Bailey Testing Framework

An automated graphic based GUI testing framework for TDD process.

Name: Yuen Kit Lai (Jacky)

Student ID: 007319910

Table of Contents

1. Abstract

Test-Driven Development (TDD) is aim to control the quality of software by emphasizing on testing requirements during development process. Most TDD toolset are geared towards text-based functional testing.

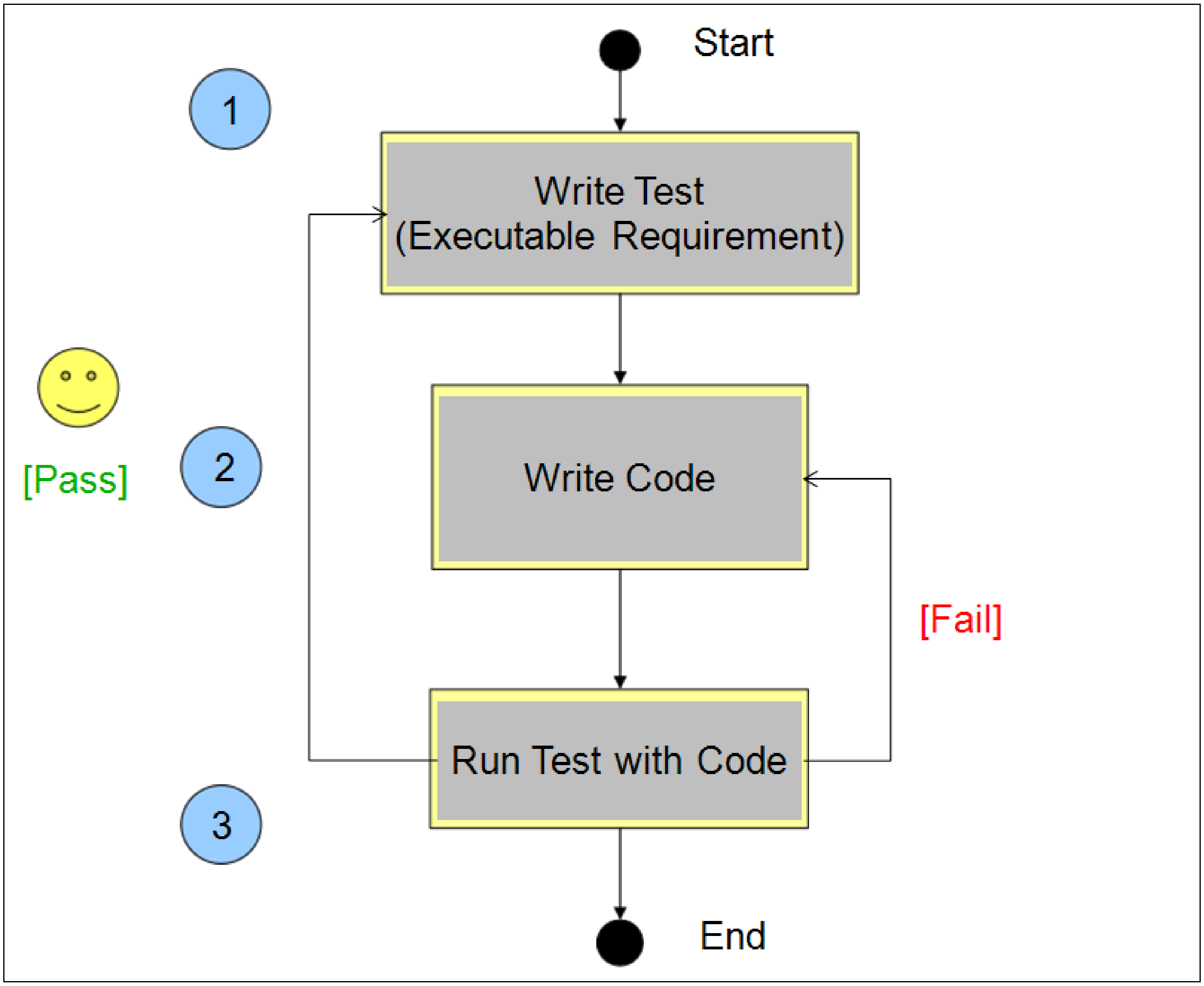
Most mainstream GUI testing frameworks such as SWTBot or Selenium are testing the existent of particular UI elements, regardless the position of element on the actual layout (visual position). A button which is positioned out viewable area can be considered passing the test, which is not the intended result.

Graphic based testing is normally run by a team of testers, by inspecting the screen layout visually. This approach could introduce human error, and inefficient in terms of productivity.

In this report, I will introduce Bailey Testing Framework, an automated graphic testing method by image comparison and how it complements the current TDD workflow.

I will first explain TDD concept in general in Chapter 2, current GUI testing trend on chapter 3, and start from chapter 4, I will introduce a home-brew graphic based GUI testing framework, called Bailey Testing Framework.

2. What is TDD?



TDD (Test Driven Development) is a software development process which creating test cases before implementing the design code. The magic of writing the test code first is that, since no implementation code is written, the only thing that will influence test code is the requirements. Psychologically, it forces the developers to think about what to test, and spend time on thinking the input / output of the design according to requirement. As a result, the test cases are sometime called "Executable Requirements".

After creating test code, when the developer try to implement the code, their code will be influenced by test code, and indirectly influenced by requirements. The end result is the implementation code will be more requirements compliant.

Even though, theoretically speaking, any sort of testing can be used. In practice, in order to implement TDD effectively, testing framework is a must.

In our discussion, we will focus on GUI testing framework. I will briefly explain the text based GUI testing framework in general (which is quite common in the market), and I will introduce the idea of graphic based GUI testing, and how it complement the TDD process.

3. Text based GUI Testing Framework

The common framework is JUnit testing framework. It's a mother of all derived testing framework.

The basic unit is unit test. It's is testing if a certain output is true or false. The concept is simple. The challenging part is where to test, among all the function and code.

There are a lot of GUI testing frameworks, for example SWTBot, selenium, HttpUnit, HtmlUnit, Cactus. Those are specialized form of JUnit testing, serving different purposes. They are all using the same concept, which is by testing certain point of code (spot) to see if its output is expected. And again, it's the developer's responsibility to find the sweet spot to test. The test quality will be greatly determined by the ability of developer to find the correct place to test.

One of the common techniques is placing unit test unconditionally to all over the code, with the hope that it can cover all the possibility of errors. The downside of this has serious impact on developer's productivity, as there is a huge test code to create, and to maintain. On the contrary, if placing too little unit test cases, we might run into the risk of missing test on some important spots.

In most Text Based GUI Testing framework, e.g. SWTBot (SWT) and Selenium (Web UI), an action which is used for triggering an event is produced by coding it, using GUI API. The result of execution will be inspected by querying the state of a component. As it turns out, the features of a text based testing framework greatly depends on the underlying GUI API.

4. Introducing Bailey Testing Framework (Graphic based GUI Testing Framework)

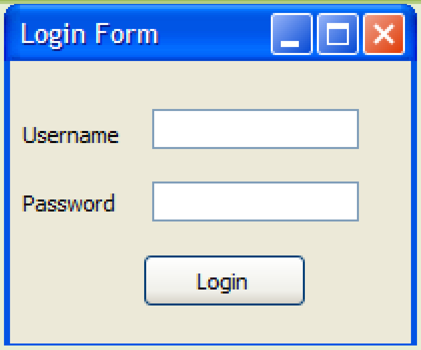
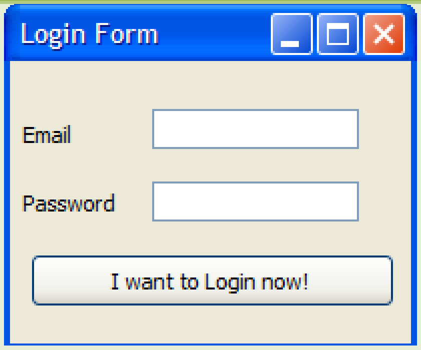
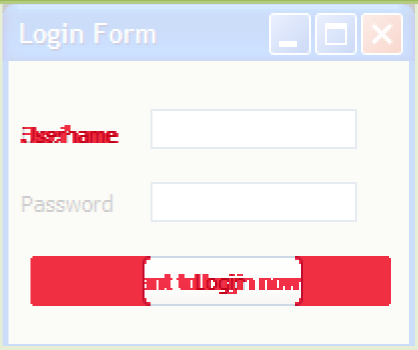
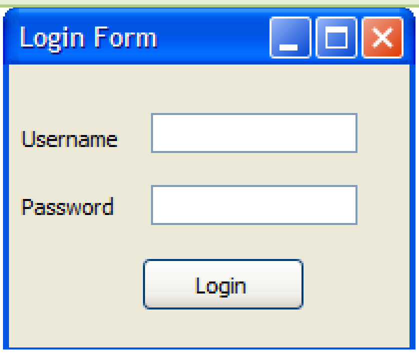
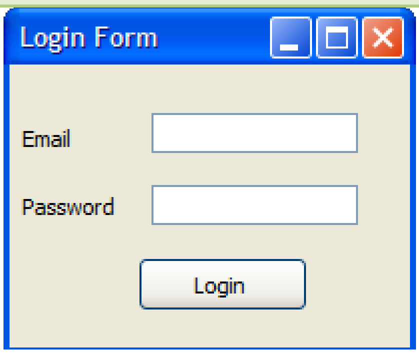
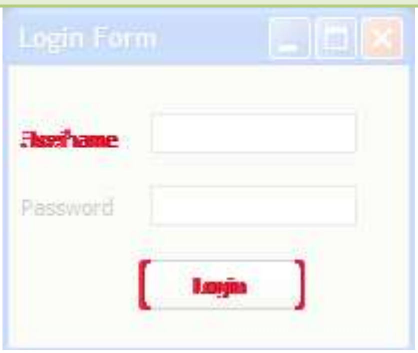
The concept of Graphic based GUI testing framework is based on the fact that, GUI is a visual element, so it would be naturally test the graphic, rather that the function point. It mimics how human inspect the product visually. Besides, a graphic contains more information than a text. By comparing images, we are comparing every single pixel on the screen.

Here I am proposing a conceptual testing framework. It's called Bailey Testing Framework. It consists of two parts: Image Comparison and Workflow Verification.

4.1 Image Comparison

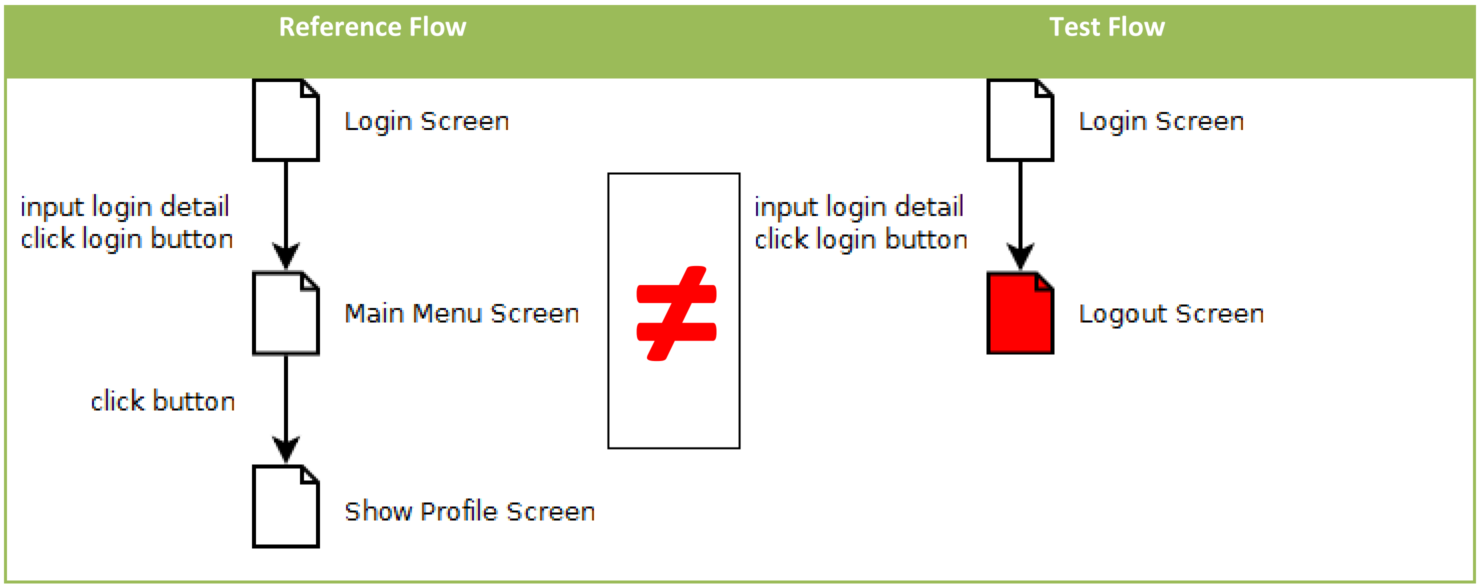
Each captured screenshot will be used to compare against a referenced image. This image is analogues to "expected result" in text-based testing framework.

The criteria of passing the test are determined by the pre-configured percentage of identical, for example, 95% of identical.

Reference Image	Test Image	Difference	Pass / Fail
			91.06% Fail
			95.87% Pass

4.2 Workflow Verification

User action and timing will be recorded first (for the reference workflow). The time and input action will be replayed in order to reproduce the program sequence. For every response shown on the screen, its screenshot will be captured, and will be used for image comparison.


Reference Flow
Test Flow
Login Screen
input login detail
click login button
Main Menu Screen
click button
Show Profile Screen
≠
Login Screen
input login detail
click login button
Logout Screen

5. How it works

Step 1: record actions

1. Open target application from within Bailey.

2. Perform actions (click, or type) on each screen.

3. Capture screenshot on each screen.

4. Close the application and stop recording actions.

Step 2: Replay actions

1. Select the test build from within Bailey.

2. Replay the actions (recorded earlier) on the test build.

Step 3: Image comparison

1. Compare the generated images.

6. Pseudo code

A. Recorder

```
// listen to onClick and onKeyUp events.

onClick(posX, posY)
{
   writeToFile(currentTime, posX, posY);
}

onKeyUp(keyCode)
{
   writeToFile(currentTime, posX, posY);
}
```

B. Player

```
while (line = file.read() != File.EOF)
{
    performAction(line);
}
```

c. Image Comparison

```
for each images in reference folder
{
   referenceImage.compareTo(testImage);
}
```

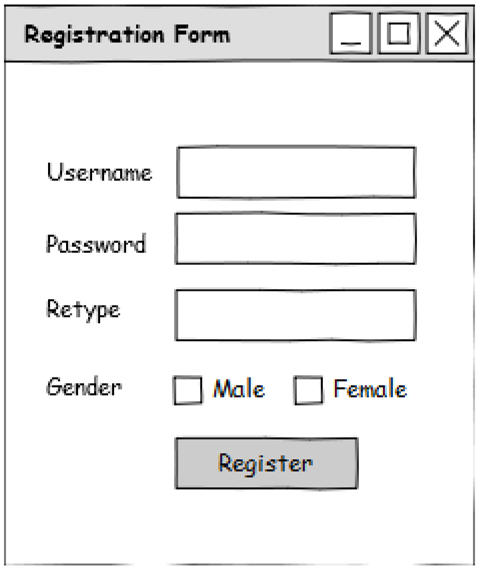
7. Text-Based GUI Testing Framework vs. Bailey Testing Framework

No.	Text-Based GUI Testing Framework	Bailey Testing Framework
	Test certain points on screen.	Complete test coverage on each screen.
	The effectiveness of the test depends on the accuracy of the test point placement.	The effectiveness of the test depends on the tolerance value range.
	Testing the existence of GUI element, within or outside viewable area. (Testing position is rare)	Testing the existence of GUI element, within visible area only.
	Test related features in several test cases (test several existence button).	Test all the feature visible on screen in one go.
	Can test non-graphical feature.	Only graphical feature.
	Learning Curve. Hard	Easy
	Developer level testing – test what developer understand	Customer level testing – test in terms what customer understand
	It is Solution oriented testing (depends on what language we use, what framework, and what method)	it's Problem oriented testing (depends solely on what customer want to see, and how it works, it's is language neutral)
	Cross project type testing – web apps, desktop apps, and mobile apps GUI test has different testing framework	Bailey can test for all, as long as there is a GUI/screen.
	It translate the graphic element into text form (by test object API), and unit test it.	Directly test on the image, pixel by pixel.
	Ability to test features are depends on the GUI API.	Ability to test is independent of GUI api.
	Action triggered by explicit method called (programmatically)	Action triggered by emulating user behavior (mouse clicking, typing etc)

8. Use Case

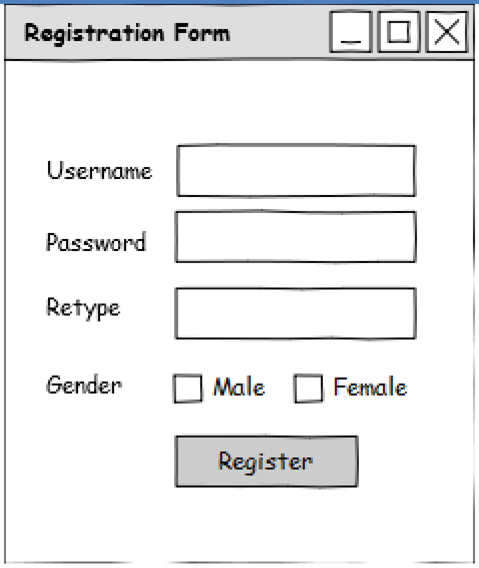
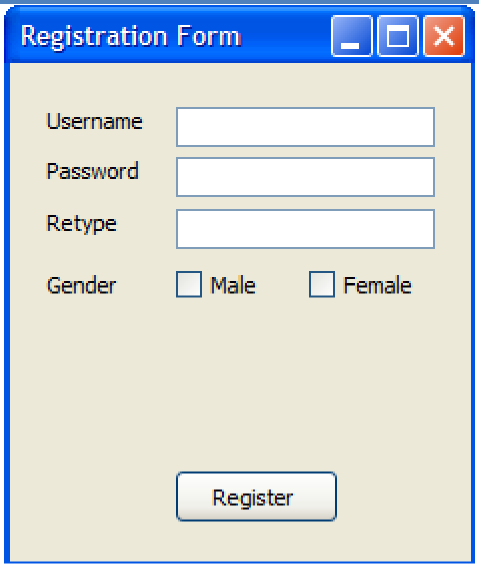
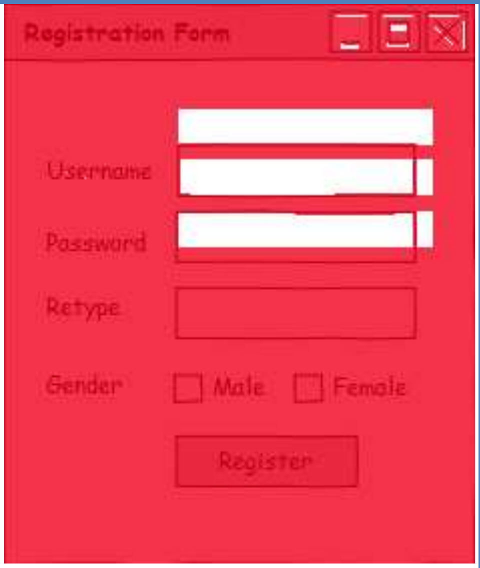
I will use one Desktop Apps Project flow in TDD, to illustrate how Bailey Testing Framework helps improve the quality of our product.

Phase 1: Discussing conceptual idea and general flow of the program with customer.



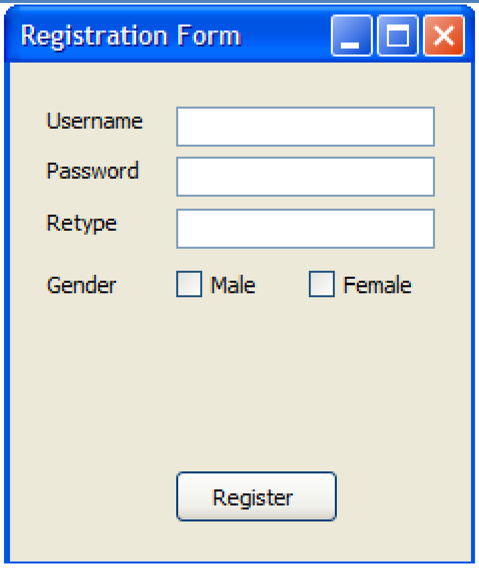
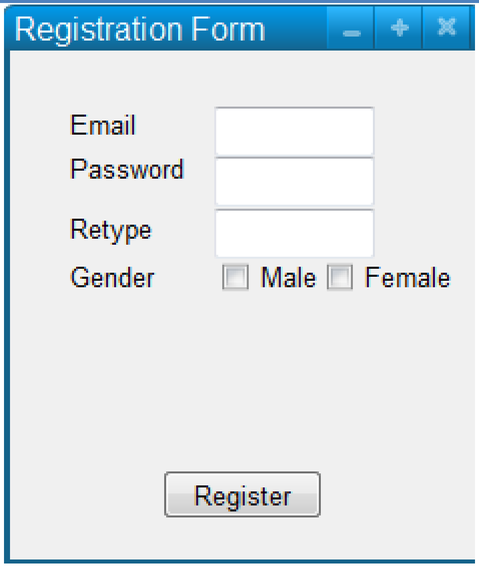
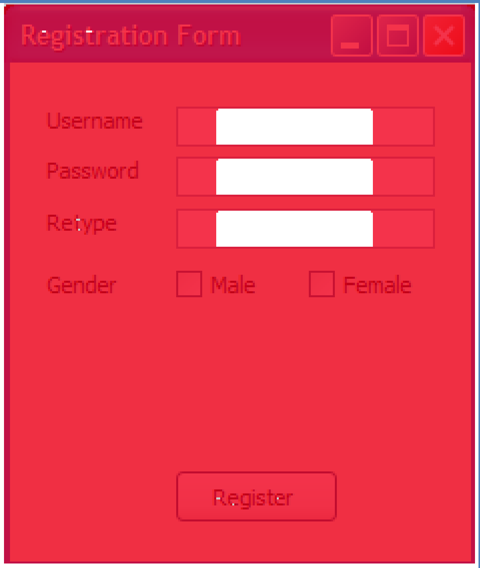
During the initial meeting with customer, wireframe mockup will generally be used to discuss the main idea.

Phase 2: Detailed Design

Mockup (reference)	Design (test)	Difference
		
		24.71%

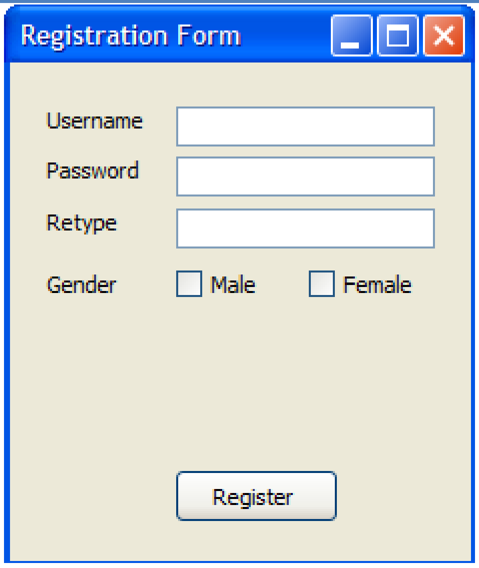
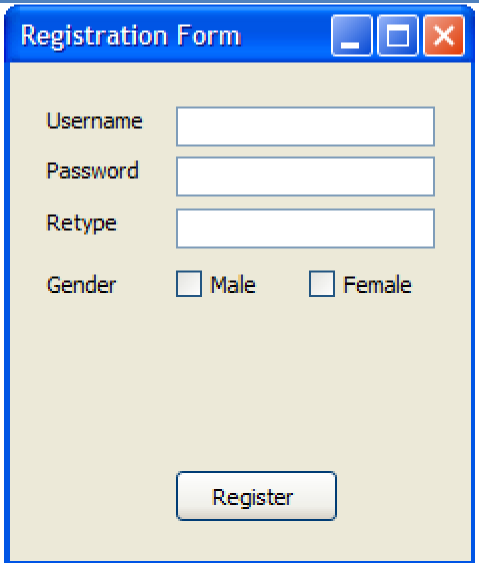
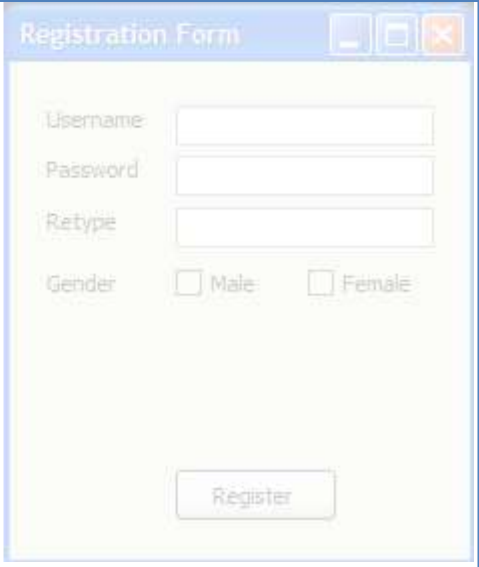
UX Designer will produce detail design, with the program flow.

Phase 3: Develop product prototype(Clarify doubt)

Design (reference)	Prototype (test)	Difference
 Registration Form Username Password Retype Gender Male Female Register	 Registration Form Email Password Retype Gender Male Female Register	 Registration Form Username Password Retype Gender Male Female Register
		67.04%

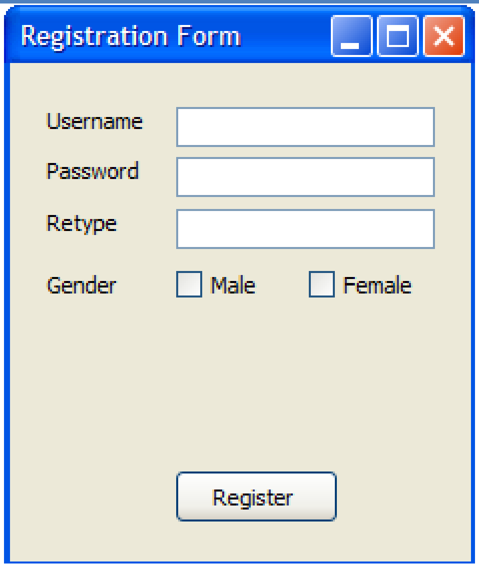
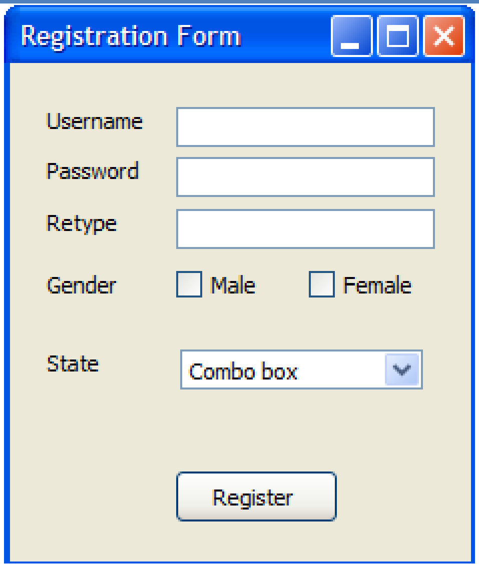
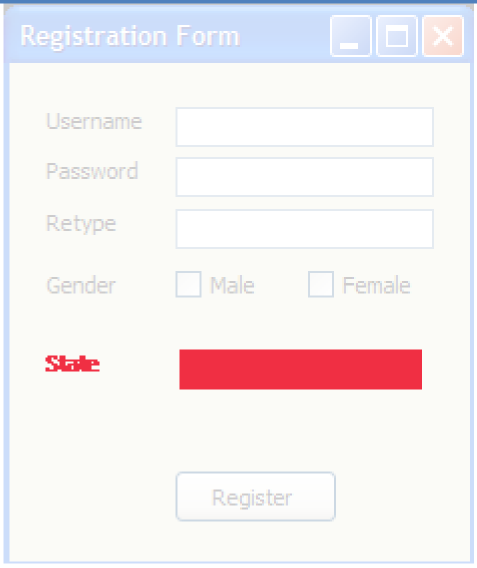
Developer will first produce a prototype, to demo to customer, to clarify some behavior ambiguity. The result of clarification will modify the UX Design (Phase 2)

Phase 4: Develop actual Product code, incrementally.

Design (reference)	Product (test)	Difference
 Registration Form Username Password Retype Gender Male Female Register	 Registration Form Username Password Retype Gender Male Female Register	 Registration Form Username Password Retype Gender Male Female Register
		100.00 %

There can be two tests: For every screenshot produced, test against the UX design, and test against the previous screenshot.

Phase 5: Maintain the product (bug fixing, adding features etc.)

Product-old (reference)	Product-new (test)	Difference
 Registration Form Username Password Retype Gender Male Female Register	 Registration Form Username Password Retype Gender Male Female State Combo box Register	 Registration Form Username Password Retype Gender Male Female State Register
		97.17%

During maintenance phase, programmer will be asked to add in new features occasionally.

7. Conclusion

Bailey Testing Framework provides a more natural way of testing GUI based software. It simulates the user input, and inspects the visual element. It is easy to use and master. It can be used as communication tool among customer and developer, and image is easier to understand.

8. Bibliography

Elias Volanakis, Ketan Padegaonkar, Fabian Steeg and Mickael Istria and others. *http://wiki.eclipse.org/SWTBot/UsersGuide.* June 2010.

Matt Stephens, Doug Rosenberg. "Design Driven Testing: Test Smarter, Not Harder." Apress, 2010.