Lakshminarayana Vudum

Automatic Bug Filing based on Test Failures

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master’s Thesis

16 August 2020

Author Title Number of Pages Date	Lakshminarayana Vudum Automatic Bug Filing based on Test Failures 36 pages 06 Nov 2020
Degree	Master of Engineering
Degree Programme	Information Technology
Instructor(s)	Ville Jääskeläinen, Principal Lecturer

In any software development methodology, a SW bug report is very essential as it shows the errors in the system. Fixing the errors in the SW helps to deliver a high-quality product to the customer. Any individual involved in product development can file a bug which can lead to inconsistent bug reports. A poor bug report requires additional effort and time from the team to either retest or debug the issue further.

The objective of this thesis is to deliver a product in the form of a script to create an automatic bug report based on the automated test case failures. When the script is run, the script will read the web-based test failures and will create a bug report based on the failures.

Solving this problem will save a lot of time for the team by filing bugs automatically and the bug reports will be consistent throughout the project. Also, the script can be altered based on the needs so that the automatic bug reporting can be improved.

The objective of the thesis was achieved by writing a Python script to read and analyze web-based test failures to create a bug report in GitHub. There were two parts involved in finishing this product, one was to gather the data from the test failure that is required to post into the bug report and the other part was to push the data into the bug tracking tool.

The results of the thesis show that automated bug reports were filed successfully fulfilling all specified project requirements. This product shows that bug filing through automation is possible for different scenarios. This product can also be altered to file bug reports for different types of failures. One improvement that was found out in this thesis is that the product should avoid creating duplicate bug reports.

Keywords	Bug report, Defect, Automation, ATC

Contents

List of Abbreviations

ATC	Automated Test Case
CMD	Command prompt
FW	Firmware
GUI	Graphical user interface
i915	A Linux kernel device driver for Intel Graphics
IGT	Intel Graphics Processing Unit (GPU) Tools is a collection of tools for development and testing of the DRM drivers
OS	Operating system
SW	Software
SYS_REQ	System Requirement

1 Introduction

A SW bug is an error in the system that causes the system to behave in unintended ways. SW bug reports contain the description of the issue and test environment including a SW release version, device information, and expected behavior. A high-quality SW bug report makes a developer's life easy and saves a lot of time for the other team members. Since bugs come from the team itself and outside of the team, it's nearly impossible to have a consistent bug report. In some cases, critical information might be missing from the report which requires re-testing and additional effort for debugging the issue.

SW bugs come from different phases of the SW development life cycle and most common phases are development or coding and testing phases. In addition, bugs come from customers or end users once the product is delivered or released. Out of these phases majority of the bugs will be identified or detected during testing phase as the test engineers will test the SW application against the software and/or system requirement specifications. Testing can happen in two ways one is manual testing and other way is automation testing. In manual testing test cases are executed step by step by a tester without test scripts whereas in automated testing a tool will execute the test cases and generates the test report. Test automation tools and frameworks to perform automated testing depends on the type of the project and requirements.

In general, manual testing consumes more time to finish testing compared to automated testing. Automated Test Cases (ATCs) can be run on any given day, for any number of cycles and perform continuous testing to find issues in the SW as quickly as possible. Any failures that occur in manual and automation testing will lead test engineers to file bugs against the SW. In any type of testing whether it's manual testing or automated testing, bugs are filed manually. A typical bug report contains a summary of the issue and steps to reproduce the issue. Apart from that, a test set up environment will be included in the bug report e.g SW version, Firmware (FW) version, Operating System (OS) details, platform etc which gives more clues to the developer. So, bug reports are crucial part in SW development in order to improve the quality of the product.

1.1 Objective and Importance of Thesis

The objective of the thesis is to:
“Deliver a product in a form of script to file bugs through automation based on web-based test results. Running the script should file bugs based on the product specifications”

Developing this product is very important to any organization who is developing SW as it saves time so that teams can avoid filing bugs manually after looking at the web-based test results. When bug reports are automated, the product can be more easily altered to meet different product specifications. This results in having consistent bug reports throughout the project so that teams will file bugs in the same format. If the situation comes to file bugs manually, teams can follow the same format as automatic bug filing. When bug reporting is automated, one doesn’t need to worry about the missing information. The product can be developed and updated according to the project needs. The script can be run all the time and that bug reports can be filed round the clock.

This product was developed using open-source tools. Web-based test results are taken from IGT (Intel Graphics Processing Unit Tools) test failures which tests Linux kernel device driver i915. The Python script was developed to file bugs through automation. There are two parts in the script, one is to gather the data from the failure according to the specification and the other part is to create a bug in GitHub and push the data to the bug.

1.2 Scope and Deliverables

The scope of the study includes only to file new bugs based on the web-based test failures through automation. When the script is run, the script will read all the failures and file bugs in GitHub. The output of this thesis is to deliver a working script that creates bugs automatically. This product can be improved further in such a way that creating duplicate bugs can be avoided, which is not part of the scope of this study.

1.3 Study Plan

For SW development, the waterfall model is the earliest SDLC approach which is also known as Linear Sequential Life Cycle model. From Figure 1 Waterfall model [1] Testing phase comes only after product development or implementation is completed. Since the majority of bugs come from the testing phase, fixing bugs in the testing phase is very expensive, which is the major drawback of developing a SW product using this waterfall model.

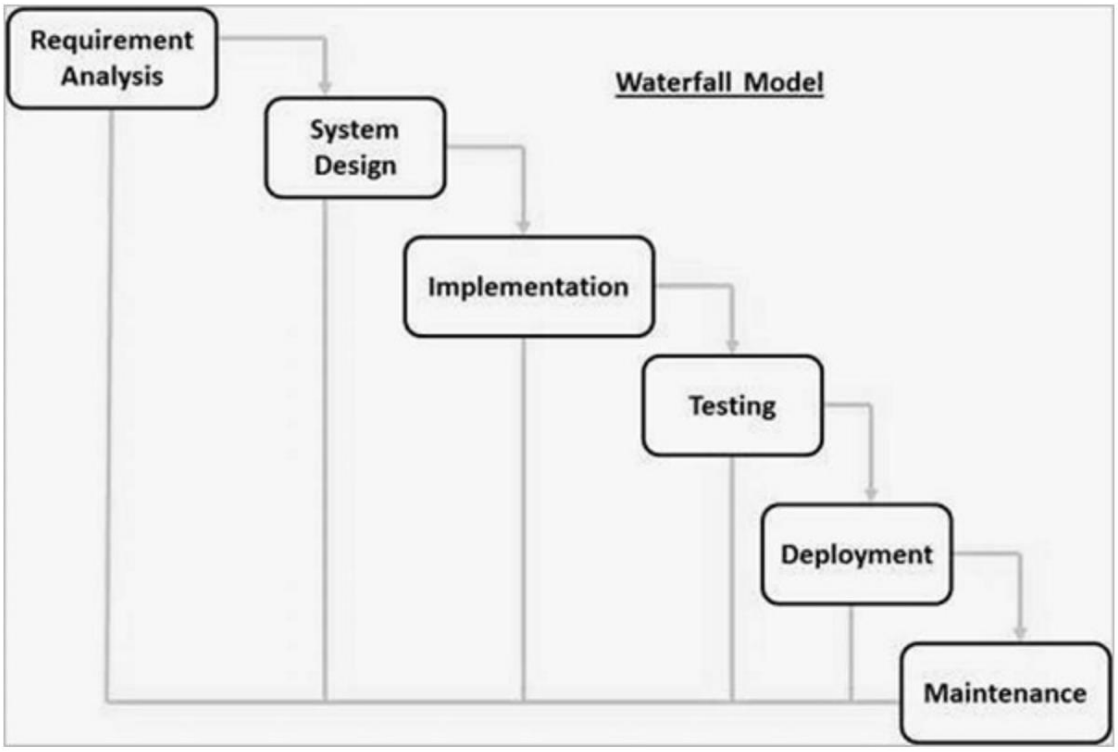


Figure 1. Waterfall model [1]

In recent times, agile software development with scrum has become the most commonly used SW development methodology which is an iterative and incremental model. From Figure 2 [2] SW testing is part of implementation which makes bug fixing more economical.

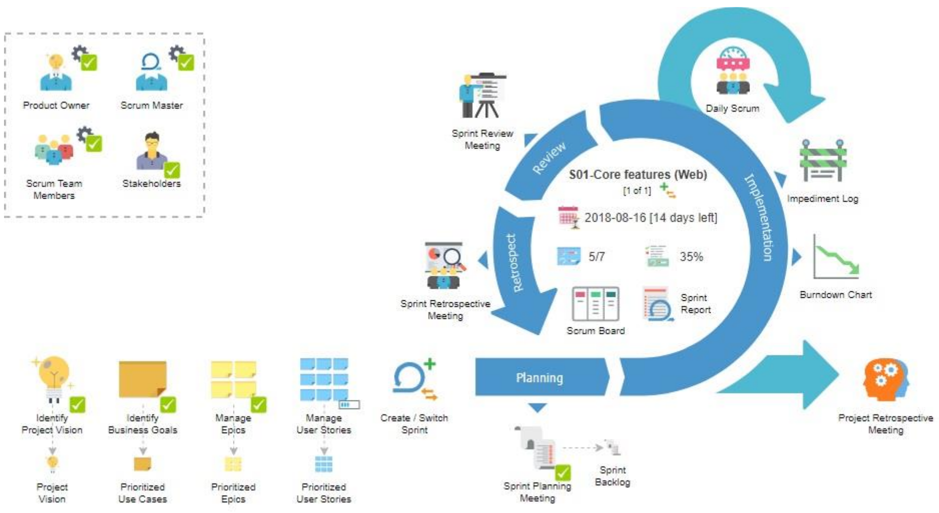


Figure 2. Scrum Approach [2]

In any SW development model, most of the bugs are found in the testing phase. SW testing is performed either by manual testing or test automation. In any type of testing, test failures are reported as bugs that are filed manually. The development team including the test engineers has to address all the test failure reports manually which is the problem that is addressed as part of this thesis study. Addressing test failures manually is time consuming and the bug reports will not be consistent throughout the project which may cause problems like missing data. This thesis study addresses the problem that web-based test failure reports are addressed manually but it has to be addressed automatically through automation. In order to address this problem, one of the requirements that have to be fixed is that web-based test failures should be addressed automatically. Also, when a bug is created, report should contain all relevant details which help the development team to understand the issue easily.

One of the challenges in order to address test failure reports through automation is that, test failure reports must be consistent. Addressing test failures will be smooth if the test failure reports are consistent and the failure gives all relevant information every time. In the process of automation, when a bug is created, the system should avoid creating duplicate bugs which is another challenge. There is a chance that the system will create duplicate bugs. So, the product should be much more intelligent to find if a matching bug is already existing in the bug management database or not which is not in the scope of this thesis study.

Addressing web-based test failure reports can be done by automation. When a test failure is available, the system shall read the failure and file a bug automatically. Addressing the failures and filing bugs can be automated by many ways by choosing different scripting languages. Since this study is addressing web-based test failures, for example, Python, JavaScript, or user interface test automation tools are few different options that can be chosen to address web-based test failures.

The solution to the above-defined requirements has to be verified as part of this thesis study. Below test case has to be executed to verify the product requirements. Verifying the developed product shall meet the expected criteria defined in the test case.

Test case:

Pre-condition:

1. Ensure the test setup is ready.
   Refer Chapter 4 for Test Environment
2. Web-based Test failures (URL's) are available in a file.
3. Internet connection is available.

Test steps:

1. Run the script to address the failures.

Expected Result:

1. The product shall address all the test failure reports automatically.
2. Bug reports are available for each failure in GitHub.
3. Bug report shall contain all relevant details about the failure.

1.4 Structure of Thesis

This thesis consists of six chapters. The first chapter gives an overview of the topic's objective, scope, deliverables of the thesis, and thesis study plan. Chapter two illustrates the current state analysis and lists the product specifications that the result should meet. Furthermore chapter 2 illustrates the solution architecture. Chapter three explains the development and test environment respectively. This chapter explains all the required tools and the setup required to develop and test the product. Chapter four explains the product design in detail so that it’s easy to understand what the code is doing. Test results and analysis are explained in chapter five. This includes the screenshots of the results and showcases the evidence that the product meets the specifications. The last chapter of the thesis includes the conclusion and findings of the thesis.

In most of the companies bugs are filed manually which can be automated. Especially failures from automated testing can be read automatically and bug filing can be automated. This could be an interesting part to be developed particularly to a project where numerous bugs are expected. The product of this thesis study could be useful for any project that needs to automate bug filing.

2 Product Specifications

The purpose of the product is to file a bug automatically when web-based test failures are available. This product will save time and effort of filing bugs manually from engineering teams.

2.1 Current State Analysis

In any SW development project, if testing is performed by manual testing, the development team would be aiming to implement test automation to avoid manual testing. When a failure appears in manual testing, the test engineer will file bugs manually. At the same time, if the testing is performed by test automation and when an ATC is failed, still bugs have to be failed manually. This is the most common trend in SW development.

In some cases, bug management tools such as Github, Jira, Trac would be integrated to test management tools such as Test Rail, Quality Center so that when a test step result is not matching to the expected result a bug will be automatically created based on the actual result of the test step. Even in this case, test engineer has to enter the actual result of the test step so that bugs will be filed manually. So, manual steps are still needed in this case. Whereas if an ATC is failed, the team must address the failure and file a bug for the test failure including the test case name, test environment, and description of the issue.

In either way, bug filing is done manually which is the problem addressed by this thesis study. Filing bugs manually takes time and effort from the teams which can be avoided partially if not completely. In the current situation if the automated testing happens overnight or during the weekend and if there are any failures, the team will address the failures only on the next working day. Looking at the current situation addressing test failures manually is not only time consuming but also not up to date are the problems that are addressed as part of this thesis study.

Developing a product or system to address this problem has a few challenges. Firstly, the test failure reports must be consistent on every run and every time. This helps to design a product to read the test failures which are uniform and stable. If the test failure is not persistent, the design of the product will keep on change which eventually delays the completion of the design. Also, the failure reports should be uniform for every run, if not, the development of the product will never end. So, choosing the test failures in order to complete the thesis study is critical.

Secondly, addressing the test failures and filing bugs automatically will save time to the development team but simultaneously the product should avoid filing duplicate bugs. Avoid creating duplicate bugs is another challenge to this thesis study which is not part of the thesis study. Lastly, the product has to be tested rigorously to find out all the issues and the issues must be fixed in order to deliver a high-quality product. Fixing all the bugs in this product as part of the thesis study will be challenging as it's time-dependent.

2.2 Solution Requirements

As the objective of this thesis is to address the test failures automatically and make bug filing through automation, the scope of the thesis is to consider only ATC failures which are available in a web-based format. The plan is to read the failure and extract the bug title and bug description from the web-based failure and create a bug in GitHub automatically through Python script. Bug title includes test name, failure status (fail, crash, etc.), and failure summary whereas bug description includes test environment including a link to the failure and failure description. The system should read the failure and gather bug title and bug description as shown in the figure 3. Solution Architecture. In addition, the script will push these details to GitHub and file a bug automatically. Below are the functional requirements that have to be implemented to finish the product.

Functional Requirements:

SYS_REQ_1	The system shall file a bug automatically based on web-based test failure.
	a) Bug title shall include the name of the test, status type, and failure summary.
	b) Bug description shall contain test environment and failure description.

The product has to be verified against all the above functional requirements and test results should provide evidence that the above requirements are fulfilled by the product. Non-functional requirements are not part of this thesis study.

During the implementation of this product, if any bugs are encountered, all the bugs should be documented and fixed in order to finish the product development.

2.3 Solution Architecture

Filing bugs automatically can be done using many ways e.g.by integrating bug management tool to test management tool. If an actual test step result is not matching to the expected result, the Test engineer will fail the test step and enter the actual result in the test management tool to push the details to the bug management tool in order to create a bug. Doing by this way, there are two drawbacks. One is that not all test management tools and bug management tools are open source. Test management tool which integrates with bug management wouldn't be available for free of charge. Another drawback is that even if the tools are available for free of charge, the actual result and environment shall be added to the test failure step in the test management tool manually and then push the details to the bug management tool to file a bug. So, manual steps are still needed. This option doesn't really solve the purpose of this thesis because of the cost and manual steps involved.

Another potential option to address the failures and automate bug filing is by choosing test automation tools like Selenium, Test Complete, Robot framework, etc. But these kinds of test automation tools are generally used to automate windows application, and/or web-based applications but not suitable for this thesis study. These automation tools are heavy to download and install packages, libraries or dependencies to address the test failures. Also, some of the tools are not open source. Test automation tools are not a suitable option for this thesis study.

Lastly, the product can be developed using a scripting language such as Python or Javascript. Since the test failures are web-based the product in this study was chosen to be developed using python. The reason for choosing Python 3 is explained in Chapter 3.2. So, this product will be developed using Python 3 script and with other open-source tools. Once the script is run in cmd, the script shall read the failure and file a bug automatically in GitHub. The script shall be run in Windows OS. There are no browser-specific restrictions, test failures can be opened from any available browser.

From Figure 3 Solution Architecture it can be found that the first step is to have web-based test failures and the final goal is to file a bug automatically through automation. When the web-based test failures are available e.g in a file, the script will go through each failure and read the test failure report using the URL given in the file. Python 3 script will extract the bug title and bug description after going through the failure report.

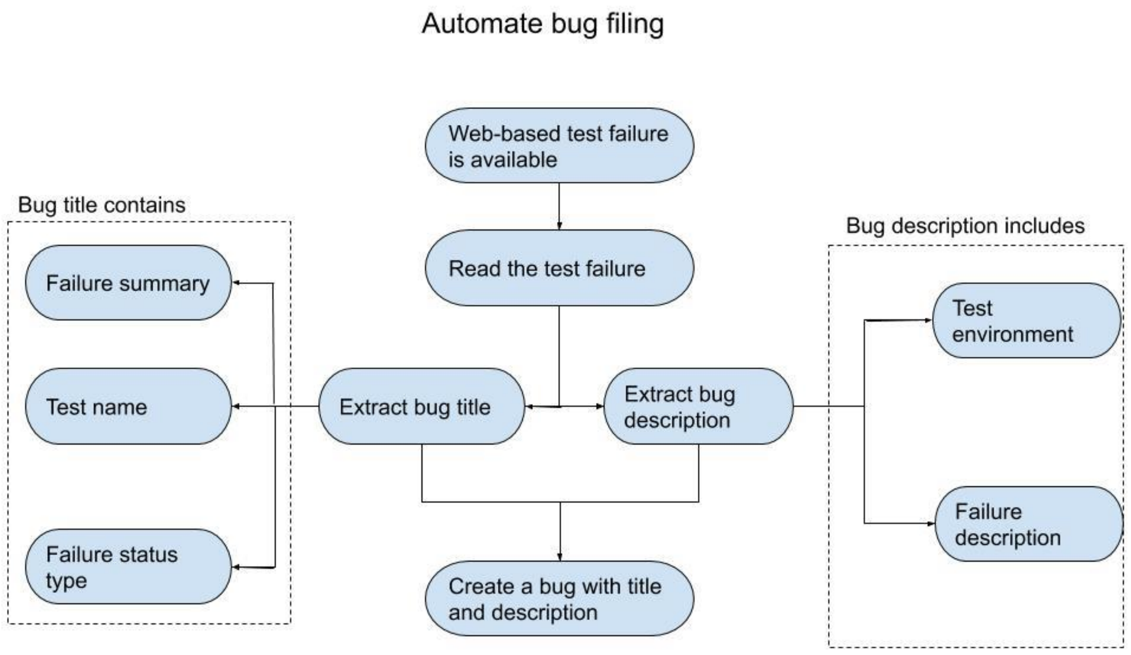


Figure 3. Solution Architecture

For any bug report, the bug title should be clear and accurate enough to understand the issue. In a bug report, the bug title is more often read compared to any other part of the bug report. A misleading bug title will lead to creation of a duplicate bug which adds additional work to the team to close the issue as a duplicate of the original issue. If a team member is searching for an existing bug in the database, a good bug title will appear on top of the search list if the title is informative and has all relevant keywords in the title. So, the bug title is crucial for a bug report as it helps the whole team in many ways.

Once the failure is read by the script the first thing that the script does is to extract the bug title from the test failure report. From Figure 3 Solution Architecture it can be found that the bug title contains failure summary, test name, and failure status. The failure summary explains what is the error when the test was failed the name of the failed test, and the failure status type such as a crash, fail, skip, etc. Reading the bug title, it can be easily understood what test failed and how it failed which gives an overall picture of the bug. As part of the bug title the script will extract the failure summary, test name, status type and store the information in the bug title which is eventually pushed to GitHub when creating a bug report.

Aside from the bug title, the script will extract the bug description from the test failure report. From Figure 3 Solution Architecture, bug description includes test environment and failure description. Bug description should explain the test environment so that when a development team looks into an issue they understand where the issue occurred. In some cases, the test environment might have an issue that some changes might not have been ready so the failure is expected, or the test environment might not have all the changes that are required to run the test. The test environment gives clues to set the priority of the issue. For example, if an issue appears in a particular setup which is not a priority to the program, in that case, the priority of the issue can be dropped so that the team can focus on high priority bugs.

Test environment information is very vital to include in a bug report. In addition to the test environment, the bug report should include the failure description to indicate what is the failure and how it was failed. These details will be very helpful to the development to triage or debug the issue and eventually to fix the issue. With this information, the development team can try to reproduce the issue to see in their test machines. The team will find more clues when they see the issue on live where they can have more debugging logs.

Once the bug title and bug description are extracted from the failure a bug will be filed in GitHub with that information. If the text file has more than one failure, the script will address the next failure and goes on to address all the failures.

3 Development Environment

To develop the product, the following applications are required to install:

1. PyCharm Community.
2. Python 3 including pip3.
3. Chrome browser.
4. Internet connection.

3.1 Install PyCharm Community

PyCharm Community is an open-source Python script editor tool. The product was developed using this tool. PyCharm has community and professional editions; the community edition is free of charge whereas the professional edition has a license fee. In this thesis PyCharm 2020.2 (Community Edition) was used.

Instructions to install PyCharm Community edition includes:
- Download windows version of PyCharm Community edition [3]
- Run the PyCharm 2020.2 exe file which starts the installation wizard.
- Follow the instructions to finish the installation successfully.

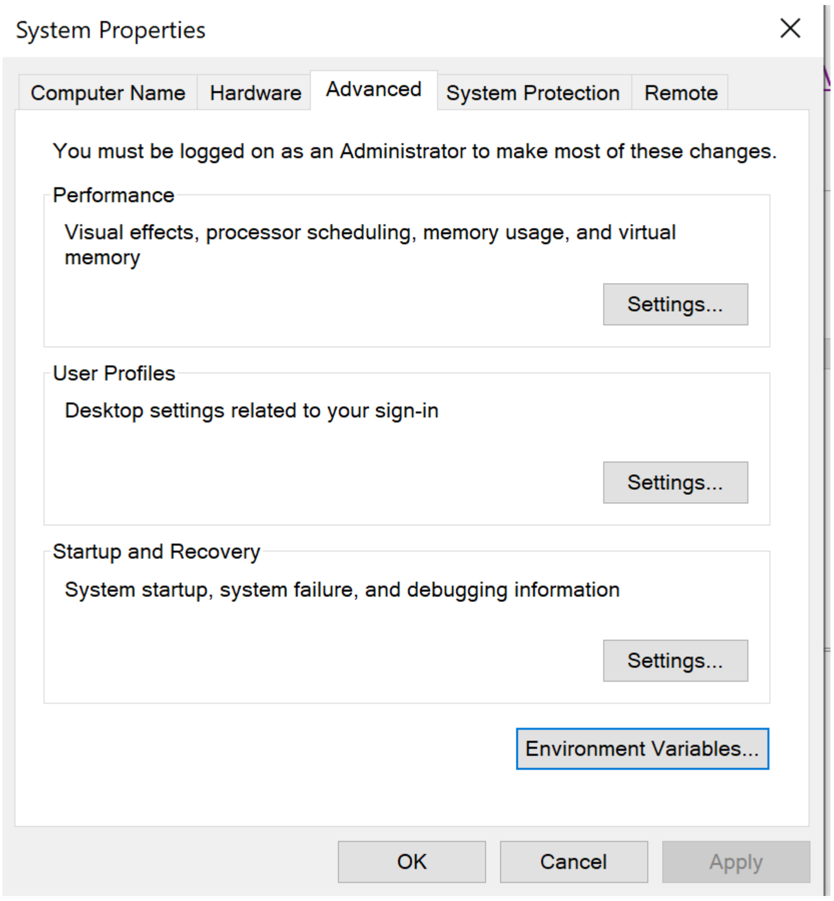
3.2 Install Python

Python has become one of the most popular and powerful language in the last few years. Python is an open-source language with easily accessible which makes it popular and easy to develop or design software and automate processes. Python is easy to learn as its syntax is not complex, unlike some other programming languages. Software can be developed with minimum lines of code with Python compared to other programming languages. Using Python, SW can be developed in windows OS, Linux, and MAC. One of the main advantages of developing SW using Python is that it has got a wide range of libraries and frameworks. One can find Python libraries for almost everything e.g. data analysis, machine learning, artificial intelligence and data visualization, and many more. Python supports frameworks such as Django for web development [4].

Python usage is growing constantly which helps to develop the language with each passing day. Python has got corporate sponsors from large IT-companies such as Google which helps to develop the language rapidly and continuously.

Python 3 was used to develop the product. Instructions to install Python 3 in windows OS are given below:

1. Download Python for Windows OS [6].
2. Run the exe file which opens the installation wizard.
3. Follow the instructions and install Python 3 successfully [5].
4. Go to Environment variables and add Python path.
    - Navigate to System Properties->Advanced->Environment Variables in Windows OS e.g Windows 10



System Properties

You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles

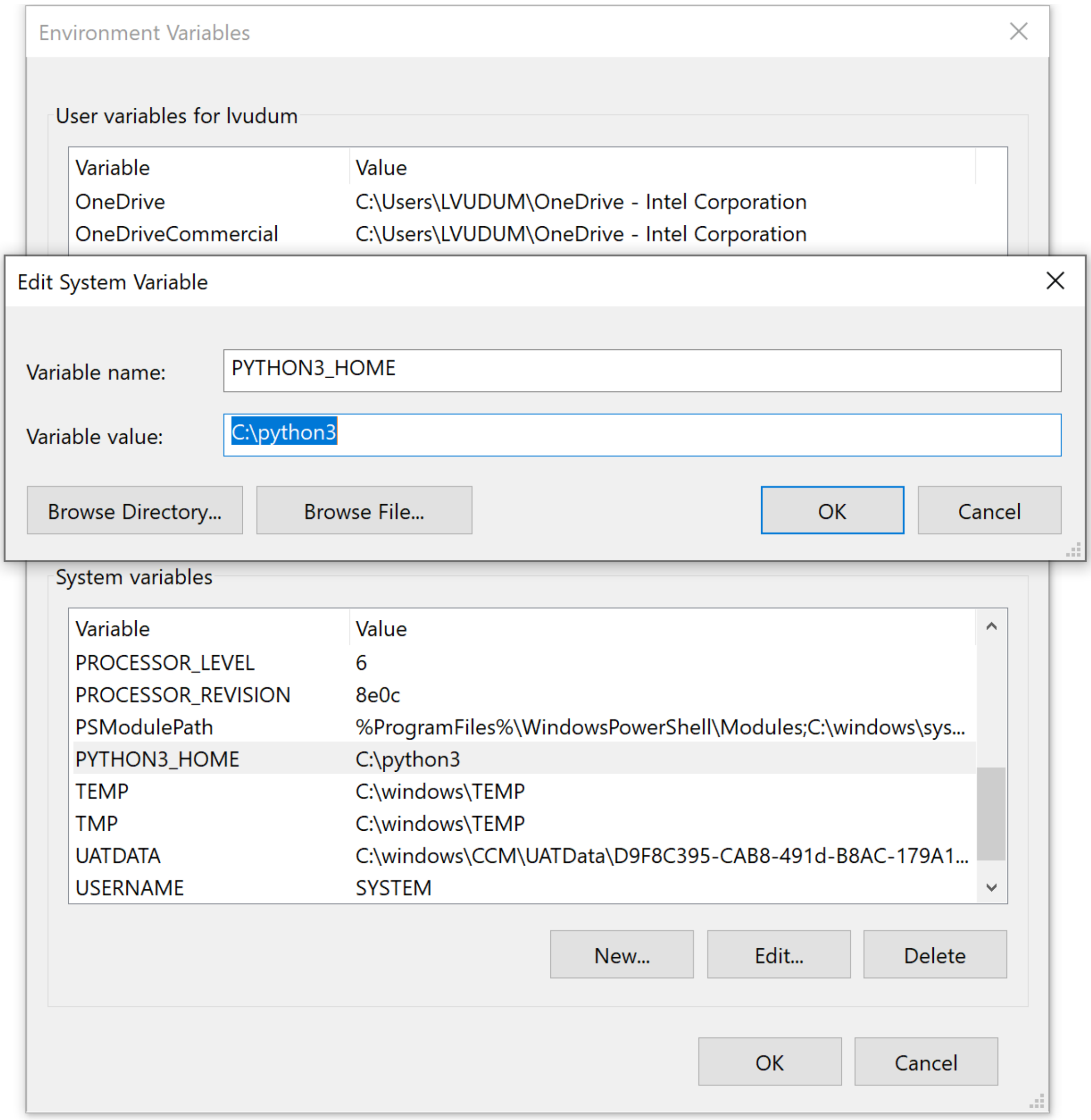
Desktop settings related to your sign-in

Startup and Recovery

System startup, system failure, and debugging information

Figure 4. Windows Environment Variables

    - Add System variable for Python 3.



Environment Variables

User variables for lvudum

Variable	Value
OneDrive	C:\Users\LVUDUM\OneDrive - Intel Corporation
OneDriveCommercial	C:\Users\LVUDUM\OneDrive - Intel Corporation

Edit System Variable

Variable name: PYTHON3_HOME

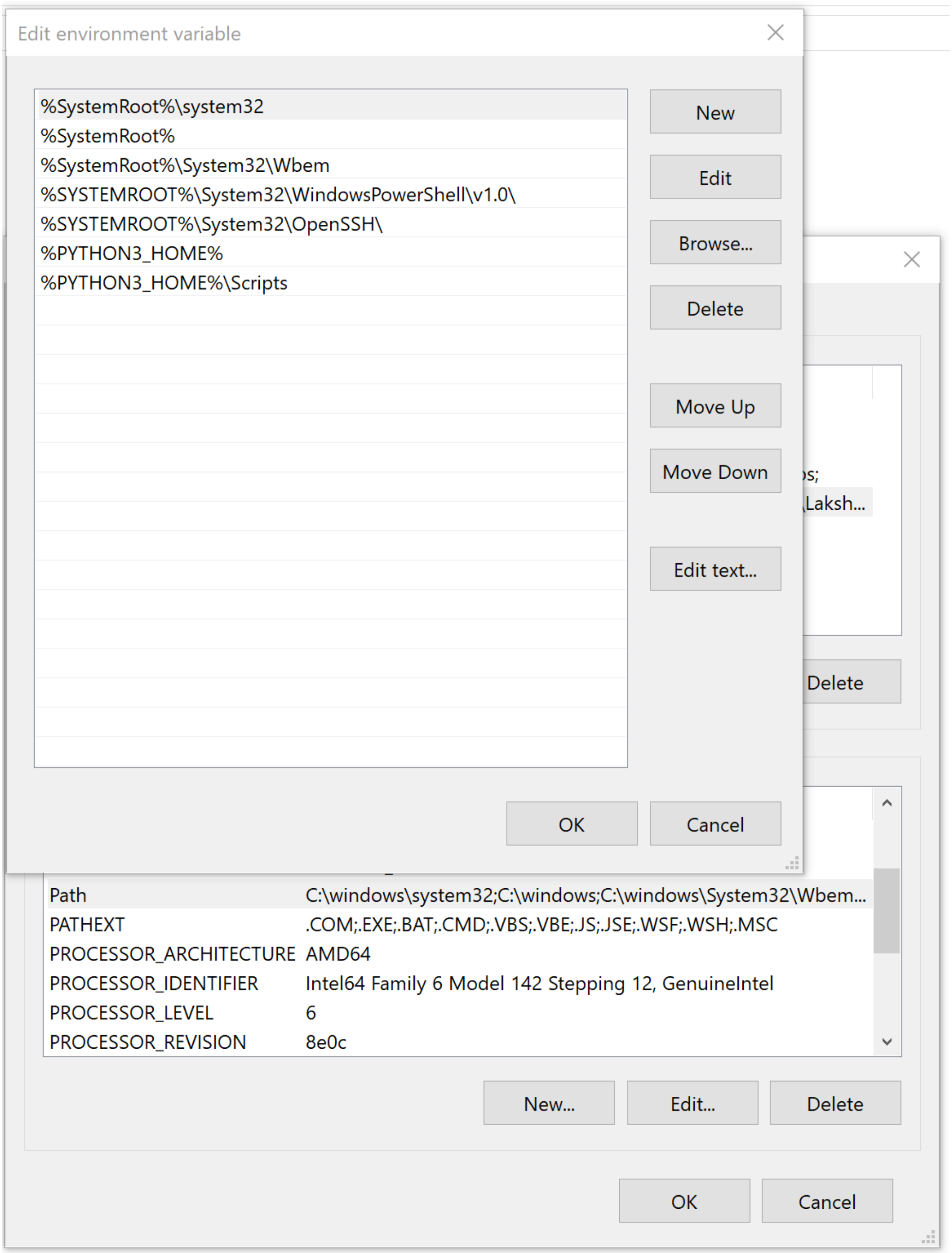
Variable value: C:\python3

System variables

Variable	Value
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	8e0c
PSModulePath	%ProgramFiles%\WindowsPowerShell\Modules;C:\windows\sys...
PYTHON3_HOME	C:\python3
TEMP	C:\windows\TEMP
TMP	C:\windows\TEMP
UATDATA	C:\windows\CCM\UATData\D9F8C395-CAB8-491d-B8AC-179A1...
USERNAME	SYSTEM

Figure 5. Add System variable for Python3

- Add Python 3 path to the environment variable.

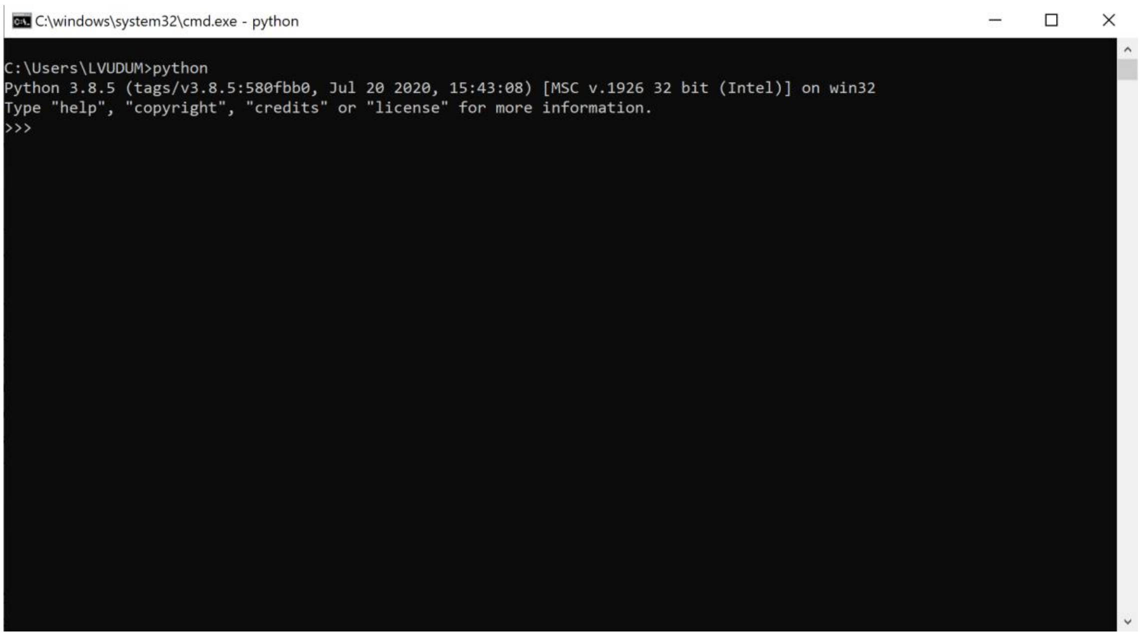


```
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
%PYTHON3_HOME%
%PYTHON3_HOME%\Scripts
```

Path	C:\windows\system32;C:\windows;C:\windows\System32\Wbem...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 142 Stepping 12, GenuineIntel
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	8e0c

Figure 6. Add Python3 path to the environment variable.

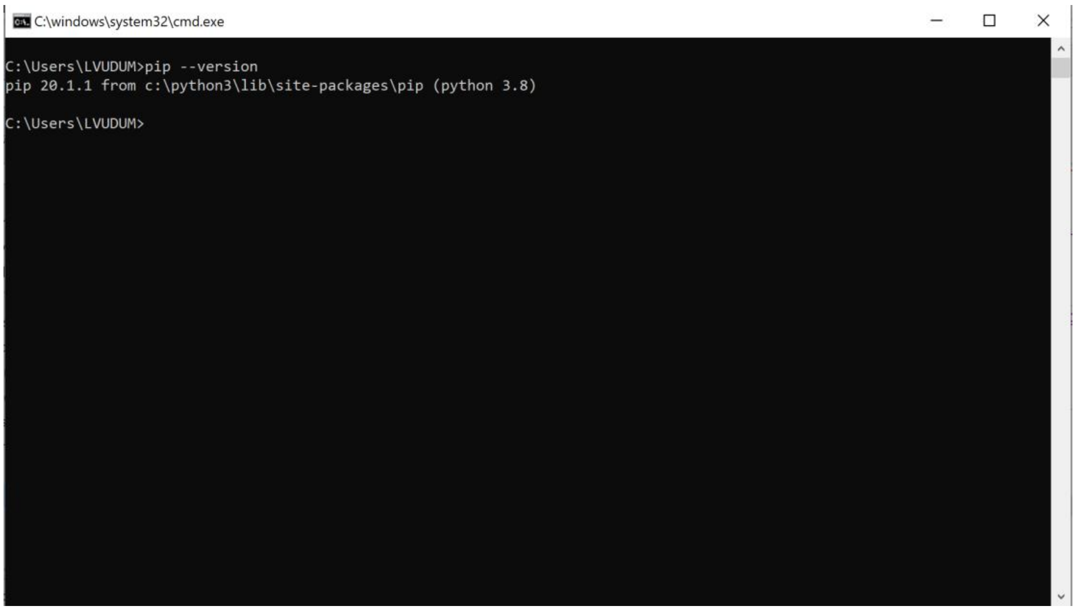
5. To check the python version, go to cmd and type “python” and enter.



C:\windows\system32\cmd.exe - python

```
C:\Users\LVUDUM>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 7. Python version

6. Install pip on windows OS
    - Download get-pip.py file [7].
    - In cmd, navigate to the location of get-pip.py file.
    - Type ”python get-pip.py” command
    - To check pip version, enter “pip --version”
    - 

C:\windows\system32\cmd.exe

```
C:\Users\LVUDUM>pip --version
pip 20.1.1 from c:\python3\lib\site-packages\pip (python 3.8)

C:\Users\LVUDUM>
```

Figure 8. pip version

Once python 3 is installed, install requests package. One of the most downloaded Python packages is requests. Requests package is used to send HTTP request and get data from the websites. Python 3 and pip3 should be installed prior to installing requests in Python. One needs to run the following command to install python requests package "python -m pip install requests"

3.3 Test Environment

The following applications are required to test the product in the test environment. Not just testing the product, even to create a automatic bug through this product, following applications are needed.

1. Python 3 including pip3.
2. Requests in Python
3. Chrome browser.
4. Internet connection.

Refer to chapter 3 for instructions to install the above applications. In addition to these applications, download and install git including gitbash on windows machine [13]. Git is the most popular and commonly used open-sourced version control system. This tool will keep track of all the changes made to the application and it's easy to revert specific changes done to the application. Git allows changes from multiple people and merged into one source. One of the benefits of using git is that git can be accessed either via terminal or GUI. Once these applications are installed in the test machine, clone the development repository from GitHub https://github.com/Vudum/Automatic-bug-filing.git. Instructions to run the script will be explained in detail in the forthcoming chapter.

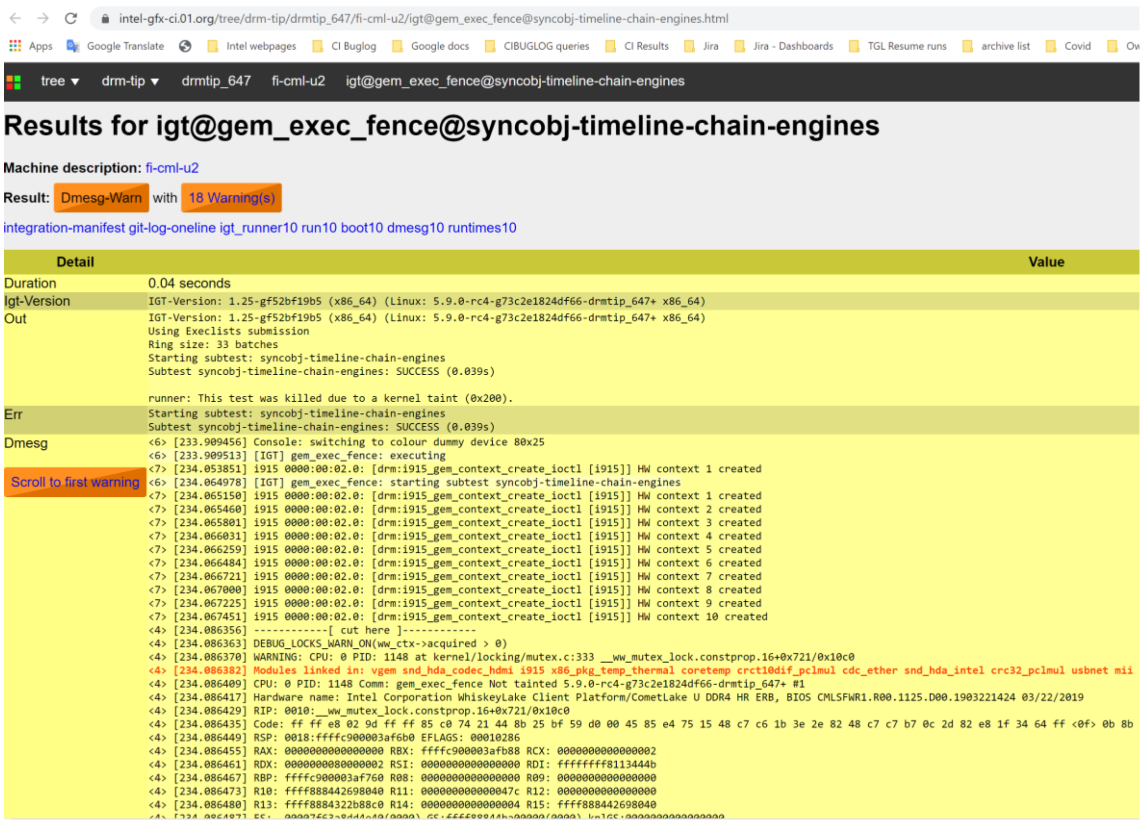
Once the script is run, a bug should have been filed in GitHub. Navigate to GitHub https://github.com/Vudum/Automatic-bug-filing/issues to view the newly created issue and refer chapter 5 for test results and defects that are found during the testing of this product.

4 Product Design

The scope of this thesis is applicable only to web-based ATC test failures. If the test failure report is well organized, neat, and graceful, developing the product will be simple and clear. So, selecting the test failures for this product plays an important role to finish this thesis study. If the test failure reports are uncoordinated and inelegant it will be complex to develop the script to read the failure. In some cases, test cases must be updated so that the test results are organized and legible. With that being said, the selection of test failures will help to develop the product without obstacles.

To develop the product, IGT test failures were chosen to read the failure and file the bug automatically in GitHub. IGT GPU tools are a cumulation of tools to develop and test Linux kernel DRM drivers. IGT tests are written to develop and test Linux kernel DRM/i915 drivers [9]. Linux kernel DRM/i915 is a graphics driver that supports integrated GFX chipsets with both Intel display and rendering blocks [9]. IGT tests [8] are open source which is one of the reasons to choose these test failures. In addition, these test results [10] are well-organized, neat, and developed by experienced engineers which makes bug filling automation possible.

Figure 9 is an example of an IGT test failure. From the failure, it can be noticed that the name of the test is “igt@gem_exec_fence@syncobj-timeline-chain-engines” status name is “`Dmesg-warn`” `and failure summary is` “`DEBUG_LOCKS _WARN_ON(ww_ctx->acquired > 0)`”.



Results for igt@gem_exec_fence@syncobj-timeline-chain-engines

Machine description: fi-cml-u2

Result: Dmesg-Warn with 18 Warning(s)

integration-manifest git-log-oneline igt_runner10 run10 boot10 dmesg10 runtimes10

Detail	Value
Duration	0.04 seconds
Igt-Version	`IGT-Version: 1.25-gf52bf19b5 (x86_64) (Linux: 5.9.0-rc4-g73c2e1824df66-drmtip_647+ x86_64)`

Out
```
IGT-Version: 1.25-gf52bf19b5 (x86_64) (Linux: 5.9.0-rc4-g73c2e1824df66-drmtip_647+ x86_64)
Using Execlists submission
Ring size: 33 batches
Starting subtest: syncobj-timeline-chain-engines
Subtest syncobj-timeline-chain-engines: SUCCESS (0.039s)

runner: This test was killed due to a kernel taint (0x200).
```

Err
```
Starting subtest: syncobj-timeline-chain-engines
Subtest syncobj-timeline-chain-engines: SUCCESS (0.039s)
```

Dmesg
```
<6> [233.909456] Console: switching to colour dummy device 80x25
<6> [233.909513] [IGT] gem_exec_fence: executing
<7> [234.053851] i915 0000:00:02.0: [drm:i915_gem_context_create_ioctl [i915]] HW context 1 created
<6> [234.064978] [IGT] gem_exec_fence: starting subtest syncobj-timeline-chain-engines
<7> [234.065150] i915 0000:00:02.0: [drm:i915_gem_context_create_ioctl [i915]] HW context 1 created
<7> [234.065460] i915 0000:00:02.0: [drm:i915_gem_context_create_ioctl [i915]] HW context 2 created
<7> [234.065801] i915 0000:00:02.0: [drm:i915_gem_context_create_ioctl [i915]] HW context 3 created
<7> [234.066031] i915 0000:00:02.0: [drm:i915_gem_context_create_ioctl [i915]] HW context 4 created
<7> [234.066259] i915 0000:00:02.0: [drm:i915_gem_context_create_ioctl [i915]] HW context 5 created
<7> [234.066484] i915 0000:00:02.0: [drm:i915_gem_context_create_ioctl [i915]] HW context 6 created
<7> [234.066721] i915 0000:00:02.0: [drm:i915_gem_context_create_ioctl [i915]] HW context 7 created
<7> [234.067000] i915 0000:00:02.0: [drm:i915_gem_context_create_ioctl [i915]] HW context 8 created
<7> [234.067225] i915 0000:00:02.0: [drm:i915_gem_context_create_ioctl [i915]] HW context 9 created
<7> [234.067451] i915 0000:00:02.0: [drm:i915_gem_context_create_ioctl [i915]] HW context 10 created
<4> [234.086356] ------------[ cut here ]------------
<4> [234.086363] DEBUG_LOCKS_WARN_ON(ww_ctx->acquired > 0)
<4> [234.086370] WARNING: CPU: 0 PID: 1148 at kernel/locking/mutex.c:333 __ww_mutex_lock.constprop.16+0x721/0x10c0
<4> [234.086382] Modules linked in: vgem snd_hda_codec_hdmi i915 x86_pkg_temp_thermal coretemp crct10dif_pclmul cdc_ether snd_hda_intel crc32_pclmul usbnet mii
<4> [234.086409] CPU: 0 PID: 1148 Comm: gem_exec_fence Not tainted 5.9.0-rc4-g73c2e1824df66-drmtip_647+ #1
<4> [234.086417] Hardware name: Intel Corporation WhiskeyLake Client Platform/CometLake U DDR4 HR ERB, BIOS CMLSFWR1.R00.1125.D00.1903221424 03/22/2019
<4> [234.086429] RIP: 0010:__ww_mutex_lock.constprop.16+0x721/0x10c0
<4> [234.086435] Code: ff ff e8 02 9d ff ff 85 c0 74 21 44 8b 25 bf 59 d0 00 45 85 e4 75 15 48 c7 c6 1b 3e 2e 82 48 c7 c7 b7 0c 2d 82 e8 1f 34 64 ff <0f> 0b 8b
<4> [234.086449] RSP: 0018:ffffc900003af6b0 EFLAGS: 00010286
<4> [234.086455] RAX: 0000000000000000 RBX: ffffc900003afb88 RCX: 0000000000000002
<4> [234.086461] RDX: 0000000080000002 RSI: 0000000000000000 RDI: ffffffff8113444b
<4> [234.086467] RBP: ffffc900003af760 R08: 0000000000000000 R09: 0000000000000000
<4> [234.086473] R10: ffff888442698040 R11: 000000000000047c R12: 0000000000000000
<4> [234.086480] R13: ffff8884322b88c0 R14: 0000000000000004 R15: ffff888442698040
<4> [234.086487] FS:  00007f63a8dd4e40(0000) GS:ffff88844ba00000(0000) knlGS:0000000000000000
```

Figure 9. Example of an IGT Test Failure

These are the three items which need to be incorporated into bug title according to SYS_REQ_1a from chapter 2.2 Functional Requirements. Based on the functional requirement SYS_REQ_1b bug description shall include test environment which is “`IGT-Version: 1.25-gf52bf19b5 (x86_64) (Linux: 5.9.0-rc4-g73c2e1824df66-drmtip_647+ x86_64)`” and description of the failure e.g.

```
<4> [306.448587] DEBUG_LOCKS_WARN_ON(ww_ctx->acquired > 0)
<4> [306.448592] WARNING: CPU: 7 PID: 1088 at kernel/locking/mutex.c:333 __ww_mutex_lock.constprop.16+0x721/0x10c0
<4> [306.448602] Modules linked in: vgem snd_hda_codec_hdmi i915 x86_pkg_temp_thermal coretemp crct10dif_pclmul snd_hda_intel crc32_pclmul snd_intel_dspcfg cdc_ether usbnet mii snd_hda_codec
```

```
ghash_clmulni_intel snd_hwdep snd_hda_core snd_pcm
e1000e ptp pps_core intel_lpss_pci prime_numbers
<4> [306.448625] CPU: 7 PID: 1088 Comm:
gem_exec_fence Not tainted 5.9.0-rc4-g73c2e1824df66-
drmtip_647+ #1
<4> [306.448632] Hardware name: Intel Corporation
WhiskeyLake Client Platform/CometLake U DDR4 HR ERB,
BIOS CMLSFWR1.R00.1125.D00.1903221424 03/22/2019
<4> [306.448641] RIP: 0010:__ww_mutex_lock.con-
stprop.16+0x721/0x10c0
```

From the above failure from figure 9, line `<4> [306.448587] DEBUG_LOCKS_WARN_ON(ww_ctx->acquired > 0)` explains what the issue is and apparently debugging starts from this point. That's one reason why it is included in bug title and bug description. From this line, <4> is the log level, [306.448587] is the timestamp in seconds which means 306.448587th second after the boot. There are 8 different log levels starting from 0 to 7 [11]. The lower the log level identifier the higher is the severity. Log level <0> is KERN_EMERG which is the highest level of severity indicating system inability or crashes. Log level <1> KERN_ALERT is the second most severe which needs user attention immediately. Next Log level <2> is identified by string KERN_CRIT is indicating the user about critical issues either in the HW or SW. Log level <3> is identified as KERN_ERR which is the next lower level of severity. This indicates the failure is a non-critical error but failed or problematic device recognition or driver related issue. The next log level is <4> which is identified as KERN_WARNING which indicates the issue to throw warnings or non-imminent issues. Log level <5> is KERN_NOTICE which is worth to see the notice whereas log level <6> is KERN_INFO that exhibits informational messages. KERN_DEBUG is got log level <7> which is used for debugging.

The product is designed in such a way that all the test failure links (HTML pages) are put into a document called failres.txt. When the script is run, the script will read each failure and finally creates a bug in GitHub based on the requirements mentioned in Chapter 2.2. Once filing a bug for the first failure, the script will proceed to address the next failure.

Below Figure 10 is the structure of the design. Development folder from Figure 10 contains failures.txt file and Bug_replication.py python file where failures.txt file contains a list of failures or links to HTML pages and Bug replication.py is the script which will login to GitHub with provided credentials and creates a session. Thereafter it will push the issue title and issue description and file a bug.

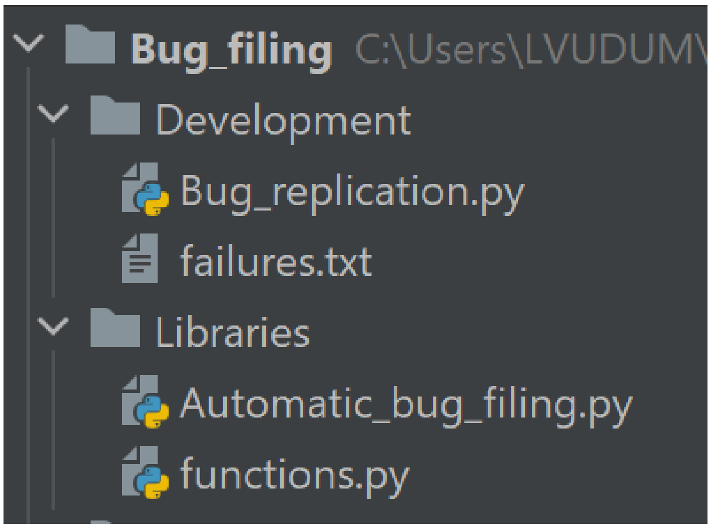
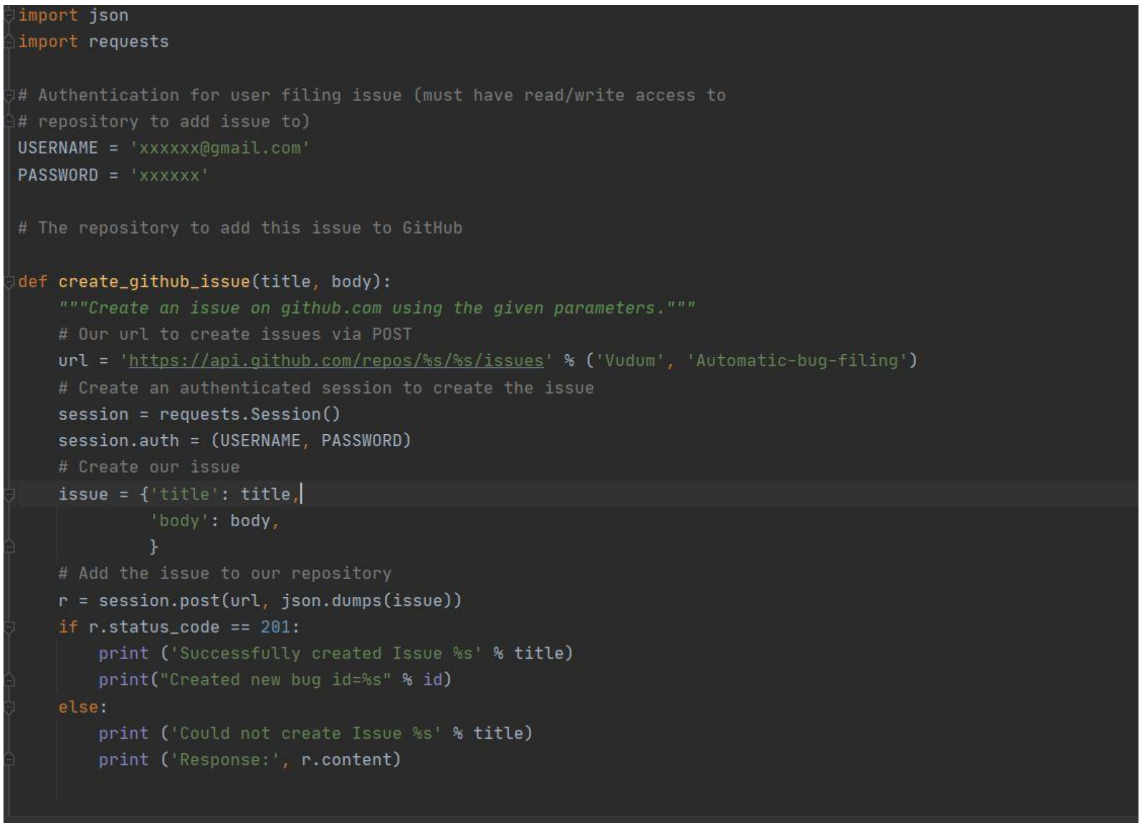


Figure 10. Design files

4.1 Bug Replication script

Below is the bug replication script written in python 3 scripting language. Bug_replication.py script will login to the GitHub by providing the default data to the request methods and create a bug based on issue title and description. Going into the details, json and request methods needs to be imported into this file. JSON is a built-in package in Python used for storing and exchange data. Json.dumps() will convert python object into string whereas requests library will allow to send HTTP requests to the website and receive response as well. Using request library, website headers, form data and few parameters



```python
import json
import requests

# Authentication for user filing issue (must have read/write access to
# repository to add issue to)
USERNAME = 'xxxxxx@gmail.com'
PASSWORD = 'xxxxxx'

# The repository to add this issue to GitHub

def create_github_issue(title, body):
    """Create an issue on github.com using the given parameters."""
    # Our url to create issues via POST
    url = 'https://api.github.com/repos/%s/%s/issues' % ('Vudum', 'Automatic-bug-filing')
    # Create an authenticated session to create the issue
    session = requests.Session()
    session.auth = (USERNAME, PASSWORD)
    # Create our issue
    issue = {'title': title,
             'body': body,
             }
    # Add the issue to our repository
    r = session.post(url, json.dumps(issue))
    if r.status_code == 201:
        print ('Successfully created Issue %s' % title)
        print("Created new bug id=%s" % id)
    else:
        print ('Could not create Issue %s' % title)
        print ('Response:', r.content)
```

Figure 11. Bug replication script [12]

There is only one function that's in Bug_replication.py that is create_github_issue which takes two arguments and creates an issue in GitHub. Two arguments are title and body and they will be coming from Automatic_bug_filing.py script. When this function is called, a session will be started and authenticated with the URL https://api.github.com/repos/Vudum/Automatic-bug-filing/issues. Once authenticated to the project in the GitHub, the function create_github_issue will dump the issue title and issue description and create an issue. There is a check in the script that if the response code is 201, a message will be shown to the user in the terminal indicating a bug has been created successfully. If the response from the website is not 201, script is designed to show a message in the terminal that it couldn't create the bug.

4.2 Functions Script

Functions script contains a class 'extract' and different methods under the class extract. A class is an object constructor and whereas method is a piece of code that is written and defined as part of the defined class. Python is an object-oriented programming where everything in python is an object which can have properties and methods. In this project, all the objects are created as part of the script Automatic_bug_filing.py which uses the class ‘extract’ present in functions.py script. More details about the objects will be followed in chapter 4.3 Automatic_bug_filing.py whereas object methods are discussed in this section.

Class ‘extract’ contains following methods updated_lines, test_name, bug_status, and error_description. There is a reason for having all these methods and all the methods are used in order to develop the product. All the methods are executed via objects which are created in Automatic_bug_filing.py.

Below is the functions.py script which contains all the class methods for ‘extract’. The first defined method in the script is ‘updated_lines’. The purpose of this method is to convert each line of the failure to a sequence of characters i.e. string. By default, each line of the failure is in a sequence of bytes which is a machine-readable format. So, each line had to be converted to a human-readable format. For that, decode(‘utf-8’) has to be done to convert bytes into strings which is human-readable. This makes it easy to find a part of the line and easy to read the text that appears in the terminal which eventually appears in the bug title and description. If all the lines in the failures are not converted to human-readable format i.e. strings it would be difficult to read and understand the bug title and description which adds more confusion to the readers.

```
import re
import urllib
import urllib.request

class extract:

    def updated_lines(self, Read_failure, change_fail_format):
        self.Read_failure = Read_failure
        self.change_fail_format = change_fail_format
        #print(Read_failure)
        for i in Read_failure:
            #print(i)
            list_format = i.decode('utf-8')
            change_fail_format.append(list_format)
        return change_fail_format

    def test_name(self, failure, igt_test):
        self.failure = failure
        self.igt_test = igt_test
        for line in failure:
            if 'Results for' in line:
                title = line[line.find('igt@'):line.find("</h1>")]
                igt_test.append(title)
        return igt_test
    def bug_status(self, failure, status):
        self.failure = failure
        self.status = status
        for line in failure:
            if 'Result:' in line:
                if '>Fail<' in failure:
                    status.append('Fail')
                elif '>Dmesg-Fail<' in line:
                    status.append('Dmesg-Fail')
                elif '>Dmesg-Warn<' in line:
                    status.append('Dmesg-Warn')
                elif '>Incomplete<' in line:
                    status.append('incomplete')
                elif '>Skip<' in line:
                    status.append('Skip')
        return status

    def error_description(self, failure, status, description, de-
tails):
        self.failure = failure
        self.status = status
        self.description = description
        l = 0
        for line in failure:
            if 'IGT-Version:' in line:
                details.append(line)
                break
        if status == 'Dmesg-Fail' or 'Dmesg-Warn':
            for line in failure:
                if 'lt;3&gt' in line and not 'cut here' in line:
                    text = line[line.find(']') +
1:line.find("</span></div>")]
                    details.append(failure[l-3:l+6])
                    description.append(text)
                    break
                elif 'lt;4&gt' in line and not 'cut here' in line:
                    text = line[line.find(']') +
1:line.find("</span></div>")]
                    details.append(failure[l:l+6])
                    description.append(text)
                    #details.append(full_details)
                    break
                l = l+1
        return description
```

Listing 1. Functions script

The next method under the class ‘extract’ is ‘test_name’ that has arguments ‘failure’ which is the test failure itself and ‘igt_test’ which is an empty list. The objective of this method is to extract the failed test name from the test result webpage. This method will scan each in the test result page and find out the line that contains ‘Results for’ and extract the test name from the line which starts with ‘igt@’. Once the test name is fetched from the results page, the test name will be appended to ‘igt_test’ which is an empty list. More details about ‘igt_test’ argument in test_name method will be elaborated in the next section 4.3 Automatic bug filing. The next following method in ‘extract’ class is bug_status that has two arguments ‘failure’ same as in method test_name and ‘status’ which is an empty list that is used to store the status of the failure. All the test results that are not passed are failed but the failure status can be fail, skip, notrun, incomplete, warning, and crash. All these statuses are considered as test failed. This method will go through the test failure and find out the status in the line that contains ‘Result:’ and store the value in the ‘status’ list.

As of now, test name and fail statuses are found from the test result. The next method ‘error_description’ will find out the summary of the issue which gives an idea about the failure when someone reads the bug title and bug description. This method has arguments ‘failure’ that is the test failure, ‘status’ the failure status, ‘description’ that will be included in the bug title, and ‘details’ which gives details about the failure in the bug description. From this method, ‘details’ argument collects the test environment and adds to the details list, and appends the failure details along with the test environment. The script is written to find out the error details when the status is either dmesg-warn or dmesg-fail. The script has to be updated to find out error details if the failure status is other than dmesg-warn and dmesg-fail. For now, if the status is other than the one mentioned above, the script will fail to execute completely.

So, from functions.py script test name, status name, error summary and error description are obtained.

4.3 Automatic Bug Filing

Automatic_bug_filing.py is a python file that imports all the libraries that are needed to create a bug automatically. This script is designed in such a way that running Automatic_bug_filing.py from a location for example as shown in Figure 10. Design files will create a bug in GitHub here https://github.com/Vudum/Automatic-bug-filing/issues.

Below is the Automatic_bug_filing.py script

```
from Development.Bug_replication import *
import urllib.request
from Libraries.functions import *

file = open('C:\\Users\\LVUDUM\\OneDrive - Intel Corporation\\Desk-
top\\Lakshmi\\Masters\\Thesis\\Bug_filing\\Development\\failures.txt',
'rU')
failures_list = file.readlines()
github_title = []
failure_status = []
test = []
failure_description = []
changed_failure_format = []
github_description = []

read_lines = extract()
print(failures_list)
l = 0
for f in failures_list:
    print(f)
    Openurl = urllib.request.urlopen(failures_list[l])
    Read_failure = Openurl.readlines()

    # changed_failure_format = Change_failure_format is a list where
each line is stored as an element after coverting bytes to string.
    read_lines.updated_lines(Read_failure, changed_failure_format)

    # test = It's name of the test which got failed.
    read_lines.test_name(changed_failure_format, test)

    # failure_status = It's status of the failure e.g, crash, fail,
skip,
    read_lines.bug_status(changed_failure_format, failure_status)

    # Failure_description = This is the first line of the fine that
indicates the failure that will be shown in the bug title in github.
    # github_description = first five lines that indicates the failure
and will be pushed to github issue description.
    read_lines.error_description(changed_failure_format, failure_sta-
tus[0], failure_description, github_description)

    # github_title = It contains name of the test, status of the fail-
ure and failure desription.
    github_title = test[0] + ' -' + failure_status[0] + ' -' + fail-
ure_description[0]

    # convert_github_description = It converts a list to string so
that we can concatenate two stings.
    convert_github_description = failures_list[l]+''.join(github_de-
scription[0])+''.join(github_description[1])

    # Below line of code will create bug with title as well as issue
description. Issue desription contains the link to the failure.
    create_github_issue(github_title, convert_github_description)

    #Following lines of code will clear test name, status of the fail-
ure, failure description, github issue description. So, it is ready to
collect data for the next failure.
    failure_status.clear()
    test.clear()
    failure_description.clear()
    changed_failure_format.clear()
    github_description.clear()
    l = l+1
```

Listing 2. Automatic bug filing.

This script imports Bug_replication.py which has the functionality to replicate the bug to GitHub once the bug title and bug description are fetched from the failure as documented in chapter 4.1 Bug replication script. In addition, this script will also import functions.py library which is capable of extracting bug title and bug description of the issue from the failure as explained in chapter 4.2 Functions script. Firstly, the location of the failures is given which is a file that contains one or more failure links i.e. html pages. failures_list is a list that contains each failure as an element in the list where the no: of failures can be at least 1 or more. Next, empty lists like github_title, failure_status, test, failure_description, changed_failure_format, github_description are created to store the following data:

- changed_failure_format: This list will store each line of the failure as an element in the list where element will be in human-readable format.
- test: This list will store the name of the test that got failed.
- failure_status: This will store only the status of the failure for example fail, dmesg-arn, dmesg-fail, skip, etc.
- failure description: This list will store the main error or failure from the test failure i.e. the summary of the error/failure.
- github_title: This list will store the whole GitHub issue title that includes test name, status name and, failure summary.
- github_description: This list will store failure description which explains where and when the failure started happening.

In this code, class extract from functions.py is used to create a new object instance and assigned to a variable read_lines. This object is used to access all the different attributes of the extract class. Once the Automatic_bug_filing.py is run, the script will open and read the test failure and convert each line of the failure to human-readable format i.e. strings. Once each line of the failure is converted to strings, using extract class attributes, all required details are gathered and stored into the empty lists to file a bug which is also mentioned in the script above. Finally, a bug will be created in GitHub. Every time when a bug is created all the created empty lists will be reset as they store values from the previous test failure. Values are reset so that these lists will again store values for the next test failure if exists any. This will repeat until all failures are addressed and bugs are filed for all the failures.

5 Test Results and Analysis

After developing the product, the test case mentioned in chapter 1.3 has been executed. Testing has undergone for few runs to develop a high-quality product. Here are the test results from round 1.

Pre-condition:

1. Ensure the test setup is ready.
   Refer Chapter 4 for Test Environment
2. Web-based Test failures (URL's) are available in a file.
3. Internet connection is available.

Test steps:

1. Run the script to address the failures.

Expected Result:

1. The product shall address all the test failure reports automatically.
2. Bug reports are available for each failure in GitHub.
3. Bug report shall contain all relevant details about the failure.

Test status (Pass/Fail): **Fail**

Actual Result:

- Bug title contains a link to the test failure instead of failure summary.
  - Refer bug https://github.com/Vudum/Automatic-bug-filing/issues/20

During the first round of testing bug #20 has been found. According to the requirement, SYS_REQ_1a bug title shall contain the name of the test, status type, and failure summary but the bug title contains a link to like "igt@kms_atomic_transition@1x-modeset-transitions - Dmesg-Warn - https://intel-gfx-ci.01.org/tree/drm-tip/drmtip_610/fi-tgl-dsi/igt@kms_atomic_transition@1x-modeset-transitions.html" which shouldn't be in the bug title. Hence, the test case has been failed in the first round. The issue has been fixed as there was a bug in the script. Once the bug has been fixed testing has been started to perform the second round of testing. Here are the results from round 2 testing.

Pre-condition:

1. Ensure the test setup is ready.
   Refer Chapter 4 for Test Environment
2. Web-based Test failures (URL's) are available in a file.

3. Internet connection is available.

Test steps:

1. Run the script to address the failures.

Expected Result:

1. The product shall address all the test failure reports automatically.
2. Bug reports are available for each failure in GitHub.
3. Bug report shall contain all relevant details about the failure.

Test status (Pass/Fail): **Fail**

Actual Result:

- The test name in the bug title is not matching in bug description. The link to the test failure must be appropriate.
  - Refer bug https://github.com/Vudum/Automatic-bug-filing/issues/43

In the second round of testing, there was a new bug found where the test failure link in the bug description is incorrect, pointing towards another test failure rather than the test mentioned in the bug title. This will create confusion for the developer when investigating the issue and the developer must put additional effort to find the appropriate link to debug the issue appropriately. So, this issue has been fixed for the third round of testing. Below are the round 3 test results.

Pre-condition:

4. Ensure the test setup is ready.
   Refer Chapter 4 for Test Environment
5. Web-based Test failures (URL's) are available in a file.
6. Internet connection is available.

Test steps:

2. Run the script to address the failures.

Expected Result:

4. The product shall address all the test failure reports automatically.
5. Bug reports are available for each failure in GitHub.

6. Bug report shall contain all relevant details about the failure.

Test status (Pass/Fail): **Fail**

Actual Result:

- Issue summary in the bug title is not informative.
    - Refer bug https://github.com/Vudum/Automatic-bug-filing/issues/65

In the third round of testing, a very interesting bug was found as this issue might happen every time. Bug title contains "igt@gem_exec_reloc@basic-gtt-wc-active -Dmesg-Fail -------------[cut here]------------". The summary of the failure doesn't really indicate anything about the failure. This had to be fixed according to the requirement SYS_REQ_1a. This issue happens only for certain test failures and this has to be fixed so that it shouldn't happen for any type of failure. The issue has been fixed for the fourth round of testing. Here are the test results from the fourth round of testing.

Pre-condition:

4. Ensure the test setup is ready.
   Refer Chapter 4 for Test Environment
5. Web-based Test failures (URL's) are available in a file.
6. Internet connection is available.

Test steps:

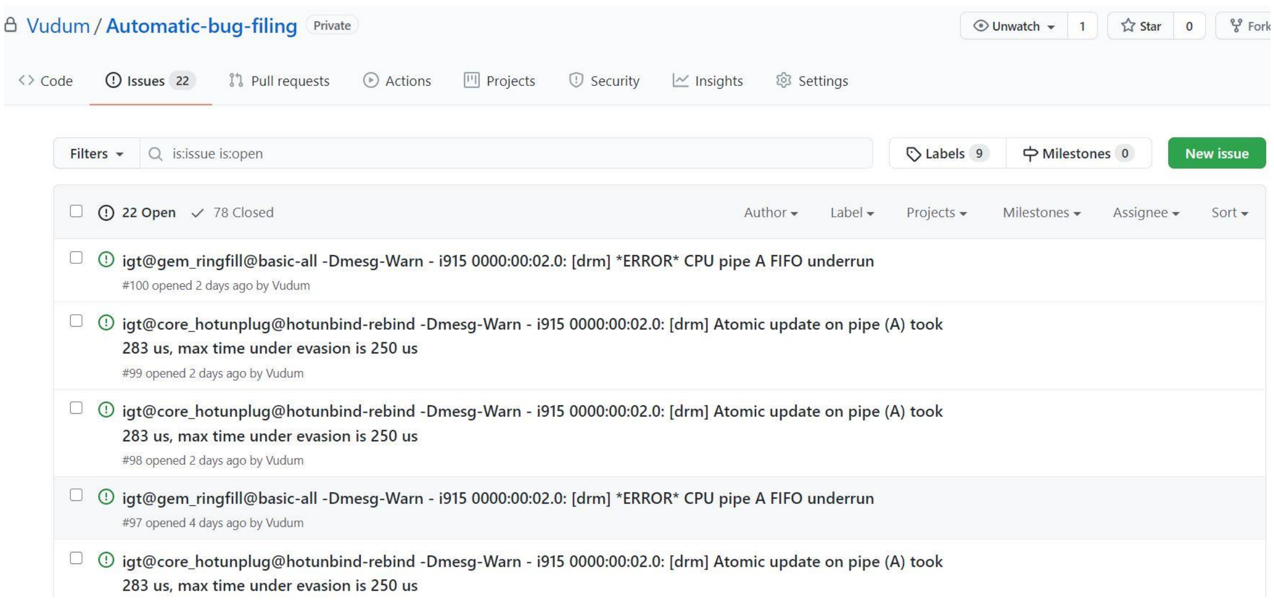
2. Run the script to address the failures.

Expected Result:

4. The product shall address all the test failure reports automatically.
5. Bug reports are available for each failure in GitHub.
6. Bug report shall contain all relevant details about the failure.

Test status (Pass/Fail): **PASS**

The fourth round of testing has been passed and testing has been completed successfully. There are no open bugs for this product and all the issues found during testing have been fixed and verified. Below a couple of screenshots are attached that shows how bugs look like when created using this product.



igt@gem_ringfill@basic-all -Dmesg-Warn - i915 0000:00:02.0: [drm] *ERROR* CPU pipe A FIFO underrun
#100 opened 2 days ago by Vudum

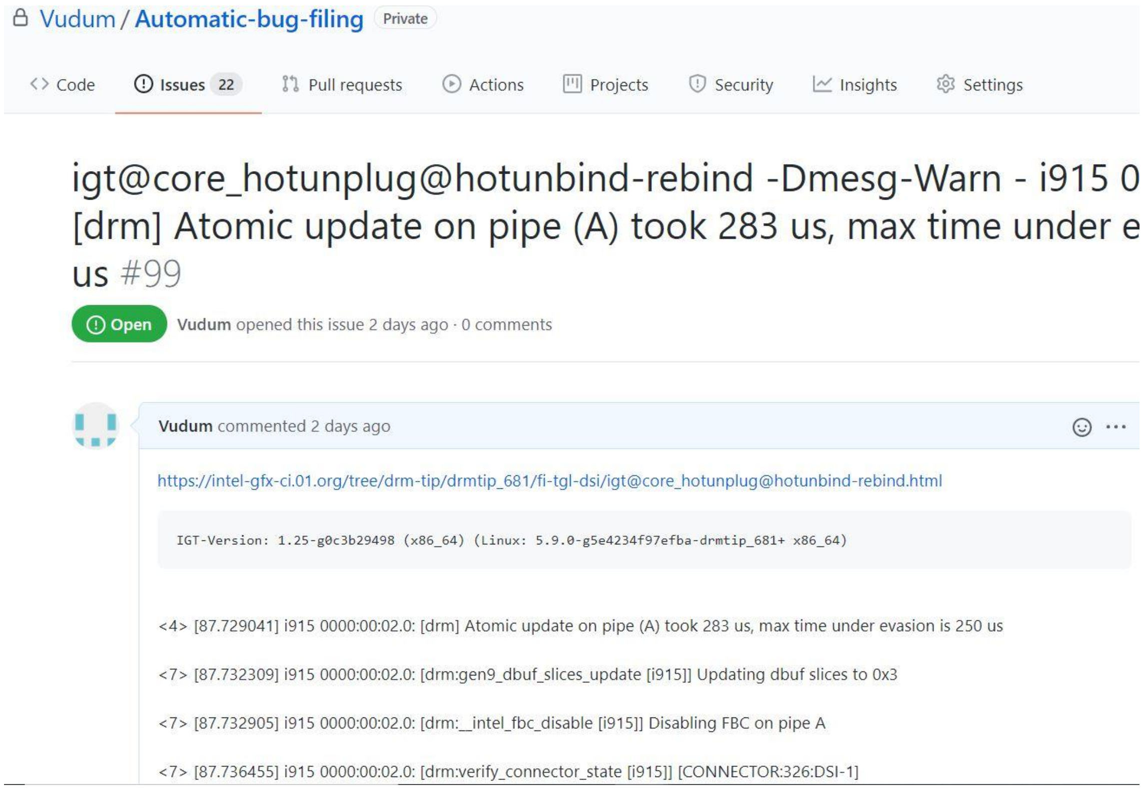
igt@core_hotunplug@hotunbind-rebind -Dmesg-Warn - i915 0000:00:02.0: [drm] Atomic update on pipe (A) took 283 us, max time under evasion is 250 us
#99 opened 2 days ago by Vudum

igt@core_hotunplug@hotunbind-rebind -Dmesg-Warn - i915 0000:00:02.0: [drm] Atomic update on pipe (A) took 283 us, max time under evasion is 250 us
#98 opened 2 days ago by Vudum

igt@gem_ringfill@basic-all -Dmesg-Warn - i915 0000:00:02.0: [drm] *ERROR* CPU pipe A FIFO underrun
#97 opened 4 days ago by Vudum

igt@core_hotunplug@hotunbind-rebind -Dmesg-Warn - i915 0000:00:02.0: [drm] Atomic update on pipe (A) took 283 us, max time under evasion is 250 us

Figure 12. Open issues



igt@core_hotunplug@hotunbind-rebind -Dmesg-Warn - i915 0
[drm] Atomic update on pipe (A) took 283 us, max time under e
us #99

Open Vudum opened this issue 2 days ago · 0 comments

Vudum commented 2 days ago

https://intel-gfx-ci.01.org/tree/drm-tip/drmtip_681/fi-tgl-dsi/igt@core_hotunplug@hotunbind-rebind.html

```
IGT-Version: 1.25-g0c3b29498 (x86_64) (Linux: 5.9.0-g5e4234f97efba-drmtip_681+ x86_64)
```

```
<4> [87.729041] i915 0000:00:02.0: [drm] Atomic update on pipe (A) took 283 us, max time under evasion is 250 us

<7> [87.732309] i915 0000:00:02.0: [drm:gen9_dbuf_slices_update [i915]] Updating dbuf slices to 0x3

<7> [87.732905] i915 0000:00:02.0: [drm:__intel_fbc_disable [i915]] Disabling FBC on pipe A

<7> [87.736455] i915 0000:00:02.0: [drm:verify_connector_state [i915]] [CONNECTOR:326:DSI-1]
```

Figure 13. Issue created using tool

6 Discussions and Conclusions

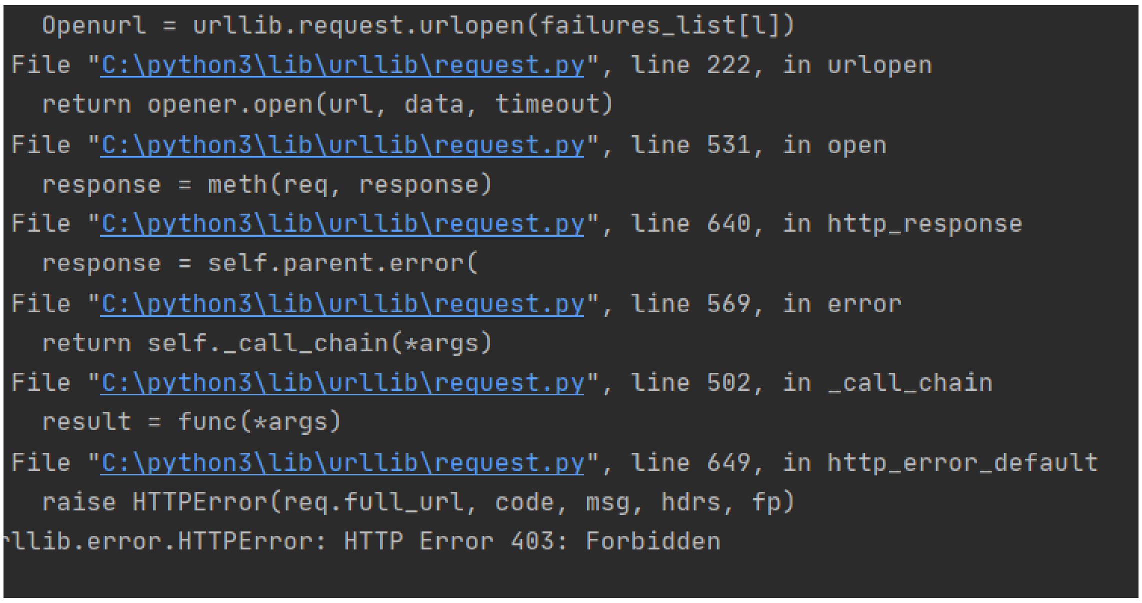
Based on the test results from chapter 5, when the script is run from the test or development environment, web-based test failures are addressed, and bugs are filed automatically. The objective evidence from the test results shows that the developed product fulfills the product specifications.

In spite of the fact that the product meets the specifications, there are few improvements that can still be made to the product which is explained in this chapter. Also, there were a few challenges that were faced during the development of the product which are also explained in this chapter.

Firstly, finding appropriate web-based test failures was very challenging as this plays a very vital role in developing this product. If not, this thesis would have become a never-ending project with continuous changes dealing with inconsistent test failures. IGT test failures are the best suitable results to develop this product as they are neat, uniform, and structured. Another challenge is that testing has to be performed thoroughly. Since all the test failures might not be well organized, this product has to be tested heavily to deliver a high-quality product to meet the product specifications. All failures might not be the same and a good example is a bug https://github.com/Vudum/Automatic-bug-filing/issues/65 . Finding these kinds of bugs are very important as this can occur in real-time. Apart from the challenges, there are potential areas identified to improve the functionality, usability, and compatibility of the product.

The first and foremost area to improve is that the product should avoid creating duplicate bugs. At the moment a bug is created based on the failure, but the product should ensure whether a similar bug exists already in the system or not. This is an enormous feature and implementing this would require a lot a time for the development and testing the feature. If the product generates duplicate bugs, the development team has to put additional effort to find the original bug and close the issue. This can be easily avoided given that this feature would be implemented. On the other side, implementing this feature is very challenging and time taking. Adding this functionality to the product will give a lot of value to the product. At the moment this product files bugs for only a couple of statuses such as dmesg-warn and dmesg-fail which can be improved further to cover other statuses such as fail, skip, crash, etc.

The next area that can be developed is that, for some reason, if the product is not able to file a bug automatically for a failure, the product should skip the current failure and go to the next failure so that a problematic failure won't block the rest of the execution of the script. The Figure 14 Execution interruption shows how the logs looks like when the execution is interrupted. This is a significant feature that can be implemented to the current product as this will save a lot of execution time and avoids manual intervention regularly.



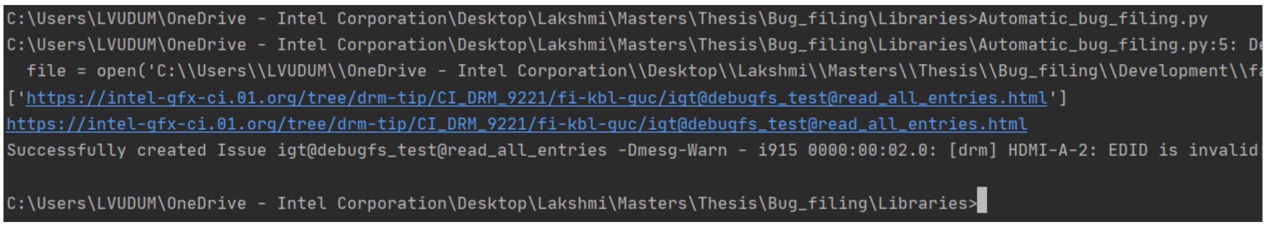
```
  Openurl = urllib.request.urlopen(failures_list[l])
File "C:\python3\lib\urllib\request.py", line 222, in urlopen
  return opener.open(url, data, timeout)
File "C:\python3\lib\urllib\request.py", line 531, in open
  response = meth(req, response)
File "C:\python3\lib\urllib\request.py", line 640, in http_response
  response = self.parent.error(
File "C:\python3\lib\urllib\request.py", line 569, in error
  return self._call_chain(*args)
File "C:\python3\lib\urllib\request.py", line 502, in _call_chain
  result = func(*args)
File "C:\python3\lib\urllib\request.py", line 649, in http_error_default
  raise HTTPError(req.full_url, code, msg, hdrs, fp)
llib.error.HTTPError: HTTP Error 403: Forbidden
```

Figure 14. Execution interruption

Furthermore, this product was developed and tested in Windows environment as defined in the scope of the thesis. The product should be tested also in Linux and MAC OS and see if the product gives the same result as in Windows. If changes are needed, those can be made to the product so that the coverage of this product increases substantially. As part of this thesis, this product was tested only on a Windows machine.

One last improvement that was identified during the testing is that when a bug is created, it would be really good to see the bug id in the log. See Figure 15 Execution log to view the output of the execution of the script. The log shows the bug title which includes a test name, a status name, and a failure summary. It would be good to see the bug id as well in the log so that looking at the log one can navigate to the most interesting or some other important bug. If this feature is not implemented one has to navigate to the GitHub and search for the bug.

These are the few other improvements that identified during the development and testing phase, but they were left out from this study.



```
C:\Users\LVUDUM\OneDrive - Intel Corporation\Desktop\Lakshmi\Masters\Thesis\Bug_filing\Libraries>Automatic_bug_filing.py
C:\Users\LVUDUM\OneDrive - Intel Corporation\Desktop\Lakshmi\Masters\Thesis\Bug_filing\Libraries\Automatic_bug_filing.py:5: De
  file = open('C:\\Users\\LVUDUM\\OneDrive - Intel Corporation\\Desktop\\Lakshmi\\Masters\\Thesis\\Bug_filing\\Development\\fa
['https://intel-gfx-ci.01.org/tree/drm-tip/CI_DRM_9221/fi-kbl-guc/igt@debugfs_test@read_all_entries.html']
https://intel-gfx-ci.01.org/tree/drm-tip/CI_DRM_9221/fi-kbl-guc/igt@debugfs_test@read_all_entries.html
Successfully created Issue igt@debugfs_test@read_all_entries -Dmesg-Warn - i915 0000:00:02.0: [drm] HDMI-A-2: EDID is invalid

C:\Users\LVUDUM\OneDrive - Intel Corporation\Desktop\Lakshmi\Masters\Thesis\Bug_filing\Libraries>
```

Figure 15. Execution log

One of the key learnings while doing this thesis is that, how to portrait a problem statement and develop a solution for that statement. In addition to that, python programming language is another key learning from this thesis where I learnt some concepts and implemented them. Also, I used GitHub to file bugs and maintain code which I had never used in my workplace. This thesis gave me a path to become an automation engineer and helps my career to grow further.

References

1 Tutorialspoint, SDLC-Waterfall Model. Available from: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

2 Visual Paradigm, Scrum vs Waterfall vs Agile vs Lean vs Kanban. Available from:https://www.visual-paradigm.com/scrum/scrum-vs-waterfall-vs-agile-vs-lean-vs-kanban/

3 PyCharm, Download PyCharm. Available from: https://www.jetbrains.com/pycharm/download/#section=windows

4 STXNEXT, Python vs. Other Programming Languages. Available from: https://www.stxnext.com/Python-vs-other-programming-languages/

5 Phoenixnap, How To Install Python 3 on Windows 10. Available from: https://phoenixnap.com/kb/how-to-install-python-3-windows

6 Python, Download the latest version for Windows. Available from: https://www.python.org/downloads/

7 Bootstrap, Install pip 3. Available from: https://bootstrap.pypa.io/get-pip.py

8 Gitlab, IGT GPU Tools. Available from: https://gitlab.freedesktop.org/drm/igt-gpu-tools/-/blob/master/README.md

9 01.org INTEL OPEN SOURCE, Intel Graphics Driver. Available from: https://01.org/linuxgraphics/gfx-docs/drm/gpu/i915.html

10 Intel GFX CI, IGT test results. Available from: https://intel-gfx-ci.01.org/tree/drm-tip/drmtip_647/fi-cml-u2/igt@gem_exec_fence@syncobj-backward-timeline-chain-engines.html

11 Linux Config, Introduction to Linux kernel log levels. Available from: https://linuxconfig.org/introduction-to-the-linux-kernel-log-levels

12 GitHub, Make an issue on github using API V3 and Python. Available from: https://gist.github.com/JeffPaine/3145490

13 Git, Downloading Git. Available from: https://git-scm.com/download/win