MTAT.03.159-Software Testing

Lab. #1 – Defect (Bug) Reporting, Ad-hoc Testing, Manual Testing, and Regression Testing

Instructor: Dietmar Pfahl (dietmar.pfahl@ut.ee)
Teaching Assistant: Svetlana Omelkova (svetlana.omelkova@ut.ee)
Institute Of Computer Science
University of Tartu

TABLE OF CONTENTS

1 INTRODUCTION 2
1.1 Objectives 2
1.2 Group-work Specification 2
1.3 Due Date and Late Marking Policy 2
1.4 Pre-lab 3
1.4.1 Definition of Ad-Hoc Testing 3
1.4.2 Defect (Bug) Tracking Tool 3
1.4.3 Example Repositories of Real Defects 3
1.4.4 System Under Test 3
2 INSTRUCTIONS 4
2.1 Familiarization with the System Under Test 4
2.1.1 Perform a deposit 4
2.2 Familiarization with Bugzilla 5
2.2.1 Logging in 5
2.2.2 Reporting a defect 5
2.2.3 Displaying all defects for a product 6
2.2.4 Editing a product 6
2.3 Ad-Hoc Functional Testing 6
2.4 Manual Functional Testing 7
2.5 Defect Correction Verification and Regression Testing 7
2.6 Summary 7
3 DELIVERABLES AND GRADING 7
3.1 Bugzilla Defect Reports (50%) 8
3.2 Lab Report (50%) 8
4 INTERESTING QUOTES AND WEB LINKS 8
5 ACKNOWLEDGEMENTS 9
6 REFERENCES 9
7 APPENDIX A – USER INTERFACE FOR ATM SIMULATION SYSTEM 11
8 APPENDIX B – DEFECT LIFECYCLE FROM INITIAL REPORT TO CLOSED 12
9 APPENDIX C – REQUIREMENTS FOR THE ATM SIMULATION SYSTEM 13
9.1 High Level Requirements 13
9.2 Use Case Diagram 13
10 APPENDIX E - FUNCTIONAL TEST SUITE 14

1 INTRODUCTION

This lab can be divided into four main parts:

- Familiarization
- Ad-hoc testing
- Manual functional testing, and finally,
- Regression testing

In the familiarization stage, students will explore the software system to be tested (System Under Test, or SUT), as well as a popular defect tracking system (Bugzilla).

During the ad-hoc testing phase, students will be free to test the system in any manner that they choose (and are able). Once students reach the manual functional testing phase, they will be required to use a predefined test suite to test the SUT.

Finally, students will perform some simple regression testing on an altered version of the system (corrected by imaginary developers in response to a list of defect reports), and record differing system behavior in the defect tracking system appropriately.

1.1 OBJECTIVES

This lab is a simple introduction to some of the concepts inherent to software testing. Specifically, students should gain an understanding of some fundamentals involved in testing. This includes hands on experience in testing an example software system, and following industrial defect tracking practices through several phases of defects' life cycles. This lab will also allow students to compare ad-hoc, manual, and regression testing strategies.

1.2 GROUP-WORK SPECIFICATION

Sections 2.1 through 2.5 will be performed as a group as specified. The lab report will be completed and submitted by each student group. Make sure to include the names of both students in the lab report.

1.3 DUE DATE AND LATE MARKING POLICY

Lab report should be submitted through the submission form on the course home page (http://courses.cs.ut.ee/2013/SW-Test/spring/Main/HomePage). Deadline for submissions is before the next lab session.

Late reports: There will be 50% penalty for one day delay. There will be no credit points for reports turned after the one day delay date.

1.4 PRE-LAB

1.4.1 Definition of Ad-Hoc Testing

Ad-hoc testing is a commonly used term for software testing performed without planning and documentation.

The tests are intended to be run only once, unless a defect is discovered. Ad-hoc testing can be considered as an element of exploratory testing, being the least formal of test methods. Ad-hoc testing has been criticized because it isn't structured, but this can also be a strength: important things can be found quickly. It contrasts to regression testing that looks for a specific issue with detailed reproduction steps, and a clear expected result. Ad-hoc testing is most often used as a complement to other types of testing.[1]

1.4.2 Defect (Bug) Tracking Tool

The only testing tool required for this lab is the defect tracking system (Bugzilla). The Bugzilla server for this course is accessible via the following link: https://app.devzing.com/UT/bugzilla/. Bugzilla is an open source defect tracking system which has features geared towards use in industry. Some of its main features are: ability to prioritize and assign defects to developers, an advanced query tool to search for defects based on any number of parameters, and integrated email capabilities to inform stakeholders and developers of relevant information.

Bugzilla tracks the state of any defects which have been reported for any given product. In Bugzilla, a product is any system under test (usually software).

For more information on Bugzilla, visit http://www.bugzilla.org.

1.4.3 Example Repositories of Real Defects

The following are example repositories of real defects for real projects in the software industry. Make sure to review many examples to see how other experienced testers write defect reports.

- https://bugzilla.mozilla.org
- http://bugzilla.kernel.org
- https://issues.apache.org/bugzilla
- http://www.openoffice.org/issues/query.cgi
- https://bugs.eclipse.org/bugs

1.4.4 System Under Test

The system under test for this lab is an ATM simulation system. To get started with this system, download the archive SUT versions.zip from course page, under the folder Lab 1/Files needed.

This system was originally developed in order to demonstrate an entire iteration of an object-oriented methodology. As this is a system designed specifically as a case study, the main purpose is to learn from it. For the purpose of this lab, however, we will treat this system as a real ATM. From this perspective the purpose of the system is to allow the user to deposit, withdraw, query and transfer funds to/from their bank account(s).

To use the ATM simulation system simply download “ATM System – Lab 1 Version 1.0.jar” from course page, save it in a known location, and run that file from the saved location. The system should begin

[1] Ad-hoc testing - Wikipedia

execution with the GUI as shown in Appendix A. The main systems of the ATM which have been simulated are labeled.

There are two valid hard-coded card numbers and PINs:

Card Number: 1 PIN: 42 Available Accounts: Checking and Savings

Card Number: 2 PIN: 1234 Available Accounts: Checking and Money Market

Note: Both of these cards access the same checking account.

The initial balances are:

Checking: $100 Savings: $1000 Money Market: $5000

2 INSTRUCTIONS

This section details the instructions for executing the lab. Perform Sections Familiarization with the System Under Test as a group as specified.

2.1 FAMILIARIZATION WITH THE SYSTEM UNDER TEST

1. If you haven't done so already, download the ATM simulation system version 1.0 from course page.
2. Run the JAR file version 1.0 file to show the GUI as shown in Appendix A.

2.1.1 Perform a deposit

3. Turn the system on using the "On" button.
4. Enter the number of $20 bills that the system is assumed to start off with, noting that this is the number of bills, not the total value of the bills. Entering a value of 10 for instance indicates that the ATM is starting with $200 (10 twenty dollar bills). Any number greater than 0 will suffice for now.
5. Click on the "Click to insert card" button which has now appeared on the main interface below the simulated ATM display.
6. The screen now changes to a prompt for the user to input the card number (since there is no actual physical card reader). Enter 1 for the card number and press Enter. Upon returning to the main screen, the display is now requesting the PIN be entered.
7. Type 42 using the simulated keypad and press Enter. The display now prompts the user to perform one of four transactions: withdraw, deposit, balance inquiry, or transfer.
8. Press 2 on the simulated keypad to perform a deposit. The display now prompts the user to indicate which account they would like to deposit to: checking, savings, or the money market account.
9. Press 2 on the simulated keypad to deposit to the savings account. The display now prompts the user to enter the deposit amount.
10. Enter any positive amount and press *Enter*. A button which represents the user inserting the deposit envelope now appears.
11. Click that button to simulate inserting the envelope. The display now prompts the user whether they wish to perform another transaction or not.
12. Press 2 on the simulated keypad to indicate you do not wish to perform another transaction. The main window shows a button appearing, simulating the ejecting of the user's card.

13. Press the System Power Button once again to turn the ATM system off.

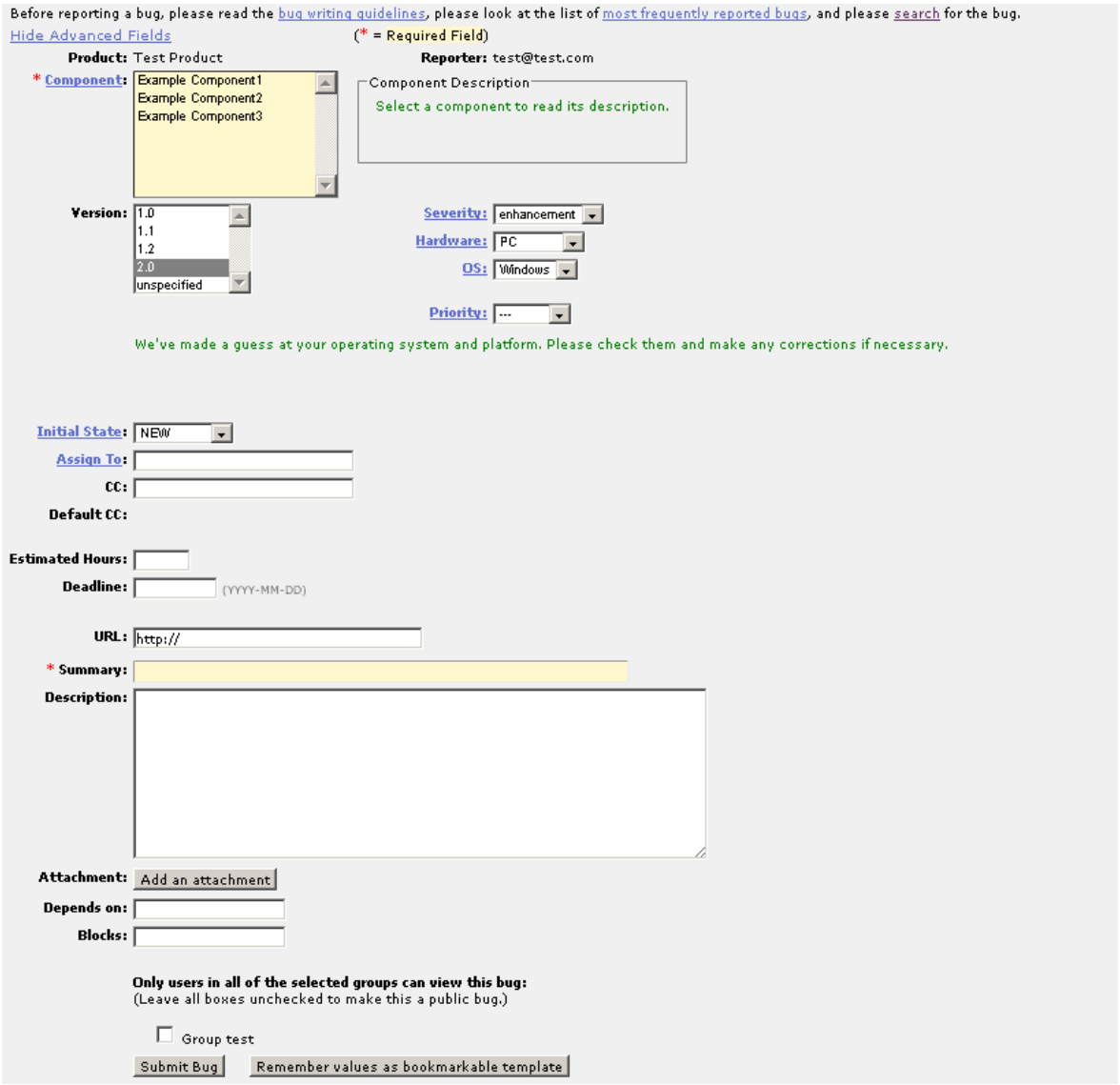
2.2 Familiarization with Bugzilla

2.2.1 Logging in

14. If you haven't already done so, navigate to our Bugzilla server in your web browser.
15. Enter your username and password on the login page. Your username is the email address which you use in ÕIS (if you are not sure which email address your account uses, please ask the TA for assistance). Your password will be your University of Tartu student ID number, e.g., A112233.
16. You should now be logged in and viewing the Bugzilla main page. Note the navigation options on the main page; there is a navigation and search bar at the top and bottom of every page, as well as several commonly used links as the content on the main page.

2.2.2 Reporting a defect

17. Begin by clicking on the "New" link on the navigation bar, or the "Enter a new bug report" link on the main page to begin reporting a new defect for a specific product.
18. The browser now shows a page which requires you to choose what product you are logging a defect for. Click on the "Test Product" link to report a defect for this non-existent product.
19. A form is now displayed in the browser which is to be filled out indicating the details of this particular defect. A screen shot of the form is shown below.



20. Note all the fields which can be entered:
    a. Component – for large products which are composed of multiple components this can be useful for searching for bugs in specific areas of the product.
    b. Version – for many products, defects will be introduced in a specific version, and must then be fixed in future versions.
    c. Severity – the severity (state of quality of being severe; severe implies adherence to rigorous standards or high principles) of a defect is likely going to determine its priority when fixing defects.
    d. Hardware & OS – a defect may be specific to only a single platform or exist in multiple platforms.
    e. Priority – the priority of a defect to be resolved.
    f. Status – a defect starts off as new, and then is assigned when it has been accepted to be fixed. Refer to the diagram in Appendix B for more information.

g. Assignee – this is the Bugzilla username of the person assigned to fix, verify, or analyze the defect.

h. CC & Default CC – one feature of Bugzilla is to notify interested parties of changes to defect statuses via email. Entering a valid Bugzilla username in this field will turn on such notifications.

i. URL – this field is often used for online products, but could be used for any address relevant to the defect.

j. Summary – this is one of the most important fields when reporting a defect. This field will be used to essentially give the defect a title for quick and easy browsing.

k. Description – although this field is called description, there are several pieces of information which should be entered in this field. Some information which should be included here is: high level description of the defect, the steps to reproduce, the actual results (what happened), and the expected results (what should have happened).

l. Add an attachment – sometimes it is useful to include log files, screenshots, and other relevant files along with the defect report.

m. Depends on & Blocks – often it is the case that defects are related to each other with different types of dependencies. Two such dependencies are: defect A's resolution depends on the resolution of defect B, and defect A's resolution is blocked by defect B's resolution. To use these fields, the defect ID of the dependent or blocked defect would be entered.

21. Think of a defect that you have seen (or make one up) in your least favorite piece of software, and enter details in the Summary and Description fields. When filling in the description, try to include all information listed in item 'j' above.

22. Click the "Submit Bug" button at the bottom of the page. A confirmation page is then displayed indicating the defect number and a few key details. A form is also displayed to edit other information for this defect, but this can be ignored as it does not pertain to testing. These defect reports will be visible to all students as well as the instructor, so they should still be professional in nature.

2.2.3 Displaying all defects for a product

23. Click on the "Search" link in one of the navigation bars.

24. A page is displayed with two large tabs. The default tab is to "Simple Search" the other tab is "Advanced Search." Click on the "Advanced Search" link.

25. The Advanced Search page is now displayed with options to search for bugs based on any of the fields available when reporting a defect. Select the "Test Product" product from the product list, and then click the "Search" button.

26. A list of all defects logged for the product is displayed. Look through the list of defects and verify that the defect logged by you earlier is present in that list.

2.2.4 Editing a product

27. To edit the settings of a product, first click on the "Administration" link on one of the navigation bars.
28. A page is displayed where most options are disabled (since you do not belong to the administrators group). Click on the "Products" link.
29. A page is displayed listing all products which you have the user rights to edit. Click on any of the products.
30. A form is displayed with a few fields for changing the product name and description. Along the left side of the page are a series of links to edit the components, versions or group access rights. This is where you will alter the product settings later on (to add a new version). For now just make note of the options available to you.

2.3 AD-HOC FUNCTIONAL TESTING

31. In order to perform any testing, the requirements must first be known. Read over the requirements for the ATM simulation system as outlined in Appendix C before continuing with the rest of this section.
32. Before beginning testing, try to come up with a high level plan for how you intend to test the system. Record key details of this plan, as it will be required in the lab report. This plan could include but is not limited to, information such as: functions being targeted, the approach to be taken (test most functions a little bit, or test a few functions extensively, etc.), and how you plan to come up with test cases (test most common paths, or exceptional paths, etc.). Keep in mind that this does not need to necessarily be the best plan, as long as it is justifiable.
33. Reporting defects: Carry out your devised testing plan for roughly half an hour. Each group needs to perform ad-hoc testing and record defects using their own account. The list of defects and also the names of report creators in Bugzilla will be reviewed by the Prof. and/or the TA after the lab to assess each student's activity. While performing your testing, record the following information for each test case:
    a. The function being tested (e.g., Login)
    b. The initial state of the system (e.g., System is on and is idle, i.e., not already serving a customer)
    c. The action to take (e.g., Insert a card, enter correct card number and PIN)
    d. The expected outcome (e.g., the system successfully accepts the customer, and shows the banking menu)

    While performing the tests, if any of the actual results differ from the expected results, report that as a defect in Bugzilla. The product to report it under is named ATM System along with your group number. Reporting defects as they are found follows the heuristic of reporting defects promptly[2] (before forgetting the defect and its detailed conditions).

    Reporting defects in a simple, concise manner ensures that the developer who reads the defect report will know what the issue is, and will be more likely to fix it. When reporting defects, follow the guidelines in the Sample Bug Report article

[2] "Lesson 67: Report defects promptly" – from the book "Lessons Learned in Software Testing"

(http://www.softwaretestinghelp.com/sample-bug-report), the one provided by Bugzilla (https://landfill.bugzilla.org/bugzilla-tip/page.cgi?id=bug-writing.html), as well as the guidelines explained earlier in the familiarization stage.

2.4 Manual Functional Testing

This section is to be performed as a group. One student can 'drive' the testing (operate the computer executing the system under test), while the other student keeps track of which tests have been performed, reports any defects found, and determines what order to execute tests in. Keep track of what order the tests are executed in, as it will be useful information later on. Note that it does not matter which student reports the defects, as it is a group effort.

34. In Appendix E, a basic test suite has been provided for this SUT. Execute each of the test cases at least once, verifying that the actual results match the expected results for each case. Report any defects found as described in step 33. In order to differentiate between defects found during this stage and the previous stage, in the summary field type "MFT:" (Manual Functional Testing) before the summary of the defect. Do not report defects which have already been found by your group during the ad-hoc testing phase, however you may wish to take note of which defects are found using both testing methods as it may be relevant in your report.
35. Upon completion of testing, review all defect reports created. To do this, perform a search in Bugzilla for defects containing "MFT:" in the summary field. This will produce a list of the newly added defect reports. The student who was previously executing the tests should now be the main participant in reviewing the defect reports.

2.5 Defect Correction Verification and Regression Testing

This section is to be performed as a group. The defects reported in the two previous stages of testing can be divided among the group and can be retested individually.

36. Download the updated version (version 1.1) of the ATM simulation system from blackboard. This version of the system has been partially fixed by imaginary developers based on the defect reports previously existing.
37. Navigate to the product editing page in Bugzilla, and edit your group's ATM system product to have an additional version (1.1).
38. Perform a search in Bugzilla for all defects reported by your group for version 1.0 of the ATM system product.
39. Retest each of these defects to determine which have been fixed and which have not. To do this we must follow the defect lifecycle as shown in Appendix B. Since we do not know which defects have been fixed exactly, assume that all defects have had an attempt to fix them. Update the defect status to Resolved (Fixed) by opening that defect for editing, and changing its status appropriately. If the defect has actually been fixed in the ATM system version 1.1, change the status once again to Verified. If the defect has not actually been fixed in the ATM system version 1.1, change the status to Reopened and write a comment stating "Defect still exists in version 1.1".
40. Execute steps 34 and 35 (Manual Functional Testing) once again, looking only for new defects that have been created. If a defect is found which had previously been reported, do not report it again. When reporting these defects, ensure that version 1.1 is selected.

2.6 SUMMARY

Within your group, you should now each be familiar with the main features of Bugzilla, and have a general understanding of how to use it to effectively report and track defects. You have also progressed through a short iteration of ad-hoc testing, two iterations of manual functional testing and an iteration of regression testing.

3 DELIVERABLES AND GRADING

Both students of each group must attend the lab session. If one student is missing, then the missing student can at most get 50% of the marks, if he/she helps writing the lab report. The maximum number of points is 8. Points are allocated as follows:

3.1 BUGZILLA DEFECT REPORTS (50%)

Students will be graded on their defect reports AS SAVED IN THE BUGZILLA SYSTEM. The grading criteria (rubric) for defect reports are as follows. Please do not repeat your bug reports in your lab reports!

Bugzilla Defect Reports (50%)	
Correctness: Do the defect reports contain the detailed defect information? Does the report contain all the defects in the same level of detail? Does it contain the input, the expected output, and the faulty output for each defect?	15%
Clarity and adherence to defect reporting guidelines: Is it obvious where to start (what state to bring the program to) to replicate the defect? Is it obvious what you would type? Is it obvious what files to use (if any)?	15%
Number of defects found: Note that not all defects need to be found. But if it appears that not enough effort was made in finding defects, marks may be deducted.	20%

3.2 LAB REPORT (50%)

Important: Lab report should be submitted through the submission form on the course home page. Lab reports should be submitted in PDF format only. Deadline for submissions is before the next lab session.

Students will be required to submit one report from group. In the report should be included:

Lab Report (50%)	
1. A high-level description of the ad-hoc testing plan	10%
2. The test cases performed during ad-hoc testing (in a table similar to in Appendix E would be best)	15%
3. A comparison of ad-hoc and manual functional testing (based on the provided test suite) from several perspectives (e.g., benefits, tradeoffs, effectiveness, efficiency, etc.)	15%
4. A discussion on how the team work/effort was divided and managed. Any lessons learned from your teamwork on this lab?	5%
5. Any difficulties encountered, challenges overcome, and lessons learned from performing the lab	5%
6. General comments and conclusions on performing the lab. Did you find it a useful practice? Was it easy to follow? Did you spend too little/too much time on it? Should some parts be dropped or explained better? Is something missing? Etc. Please try to be constructive.	5%

A portion of the grade for the lab report will be allocated to organization and clarity.

4 INTERESTING QUOTES AND WEB LINKS

- *"Bug Advocacy: Lesson 59 - Take the time to make your bug reports valuable"*
- *"Bug Advocacy: Lesson 57 - Make your bug report an effective sales tool"*
- *"Bug Advocacy: Lesson 62 - Report perceived quality gaps as bugs"*

- *"Bug Advocacy: Lesson 67 - Report defects promptly"*
- *"Bug Advocacy: Lesson 68 - Never assume that an obvious bug has already been filed"*
- *"Bug Advocacy: Lesson 72 - Minor bugs are worth reporting and fixing"*
- *"Bug Advocacy: Lesson 84 - Never exaggerate your bugs"*
- *"Bug Advocacy: Lesson 85 - Report the problem clearly, but don't try to solve it"*
- *"Bug Advocacy: Lesson 94 - Verify bug fixes promptly"*

From Cem Kaner's book "Lessons Learned in Software Testing"

- Cem Kaner's slides on Bug Advocacy: http://www.kaner.com/pdfs/bugadvoc.pdf
- Sun Developer Network (SDN) Article - How to Write a Helpful Bug Report: http://java.sun.com/developer/technicalArticles/bugreport_howto
- Bug Reporting Best Practices by Apple Developer Connection (for testers of iPod and iPhone!): http://developer.apple.com/bugreporter/bugbestpractices.html

5 ACKNOWLEDGEMENTS

This lab instructions was originally performed by prof. Vahid Garousi from University of Calgary.

This lab is part of a repository of a large software testing laboratory courseware available under a Creative Commons license for other testing educators at:

http://www.softqual.ucalgary.ca/projects/testing_labs

The courseware's author, Vahid Garousi, would like to thank Yuri Shewchuk and Riley Kotchorek for their helps in developing and testing these exercises and also Negar Koochakzadeh and students of the courses SENG 437 and 521 in the last few years for their careful reviews and feedbacks on this set of lab manuals.

6 REFERENCES

[1] "Bugzilla," Internet: http://www.bugzilla.org

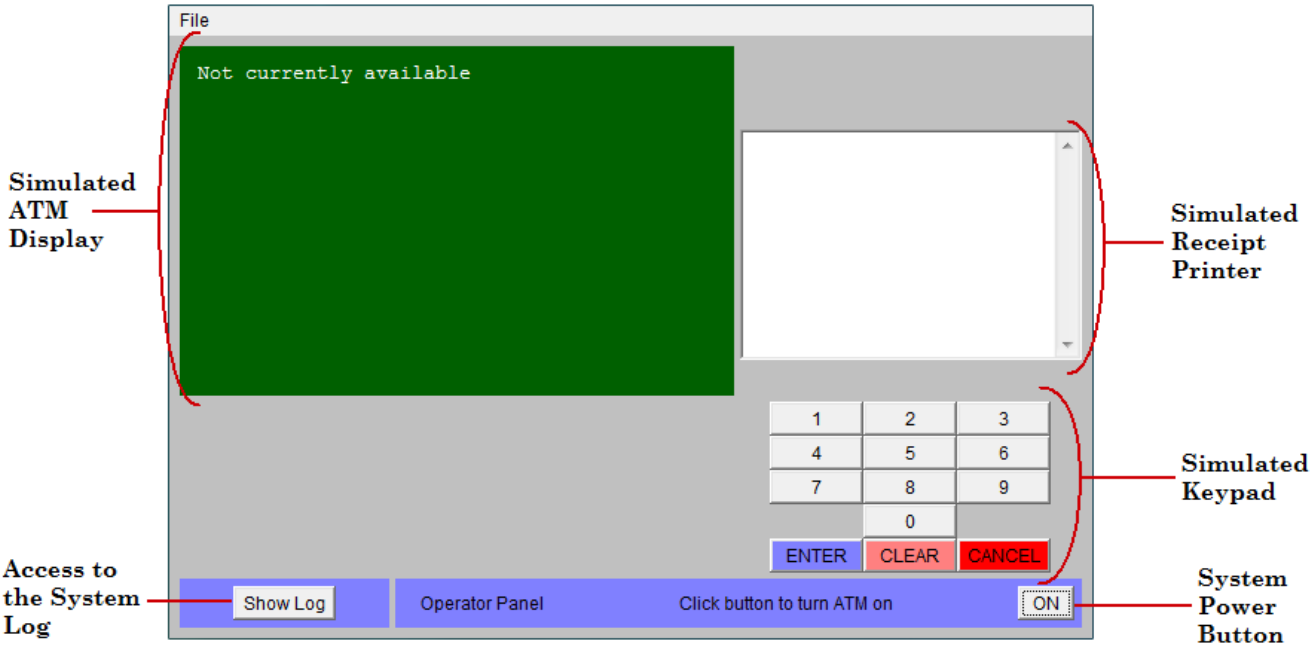
[2] R. C. Bjork, "Example ATM Simulation System," Internet: http://www.math-cs.gordon.edu/local/courses/cs211/ATMExample [June 26, 2008]

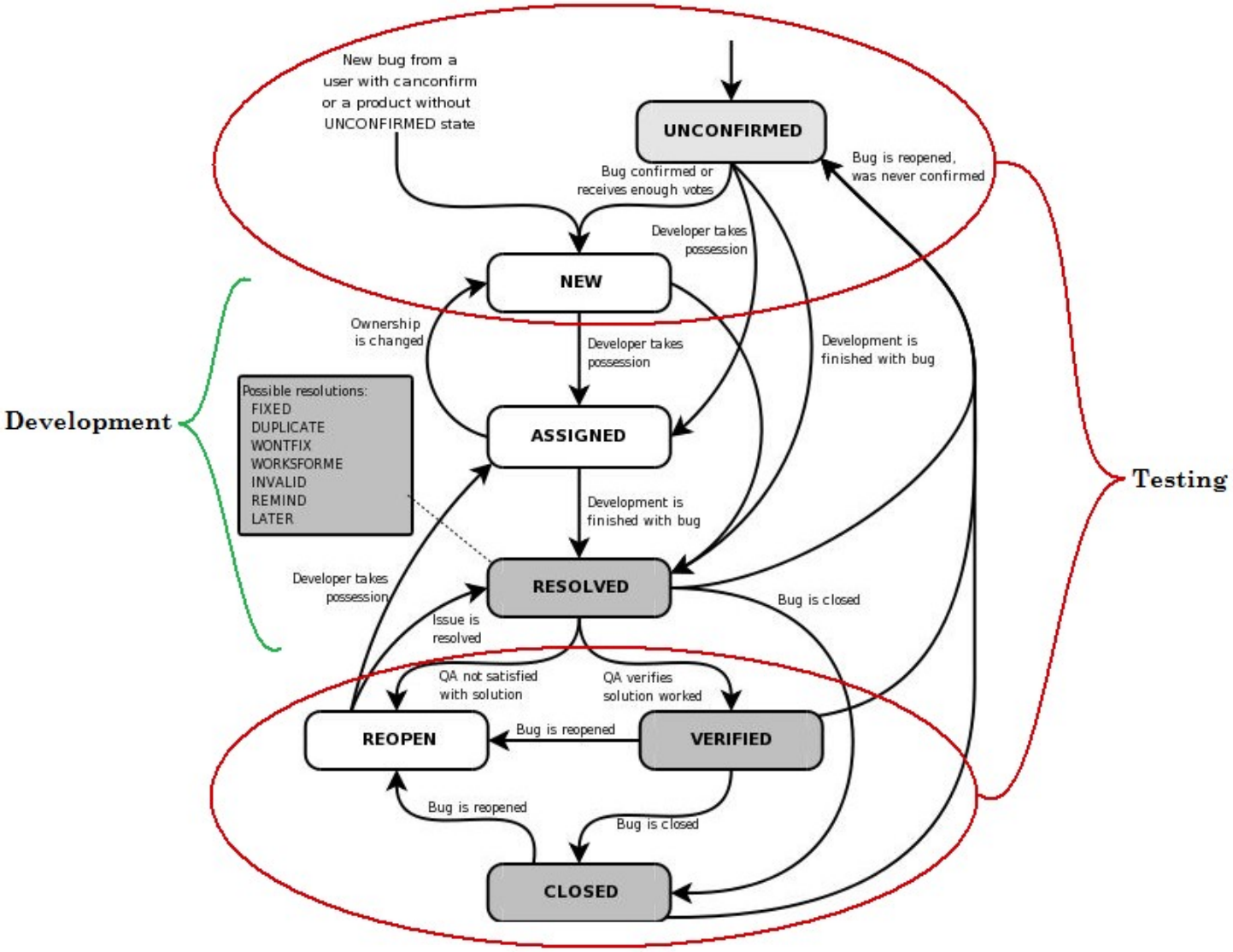
[3] J. B. Cem Kaner, Bret Pettichord, "Chapter 4 - Bug Advocacy," in *Lessons Learned in Software Testing* New York: John Wiley & Sons Inc., 2002.

[4] C. Kaner, "Assignment - Replicate and Edit Bugs," 2008.

[5] Wikipedia, "Ad-hoc testing," Internet: http://en.wikipedia.org/wiki/Ad_hoc_testing [June 26, 2008]

7 APPENDIX A – USER INTERFACE FOR ATM SIMULATION SYSTEM





8 APPENDIX B – DEFECT LIFECYCLE FROM INITIAL REPORT TO CLOSED

9 APPENDIX C – REQUIREMENTS FOR THE ATM SIMULATION SYSTEM

9.1 HIGH LEVEL REQUIREMENTS

The software to be designed will control a simulated automated teller machine (ATM) having a magnetic stripe reader for reading an ATM card, a customer console (keyboard and display) for interaction with the customer, a slot for depositing envelopes, a dispenser for cash (in multiples of $20), a printer for printing customer receipts, and a key-operated switch to allow an operator to start or stop the machine. The ATM will communicate with the bank's computer over an appropriate communication link. (The software on the latter is not part of the requirements for this problem.)

The ATM will service one customer at a time. A customer will be required to insert an ATM card and enter a personal identification number (PIN) - both of which will be sent to the bank for validation as part of each transaction. The customer will then be able to perform one or more transactions. The card will be retained in the machine until the customer indicates that he/she desires no further transactions, at which point it will be returned - except as noted below.

The ATM must be able to provide the following services to the customer:

- A customer must be able to make a cash withdrawal from any suitable account linked to the card, in multiples of $20.00. Approval must be obtained from the bank before cash is dispensed.
- A customer must be able to make a deposit to any account linked to the card, consisting of cash and/or checks in an envelope. The customer will enter the amount of the deposit into the ATM, subject to manual verification when the envelope is removed from the machine by an operator. Approval must be obtained from the bank before physically accepting the envelope.
- A customer must be able to make a transfer of money between any two accounts linked to the card.
- A customer must be able to make a balance inquiry of any account linked to the card.
- A customer must be able to abort a transaction in progress by pressing the Cancel key instead of responding to a request from the machine.

The ATM will communicate each transaction to the bank and obtain verification that it was allowed by the bank. Ordinarily, a transaction will be considered complete by the bank once it has been approved. In the case of a deposit, a second message will be sent to the bank indicating that the customer has deposited the envelope. (If the customer fails to deposit the envelope within the timeout period, or presses cancel instead, no second message will be sent to the bank and the deposit will not be credited to the customer.)

If the bank determines that the customer's PIN is invalid, the customer will be required to re-enter the PIN before a transaction can proceed. If the customer is unable to successfully enter the PIN after three tries, the card will be permanently retained by the machine, and the customer will have to contact the bank to get it back.

If a transaction fails for any reason other than an invalid PIN, the ATM will display an explanation of the problem, and will then ask the customer whether he/she wants to do another transaction.

The ATM will provide the customer with a printed receipt for each successful transaction, showing the date, time, machine location, type of transaction, account(s), amount, and ending and available balance(s) of the affected account ("to" account for transfers).

The ATM will have a key-operated switch that will allow an operator to start and stop the servicing of customers. After turning the switch to the "on" position, the operator will be required to verify and enter the total cash on hand. The machine can only be turned off when it is not servicing a customer. When the

switch is moved to the "off" position, the machine will shut down, so that the operator may remove deposit envelopes and reload the machine with cash, blank receipts, etc.

The ATM will also maintain an internal log of transactions to facilitate resolving ambiguities arising from a hardware failure in the middle of a transaction. Entries will be made in the log when the ATM is started up and shut down, for each message sent to the Bank (along with the response back, if one is expected), for the dispensing of cash, and for the receiving of an envelope. Log entries may contain card numbers and dollar amounts, but for security will *never* contain a PIN.

9.2 USE CASE DIAGRAM