:00:01	🟩
Robot Files . TestSuite	2	2	0	0	00:00:00	🟩

Test Execution Log

```
− SUITE Robot Files                                                          00:00:00.768
    Full Name:            Robot Files
    Start / End / Elapsed: 20220327 20:21:03.192 / 20220327 20:21:03.960 / 00:00:00.768
    Status:               2 tests total, 2 passed, 0 failed, 0 skipped
  − SUITE TestSuite                                                          00:00:00.368
      Full Name:            Robot Files.TestSuite
      Documentation:        A test suite for valid login.

                            Keywords are imported from the resource file
      Start / End / Elapsed: 20220327 20:21:03.590 / 20220327 20:21:03.958 / 00:00:00.368
      Status:               2 tests total, 2 passed, 0 failed, 0 skipped
    − TEST Login User with Password                                          00:00:00.039
        Full Name:            Robot Files.TestSuite.Login User with Password
        Tags:                 positive
        Start / End / Elapsed: 20220327 20:21:03.882 / 20220327 20:21:03.921 / 00:00:00.039
        Status:               PASS
      − KEYWORD keywords.Connect to Server                                   00:00:00.023
          Start / End / Elapsed: 20220327 20:21:03.883 / 20220327 20:21:03.906 / 00:00:00.023
        + KEYWORD CustomLibrary.Connect fe80::aede:48ff:fe00:1122            00:00:00.021
      + KEYWORD keywords.Login User ironman, 1234567890                      00:00:00.007
      + KEYWORD keywords.Verify Valid Login Tony Stark                       00:00:00.006
      + TEARDOWN keywords.Close Server Connection                            00:00:00.002
    + TEST Denied Login with Wrong Password                                  00:00:00.036
```

Figure 6. Test Execution Log (Robot Framework - Simple example N.d.)

3 Assignment

3.1 Robocop

Robocop is a linter that performs static code analysis to Robot Framework code. First version of it was released in September 2020. It has been developed around the same values as the cybernetic police officer from the movie RoboCop who had 3 prime directives: serve the public trust, protect the innocent and uphold the law. Robocop tool serves the developers and testers so they can create applications that they are able to trust. Testers and developers can produce unacceptable code and Robocop protects them from that. If developers and testers follow the guidelines from Robocop they can keep the code in order, comprehensible and logical and thus upholding the law. (Robocop repository 2022.)

In Robocon 2021 conference Nojek Mateusz stated that Robocop uses Robot Framework parsing API to parse files, and it checks the code for violations of the code quality standards or potential errors. Rules are used as a code quality standard and those can be modified with configuration file by the user if they use different standards. Built-in rules can also be extended by creating new custom rules. Nojek also mentioned that Robocop reports can be redirected to a different location and can use different output formats. It can also be set to filter rules out by severity level and is able to ignore specific files or patterns. (Hirsz, B., Nojek, M. 2021.)

To interpret how the Robocop tool is used, a flowchart was created (see Figure 7). Robocop starts to examine the file with the rules that it has been given and checks the line of code for issues. If there is an issue, it is then added to the report. Same steps are then repeated to verify that the current line does not have any issues before continuing to the next line. If there are no longer lines to be examined then the report is complete and the test automation person can read the report.

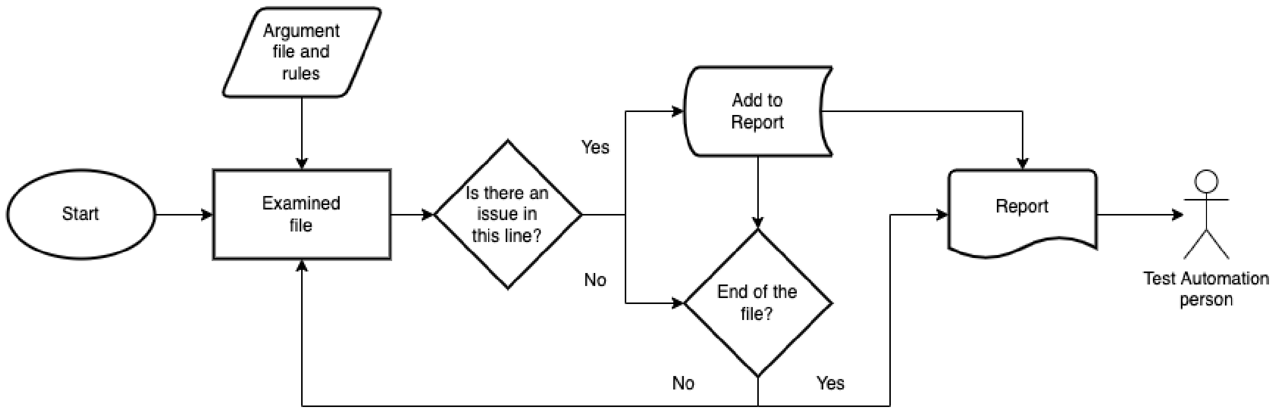


Figure 7. Robocop flowchart

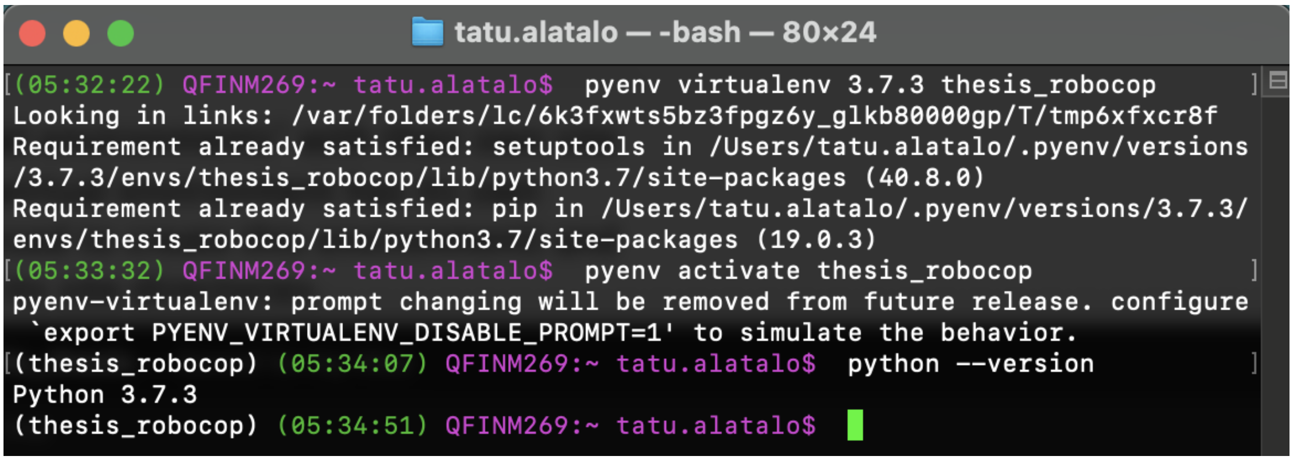
3.2 Alternative linter

Before starting assignment with Robocop, it is necessary to look for similar static code analysis tools, also known as linters, that are intended to be used with Robot Framework. There is at least one known tool called Robot Framework linter (rflint) that was first released in October 2014 by Bryan Oakley and current version 1.1 was released in June 2020. Like Robocop, it contains built-in rules and some of those are configurable e.g., “LineTooLong” that verifies that line of code is within character limit. It also supports argument file where rules can be listed by what user wants to e.g., disable some rules, configure a set of rules, or create different argument files for different robot-files. User can also create custom rules with simple python classes. (Rflint repository 2019.)

Main differences between Robocop and Rflint is that Robocop comes with more built-in rules, can be integrated to IDE tools, uses official Robot Framework Parsing API, can redirect output to a file and claims to be easier to extend with new rules (Robocop repository 2022). Rflint was updated last time in June 2020 (Rflint repository 2019) and Robocop in March 2022 (Robocop repository 2022). Therefore, assignment continues with Robocop as it seems to be a more versatile out of the box solution for this assignment.

3.3 Installing Robocop

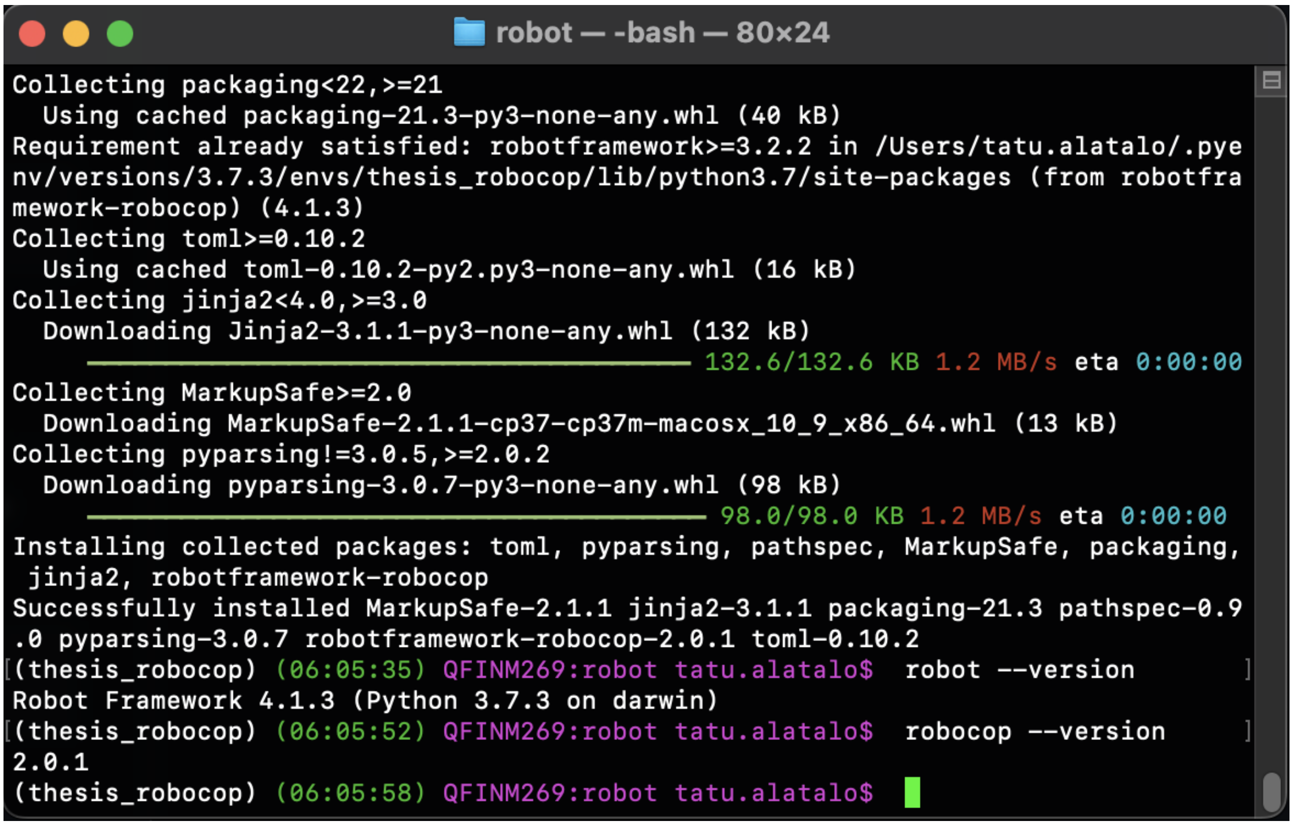
Before installing Robocop, a new virtual environment was needed. Virtual environment (venv) is a Python environment where Python interpreter, libraries and scripts are installed virtually but are isolated from other virtual environments that may have different Python versions, libraries and scripts (Venv 2022). Test automation team in the Qvantel customer program uses pyenv for handling virtual environments and python version 3.7.3, so a new virtual environment named thesis_robocop was created (see Figure 8).



```
(05:32:22) QFINM269:~ tatu.alatalo$  pyenv virtualenv 3.7.3 thesis_robocop
Looking in links: /var/folders/lc/6k3fxwts5bz3fpgz6y_glkb80000gp/T/tmp6xfxcr8f
Requirement already satisfied: setuptools in /Users/tatu.alatalo/.pyenv/versions
/3.7.3/envs/thesis_robocop/lib/python3.7/site-packages (40.8.0)
Requirement already satisfied: pip in /Users/tatu.alatalo/.pyenv/versions/3.7.3/
envs/thesis_robocop/lib/python3.7/site-packages (19.0.3)
(05:33:32) QFINM269:~ tatu.alatalo$  pyenv activate thesis_robocop
pyenv-virtualenv: prompt changing will be removed from future release. configure
 `export PYENV_VIRTUALENV_DISABLE_PROMPT=1' to simulate the behavior.
(thesis_robocop) (05:34:07) QFINM269:~ tatu.alatalo$  python --version
Python 3.7.3
(thesis_robocop) (05:34:51) QFINM269:~ tatu.alatalo$
```

Figure 8. Setting up virtual environment

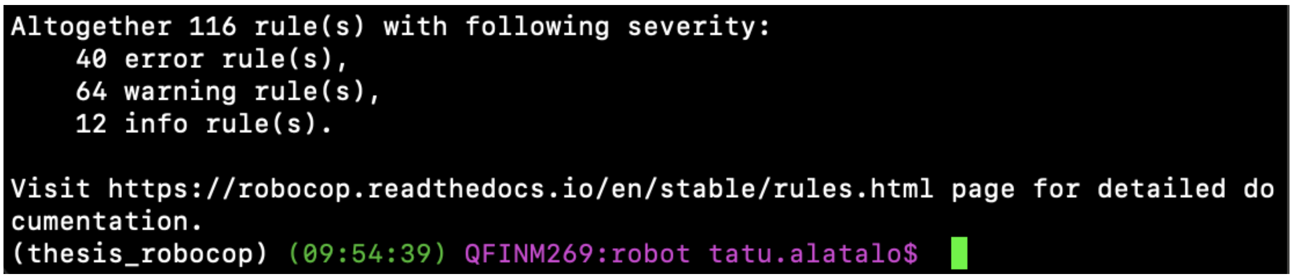
After the virtual environment was created, it was necessary to install required libraries such as robot framework, selenium library and others required to run robot framework tests correctly. Those were installed using requirements.pip file that is used in the customer program and can be found from their repository. Following Robocop documentation from their github-page, robocop was installed afterwards (see Figure 9).



```
Collecting packaging<22,>=21
  Using cached packaging-21.3-py3-none-any.whl (40 kB)
Requirement already satisfied: robotframework>=3.2.2 in /Users/tatu.alatalo/.pye
nv/versions/3.7.3/envs/thesis_robocop/lib/python3.7/site-packages (from robotfra
mework-robocop) (4.1.3)
Collecting toml>=0.10.2
  Using cached toml-0.10.2-py2.py3-none-any.whl (16 kB)
Collecting jinja2<4.0,>=3.0
  Downloading Jinja2-3.1.1-py3-none-any.whl (132 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 132.6/132.6 KB 1.2 MB/s eta 0:00:00
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.1.1-cp37-cp37m-macosx_10_9_x86_64.whl (13 kB)
Collecting pyparsing!=3.0.5,>=2.0.2
  Downloading pyparsing-3.0.7-py3-none-any.whl (98 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.0/98.0 KB 1.2 MB/s eta 0:00:00
Installing collected packages: toml, pyparsing, pathspec, MarkupSafe, packaging,
 jinja2, robotframework-robocop
Successfully installed MarkupSafe-2.1.1 jinja2-3.1.1 packaging-21.3 pathspec-0.9
.0 pyparsing-3.0.7 robotframework-robocop-2.0.1 toml-0.10.2
(thesis_robocop) (06:05:35) QFINM269:robot tatu.alatalo$  robot --version
Robot Framework 4.1.3 (Python 3.7.3 on darwin)
(thesis_robocop) (06:05:52) QFINM269:robot tatu.alatalo$  robocop --version
2.0.1
(thesis_robocop) (06:05:58) QFINM269:robot tatu.alatalo$
```

Figure 9. Installing Robocop

By default, Robocop has a set of built-in rules that are active. Amount of those rules are 116 which consists of 40 error rules, 64 warning rules and 12 info rules (see Figure 10).



```
Altogether 116 rule(s) with following severity:
    40 error rule(s),
    64 warning rule(s),
    12 info rule(s).

Visit https://robocop.readthedocs.io/en/stable/rules.html page for detailed do
cumentation.
(thesis_robocop) (09:54:39) QFINM269:robot tatu.alatalo$
```

Figure 10. Amount of built-in rules

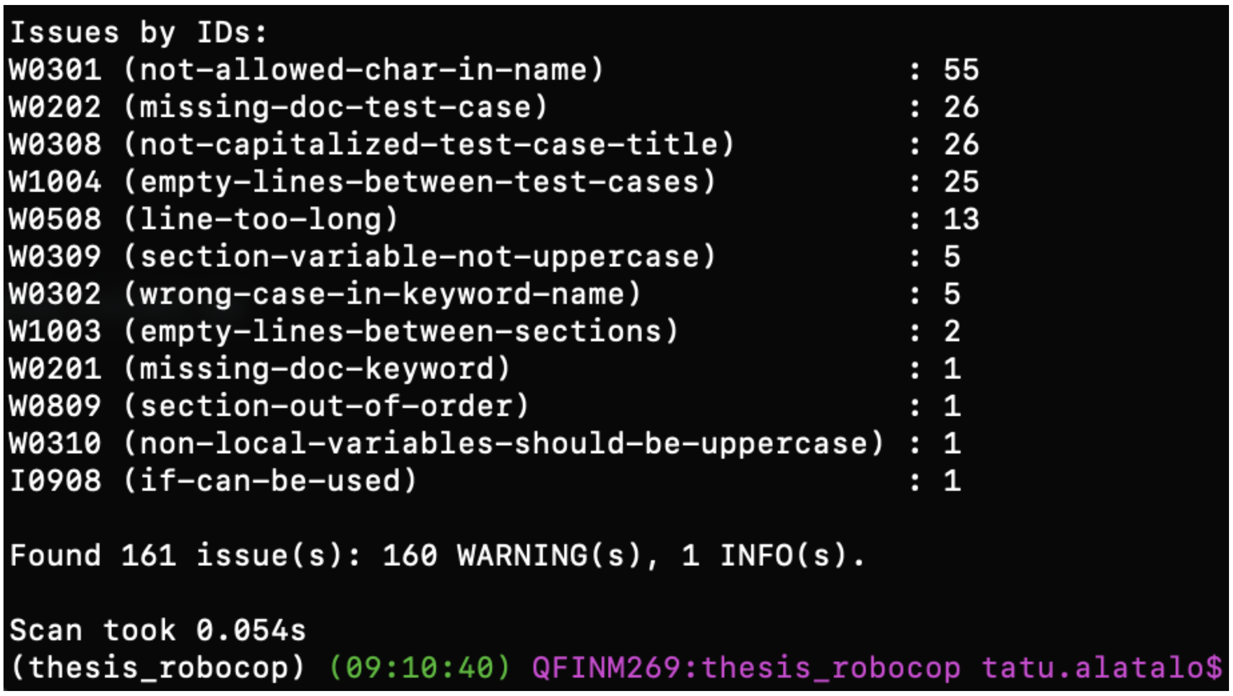
3.4 Test data

Due to the number of available rules, it was agreed that these rules are run against a set of test data and see if there are similar errors, warnings or info that are caught with Robocop. This way it may be found out the common issues that there may be. Customer programs' main regression set has over 60 test suites that covers over 2100 test cases and there are also other sets with more cases for different applications including API regression tests making test suites total number over 100. It was agreed that 3 files will be selected randomly: one regression test suite, one API regression test suite and one resource file. Those that were selected were "Foreign_address.robot", "API_Shopping-basket_with_several_subscriptions.robot" and "salestool_keywords.resource".

"Foreign_address.robot" (later referred to as the first suite) regression test suite tests how a customer with foreign address is handled in customer management (CM), order management (OM) and self-service applications using graphical user interface. In "API_Shopping-basket_with_several_subscriptions.robot" (later referred to as the second suite) test suite several subscriptions are ordered using requests through the application programming interface. "Salestool_keywords.resource" (later referred to as a resource file) is a resource file where different kinds of keywords are stored that are meant to be used in Salestool test cases.

After test data was selected, Robocop could be executed with these files with default built-in rules. First execution was made with hook report all so the Robocop prints a compilation of the issues at the end of the console print. In the first suite there were a total of 161 issues (see Figure

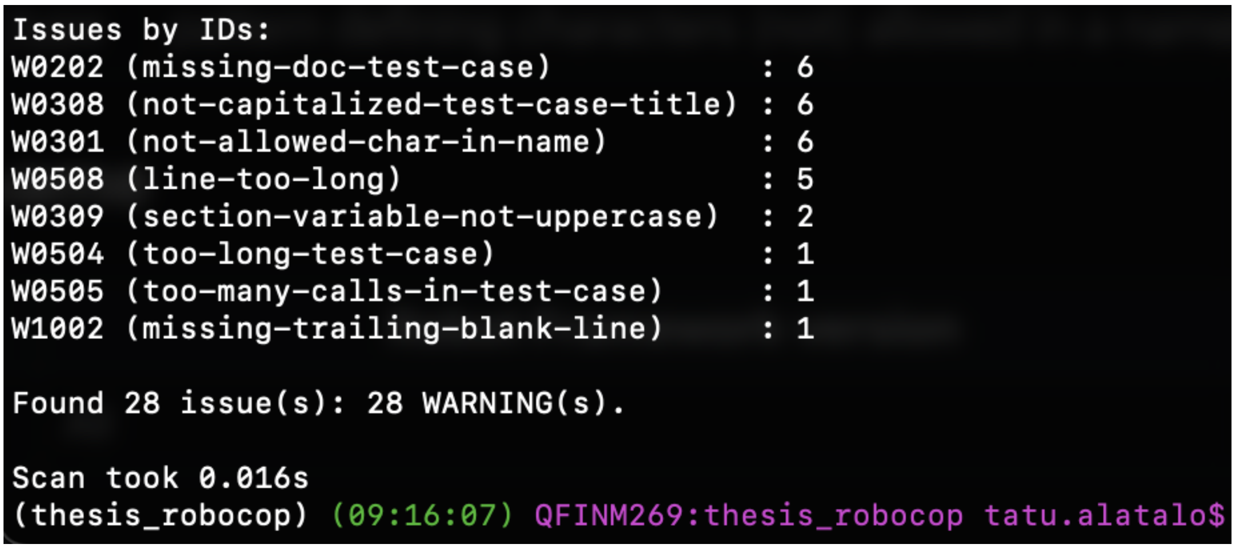
11) where 160 of those were warnings and one info issue. This first suite is over 240 lines long and contains 27 different test cases.



```
Issues by IDs:
W0301 (not-allowed-char-in-name)                   : 55
W0202 (missing-doc-test-case)                      : 26
W0308 (not-capitalized-test-case-title)            : 26
W1004 (empty-lines-between-test-cases)             : 25
W0508 (line-too-long)                              : 13
W0309 (section-variable-not-uppercase)             : 5
W0302 (wrong-case-in-keyword-name)                 : 5
W1003 (empty-lines-between-sections)               : 2
W0201 (missing-doc-keyword)                        : 1
W0809 (section-out-of-order)                       : 1
W0310 (non-local-variables-should-be-uppercase)    : 1
I0908 (if-can-be-used)                             : 1

Found 161 issue(s): 160 WARNING(s), 1 INFO(s).

Scan took 0.054s
(thesis_robocop) (09:10:40) QFINM269:thesis_robocop tatu.alatalo$
```

Figure 11. First suite ran with built-in rules

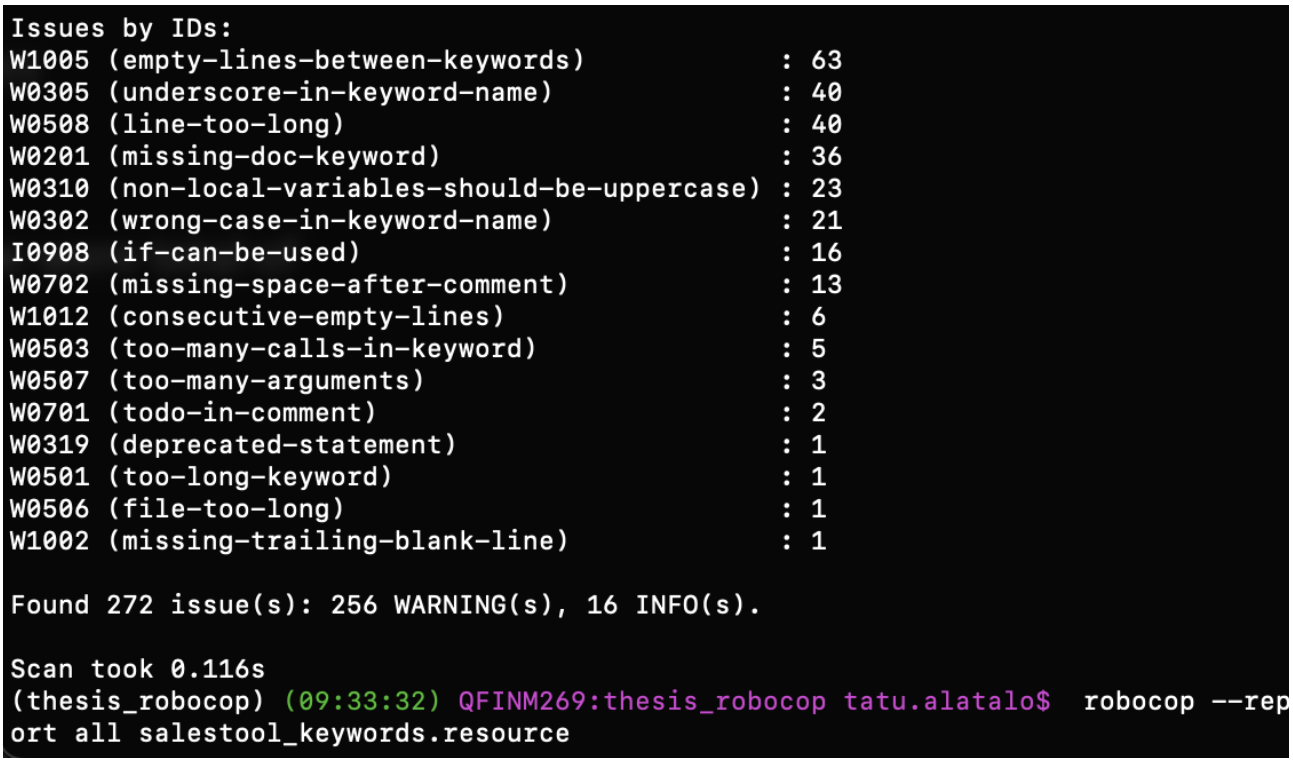
After this, the second suite was executed and there were 28 different issues with 28 warnings (see Figure 12). Second suite is smaller than the first one, with length a of 66 lines and there are 6 test cases.



```
Issues by IDs:
W0202 (missing-doc-test-case)            : 6
W0308 (not-capitalized-test-case-title)  : 6
W0301 (not-allowed-char-in-name)         : 6
W0508 (line-too-long)                    : 5
W0309 (section-variable-not-uppercase)   : 2
W0504 (too-long-test-case)               : 1
W0505 (too-many-calls-in-test-case)      : 1
W1002 (missing-trailing-blank-line)      : 1

Found 28 issue(s): 28 WARNING(s).

Scan took 0.016s
(thesis_robocop) (09:16:07) QFINM269:thesis_robocop tatu.alatalo$
```

Figure 12. Second suite ran with built-in rules

Finally, the resource file was executed and there were a total of 272 issues where 256 were warnings and 16 info issues (see Figure 13). Resource file is a collection of keywords with a length of over 650 lines and over 60 different keywords.



```
Issues by IDs:
W1005 (empty-lines-between-keywords)             : 63
W0305 (underscore-in-keyword-name)               : 40
W0508 (line-too-long)                            : 40
W0201 (missing-doc-keyword)                      : 36
W0310 (non-local-variables-should-be-uppercase)  : 23
W0302 (wrong-case-in-keyword-name)               : 21
I0908 (if-can-be-used)                           : 16
W0702 (missing-space-after-comment)              : 13
W1012 (consecutive-empty-lines)                  : 6
W0503 (too-many-calls-in-keyword)                : 5
W0507 (too-many-arguments)                       : 3
W0701 (todo-in-comment)                          : 2
W0319 (deprecated-statement)                     : 1
W0501 (too-long-keyword)                         : 1
W0506 (file-too-long)                            : 1
W1002 (missing-trailing-blank-line)              : 1

Found 272 issue(s): 256 WARNING(s), 16 INFO(s).

Scan took 0.116s
(thesis_robocop) (09:33:32) QFINM269:thesis_robocop tatu.alatalo$  robocop --rep
ort all salestool_keywords.resource
```

Figure 13. Resource file ran with built-in rules

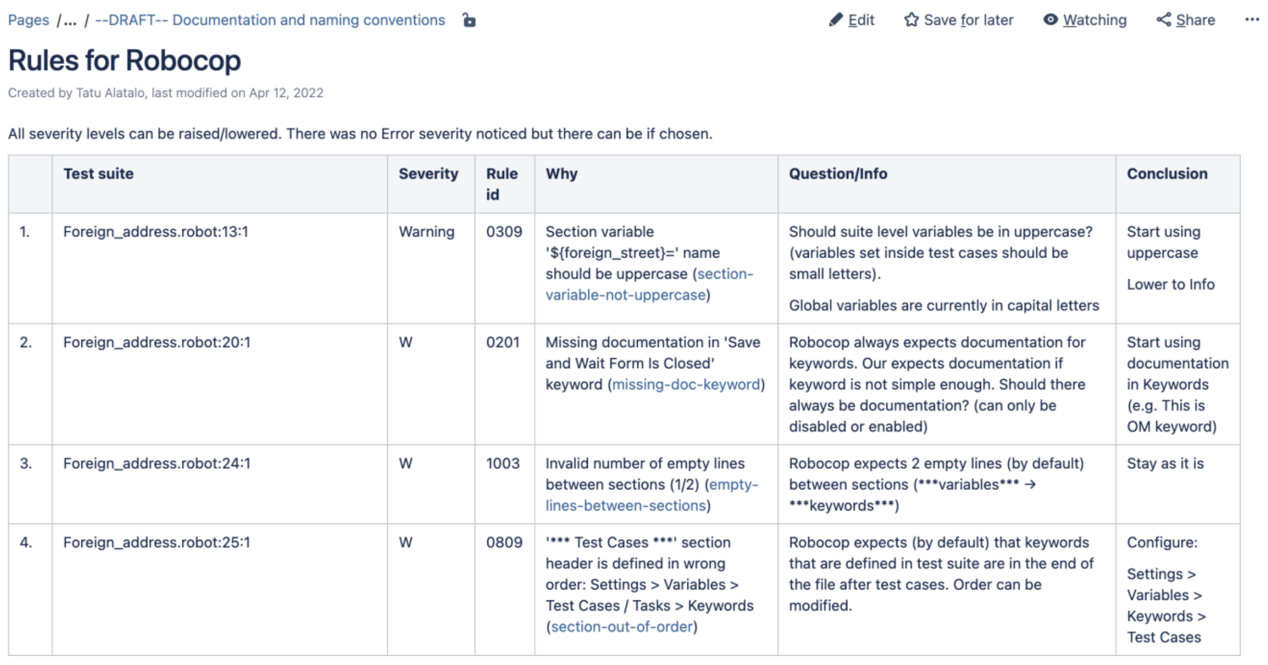
3.5 Rules for Robocop

After test data was executed with built-in rules it was noticed that built-in rules should be modified for the test automation team as they use different standard (convention) for Robot Framework. E.g., Robocop built-in rule for the number of empty lines between test cases expects only one empty line and the test automation team's own coding convention requires two. This was printed as an issue in the first suite where there were two empty lines between test cases but not in the second suite where were only one empty line between test cases. To get a clarification what and how the rules should be modified, it was agreed to have a workshop meeting where the rules for the Robocop can be decided. It was also necessary to document these as the convention that is used in test automation team is mostly undocumented and knowledge of it is shared verbally.

To get a list of rules that should be modified, it was needed to find out what unique rules were in the executions. First suite had 12 different issue ids, second suite 8 and the resource file had 16. These all were combined into a list and duplicate ids were then removed. After that, it was noticed that there were a total of 26 unique issues.

It was agreed that a wiki-page in Qvantel intranet could be utilized to hold a table where these 26 rules can be added as rows. Table consisted of 6 columns: test suite (1), severity (2), rule id (3), why (4), question/info (5) and lastly, conclusion (6). In the first column it was written the file name and the line and the column where Robocop found the issue. Severity column marked the severity of the issue which could have been error, warning or info. In rule id column was marked the id of the rule. In the 'why' column it was marked why the issue was raised by Robocop and a link to the Robocop documentation page where the rule can be seen in more detail. Question/info column holds the question or info to the test automation team that leads to a question what to do with this issue. In this column it was also mentioned if the rule is configurable. Last column was used to mark down the conclusion of what to do with the rule.

When all the rules were listed on the page an email invitation to the test automation team was sent with a link to the rules for the Robocop wiki-page and to the Robocop documentation page. In the message it was mentioned that every team member should at least glance through all the issues beforehand to get an idea what will be talked about in the meeting. When the workshop started in Microsoft Teams, it was shown how the Robocop works and what it is used for with an example run. and then it was started to talk about the rules for two hours. Conclusions were marked into the conclusion column while the meeting was ongoing (see Figure 14).


Pages / ... / --DRAFT-- Documentation and naming conventions
Edit
Save for later
Watching
Share

Rules for Robocop

Created by Tatu Alatalo, last modified on Apr 12, 2022

All severity levels can be raised/lowered. There was no Error severity noticed but there can be if chosen.

	Test suite	Severity	Rule id	Why	Question/Info	Conclusion
1.	Foreign_address.robot:13:1	Warning	0309	Section variable '${foreign_street}=' name should be uppercase (section-variable-not-uppercase)	Should suite level variables be in uppercase? (variables set inside test cases should be small letters). Global variables are currently in capital letters	Start using uppercase Lower to Info
2.	Foreign_address.robot:20:1	W	0201	Missing documentation in 'Save and Wait Form Is Closed' keyword (missing-doc-keyword)	Robocop always expects documentation for keywords. Our expects documentation if keyword is not simple enough. Should there always be documentation? (can only be disabled or enabled)	Start using documentation in Keywords (e.g. This is OM keyword)
3.	Foreign_address.robot:24:1	W	1003	Invalid number of empty lines between sections (1/2) (empty-lines-between-sections)	Robocop expects 2 empty lines (by default) between sections (***variables*** → ***keywords***)	Stay as it is
4.	Foreign_address.robot:25:1	W	0809	'*** Test Cases ***' section header is defined in wrong order: Settings > Variables > Test Cases / Tasks > Keywords (section-out-of-order)	Robocop expects (by default) that keywords that are defined in test suite are in the end of the file after test cases. Order can be modified.	Configure: Settings > Variables > Keywords > Test Cases

Figure 14. Rules for the Robocop wiki-page in Qvantel Confluence

In the meeting it was also discussed about the severity of the issues. It was agreed that info-level issues are “good to know” and are not mandatory to fix but warning and especially error issues should be fixed every time these occur. While discussing this, it was also mentioned that severity of every built-in and external rule can be raised or lowered if wanted. After the meeting it was discussed that the test automation team would like to try Robocop themselves and it was agreed that during this assignment there will be a user guide created for that.

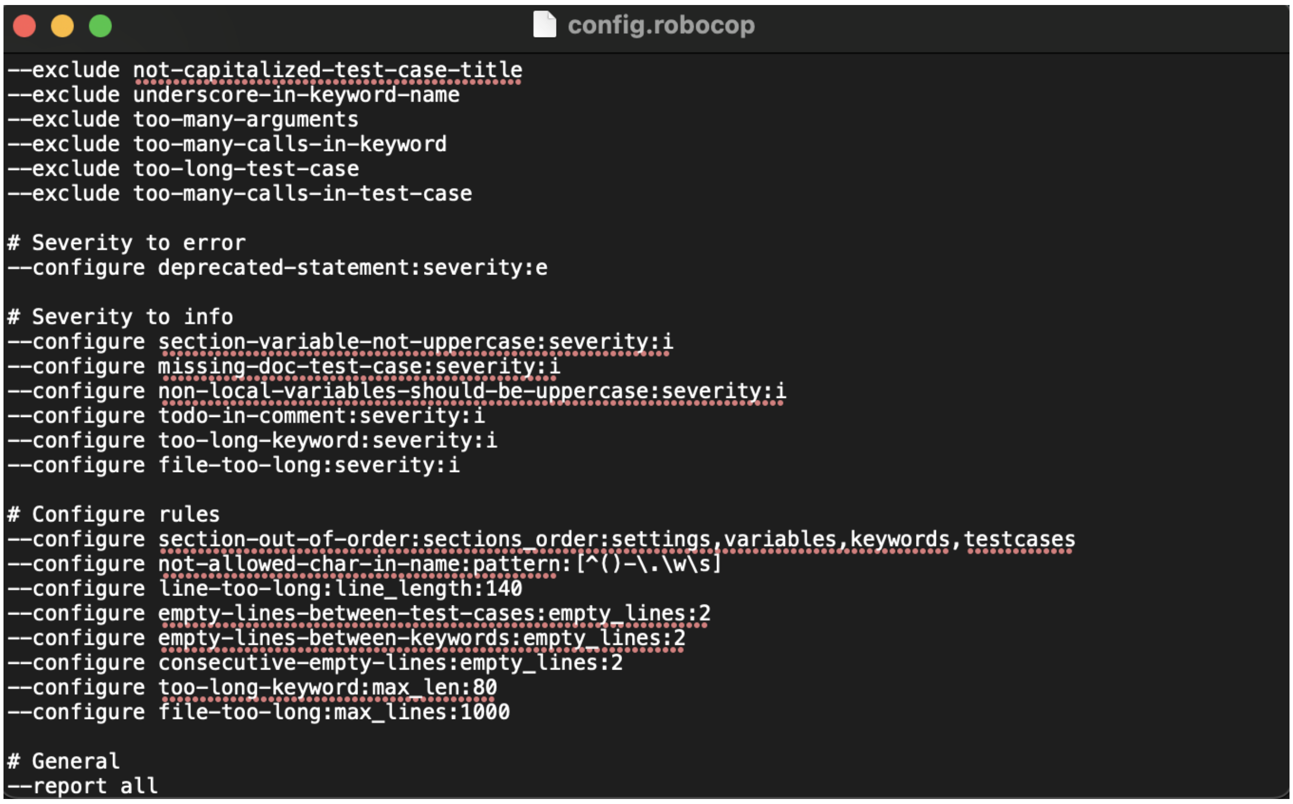
3.6 Creating configure file

After the rules for the Robocop were decided in the meeting it was necessary to create a configure file where the built-in rules can be modified. This file is being used as an argument-file so that there is no need to list every modified rule in the command line. There were 5 rules that did not need any modification so there was no need to write them in to the configure file. Report all hook was added to the end of the configure file so every time configure-file is being used there will be a clean report.

It was agreed that 6 rules can be totally excluded as those either are against current coding convention or are unnecessary. Excluded rules were as follows: not capitalized test case title, underscore in keyword name, too many arguments, too many calls in keyword, too long test case and too many calls in test case. These were added into the configure file with exclude hook.

Secondly, every built-in rule severity could be raised or lowered, and it was decided that deprecated statements -rule should be error level instead of warning level. Total of 6 rules were decided to lower into info-level and those were following: section variable not uppercase, missing doc test case, non-local variables should be uppercase, todo in comment, too long keyword and file too long. These were added into the configure file with a configure hook where severity level was given with a letter.

Rules that needed configuring were following 8 rules. Section out of order -rule (1), it was agreed that sections within the test suite are as follows: settings, variables, keyword and test cases. Default rule expects that keywords introduced in the test suite are at the end of the suite. Line too long -rule (2) was agreed to extend to 140 characters instead of 120. Empty lines between test cases (3), empty lines between keywords (4) and consecutive empty lines -rule (5) were raised from one to two empty lines. Too long keyword -rule (6) name character limit was raised from 40 to 80 characters and file too long (7) -rule was raised from 400 to 1000 lines. Not allowed char in name (8) -rule needed a new regular expression (regex) pattern as it by default only checks that suite, test case or keyword name does not have a period or a question mark. It was modified with a pattern where it is checked that the name only consists of characters that are allowed: period mark, round brackets, hyphen, word characters and any whitespace characters. These were also added to the configure file with a configure hook (see Figure 15).



config.robocop

```
--exclude not-capitalized-test-case-title
--exclude underscore-in-keyword-name
--exclude too-many-arguments
--exclude too-many-calls-in-keyword
--exclude too-long-test-case
--exclude too-many-calls-in-test-case

# Severity to error
--configure deprecated-statement:severity:e

# Severity to info
--configure section-variable-not-uppercase:severity:i
--configure missing-doc-test-case:severity:i
--configure non-local-variables-should-be-uppercase:severity:i
--configure todo-in-comment:severity:i
--configure too-long-keyword:severity:i
--configure file-too-long:severity:i

# Configure rules
--configure section-out-of-order:sections_order:settings,variables,keywords,testcases
--configure not-allowed-char-in-name:pattern:[^()-\.\w\s]
--configure line-too-long:line_length:140
--configure empty-lines-between-test-cases:empty_lines:2
--configure empty-lines-between-keywords:empty_lines:2
--configure consecutive-empty-lines:empty_lines:2
--configure too-long-keyword:max_len:80
--configure file-too-long:max_lines:1000

# General
--report all
```

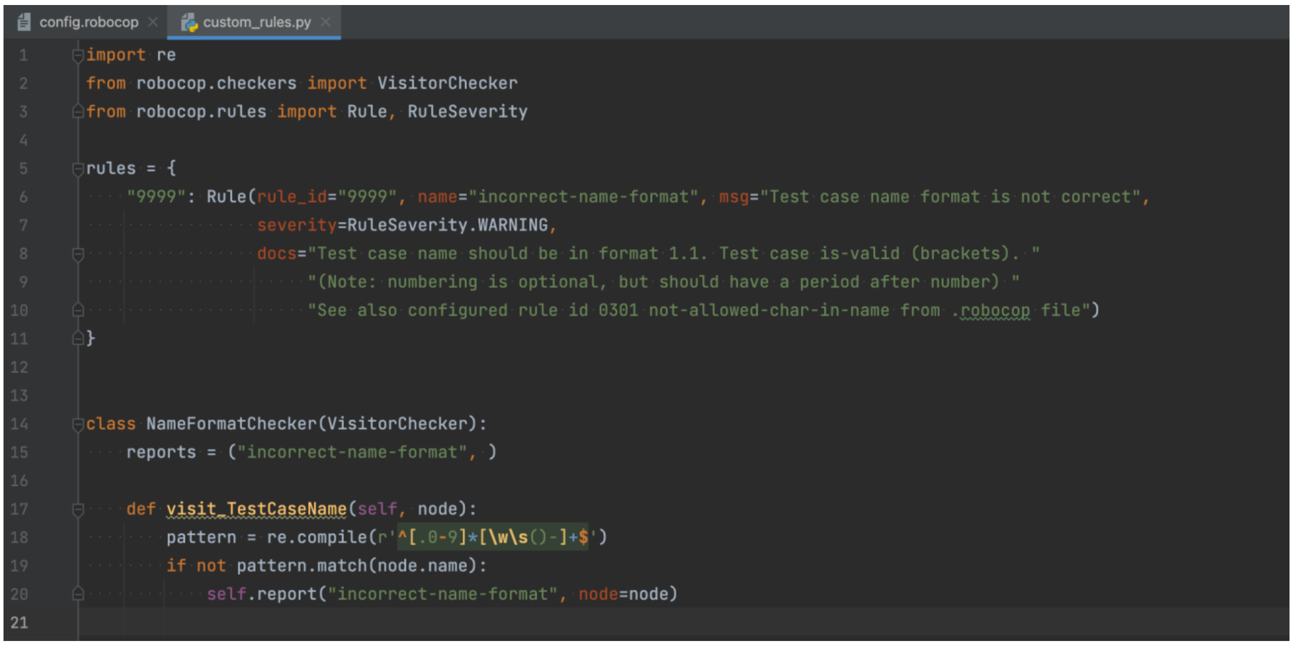
Figure 15. Config.robocop file

3.7 Creating custom rules

When creating a regular expression pattern for “not allowed char in name” -rule and period mark now allowed, it was noticed that Robocop did not notice if the keyword or a test case name has consecutive period-marks in it. E.g., “1.1. This is a test case name…” was a valid test case name according to the Robocop but by test automation team coding convention it is a bad test case name. For this reason, a custom keyword was needed.

Robocop supports external rules that can be made by the user, and these custom rule python-files use classes from the Robocop library itself (Robocop external rules 2022). Basics of the rule were created using classes Rule and RuleSeverity where the rule id, number, name, message to the console, severity and the documentation for the rule were given. After that the logic of the rule was created by implementing a class called NameFormatChecker that uses Robocop class VisitorChecker as a parameter. Created class has a function called visit_TestCaseName where the regular expression pattern is used to check that the case name is in the correct format.

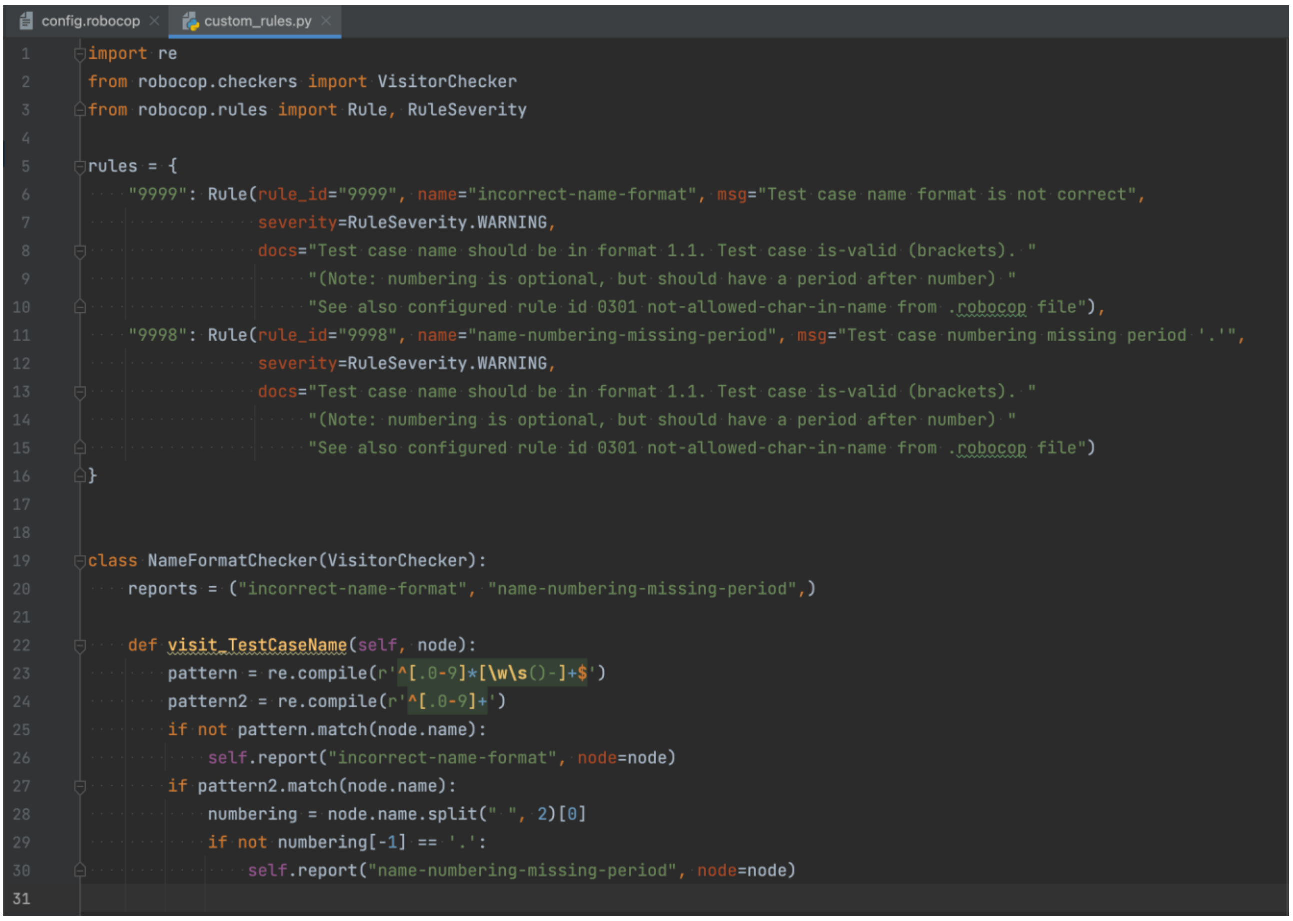
In the customer program test automation coding convention, it is agreed that test case names can contain a numbering before the test case name. Therefore, the pattern checks that in the numbering it is only allowed numbers and period-marks. After the first whitespace character the pattern checks that the rest of the name contains only allowed characters. If the test case name is not in the correct format, a warning is then raised (see Figure 16).



```python
import re
from robocop.checkers import VisitorChecker
from robocop.rules import Rule, RuleSeverity

rules = {
    "9999": Rule(rule_id="9999", name="incorrect-name-format", msg="Test case name format is not correct",
                 severity=RuleSeverity.WARNING,
                 docs="Test case name should be in format 1.1. Test case is-valid (brackets). "
                      "(Note: numbering is optional, but should have a period after number) "
                      "See also configured rule id 0301 not-allowed-char-in-name from .robocop file")
}


class NameFormatChecker(VisitorChecker):
    reports = ("incorrect-name-format", )

    def visit_TestCaseName(self, node):
        pattern = re.compile(r'^[.0-9]*[\w\s()-]+$')
        if not pattern.match(node.name):
            self.report("incorrect-name-format", node=node)

```

Figure 16. Custom rule W9999 created in PyCharm

After custom rule id 9999 named “incorrect name format” was created it was soon noticed that the rule does not notice if the numbering does not have a period mark as the last character. And due to that, a second custom rule was needed. This was done in a similar way as rule id 9999 starting with the basics of the rule. It uses the same function as rule id 9999 and checks that if the name has numbering then the last character should be a period mark. If not, then a warning named “name-numbering-missing-period” is raised (see Figure 17).



```python
import re
from robocop.checkers import VisitorChecker
from robocop.rules import Rule, RuleSeverity

rules = {
    "9999": Rule(rule_id="9999", name="incorrect-name-format", msg="Test case name format is not correct",
                 severity=RuleSeverity.WARNING,
                 docs="Test case name should be in format 1.1. Test case is-valid (brackets). "
                      "(Note: numbering is optional, but should have a period after number) "
                      "See also configured rule id 0301 not-allowed-char-in-name from .robocop file"),
    "9998": Rule(rule_id="9998", name="name-numbering-missing-period", msg="Test case numbering missing period '.'",
                 severity=RuleSeverity.WARNING,
                 docs="Test case name should be in format 1.1. Test case is-valid (brackets). "
                      "(Note: numbering is optional, but should have a period after number) "
                      "See also configured rule id 0301 not-allowed-char-in-name from .robocop file")
}


class NameFormatChecker(VisitorChecker):
    reports = ("incorrect-name-format", "name-numbering-missing-period",)

    def visit_TestCaseName(self, node):
        pattern = re.compile(r'^[.0-9]*[\w\s()-]+$')
        pattern2 = re.compile(r'^[.0-9]+')
        if not pattern.match(node.name):
            self.report("incorrect-name-format", node=node)
        if pattern2.match(node.name):
            numbering = node.name.split(" ", 2)[0]
            if not numbering[-1] == '.':
                self.report("name-numbering-missing-period", node=node)

```

Figure 17. Completed custom rules in PyCharm

For an uncomplicated presentation, a flowchart interpreting the usage of both custom rules was created (see Figure 18). In the line that is being examined the first thing Robocop does when using this rule is to check that does the test case name include forbidden characters. If yes, raise a warning with id 9999 and continue to the next verification where is checked does the test case name contain a numbering. If yes, get the numbering and verify that there is a period mark at the end of the numbering. If there is no period mark, then raise a warning with ID 9998 and continue to the any rule that is next in line.

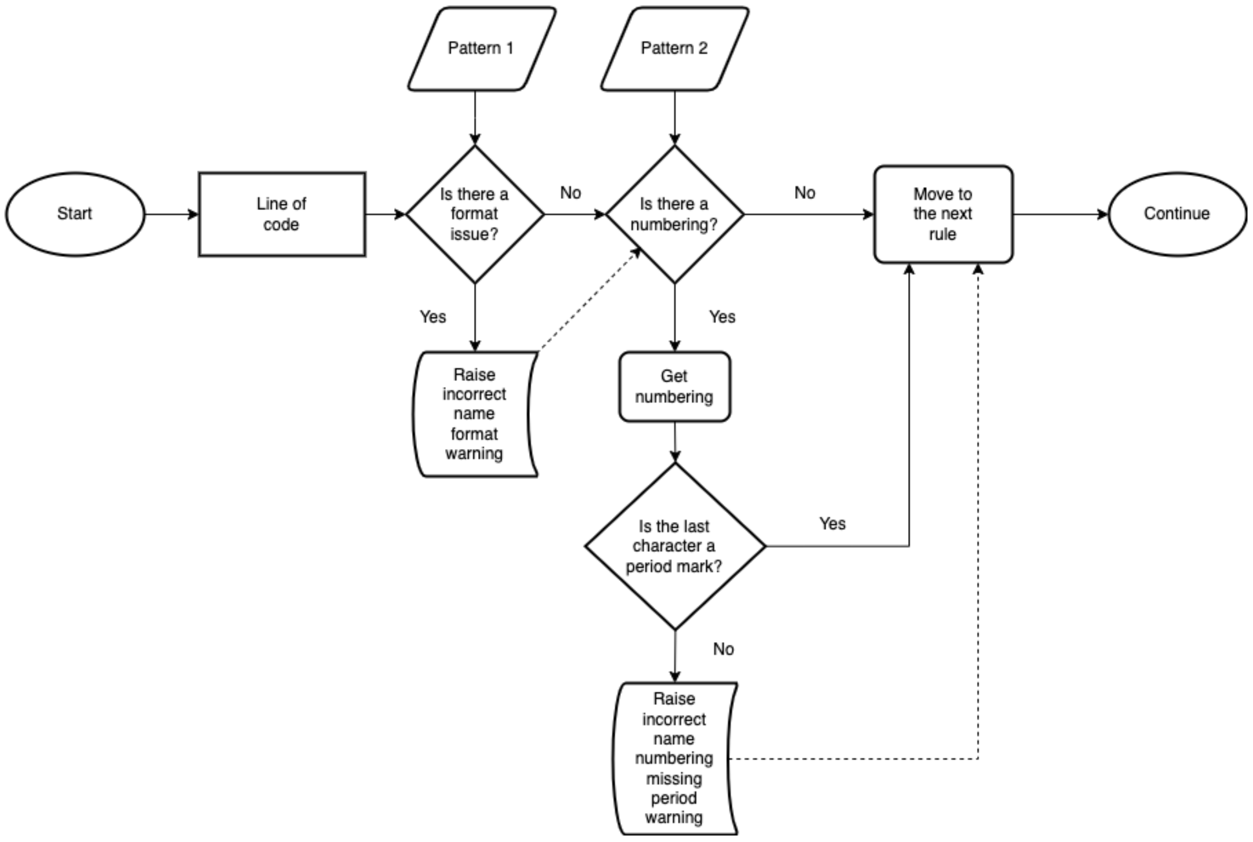


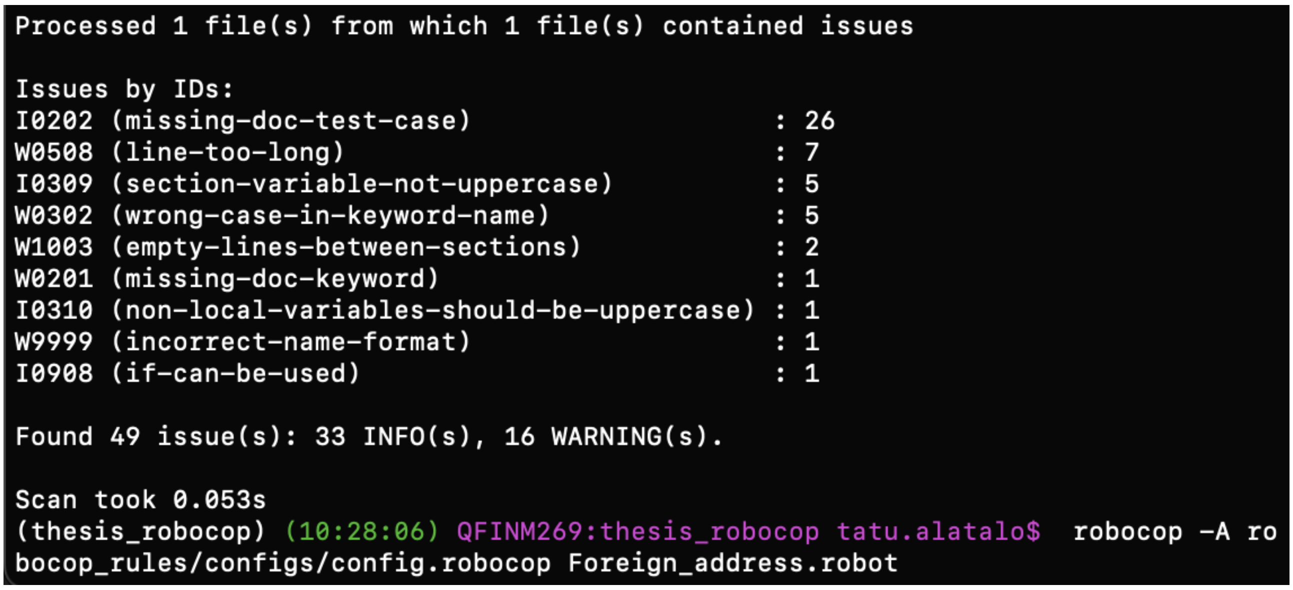
Figure 18. Completed custom rules flowchart

Finally, a folder named robocop_rules was created that contains two folders: configs folder that contains the config file and external_rules that contains custom_rules.py file. To take in use the external rules, a path to the custom_rules.py file was added to the end of the config file.

3.8 Running test data with the configured rules

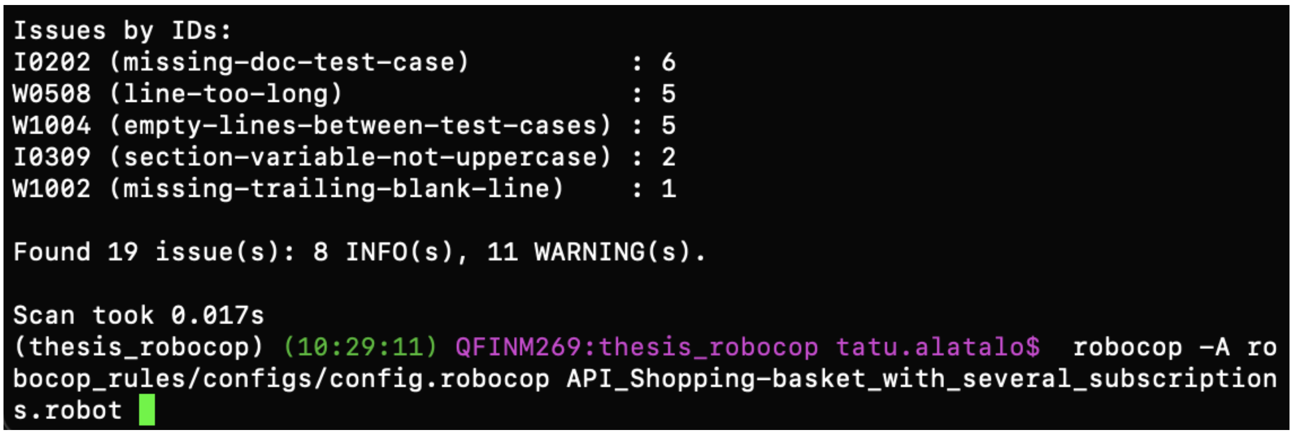
After the config file and the custom rules were created it was necessary to test again with the same test data but using the config file. It was also mandatory to test that the custom rules also work as expected.

First suite was executed with the modified rules and there were a total of 49 issues (see Figure 19) where 16 of those were warnings and 33 info issues. After that the second suite was run through Robocop and the suite had 19 issues (see Figure 20) where 11 of those were warnings and 8 info issues. Finally, the resource file had 155 issues (see Figure 21) where there were 113 warnings, 1 error and 41 info issues.



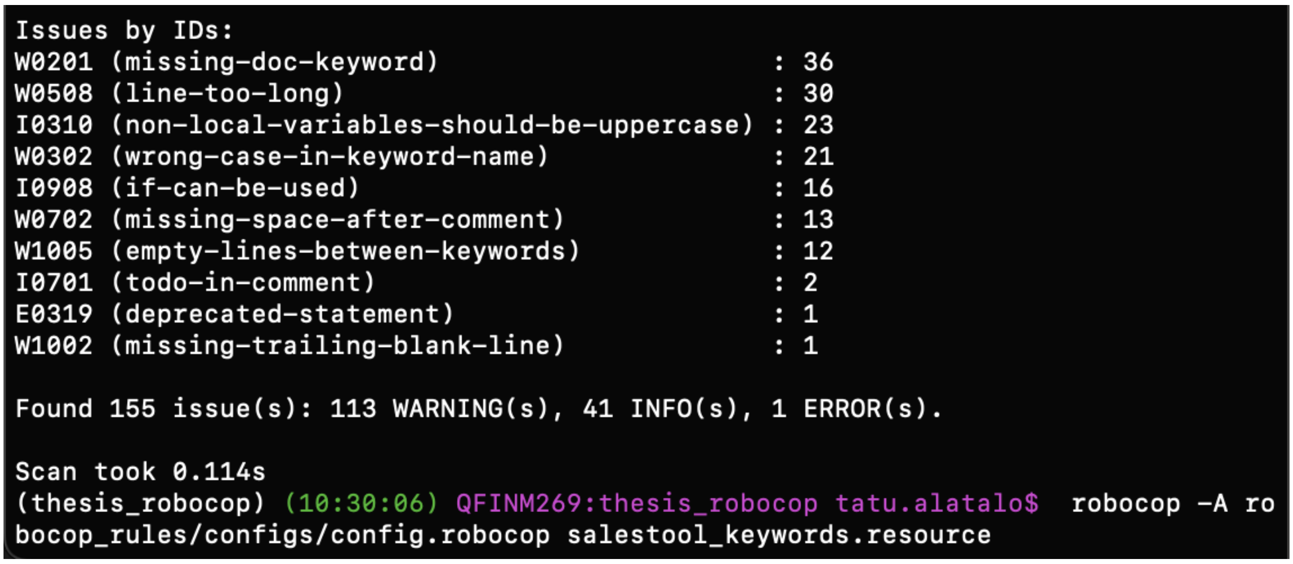
```
Processed 1 file(s) from which 1 file(s) contained issues

Issues by IDs:
I0202 (missing-doc-test-case)                       : 26
W0508 (line-too-long)                               : 7
I0309 (section-variable-not-uppercase)              : 5
W0302 (wrong-case-in-keyword-name)                  : 5
W1003 (empty-lines-between-sections)                : 2
W0201 (missing-doc-keyword)                         : 1
I0310 (non-local-variables-should-be-uppercase) : 1
W9999 (incorrect-name-format)                       : 1
I0908 (if-can-be-used)                              : 1

Found 49 issue(s): 33 INFO(s), 16 WARNING(s).

Scan took 0.053s
(thesis_robocop) (10:28:06) QFINM269:thesis_robocop tatu.alatalo$  robocop -A ro
bocop_rules/configs/config.robocop Foreign_address.robot
```

Figure 19. First suite with modified rules



```
Issues by IDs:
I0202 (missing-doc-test-case)           : 6
W0508 (line-too-long)                   : 5
W1004 (empty-lines-between-test-cases) : 5
I0309 (section-variable-not-uppercase) : 2
W1002 (missing-trailing-blank-line)     : 1

Found 19 issue(s): 8 INFO(s), 11 WARNING(s).

Scan took 0.017s
(thesis_robocop) (10:29:11) QFINM269:thesis_robocop tatu.alatalo$  robocop -A ro
bocop_rules/configs/config.robocop API_Shopping-basket_with_several_subscription
s.robot
```

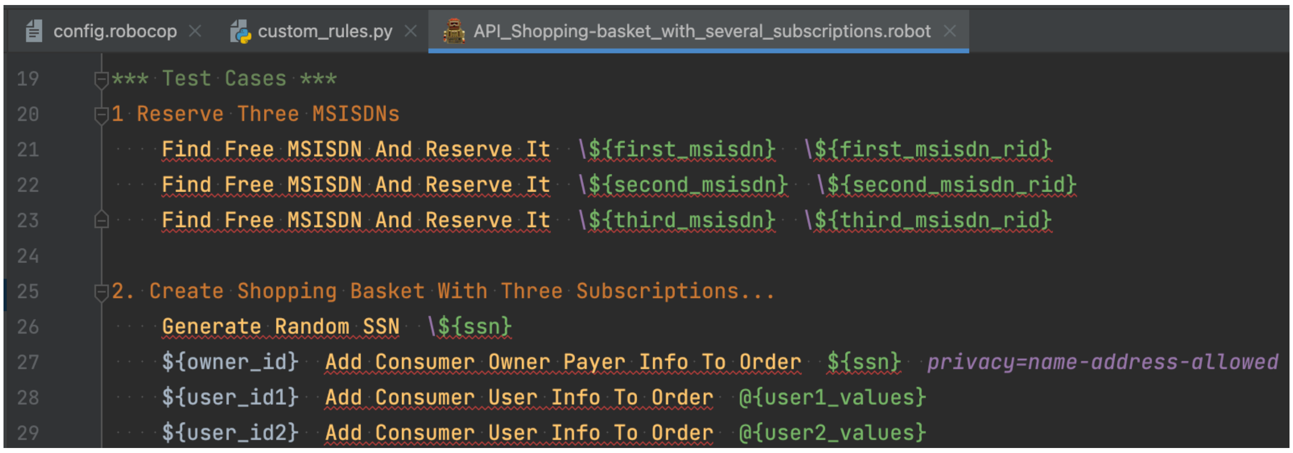
Figure 20. Second suite with modified rules



```
Issues by IDs:
W0201 (missing-doc-keyword)                      : 36
W0508 (line-too-long)                            : 30
I0310 (non-local-variables-should-be-uppercase)  : 23
W0302 (wrong-case-in-keyword-name)               : 21
I0908 (if-can-be-used)                           : 16
W0702 (missing-space-after-comment)              : 13
W1005 (empty-lines-between-keywords)             : 12
I0701 (todo-in-comment)                          : 2
E0319 (deprecated-statement)                     : 1
W1002 (missing-trailing-blank-line)              : 1

Found 155 issue(s): 113 WARNING(s), 41 INFO(s), 1 ERROR(s).

Scan took 0.114s
(thesis_robocop) (10:30:06) QFINM269:thesis_robocop tatu.alatalo$  robocop -A ro
bocop_rules/configs/config.robocop salestool_keywords.resource
```

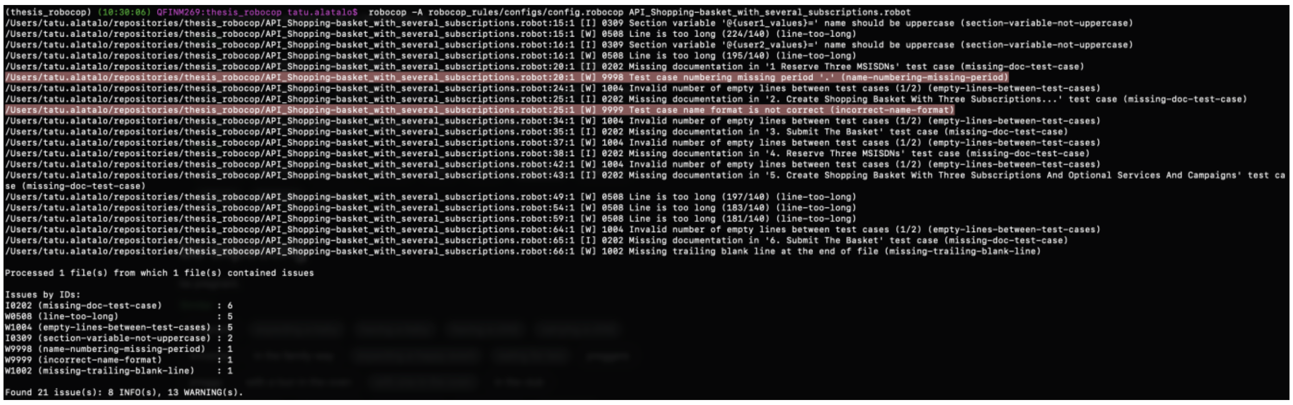
Figure 21. Resource file with modified rules

Validity of custom rules was tested by editing temporarily test case names in the second suite to have a test case name where numbering was missing period and one where there were consecutive period marks after the test case name. “1. Reserve Three MSISDNs” -test case name in line 20 became “1 Reserve Three MSISDNs” and “2. Create Shopping Basket With Three Subscriptions” in line 25 became “2. Create Shopping Basket With Three Subscriptions...” (see Figure 22). After this change, the second suite was run again, and the results contained 2 new warnings with the identifications W9998 name numbering mission period mark and W9999 incorrect name format (see Figure 23).



```
19  *** Test Cases ***
20  1 Reserve Three MSISDNs
21      Find Free MSISDN And Reserve It    \${first_msisdn}    \${first_msisdn_rid}
22      Find Free MSISDN And Reserve It    \${second_msisdn}    \${second_msisdn_rid}
23      Find Free MSISDN And Reserve It    \${third_msisdn}    \${third_msisdn_rid}
24
25  2. Create Shopping Basket With Three Subscriptions...
26      Generate Random SSN    \${ssn}
27      ${owner_id}    Add Consumer Owner Payer Info To Order    ${ssn}    privacy=name-address-allowed
28      ${user_id1}    Add Consumer User Info To Order    @{user1_values}
29      ${user_id2}    Add Consumer User Info To Order    @{user2_values}
```

Figure 22. Editing test case names in test suite 2



```
(thesis_robocop) (10:30:06) QFINM269:thesis_robocop tatu.alatalo$  robocop -A robocop_rules/configs/config.robocop API_Shopping-basket_with_several_subscriptions.robot
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:15:1 [I] 0309 Section variable '@{user1_values}=' name should be uppercase (section-variable-not-uppercase)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:15:1 [W] 0508 Line is too long (224/140) (line-too-long)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:16:1 [I] 0309 Section variable '@{user2_values}=' name should be uppercase (section-variable-not-uppercase)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:16:1 [W] 0508 Line is too long (195/140) (line-too-long)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:20:1 [I] 0202 Missing documentation in '1 Reserve Three MSISDNs' test case (missing-doc-test-case)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:20:1 [W] 9998 Test case numbering missing period '.' (name-numbering-missing-period)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:24:1 [W] 1004 Invalid number of empty lines between test cases (1/2) (empty-lines-between-test-cases)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:25:1 [I] 0202 Missing documentation in '2. Create Shopping Basket With Three Subscriptions...' test case (missing-doc-test-case)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:25:1 [W] 9999 Test case name format is not correct (incorrect-name-format)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:34:1 [W] 1004 Invalid number of empty lines between test cases (1/2) (empty-lines-between-test-cases)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:35:1 [I] 0202 Missing documentation in '3. Submit The Basket' test case (missing-doc-test-case)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:37:1 [W] 1004 Invalid number of empty lines between test cases (1/2) (empty-lines-between-test-cases)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:38:1 [I] 0202 Missing documentation in '4. Reserve Three MSISDNs' test case (missing-doc-test-case)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:42:1 [W] 1004 Invalid number of empty lines between test cases (1/2) (empty-lines-between-test-cases)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:43:1 [I] 0202 Missing documentation in '5. Create Shopping Basket With Three Subscriptions And Optional Services And Campaigns' test ca
se (missing-doc-test-case)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:49:1 [W] 0508 Line is too long (197/140) (line-too-long)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:54:1 [W] 0508 Line is too long (183/140) (line-too-long)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:59:1 [W] 0508 Line is too long (181/140) (line-too-long)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:64:1 [W] 1004 Invalid number of empty lines between test cases (1/2) (empty-lines-between-test-cases)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:65:1 [I] 0202 Missing documentation in '6. Submit The Basket' test case (missing-doc-test-case)
/Users/tatu.alatalo/repositories/thesis_robocop/API_Shopping-basket_with_several_subscriptions.robot:66:1 [W] 1002 Missing trailing blank line at the end of file (missing-trailing-blank-line)

Processed 1 file(s) from which 1 file(s) contained issues

Issues by IDs:
I0202 (missing-doc-test-case)           : 6
W0508 (line-too-long)                   : 5
W1004 (empty-lines-between-test-cases) : 5
I0309 (section-variable-not-uppercase) : 2
W9998 (name-numbering-missing-period)  : 1
W9999 (incorrect-name-format)           : 1
W1002 (missing-trailing-blank-line)    : 1

Found 21 issue(s): 8 INFO(s), 13 WARNING(s).
```

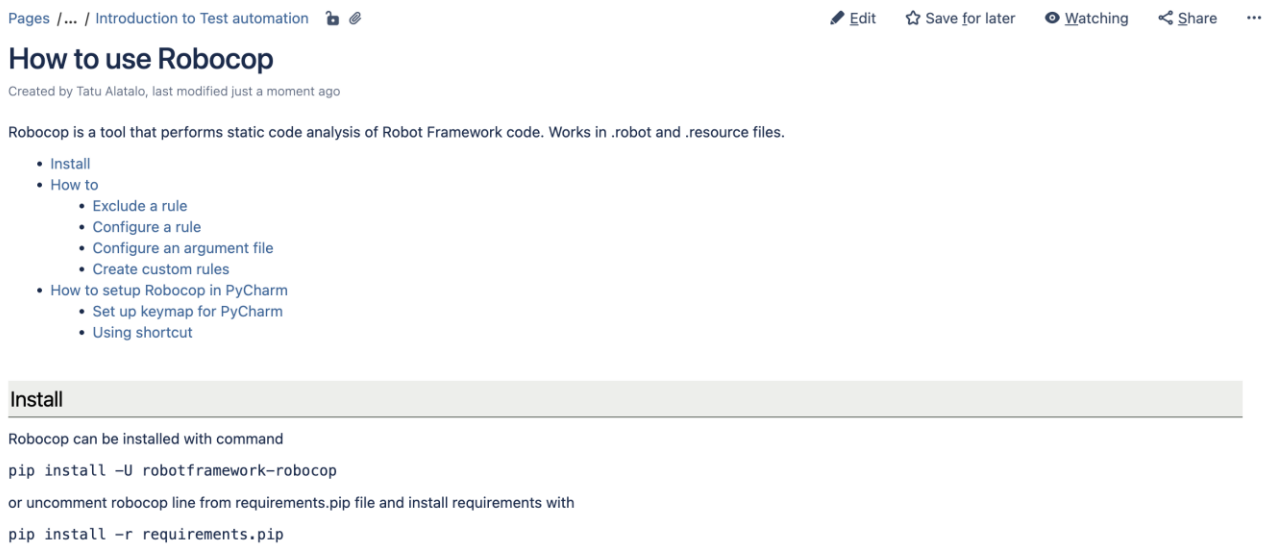
Figure 23. Testing custom rules in test suite 2

After testing that configure file and custom rules work as expected, these new created files and changes in the requirements file were committed into the robocop branch in the test automation repository in Qvantels Gitbucket and later merged into the develop branch.

3.9 Robocop user guide for test automation team

User guide was needed so the test automation team could try out Robocop in their daily work. It was also created as a wiki page in Qvantel Confluence (see Figure 24.), and it is made of 3 main parts: how to install Robocop, how to use it and how to setup it with PyCharm. User guide is written in high level and with links to the official Robocop documentation.

How to install section guides test automation team member to install Robocop from the requirements file from the repository or with a command line command. In the how to use section it is guided how to exclude rules, how to configure a built-in rule, how to edit configure file and how to create custom rules. In the final section it is guided how Robocop can be used in PyCharm as an external tool and how to enable plus use it with a keyboard shortcut.


Pages /... / Introduction to Test automation

Edit Save for later Watching Share

How to use Robocop

Created by Tatu Alatalo, last modified just a moment ago

Robocop is a tool that performs static code analysis of Robot Framework code. Works in .robot and .resource files.

- Install
- How to
  - Exclude a rule
  - Configure a rule
  - Configure an argument file
  - Create custom rules
- How to setup Robocop in PyCharm
  - Set up keymap for PyCharm
  - Using shortcut

Install

Robocop can be installed with command

```
pip install -U robotframework-robocop
```

or uncomment robocop line from requirements.pip file and install requirements with

```
pip install -r requirements.pip
```

Figure 24. User guide for how to use Robocop

4 Result analysis

Robocop seems to be a suitable static code analysis tool for the test automation team as they took it in use, and it was straightforward to modify as well as extend by creating custom rules. It seems also to be actively updated compared to rflint.

When comparing the results of test suite 1 (see Table 1.) it is noticed that there are almost 70% less issues raised by Robocop from 161 issues to 49 issues. It is possible that the default built-in rules are stricter compared to what was modified and therefore there are a lot more issues raised with built-in rules. Less issues with modified rules could be explained due to the reason that some of the rules were decided to be excluded, severities of some rules were lowered, and some rules were customised accordingly. With built-in rules there were 55 warnings only because Robocop noticed a period-mark in test case names and that was now allowed. There were also 26 warnings about numbering of test cases as Robocop expected test case names to start with a letter and now numbering of test case names are permitted.

Table 1. Comparing results of test suite 1

	Test suite 1 (built-in rules)	Test suite 1 (modified rules)
Error count	0	0
Warning count	160	16
Info count	1	33
Total issues	161	49

In test suite 2 there were a total of 28 issues with built-in rules and 19 issues with modified rules (see Table 2) and that is over 32% less issues. This could be due to the same reason as with test suite 1 that severities of warnings were lowered, and some rules were excluded or customised.

Table 2. Comparing results of test suite 2

	Test suite 2 (built-in rules)	Test suite 2 (modified rules)
Error count	0	0
Warning count	28	11
Info count	0	8
Total	28	19

Resource file had 43% less issues from 272 issues to 155 (see Table 3). Same reasons apply here as with previous files and there is one error issue with modified rules. That is because one rule regarding using of deprecated keywords from Robot Framework was raised from warning to error severity level. Most of the warnings that were not reported with modified rules were empty lines between keywords (63 warnings) and underscore in keyword name (40 warnings). First mentioned rule was customised from one empty line to two empty lines and the second rule was excluded.

Table 3. Comparing results of resource file

	Resource file (built-in rules)	Resource file (modified rules)
Error count	0	1
Warning count	256	113
Info count	16	41
Total	272	155

In all test data ran with modified rules it was noticed that there were less warnings, and more info level issues. This does not mean that there are warning level issues hidden and not reported by Robocop anymore, but warnings were raised as an info level issue or not at all due to the agreed rules by the test automation team. Errors and warnings that are now raised with the modified rules are now valid issues that must be fixed in the test suite. Agreed rules in the configure file also makes it easier for the test automation team member to check the file for real issues and info level issues help the user to make improvements to the file.

Taking into account that there are many issues in the test data after taking in use the configured rules, it is possible that the amount of work that is required to edit all the test suites and resource files is excessive. Even the smallest test data file had 11 warnings that should be taken care of and there are at least over 100 test suites in use. Although, all the issues that Robocop reports can be relatively effortless to fix, as those are only related to the coding standards that were created in the configure file. By this number of issues, it still takes quite a lot of time to fix all the issues.

Validity of the created coding standard is correct because it was agreed with team members who are working in test automation and are experts in their field. It is now written down how the test suites and resource files should look like and that can be verified with Robocop.

5 Conclusion

Objectives of this thesis were to find out if the Robocop is a suitable static code analysis tool for the test automation team, what is the current coding standard used by the team and how much there is to fix in the test suites and in resource files after coding standard is found out.

All the objectives were met as the Robocop was taken in use by the test automation team, coding standard was created as a configure file that can be modified and extended by the team itself and test automation team can now see the issues from the files themselves using Robocop. This gives them more time during pull requests to focus on the logic side of the tests and not to the syntax issues. There are quite an excessive number of issues in all the files, and it is up to the test automation team to decide how and when they decide to fix them.

Chosen research strategy and methods were correct as it was not possible to do this research with other strategy or with only one method. Qualitative research method was used to gather information about what are the current coding standards in rules for Robocop meeting. A quantitative research method was used to handle Robocop results so they could be utilized and the knowledge foundation for this was done in the assignment and theory basis. Although, the theory basis was slightly brief but manages to give a solid ground for the assignment. Action research required that there is a problem that should be solved, although there was no direct problem to be solved in this assignment it was still in the background as a driving force for this assignment. Action research was utilized in full course during the assignment as there were before and after measurements and intervention between. It was also present when analyzing results.

In the assignment Robot Framework lint tool was not tested with Qvantel files and no other static code analysis tools for Robot Framework were not found other than that and Robocop. There could be others but most likely those are not that well known and maintained. It should be also noted that Robot Framework linter may be suitable for use but as it is not maintained actively there is a possibility that it is not compatible with the newest version of Robot Framework.

It should be also taken in consideration that are the Robocop built-in rules a standard for the Robot Framework. So far there is no official code quality standard for Robot Framework, but it might be that in the future Robocop could become one of the standards as it was promoted in Robocon 2021 and now in the upcoming Robocon 2022. People working on it are also professional quality assurance and test automation engineers (Robocop 2021). For that reason, it is possible that Robocop may be maintained for a long period of time.

From the ethical perspective it should be examined that is there a problem with personally working in the same test automation team and doing research for it. Research was done ethically by discussing with team members and listening their thoughts first before telling personal thoughts, especially when discussing in rules for Robocop meeting.

Coding standard was created from a small sample of files, but it seems to have caught most common issues that are present in files. There could have been possibly more files to execute with Robocop to catch more issue types but discussing with the team indicates that they have not yet seen issues where built-in rules should be modified. Robocop built-in rules seem to be efficient to catch new issues in other files that were not present in test data during the assignment.

User guide was an additional task for the assignment as it was agreed to be created at the end of rules for Robocop meeting. Overall, it did not require that much time to create the page as installation and usage of Robocop was familiar from the assignment. Feedback from the test automation team was positive as the user guide that was created was uncomplicated to follow and everyone got Robocop into use without asking for help. It was also mentioned that a new custom rule could be implemented that checks if the test suite or a keyword contains a debug keyword from the Robot Framework debug library and gives a warning issue due to that. The findings from this research could be utilized in other Qvantel customer programs too.

Thesis supervisor Teemu Somppi from Qvantel gave feedback that the user guide was clear to read. He also stated that the rules for Robocop meeting were arranged and held skillfully as the agreed rules were relatively clear to follow. Somppi also mentioned that the modifying of built-in rules and specifically creating the custom rules were done especially well. (Somppi 2022.)

6 Further Development

Main further development idea that was raised by the test automation team was that after the team has a continuous integration pipeline tool taken in to use, Robocop could possibly be used there. That way Robocop could check issues from pull requests automatically so the person that is reviewing pull request does not need to manually run Robocop against files that are being modified. Robocop repository (2022) seems to have some guidance how on to take it in use with Jenkins, an automation server that is also in use in Qvantel.

References

Anmol. 2021. Types of Automation Testing. User named Anmol article on EuroSTAR Huddle page. Accessed 13 January 2022. Retrieved from https://huddle.eurostarsoftwaretesting.com/types-of-automation-testing/

Bellairs, R. 2020. Perforce blogpost about what is Static code analysis. Accessed on 6 January 2022. Retrieved from https://www.perforce.com/blog/sca/what-static-analysis

Business support system. 2012. Techopedia article about what is business support system (BSS). Accessed on 6 January 2022. Retrieved from https://www.techopedia.com/definition/26873/business-support-system

Different types of testing in software. 2021. Perfecto's article about different types of testing in software on their site. Accessed 10 January 2022. Retrieved from https://www.perfecto.io/resources/types-of-testing

Hamilton, T. 2022. Automation Testing Tutorial: What is Automation Testing? Tutorial in guru99 page. Accessed 13 January 2022. Retrieved from https://www.guru99.com/automation-testing.html

Hirsjärvi, S., Remes, P., & Sajavaara, P. 2013. Tutki ja kirjoita [Research and write]. 132–133. 18th ed. Porvoo: Bookwell Oy

Hirsz, B., Nojek, M. 2021. Robocon 2021 – 1.02 How to avoid jail for nasty code. Youtube video service. Published 7.4.2021. Accessed on 30 March 2022. Retrieved from https://www.youtube.com/watch?v=vZoyi2ObM8E

Kananen, J. 2015. Kehittämistutkimus opinnäytetyönä: Kehittämistutkimuksen kirjoittamisen käytännön opas [Action research as a thesis: a practical guide to writing action research]. E-book, 34-35, 38-40, 43-45. Jyväskylä: Jyväskylä University of applied sciences. Accessed on 8 January 2022. https://janet.finna.fi

Kumar, D., Mishra, K. K. 2016. The Impacts of Test Automation on Software's Cost, Quality and Time to Market. Elsevier B.V. 2016. Accessed on 13 January 2022. Retrieved from https://doi.org/10.1016/j.procs.2016.03.003

Klärck, P. 2016. Pekka Klärck, Eliga - DEVOPS 2016 Helsinki. Video. Youtube video service. Published 12.4.2016. Accessed on 20 March 2022. Retrieved from https://www.youtube.com/watch?v=-M65Oyk0nLw

Martin, J. 2017. Using Automation to Assist –Not Replace – Manual Testing. Blog post in Smartbear page. Accessed on 13 January 2022. Retrieved from https://smartbear.com/blog/using-automa-tion-to-assist-not-replace-manual-test/

Python static analysis tools. 2021. Luminousmens article on their homepage. Accessed 6 January 2022. Retrieved from https://luminousmen.com/post/python-static-analysis-tools

Qvantel. 2022. Qvantels homepage. Accessed on 6 January 2022. Retrieved from https://www.qvantel.com/

Qvantel LinkedIn. 2022. Qvantels LinkedIn page. Accessed on 6 January 2022. Retrieved from https://www.linkedin.com/company/qvantel/about/

Rflint repository. 2019. Robot Framework linter repository. Accessed 30 March 2022. Retrieved from https://github.com/boakley/robotframework-lint

Robocop external rules. 2022. Robocop documentation for external rules. Accessed 20 April 2022. Retrieved from https://robocop.readthedocs.io/en/stable/external_rules.html

Robocop repository. 2022. Robocop repository readme file. Accessed on 29 March 2022. Retrieved from https://github.com/MarketSquare/robotframework-robocop/blob/master/README.md

Robot Framework User guide. 2021. Robot Frameworks user guide. Accessed on 27 March 2022. Retrieved from https://robotframework.org/robotframework/latest/RobotFrameworkUser-Guide.html

Robot Framework – Simple example. N.d. Uncomplicated Robot Framework example in Robot Framework homepages code playground. Accessed on 27 March 2022. Retrieved from https://ro-botframework.org/code/?codeProject=N4IgdghgtgpiBcIDKBLKAHANjABAUQA9os4AaEAExgGcBjA-JxXQBcUB7MBEEcgMxWzUEAbQC65KowBuMCgAV6bAFYxazBM3oBXGOUUAjNswBqMetXa-dEAVgB0ABhABfIA

Robot Framework – Test automation the smart way. N.d. Quintagroups article about Robot Frame-work. Accessed on 27 March 2022. Retrieved from https://quintagroup.com/cms/python/robot-framework

Somppi, T. 2022. Test Automation Manager, Qvantel Oy. Microsoft Teams message 27.04.2022. Feedback for Tatu Alatalos thesis.

Venv. 2022. Python Software Foundations documentation about what is virtual environment and how to create it. Accessed on 3 April 2022. Retrieved from https://docs.python.org/3/li-brary/venv.html

What is software testing?. N.d. IBMs article about what is software testing on IBMs homepage. Ac-cessed on 7 January 2022. Retrieved from https://www.ibm.com/se-en/topics/software-testi