Development of PyAutoGUI Tool

Yu SU
computer science
City University of Hongkong
yusu27-c@my.cityu.edu.hk

Tong Zhang
computer science
City University of Hongkong
tzhang273-c@my.cityu.edu.hk

Xinpin XU
computer science
City University of Hongkong
xinpinxu2-c@my.cityu.edu.hk

Peiyu Li
computer science
City University of Hongkong
peiyuli4-c@my.cityu.edu.hk

Jianan ZHOU
computer science
City University of Hongkong
jianazhou2-c@my.cityu.edu.hk

***Abstract*—PyAutoGui is a cross-platform GUI automation module. However, there are still some drawbacks in this prevalent tool. And the object of this project is to improve the ability of utilizing different software engineering model in the developing process. So we set PyAutoGui as the target of our project.**

***Index Terms*—PyAutoGUI, scrum, modern code review, fuzzy, test oracle, code smell, refactor**

I. Introduction

PyAutoGUI is a cross-platform GUI automation Python module for human beings. The purpose is used to programmatically control the mouse and keyboard. PyAutoGUI can simulate moving the mouse, clicking the mouse, dragging with the mouse, pressing keys, pressing and holding keys, and pressing keyboard hotkey combinations. The platform that Pyautogui supported include windows, OSX and linux. It is actually an automated test tool for the GUI related project.

II. Motivation

To practice how to develop tools by the software engineering approach, Pyautogui is a great target for us. On the one hand, the code of it is not too hard for us to modify or understand. On the other hand, it is open-source on the github, it is easy for us to get the code.

III. Object

About the development of Pyautogui, we focus mainly on the mouse control on OSX platform, since it is a large process and we only have five group members. What we decided to do is in three aspects:

- increase the code readability
- improve the performance of mouse control
- test the tolerance of each functions.

The rest of paper shows the process that how we develop this project step by step.

IV. Process

In our project we use an agile method called scrum tools and meetings to make our process.

Identify applicable funding agency here. If none, delete this.

A. Scrum

Scrum is an iterative and incremental framework for managing product development. It defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal", challenges assumptions of the "traditional, sequential approach" to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines involved.

- product backlog: is a list of features desired for a final product, the bugs to be removed, technical work to set up and maintain development environment and user site, and knowledge (e.g., learn to use a new framework) to acquire by the project.
- release burndown chart: tracks progress on a project. The chart itself is updated after each sprint. Teams can measure progress in any unit they choose.
- sprint backlog: is a list of tasks to complete during a sprint. It is updated once a day.
- task board: is a sheet that every member of the team can use and add to over the course of a sprint, and is a visual representation of every task and what phase of completion its in. Usually, task boards include columns for stories, to-dos, work in process, things needing verification and items finished. Some teams also include burndown charts, notes and tests. Hang the board on a wall or digitalize it.
- Meeting: The product owner shows up at the sprint planning meeting with the prioritized agile product backlog and describes the top items to the team. The team then determines which items they can complete during the coming sprint. The team then moves items from the product backlog to the sprint backlog.In doing so, they expand each Scrum product backlog item into one or more sprint backlog tasks

B. Our Process

In this part, I will introduce the process schedule in our project. first I will introduce the requirement in our project.

there are six main requirements: code review, testing, code smell, refactor, result compare and video. the process has four sprint:

- sprint 1:
  In this sprint, we focus on the code review and the video making about this part.(Section 3)

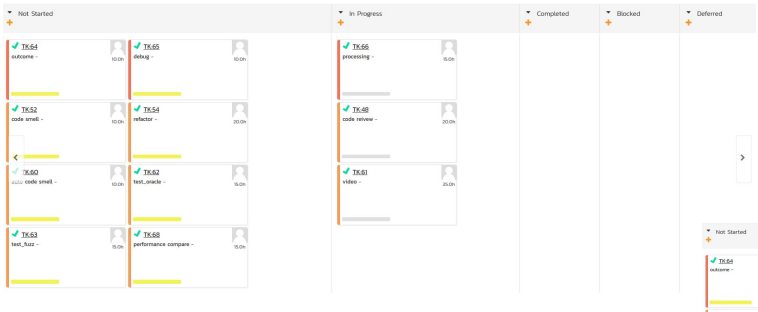

Fig. 1. before sprint 1.

Fig. 2. after sprint 1.

- sprint 2:
  In this sprint, the main task is testing and debug. testing contains fuzz and oracle two parts.(Section 4)

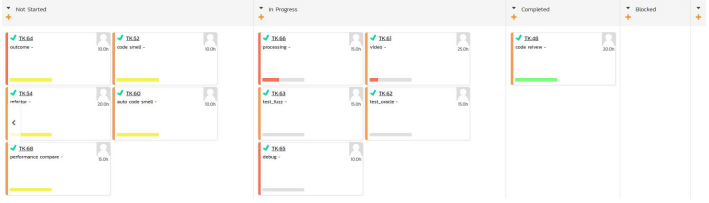

Fig. 3. before sprint 2.

Fig. 4. after sprint 2.

- sprint 3:
  In this sprint, we try to do the code smell(Section 5) and refactor.(Section 6)

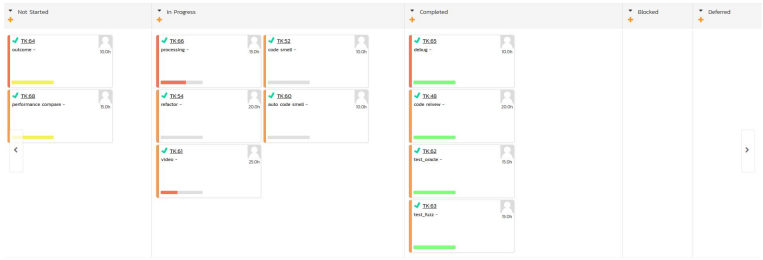

Fig. 5. before sprint 3.

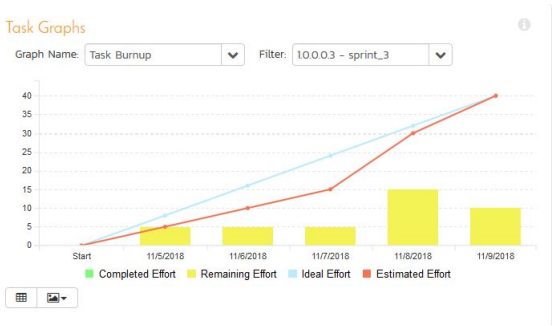

Fig. 6. after sprint 3.

- sprint 4:
  In this sprint, our goal is do some result compare and finish all the other work like video. At last we get the outcome.

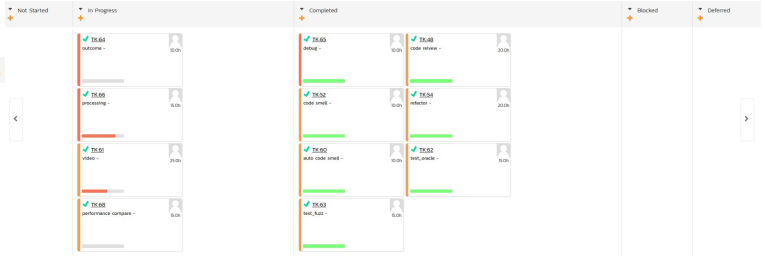

Fig. 7. before sprint 4.

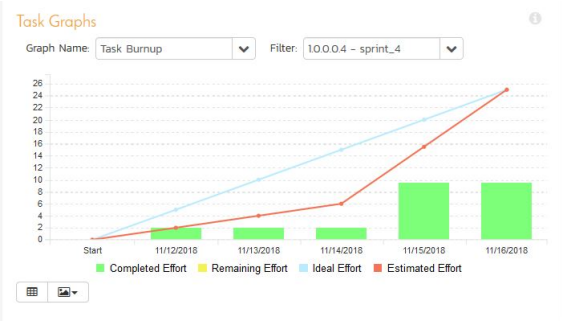

Fig. 8. after sprint 4.

V. Modern Code Review

A. preview

Modern code review is a lightweight and informal process for inspecting the source code by developers, it help development teams make intelligent, informed decisions [1]. More than that, modern code review can help developers analyze the codes asynchronous, so that author and reviewers don't need to be engaged on the same task simultaneously to improve the efficiency of the understanding of a code.

In our project, modern code review is the first process before we starting the development of the Pyautpgui Tool. It performs a better and faster code readability and maintainability, help us have a prior knowledge of the context and the code complete reviews, solve the development problem and prevent some accidence.

B. code review processes

a) the source code understanding: In the first step , one of our team member plays the author role who study and annotate the source codes. The source codes generally contains two main parts. The first init.py part defines some general functions and the mouse movement operation functions. The simplified code below shows some definition functions and the comments of each of them.

Listing 1. General functions

```
def getPointOnLine(x1, y1, x2, y2, n):
    """Returns the (x, y) tuple of the point that has
    progressed a proportion n along the line defined by
    the two x, y coordinates."""
def linear(n):
    """Trivial linear tweening function."""
def position(x=None, y=None):
    """Returns the current xy coordinates of the mouse
    cursor as a two-integer tuple."""
def size():
    """Returns the width and height of the screen."""
def onScreen(x, y=None):
    """Returns whether the given xy coordinates are
    on the screen or not."""
```

Listing 2. mouse operation functions

```
def mouseDown(x=None, y=None, button='left', duration
=0.0,tween=linear, pause=None, _pause=True):
    """Performs pressing a mouse button down ."""
def mouseUp(x=None, y=None, button='left', duration=
0.0,tween=linear, pause=None, _pause=True):
    """Performs releasing a mouse button up """
def click(x=None,y=None,clicks=1,interval=0.0,button=
'left',duration=0.0,tween=linear,pause=None,_pause=True):
    """Performs pressing a mouse button down and then
    immediately releasing it."""
def scroll(clicks,x=None,y=None,pause=None,_pause=True):
    """Performs a scroll of the mouse scroll wheel."""
def moveTo(x=None, y=None, duration=0.0, tween=linear,
pause=None, _pause=True):
"""Moves the mouse cursor to a point on the screen."""
def dragTo(x=None, y=None, duration=0.0, tween=linear,
button='left', pause=None, _pause=True,
mouseDownUp=True):
"""Performs a mouse drag to a point on the screen."""
def _mouseMoveDrag(moveOrDrag, x, y, xOffset, yOffset,
duration, tween=linear, button=None):
"""Handles the actual move or drag event."""
```

Another main file is about the mouse event operation on the OSX platform, its the correspondence to the previous file. All the functions in pyautoguiosx.py call the Quartz to implement the mouse operation event on OSX system.

b) make patches: After studying the whole source codes, the author add some comments and address some problems of the source code, then he use "diff xx/xx p.patch" command to generate the patch file. It's about the difference between the code changed before and after. In this patch below we add some comments and delete some useless syntax for better readability.After that, the author send these patch file to each team members via email.

```
diff -Naur patches1/patch.py patch/patch.py
--- patches1/patch.py   1970-01-01 08:00:00.000000000 +0800
+++ patch/patch.py      2018-11-30 14:56:33.000000000 +0800
@@ -0,0 +1,96 @@
+# 决定平台
 # The platformModule is where we reference the platform-specific functio
 if sys.platform.startswith(dataClass.JAVAPlatform):
     #from . import _pyautogui_java as platformModule
     raise NotImplementedError('Jython is not yet supported by PyAutoGUI.

+else sys.platform == dataClass.OSXPlatform:
+    from . import _pyautogui_osx as platform_module
-elif sys.platform == dataClass.WINPlatform:
-    from . import _pyautogui_win as platform_module
-    from ._window_win import Window, getWindows, getWindow
-else:
-    from . import _pyautogui_x11 as platform_module

+# TODO: Having module-wide user-writable global variables is bad. It mak
+# restructuring the code very difficult. For instance, what if we decide
+# move the mouse-related functions to a separate file (a submodule)? How
+# file will access this module vars? It will probably lead to a circular
+# import.
+
+# In seconds. Any duration less than this is rounded to 0.0 to instantly
+# the mouse.
+# If sleep_amount is too short, time.sleep() will be a no-op and the mou
+# cursor moves there instantly.
```

Fig. 9. patch

c) review patches: After receiving the patches, each team member begin to evaluate these pieces of code. During the understanding of codes, one of our team members pull a question about the why use the fail-safety feature, after discussion, we found the purpose of this feature is to prevent the mouse losing control.If you lose control and need to stop the current PyAutoGUI function, just keep moving the mouse cursor up and to the left. About the defects of code, no one found some logic or algorithm bugs, and all of us think these codes are in high enough quality, we found the project is a relatively completed project.

d) commits submission: After evaluating the codes, each team member sent back the patches, and the author collected the code review results and committed the code to Github.

VI. Test

A. Fuzz Testing

Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. The program is then monitored for exceptions such as crashes, failing built-in code assertions, or potential memory leaks. Typically, fuzzers are used to test programs that take structured inputs. [2]

In our project, we want to use the American fuzzy lop to test the stability of the pyautogui. In detail, We use fuzz to generate input to test the functions of the pyautogui. We could tell if this tool is robust by the output log which generated by the fuzz.

There are mainly three steps to use the fuzz testing.I will introduce them one by one by one function example.

1) Input initialization: At first, we need to create an initial input as the seeds. After AFL put the initial input in the input queue, it will generate the new input using the seeds to generate the new input continuely.

In this demo , what I want test is the mouse move function and the mouse click function. The input for move function is coordinate and the click function does not need the argument. So I choose to convert a list of number in str type to the int type as the input.

2) test programming: After we set the input, we need to finish the test program. In the program, I import the library that I need and use an infinite loop to process the new input continuely.

Then I call these functions of the pyautogui in the loop, and use the standard input as the input interface to get the input which fuzz genarated.

```
import afl
import sys
import pyautogui
if __name__=="__main__":
    afl.init()
    while True:
        try:
            Width, Height = pyautogui.size()
            MouseX, MouseY = pyautogui.position()
            j = sys.stdin.readline().strip('\n')
            pyautogui.moveTo(int(j),int(j)+50)
            pyautogui.click()
        except:
            break
```

3) execute and analyse: I could use the afl-python module to excute the AFL to see the detailed test information.But there are still somewhere to be improved, you can see the processing speed is very slow. In the future, maybe I could fix it. At last , we could tell if the pyautogui is a robust project from the output log.

As you can see from the figure,after more than 90 min , there is not any crash . So I could say that this tools is stable enough.

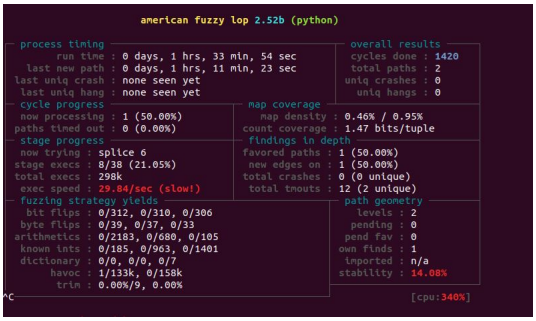

Fig. 10. fuzz result.

B. Test Oracle

Software testing is a very important software engineering activity which is widely used to find defects in programs. During the testing process, test cases are executed on an application under test (AUT) and test oracles are used to determine whether the AUT executed as expected [3]. If we only use Fuzz Testing, we can not find the logical errors of the code. Hence, we need to do Test Oracle.

The oracle might be:

- a program (separate from the system under test) which takes the same input and produces the same output
- documentation that gives specific correct outputs for specific given inputs
- a documented algorithm that a human could use to calculate correct outputs for given inputs
- a human domain expert who can somehow look at the output and tell whether it is correct
- or any other way of telling that output is correct.

An oracle is not a test runner, but a test runner could use an oracle as a source of correct output to which to compare the output of System Under Test (SUT), or as a source of constraints against which to evaluate the SUT's output.

Test Oracle may either be automated or manual. Test oracle automation is important to remove a current bottleneck that inhibits greater overall test automation. Without test oracle automation, the human has to determine whether observed behaviour is correct. The test oracle formulate either a correctness or criterion or an error condition for test cases in order to know whether the program actually passes these cases [4].Conceptually, we can consider testing a process in which the test cases are given to the test oracle and the program under testing. In our project, We choose to use python assert which is a pattern-based test oracle tool to do test oracle. Pattern-based test oracle is automated, and it is convenient for us to compare the real output with the desired output. Since we need to compare the two outputs, a test case is made of a test input and a test oracle. According to the python code file, we choose some specific inputs to test every function of mouse and compare the results with the assert statement. If there are some logical errors, the program will raise the errors.

As shown in figure 11, We set the top left corner of the screen to the origin of the coordinates, the X axis to the left

and the Y axis to the bottom.

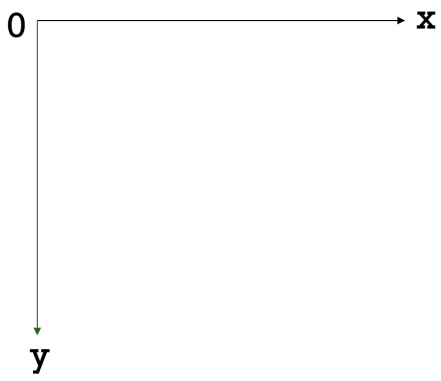

Fig. 11. The coordinate of the screen

At first, we test the interface and the size of the screen. Then we test the position of the mouse with click functionincluding click, double click and triple click. We test the left click, right click and middle click at different positions. At each position, the mouse click once,twice or four times. The detailed values are shown in figure 12 and figure 13.

Test inputs for mouse click function

x	y	times	duration	left or right
0	100	2	1	left
0	200	2	1	left
200	0	4	2	right
300	0	4	2	right
0	400	2	2	middle
400	0	2	2	middle

Fig. 12. The test inputs of click function

Test inputs for mouse double&triple click function

	x	y	times	duration	left or right
double click	100	100	1	2	left
	100	100	1	2	right
triple click	100	100	1	2	left
	100	100	1	2	right

Fig. 13. The test inputs of double and triple click function

Secondly, we test the function of drag with relocation. Since the default value of "_pause" is fault, we also change the value to true to test pause function while dragging.The detailed values are shown in figure 19.

Test inputs for mouse drag function

	x	y	_pause	duration	left or right
dragTo	200	200	false	2	left
dragTo	200	200	false	2	left
dragRel	0	0	true	2	right
dragTo	200	200	true	2	right

Fig. 14. The test inputs of drag function

Each test of the mouse function is independent so that before each test the position of the mouse is in the screen center. In this dragging test, the mouse will pause to drag several rows from the screen center.

Then we test the moveTo function, we set a lot of different inputs to move the mouse to a new position, then compared the new values of x and y with the desired position. The mouse begin moving from the screen center to some new positions and every movements duration is two seconds. The detailed values are shown in figure 15.

Test inputs for mouse moveTo function

From_x	From_y	To_x	To_y	duration
last loc	last loc	center	center	2
center	center	center	center	2
center	center	center+100	center+100	2
center+100	center+100	center	center	2
center	center	center-100	center	2
center-100	center	center-100	center-100	2
center-100	center-100	center	center	2
center	center	center+100	center+100	2
center+100	center+100	center	center	2

Fig. 15. The test inputs of moveTo function

In order to cover as many test scenarios as possible, we also test different forms of test values of x and y,such as list format and tuple format.The test of moveRel function is similar to the test of moveTo function.

After that, we also test the moveTo function with tween effect. However, we do not set as many inputs as the moveTo function. PyTweening is a collection of tweening / easing functions implemented in Python. And in this project, the author choose to use linear, easeInQuad, easeOutQuad, easeInOutQuad, easeInCubic, easeOutCubic, easeInOutCubic, easeInQuart, easeOutQuart, easeInOutQuart, easeInQuint, easeOutQuint, easeInOutQuint, easeInSine, easeOutSine, easeInOutSine, easeInExpo, easeOutExpo, easeInOutExpo, easeInCirc, easeOutCirc, easeInOutCirc, easeInElastic easeOutElastic, easeInOutElastic, easeInBack, easeOutBack, easeInOutBack, easeInBounce, easeOutBounce and easeInOutBounce effect.

The default value of tween parameter is linear, so when we test the tween effect with moveTo function, we just need to change the value of tween, and then compare the mouse

position with desired value. The detailed code is shown in Listing 4 .

Listing 4. Test moveTo function with tween effect

```
    def test_moveToWithTween(self):
        origin = self.center - P(100, 100)
        destination = self.center + P(100, 100)

        def resetMouse():
            pyautogui.moveTo(*origin)
            mousepos = P(*pyautogui.position())
            self.assertEqual(mousepos, origin)

        for tweenName in self.TWEENS:
            tweenFunc = getattr(pyautogui, tweenName)
            resetMouse()
            pyautogui.moveTo(destination.x, destination.y,
            duration=2, tween=tweenFunc)
            mousepos = P(*pyautogui.position())
            self.assertEqual(mousepos, destination,
            '%s tween move failed. mousepos set to %s instead
of %s' % (tweenName, mousepos, destination))
        print(7)
```

In addition to the above functions, we also test the scroll function of the mouse. It is very easy to set the test inputs and we just set four set of compared test inputs. Futhermore, we test the mouseDownUp function within two pairs inputs.

Listing 5. Test scroll and mouseDownUp function

```
    def test_scroll(self):
        pyautogui.scroll(100)
        pyautogui.scroll(-100)
        pyautogui.hscroll(100)
        pyautogui.hscroll(-100)
        pyautogui.vscroll(100)
        pyautogui.vscroll(-100)
        pyautogui.scroll(100, x=100, y=100)
        pyautogui.scroll(200, x=200, y=200)
        print(10)
    def test_mouseDownUp(self):
        pyautogui.mouseDown()
        pyautogui.mouseUp()
        pyautogui.mouseDown(button='right')
        pyautogui.mouseUp(button='right', x=100, y=200,
        duration=2)
        print(15)
```

In order to find out whether some test cases are not executed, we have compiled a number for each test case. Luckily, we do not find any logical problem with test oracle.

So in the code refactoring section, we do not have to make logic changes. Of course, for the code after refactoring, we have to do another test oracle.

VII. Code Smell

A. Preview

Code Smell, it is a hint that there is an error in somewhere in the code, developers can use the smell (odor) to hunt down the problem in the code. In the computer programming community, the code smell represents any sign that things are getting worse. It often marks that the code should be refactored or that all designs should be reviewed. This phrase appeared on WardsWiki, which was coined by Kent Beck. After the rise of refactoring, the usage of this phrase has increased dramatically.Determining whether or not there is a code smell is often subjective and varies with language, developer, and development theory.

B. Duplicated Code

1) Reason For The Problem: Duplication usually occurs when multiple programmers are working on different parts of the same program at the same time. Since they are working on different tasks, they may be unaware their colleague has already written similar code that could be repurposed for their own needs.

There is also more subtle duplication, when specific parts of code look different but actually perform the same job. This kind of duplication can be hard to find and fix.

Sometimes duplication is purposeful. When rushing to meet deadlines and the existing code is "almost right" for the job, novice programmers may not be able to resist the temptation of copying and pasting the relevant code. And in some cases, the programmer is simply too lazy to de-clutter.

2) Example: The simplest Duplicated Code is [two methods in the same class contain the same expression (expression)]. In pytautogui.py(see Listing 6), Obviously, the _mouseDown and _mouseUp method have no different. The solution of this situation is applying extract method to extract duplicated code and then let both locations call code patch that was extracted.

Listing 6. pyautogui.py

```
    def _mouseDown(x, y, button):
    if button == 'left':
        _sendMouseEvent(Quartz.kCGEventLeftMouseDown,
        x, y, Quartz.kCGMouseButtonLeft)
    elif button == 'middle':
        _sendMouseEvent(Quartz.kCGEventOtherMouseDown,
        x, y, Quartz.kCGMouseButtonCenter)
    elif button == 'right':
        _sendMouseEvent(Quartz.kCGEventRightMouseDown,
        x, y, Quartz.kCGMouseButtonRight)


def _mouseUp(x, y, button):
    if button == 'left':
        _sendMouseEvent(Quartz.kCGEventLeftMouseUp,
            x, y, Quartz.kCGMouseButtonLeft)
    elif button == 'middle':
        _sendMouseEvent(Quartz.kCGEventOtherMouseUp,
            x, y, Quartz.kCGMouseButtonCenter)
    elif button == 'right':
        _sendMouseEvent(Quartz.kCGEventRightMouseUp,
            x, y, Quartz.kCGMouseButtonRight)
```

3) Treatment:

- If the same code is found in two or more methods in the same class: use Extract Method and place calls for the new method in both places.
- If the same code is found in two subclasses of the same level:
  1) Use Extract Method for both classes, followed by Pull Up Field for the fields used in the method.
  2) If the duplicate code is inside a constructor, use Pull Up Constructor Body.
  3) If the duplicate code is similar but not completely identical, use Form Template Method.

4) If two methods do the same thing but use different algorithms, select the best algorithm and apply Substitute Algorithm.

- If duplicate code is found in two different classes:
  1) If the classes are not part of a hierarchy, use Extract Superclass in order to create a single superclass for these classes that maintains all the previous functionality.
  2) If it is difficult or impossible to create a superclass, use Extract Class in one class and use the new component in the other.
- If a large number of conditional expressions are present and perform the same code (differing only in their conditions), merge these operators into a single condition using Consolidate Conditional Expression and use Extract Method to place the condition in a separate method with an easy-to-understand name.
- If the same code is performed in all branches of a conditional expression: place the identical code outside of the condition tree by using Consolidate Duplicate Conditional Fragments.

C. Long Parameter List

1) Reason For Peoblem: A long list of parameters might happen after several types of algorithms are merged in a single method. A long list may have been created to control which algorithm will be run and how.

Long parameter lists may also be the byproduct of efforts to make classes more independent of each other. For example, the code for creating specific objects needed in a method was moved from the method to the code for calling the method, but the created objects are passed to the method as parameters. Thus the original class no longer knows about the relationships between objects, and dependency has decreased. But if several of these objects are created, each of them will require its own parameter, which means a longer parameter list.

It is hard to understand such lists, which become contradictory and hard to use as they grow longer. Instead of a long list of parameters, a method can use the data of its own object. If the current object does not contain all necessary data, another object (which will get the necessary data) can be passed as a method parameter.

2) Example: In init.py(see Listing 7), the parameter x, y, button, duration, tween, pause, _pause transfer into the mouseDown method simultaneous.

Listing 7. init.py

```
def mouseDown(x=None, y=None, button='left',
duration=0.0, tween=linear, pause=None, _pause=True):
    if button not in('left','middle','right',1,2,3):
    ..
```

3) Treatment:

- Check what values are passed to parameters. If some of the arguments are just results of method calls of another object, use Replace Parameter with Method Call. This object can be placed in the field of its own class or passed as a method parameter.
- Instead of passing a group of data received from another object as parameters, pass the object itself to the method, by using Preserve Whole Object.
- If there are several unrelated data elements, sometimes merging them into a single parameter object via Introduce Parameter Object.

D. Primitive Obsession

1) Reason For Problem: Like most other smells, primitive obsessions are born in moments of weakness. "Just a field for storing some data!" the programmer said. Creating a primitive field is so much easier than making a whole new class, right? And so it was done. Then another field was needed and added in the same way. Lo and behold, the class became huge and unwieldy.

Primitives are often used to "simulate" types. So instead of a separate data type, there are a set of numbers or strings that form the list of allowable values for some entity. Easy-to-understand names are then given to these specific numbers and strings via constants, which is why they are spread wide and far.

Another example of poor primitive use is field simulation. The class contains a large array of diverse data and string constants (which are specified in the class) are used as array indices for getting this data.

2) Example: In init.py(see Listing 8), the mouseDown, mouseUp, click, rightClick, middleClick and doubleClick apply the same parameters:x, y, button, duration, tween, pause.

Listing 8. init.py

```
def mouseDown(x=None, y=None, button='left',
duration=0.0, tween=linear, pause=None, _pause=True):


def mouseUp(x=None, y=None, button='left',
duration=0.0, tween=linear, pause=None, _pause=True):


def click(x=None, y=None, clicks=1, interval=0.0,
button='left', duration=0.0, tween=linear, pause=None,
_pause=True):


def rightClick(x=None, y=None, duration=0.0,
tween=linear, pause=None, _pause=True):


def middleClick(x=None, y=None, duration=0.0,
tween=linear, pause=None, _pause=True):


def doubleClick(x=None, y=None, interval=0.0,
button='left', duration=0.0,
tween=linear, pause=None, _pause=True):
```

3) Treatment:

- If there are a large variety of primitive fields, it may be possible to logically group some of them into their own class. Even better, move the behavior associated with this data into the class too. For this task, try Replace Data Value with Object.

- If the values of primitive fields are used in method parameters, go with Introduce Parameter Object or Preserve Whole Object.
- When complicated data is coded in variables, use Replace Type Code with Class, Replace Type Code with Subclasses or Replace Type Code with State/Strategy.
- If there are arrays among the variables, use Replace Array with Object.

E. Temporary Field

1) Reason For Problem: Oftentimes, temporary fields are created for use in an algorithm that requires a large amount of inputs. So instead of creating a large number of parameters in the method, the programmer decides to create fields for this data in the class. These fields are used only in the algorithm and go unused the rest of the time. This kind of code is tough to understand. Developer expects to see data in object fields but for some reason they are almost always empty.

2) Example: In pyautogui.py(see Listing 9), there are many temporary variables like left, right, or middle which are applied for determining the action of mouse. The worse is that these temporary variables are not even display once time in the code.

Listing 9. pyautogui.py

```
def _normalKeyEvent(key, upDown):
    assert upDown in ('up', 'down'),
    "upDown argument must be 'up' or 'down'"


def _mouseUp(x, y, button):
    if button == 'left':
        _sendMouseEvent(Quartz.kCGEventLeftMouseUp,
        x, y, Quartz.kCGMouseButtonLeft)
    elif button == 'middle':
        _sendMouseEvent(Quartz.kCGEventOtherMouseUp,
        x, y, Quartz.kCGMouseButtonCenter)
    elif button == 'right':
        _sendMouseEvent(Quartz.kCGEventRightMouseUp,
        x, y, Quartz.kCGMouseButtonRight)
```

3) Treatment:

- Temporary fields and all code operating on them can be put in a separate class via Extract Class. In other words, developer is creating a method object, achieving the same result as if developer would perform Replace Method with Method Object.
- Introduce Null Object and integrate it in place of the conditional code which was used to check the temporary field values for existence.

F. Comments

1) Reason For Problem: Comments are usually created with the best of intentions, when the author realizes that his or her code is not intuitive or obvious. In such cases, comments are like a deodorant masking the smell of fishy code that could be improved. The best comment is a good name for a method or class. If author feels that a code fragment cannot be understood without comments, try to change the code structure in a way that makes comments unnecessary.

2) Example: In init.py(see Listing 10), there are a moutain of comments can be seen in the file, the author wrote everything about the code, including usage, parameters and value which is returned, however, most of comments are unnecessary.

Listing 10. init.py

```
def mouseDown(x=None, y=None, button='left',
duration=0.0, tween=linear, pause=None,
_pause=True):
    """Performs pressing a mouse button down (but
    not up).
    The x and y parameters detail where the mouse
    event happens. If None, the current mouse
    position is used. If a float value, it is rounded
    down. If outside the boundaries of the screen,
    the event happens at edge of the screen.
    Args:
      x (int, float, None, tuple, optional): The x
      position on the screen where the mouse down
      happens. None by default. If tuple, this is used
      for x and y. y (int, float, None, optional):
      The y position on the screen where the mouse
      down happens. None by default. button
      (str, int, optional):
The mouse button pressed
      down. Must be one of 'left', 'middle', 'right'
      (or 1, 2, or 3) respectively. 'left' by default.
    Returns:
      None
    Raises:
      ValueError: If button is not one of 'left',
      'middle', 'right', 1, 2, or 3
    """
```

3) Treatment:

- If a comment is intended to explain a complex expression, the expression should be split into understandable subexpressions using Extract Variable.
- If a comment explains a section of code, this section can be turned into a separate method via Extract Method. The name of the new method can be taken from the comment text itself, most likely.
- If a method has already been extracted, but comments are still necessary to explain what the method does, give the method a self-explanatory name. Use Rename Method for this.
- If developer needs to assert rules about a state that is necessary for the system to work, use Introduce Assertion.

VIII. Refactoring

A. Preview

The purpose of refactoring are improve code structure and organization, which can improve code readability and reduce complexity, and improve source-code maintainability and create a more expressive internal architecture or object model to improve extensibility. The triggering point to refactoring are varies, it maybe some code quality problems found out, or the code maintenance problems. After the progress of code review, we found out that the original code is not in a good maintainable way, and some code smell problems should be solved. The biggest problem of the code is it only contains functions, but not in a class, and there are many duplicated

parts between functions. Hence, the reasons for refactoring the code would be sufficient. Duplicate code occurs frequently in computer programming, it can cause many problems if the developers are not aware of this, like it may vulnerabilities in one part, modify one part cannot fix the bug occurs in other parts, hence, it would be better to merge those duplication code into one part.

B. Duplication

Original

```
def  vscroll(clicks, x=None, y=None):
    moveTo(x, y)
    clicks = int(clicks)
    for   in range(abs(clicks) // 10):
        scrollWheelEvent = Quartz.CGEventCreateScrollWheelEvent(
            None, # no source
            Quartz.kCGScrollEventUnitLine, # units
            1, # wheelCount (number of dimensions)

            10 if clicks >= 0 else -10) # vertical movement
        Quartz.CGEventPost(Quartz.kCGHIDEventTap, scrollWheelEvent)

    scrollWheelEvent = Quartz.CGEventCreateScrollWheelEvent(
        None, # no source
        Quartz.kCGScrollEventUnitLine, # units
        1, # wheelCount (number of dimensions)
        clicks % 10 if clicks >= 0 else -1 * (-clicks % 10)) # vertical move
    Quartz.CGEventPost(Quartz.kCGHIDEventTap, scrollWheelEvent)
def  hscroll(clicks, x=None, y=None):
    moveTo(x, y)
    clicks = int(clicks)
    for   in range(abs(clicks) // 10):


        scrollWheelEvent = Quartz.CGEventCreateScrollWheelEvent(
            None, # no source
            Quartz.kCGScrollEventUnitLine, # units
            2, # wheelCount (number of dimensions)
            0, # vertical movement
            10 if clicks >= 0 else -10) # horizontal movement
        Quartz.CGEventPost(Quartz.kCGHIDEventTap, scrollWheelEvent)
```

Refactoring

```
def  scroll(self, clicks, ish=False, x=None, y=None):
    self. move to(x, y)
    clicks = int(clicks)
    if(ish):
        scroll wheel event = Quartz.CGEventCreateScrollWheelEvent(
            None,  # no source
            Quartz.kCGScrollEventUnitLine,  # units
            2,  # wheelCount (number of dimensions)
            0,  # vertical movement
            10 if clicks >= 0 else -10)  # horizontal movement
        Quartz.CGEventPost(Quartz.kCGHIDEventTap, scroll wheel event)
    else:
        scroll wheel event = Quartz.CGEventCreateScrollWheelEvent(
            None,  # no source
            Quartz.kCGScrollEventUnitLine,  # units
            1,  # wheelCount (number of dimensions)
            10 if clicks >= 0 else -10)  # vertical movement



    for   in range(abs(clicks) // 10):
        Quartz.CGEventPost(Quartz.kCGHIDEventTap, scroll wheel event)
    if (ish):
        scroll wheel event = Quartz.CGEventCreateScrollWheelEvent(
            None,  # no source
            Quartz.kCGScrollEventUnitLine,  # units
            2,  # wheelCount (number of dimensions)
            0,  # vertical movement
            (clicks % 10) if clicks >= 0 else (-1 * clicks % 10))  # horizon
    else:
```

Fig. 16. Duplication in scroll functions

In our work, there are mainly two parts are duplicated, Fig 13 is the duplication in scroll functions, function of vscroll and hscroll are mainly the same, but only different in the parameters, thus, we merge these two functions to a new function ”scroll” with a parameter to judge implement ”vscroll” or ”hscroll”.

Original

```
def  mouseDown(x, y, button):
    if button == 'left':
        sendMouseEvent(Quartz.kCGEventLeftMouseDown, x, y, Quartz.kCGMouseB
    elif button == 'middle':
        sendMouseEvent(Quartz.kCGEventOtherMouseDown, x, y, Quartz.kCGMouse
    elif button == 'right':
        sendMouseEvent(Quartz.kCGEventRightMouseDown, x, y, Quartz.kCGMouse
    else:
        assert False, "button argument not in ('left', 'middle', 'right')"
def  mouseUp(x, y, button):

    if button == 'left':
        sendMouseEvent(Quartz.kCGEventLeftMouseUp, x, y, Quartz.kCGMouseBut
    elif button == 'middle':
        sendMouseEvent(Quartz.kCGEventOtherMouseUp, x, y, Quartz.kCGMouseBu
    elif button == 'right':
        sendMouseEvent(Quartz.kCGEventRightMouseUp, x, y, Quartz.kCGMouseBu
    else:
        assert False, "button argument not in ('left', 'middle', 'right')"
```

Refactoring

```
def  mouse event(self, x, y, button, behavior):
        if(behavior == 'up'):
            event = Quartz.kCGEventLeftMouseDown
        elif(behavior == 'down'):
            event = Quartz.kCGEventLeftMouseUp


        else:
            assert False
            return

        if button == config.LEFT:
            self. send mouse event(event, x, y, Quartz.kCGMouseButtonLeft)
        elif button == config.MIDDLE:
            self. send mouse event(event, x, y, Quartz.kCGMouseButtonCenter)
        elif button == config.RIGHT:
            self. send mouse event(event, x, y, Quartz.kCGMouseButtonRight)
        else:
            assert False, config.button_argument
```

Fig. 17. Duplication in mouse functions

Duplication in function ”mouse_up” and ”mouse_down” are the same in this case, comparison between original code and code after refactoring is shown at Fig 15.

C. Readability

Readability can be a problem of the original code, some important function are not annotated clearly, and the structure of the code, like some of the functions are not so clear to the reader.

Original

```
def  position():
    loc = AppKit.NSEvent.mouseLocation()
    return int(loc.x), int(Quartz.CGDisplayPixelsHigh(0) - loc.y)

def  size():
    return Quartz.CGDisplayPixelsWide(Quartz.CGMainDisplayID()), Quartz.CGDis

```

Refactoring

```
@property
def  position(self):
    loc = AppKit.NSEvent.mouseLocation()
    return int(loc.x), int(Quartz.CGDisplayPixelsHigh(0) - loc.y)
@property
def  size(self):
    return Quartz.CGDisplayPixelsWide(Quartz.CGMainDisplayID()), Quartz.CGDis
        Quartz.CGMainDisplayID())
```

Fig. 18. Readability

Take the function position and size as an example, function position returns the position of the mouse and function size returns the size of the screen. These two function actually should be transferred to the form of ”@property” to make reader understand that this is not a function, but property instead. In this section, we change the structure of the code and add some annotations in the code to improve readability.

D. Encapsulation

Encapsulation is refers to restrict outsiders to directly access certain information or some operators, it can be a class, a module, a file and so on. If the code violate the encapsulation rule, serious vulnerabilities can be exploited, making those

variables to private is important since that. After code review, we found out that the original code indeed violate the encapsulation rule, many variables are not private, which could be changed by outsiders easily.

```
OSXPlatform = 'darwin'
JAVAPlatform = 'java'
LEFT = 'left'
RIGHT = 'right'
MIDDLE = 'middle'
MOVE = 'move'
DRAG = 'drag'
```

```
self.__OSXPlatform = 'darwin'
self.__JAVAPlatform = 'java'
self.__LEFT = 'left'
self.__RIGHT = 'right'
self.__MIDDLE = 'middle'
self.__MOVE = 'move'
self.__DRAG = 'drag'
```

Fig. 19. Encapsulation

Before refactoring, the variables are not private, both readable and writable. After refactoring, the variables are only can be readable. In python, the variable with doubel underline at head, means can only access within the class, but cannot from outside, we add some interfaces that allow the users to read this variables, therefore the encapsulation violation was fixed.

IX. PERFORMANCE ANALYSIS

We use profile to compare the performance of two python codes. It is a common code performance testing tool of python. The result of profile will show you the values of calls and running time which are related to the performance of code. It is clear that the refactored code has a smaller total running time, and the number of calls to each function is also significantly less than before refactoring. Whats more, the refactored functions also run at a smaller time than before refactoring.

NAME	Call Count(Before)	Call Count(After)	Time(Before)	Time(After)	Own Time(Before)	Own Time(After)
<objc._objc.loadBundle>	12	12	63	57	61	55
<method 'search' of '_sre.SRE_Pattern' objects>	1137	1137	25	22	25	22
_parse	972	968	62	54	23	19
_next	18463	18445	17	15	14	13
_update	1	1	13	11	13	11
_compile	1720	1716	19	15	11	8
init.py	1	1	91	74	9	7
_compile_info	395	393	22	17	9	8
init.py	1	1	24	61	8	6
<method 'append ' of 'list' objects>	48050	47996	7	4	7	5
Get	16857	16843	21	19	6	5
getwidth	2436	2432	7	6	6	5
init.py	1	1	75	114	5	4
init	13	13	110	99	4	3
_compile	1182	1180	118	97	4	3
init.py	1	1	47	34	4	3
<hasattr>	212	208	6	8	4	3
<objc._objc.regidterMetaDataForSelector>	4783	4783	4	3	4	3
<len>	55840	55778	5	4	4	3
init.py	1	1	126	40	4	3
_metadata.py	1	1	7	6	4	3
_metadata.py	1	1	5	4	3	3

Fig. 20. Performance Comparation

REFERENCES

[1] Caitlin Sadowski, Emma Sderberg,Luke Church, Michal Sipko,Alberto Bacchelli, "Modern Code Review: A Case Study at Google", ICSE-SEIP 18, May 27-June 3, 2018, Gothenburg, Sweden.

[2] American fuzzy lop (2.52b). (n.d.). Retrieved from http://lcamtuf.coredump.cx/afl/

[3] A. Memon, I. Banerjee, A. Nagarajan, "What test oracle should I use for effective GUI testing?", 18th IEEE International Conference on Automated Software Engineering, 2003.

[4] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, Shin Yoo, "The Oracle Problem in Software Testing: A Survey", IEEE Transactions on Software Engineering, Volume: 41, Issue: 5, May 1 2015.