**Lab 0-Introduction to Software Testing and Defect (Bug) Tracking**

Developed and maintained by: Dr. Vahid Garousi, since year 2007

[vgarousi@gmail.com](mailto:vgarousi@gmail.com)

[www.vgarousi.com](http://www.vgarousi.com)

Part of the Open Software Testing Laboratory Courseware:

[sites.google.com/view/software-testing-labs](https://sites.google.com/view/software-testing-labs/)

**Table of Contents**

[Important note about using Generative AI tools: 2](#_Toc186470958)

[Interesting quotes on defect (bug) reports 2](#_Toc186470959)

[1 Introduction 2](#_Toc186470960)

[1.1 Objectives 2](#_Toc186470961)

[1.2 Structure of the lab and lab document 2](#_Toc186470962)

[1.3 Pair Testing 3](#_Toc186470963)

[1.4 Group-work specification 3](#_Toc186470964)

[1.5 Submission style, due date and late marking policy 3](#_Toc186470965)

[2 Familiarizations 3](#_Toc186470966)

[2.1 Testing definitions 3](#_Toc186470967)

[Exploratory testing 3](#_Toc186470968)

[Manual scripted testing: 3](#_Toc186470969)

[Regression testing: 4](#_Toc186470970)

[Comparing exploratory testing versus other types of testing: 4](#_Toc186470971)

[2.2 System under test (SUT) 5](#_Toc186470972)

[2.2.1 Purpose of the system 5](#_Toc186470973)

[2.2.2 Usage of the system 5](#_Toc186470974)

[2.3 Lab artifacts 6](#_Toc186470975)

[2.4 An overview of issue (bug) tracking software systems 6](#_Toc186470976)

[2.5 Our choice of issue tracking software 7](#_Toc186470977)

[2.6 Example repositories of real defects 7](#_Toc186470978)

[2.7 What makes a “good” bug report? 7](#_Toc186470979)

[3 Instructions for the lab-work 8](#_Toc186470980)

[3.1 Setting up the Zoho issue tracking software 9](#_Toc186470981)

[3.1.1 First, create an “account”, a “portal” and a “project” in the issue tracking software (Zoho) 10](#_Toc186470982)

[3.1.2 Giving full access to both student accounts, to see the new “project” 13](#_Toc186470983)

[3.1.3 Submitting an example bug in the issue (bug) tracking system 14](#_Toc186470984)

[3.1.4 Install and try the mobile app of the bug tracking system on your mobile phone (optional) 15](#_Toc186470985)

[3.2 Perform a deposit in the ATM system 15](#_Toc186470986)

[3.3 Exploratory (manual non-scripted) testing 16](#_Toc186470987)

[3.3.1 Writing your exploratory test plans 16](#_Toc186470988)

[3.3.2 Execute your devised exploratory test plan and report defects 16](#_Toc186470989)

[3.3.3 Peer reviewing of the bug reports, filed during exploratory testing 17](#_Toc186470990)

[3.3.4 Identifying duplicate defects and marking them as duplicate in the system 18](#_Toc186470991)

[3.4 Manual scripted testing 19](#_Toc186470992)

[3.4.1 Peer reviewing of the bug reports, filed during manual scripted testing 20](#_Toc186470993)

[3.5 Regression testing (verification of defect fixes) 20](#_Toc186470994)

[3.6 Summary of the work and learning 22](#_Toc186470995)

[4 Deliverables and Grading 22](#_Toc186470996)

[4.1 Defect reports– to be recorded in Zoho 22](#_Toc186470997)

[4.2 Lab report 22](#_Toc186470998)

[4.3 Marking scheme 22](#_Toc186470999)

[Interesting web links 22](#_Toc186471000)

[Acknowledgements 22](#_Toc186471001)

[References 22](#_Toc186471002)

[Appendix A – Defect lifecycle: Different states/status modes: from “new” to “closed” 23](#_Toc186471003)

[Appendix B – Requirements / specifications for the ATM simulation system 24](#_Toc186471004)

[4.4 High-level requirements 24](#_Toc186471005)

[4.5 Use-case diagram of the SUT 25](#_Toc186471006)

[Appendix C - Functional Test Suite for Manual Scripted Testing 26](#_Toc186471007)

# Important note about using Generative AI tools:

Make sure to review in Canvas the “*Policy for using Generative AI tools (such as ChatGPT) in module assignments*”

# Interesting quotes on defect (bug) reports

* *“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”

[www.amazon.co.uk/Lessons-Learned-Software-Testing-Context-Driven/dp/0471081124](http://www.amazon.co.uk/Lessons-Learned-Software-Testing-Context-Driven/dp/0471081124)

# Introduction

## Objectives

This lab is a simple introduction to some of the basic concepts in software testing. Specifically, students will gain an understanding of some fundamentals involved in testing. After taking this lab, students will learn:

* Hands-on experience in manual testing of an example software system
* How to file bug (defect) reports
* The difference between exploratory (manual non-scripted) testing versus manual scripted testing, and regression testing
* Industrial defect tracking systems, processes and practices
* Different phases of defects’ lifecycle

## Structure of the lab and lab document

This lab is divided into these main parts:

* Section 2 (Familiarizations): The section will familiarize students with various concepts that they will need to perform and complete the lab work.
* Section 3 (Instructions for lab-work): The section details the instructions for performing the lab-work

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

During the exploratory testing phase (Section 2.5), students will be free to test the system in any manner that they choose (and are able). Once students reach the manual scripted-testing phase (Section 2.6), they will be required to use a predefined test suite to test the SUT.

Finally, students will perform some simple regression testing (Section 2.7) on an updated version of the system (corrected by a team of hypothetical developers in response to a list of defect reports), and if they notice any defects in the SUT, students should track the defects in the tracking system.

## Pair Testing

Two students of each group will work in “pair testing”. Pair Testing is a software development technique in which two team members work together at one keyboard to test the software application. One does the testing and the other analyzes or reviews the testing. This can be done between one Tester and Developer or Business Analyst or between two testers with both participants taking turns at driving the keyboard.

There are many online sources on “pair testing”, that you can find via Google search. Two interesting videos, that we suggest you to watch, are:

* Styles of pair testing: [youtu.be/ctRD2KBUYSI](https://youtu.be/ctRD2KBUYSI)
* Pair testing in sessions: [youtu.be/xoJaFVKhyIA](https://youtu.be/xoJaFVKhyIA)

## Group-work specification

Sections 2.1-2.5 of this lab (described in this document) should be performed individually (each student will perform on her/his own). Sections 2.6 and 2.7 should be performed as a group as specified (i.e., pair testing). The lab report will be completed and submitted by both students together, i.e., one lab report for each group.

## Submission style, due date and late marking policy

Please check the labs schedule in the website. You need to submit your lab reports and all the necessary files (e.g., report, test code, source code, as required by Section 3 of this lab document) by the deadline. The assignments in the submission system have been set such that they will stop accepting submissions from students when the due date is over.

Late reports: There will be no credit for reports turned after the due date. They will be returned unmarked.

# Familiarizations

This section will familiarize you with various concepts that you will need to perform and complete the lab work.

## Testing definitions

### Exploratory testing

Exploratory testing[[1]](#footnote-1) is a commonly used term for software testing performed without or with little planning and documentation. Exploratory testing is performed manually and usually without any test script.

The tests are intended to be run only once, unless a defect is discovered. Exploratory testing has been criticized because it is often not structured, but this can also be a strength. By doing exploratory testing, major issues can be found quickly. It is performed with “improvisation”; the tester seeks to find bugs with any means that seem appropriate. It contrasts to regression testing that looks for a specific issue with detailed reproduction steps, and a clear expected result. Exploratory testing is most often used as a complement to other types of testing.

### Manual scripted testing:

Manual scripted testing is the oldest and one of the most rigorous types of software testing. In this particular type of testing, test cases are designed and reviewed by the team before executing it. There are many variation of this basic approach, test cases can be created at the basic functionality level or they can be created at the scenario level.

Value of the scripted testing has been questioned by some experts in the field. They claim that scripted manual testing closes the mind of tester and limits their creativity during testing. Also, this approach is very heavy on the documentation and require considerable amount of resources to create the test scripts in the first place and they often get out dated because of the inevitable changes in the system under test (SUT).

Despite these drawbacks, manual scripted testing has been used and is in use in many organizations of all the sizes. They make test cases repeatable and easy enough for a new person to come on board and start testing with minimum supervision. Manual scripted testing is also used in places where contractual agreement states that written specification of the software must be met for the successful implementation of the project. Scripted test cases might be useful where tests are used for the benchmarking purpose and tests have to be executed exactly in the same way, every time.

### Regression testing:

Regression testing is any type of software testing that seeks to uncover new defects, or regressions, in the existing (old) functionality after new changes have been made to the Software Under Test (SUT), during software maintenance, such as functional enhancements, patches or configuration changes.

The intent of regression testing is to ensure that a change, such as a bug fix, did not introduce new faults. One of the main reasons for regression testing is that it is often extremely difficult for a software engineer to figure out how a change in one part of the software will echo in other parts of the software. Regression testing can be done in both manual and automated testing fashion. In this lab, we will do only manual regression testing.

### Comparing exploratory testing versus other types of testing:

We cannot say that one type of testing is better than others. At the end of the day, each type of testing approach (such as those discussed above) will target certain types of faults, which may not be found using other test approaches (techniques). Note: We will discuss more on this throughout our course. Thus, to deliver high-quality software, software engineering teams in industry apply / execute several test approaches.

Also, there are many good online resources comparing exploratory testing versus other types of testing, and we have discussed or will discuss them in the lectures. As another example, we suggest you to watch the following snippet of the video shown in Figure 1, which is comparing exploratory testing with confirmatory testing (also called: confirmation testing and re-testing). Confirmatory testing is a type of software testing that allows testers to retest a software product to make sure the previously posted bugs are fixed or not in the system. Among other types of testing, confirmatory testing can include manual scripted and also regression testing.



Figure 1- A good online video discussing exploratory testing and other types of testing: [youtube.com/watch?v=yPelznpCDc0](https://www.youtube.com/watch?v=yPelznpCDc0)

## System under test (SUT)

The software under test (SUT) for this lab is an ATM simulation software [2]. The executable file of this SUT can be found in the lab artifacts package in the labs online folder. There are two versions of the ATM system in the lab artifacts package which represent two consecutive releases of the software.

* ATM System - Version 1.0
* ATM System - Version 1.1

|  |
| --- |
| Important: An undisclosed number of defects have been intentionally injected into each version of the SUT.  **Question: How many defects (bugs) do we need to find, and submit as bug report in the defect management system?**   * **Answer:** Just like real testing projects, we never know how many defects a real SUT has. Thus, the lab instructions do not disclose the number of bugs in the system.   **Question: What types of defects (bugs) should we find?**   * **Answer:** Any types of defects; see the lecture slides / notes on “02-Introduction to Software Testing” and types of defects |

### Purpose of the system

This system was originally developed in an American college (called Gordon College) to demonstrate an entire iteration of the object-oriented software development methodology. The entire project is available open-source online, see reference [2] for the URL. The purpose of the system is to allow the user to deposit, withdraw, query and transfer funds to/from his/her hypothetical bank account(s).

### Usage of the system

To use the ATM simulation software, run JAR file *ATM System –Version 1.0.jar*. If the JAR file does not properly run in your PC / OS, it means that Java is not properly installed on your PC. See: <https://www.google.com/search?q=run+jar+files>

Once executed, the SIUT should begin execution with the GUI as shown below.

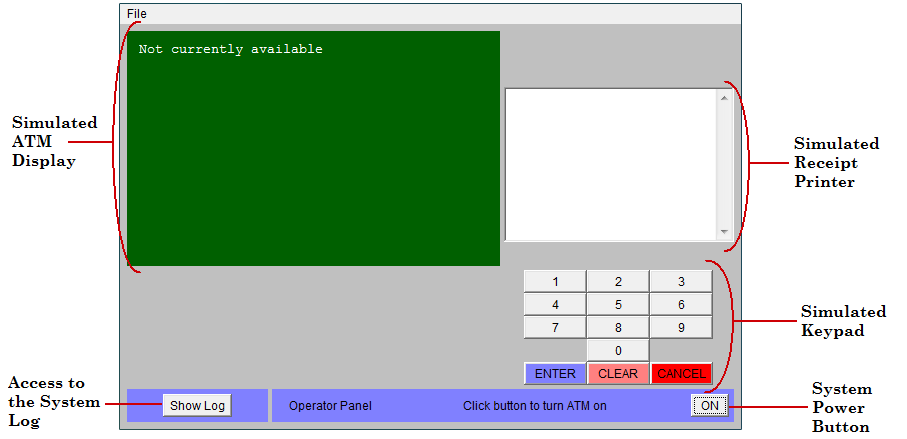


Figure 2- A screenshot from the ATM simulation software

### Login data

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

Notes:

* To learn about different types of bank accounts, see:

<https://www.google.com/search?q=account+types+Checking+and+Money+Market>

* Both the above cards access the same checking account.

Since the system was developed in the US, the supported currency is $ USD. The initial balances in the accounts are:

* Checking account: $100
* Savings account: $1,000
* Money market account: $5,000

### Performing a simple operation in the ATM system: a deposit

1. Run the JAR file *ATM System - Version 1.0.jar*. You should see the GUI as shown in Section 2.2.
2. Turn the system on using the “On” button.
3. 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 starts with $200 (10 twenty-dollar bills). Any number greater than 0 will suffice for now.
4. Click on the “Click to insert card” button which has now appeared on the main interface below the simulated ATM display.
5. 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.
6. 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.
7. 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.
8. Press 2 on the simulated keypad to deposit to the savings account. The display now prompts the user to enter the deposit amount.
9. Enter any positive amount and press *Enter*. A button which represents the user inserting the deposit envelope now appears.
10. Click that button to simulate inserting the envelope. The display now prompts the user whether they wish to perform another transaction or not.
11. 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.
12. Press the System Power Button once again to turn the ATM system off.

## Lab artifacts

Each lab in this course comes with a ZIP file or a folder named lab artifacts. Those files will include the lab artifacts for each lab, e.g., SUT executables, source code (if needed for a given lab), lab report template Word file, etc. Take the time to review the artifacts provided for the current lab.

## An overview of issue (bug) tracking software systems

There are many issue tracking systems out there, including bug (defect) tracking systems, help desk and service desk issue tracking systems. Our focus in this lab and the course are on bug (defect) tracking systems, and not the others.

A non-exhaustive list, comparing different issue tracking systems, has been provided in Wikipedia. There are literarily 100’s of such systems, both open-source and commercial. Let’s review that list:

<http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems>

## Our choice of issue tracking software

We will use the Zoho bug tracker system ([www.zoho.com/bugtracker](http://www.zoho.com/bugtracker)) in this lab. It provides a free account for one single project.

## Example repositories of real defects

Links to several example repositories of real defects for real large software projects are provided in the following. It is important for new learners to review many example defect reports to see how other experienced testers write effective and good-quality defect reports.

* Android’s public bug repository: <https://source.android.com/setup/contribute/report-bugs#bug-queues> ; then click on “BROWSE ALL PLATFORM ISSUES”
* Bug repository of the Apache platform: <https://bz.apache.org/bugzilla/> ; then click on “Search” and “Search” again
* Bug repository of the Chrome platform: <https://bugs.chromium.org/p/chromium/issues/list>

A screenshot of the Chrome issue tracking system is shown below. As you can see, in the time of taking this screenshot (September 2019), Chrome had over 52,000 open issues (too many!).

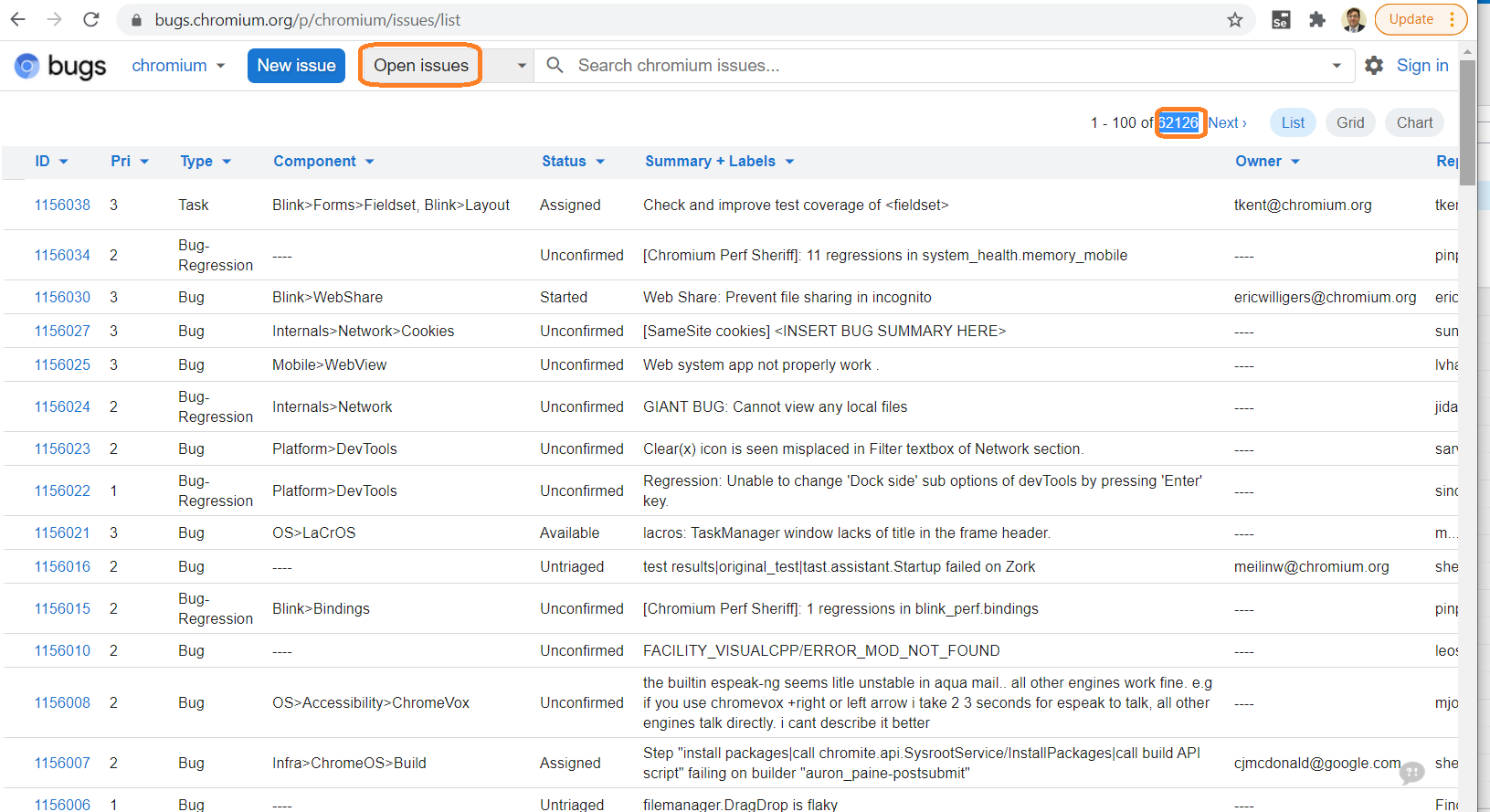


Figure 3- A screenshot of the Chrome issue-tracking system

## What makes a “good” bug report?

1. No matter what defect tracking system you are using to report defects, there are some elements that product supporters will expect and likely require. When entering your bug reports, you should be sure to include and make clear the following:

* The function being tested (e.g., Login)
* The initial state of the system (e.g., System is on and is idle, i.e., not already serving a customer)
* **Steps to reproduce the defect/bug** (e.g., Insert a card, then enter correct card number and PIN)
* What was the expected outcome? (e.g., the system successfully accepts the customer, and shows the banking menu)
* What was the actual outcome?

1. You can learn about what developers in the industry think make good bug reports. Reporting defect reports in a simple / concise manner ensures that the software engineer who reads the defect report will know what the issue is, and will be able to fix it more effectively and efficiently. There are many articles and online sources on this topic. Before reporting defects, we suggest you read, learn from, and use the guidelines from the following resources:

|  |  |
| --- | --- |
| A group of people with text  Description automatically generated  [www.slideshare.net/nicbet/the-quality-of-bug-reports-in-eclipse-etx07](http://www.slideshare.net/nicbet/the-quality-of-bug-reports-in-eclipse-etx07) | A person in a pink shirt  Description automatically generated  [www.youtube.com/watch?v=Ze\_nGboSmZ4](http://www.youtube.com/watch?v=Ze_nGboSmZ4) |
| A close-up of a document  Description automatically generated  [www.st.cs.uni-saarland.de/publications/files/bettenburg-tr-2007.pdf](http://www.st.cs.uni-saarland.de/publications/files/bettenburg-tr-2007.pdf) | A screenshot of a computer  Description automatically generated  [developer.mozilla.org/en-US/docs/Mozilla/QA/Bug\_writing\_guidelines](https://developer.mozilla.org/en-US/docs/Mozilla/QA/Bug_writing_guidelines) |

Figure 4- Some guidelines for writing good bug reports

# Instructions for the lab-work

This section provides instructions for performing the lab work. Topic of each sub-section and whether that section shall be individually (by each student of each group), or as a group, is explained next.

|  |  |  |
| --- | --- | --- |
| **Sub-section** | **Topic** | Whether that section shall be individually (by each student of each group), or as a group |
| 3.1 | Setting up your project’s bug repository in the Zoho bug tracking software | The group should have ONE SINGLE bug repository in the Zoho. But each student should have a separate account in the system, with access to that bug repository. See the sub-section for details. |
| 3.2 | Exploratory (manual non-scripted) testing | Individually by each student |
| 3.3 | Manual scripted testing | Done together as a group |
| 3.4 | Regression testing (re-testing the software after it has been changed) | Done together as a group |

|  |
| --- |
| IMPORTANT NOTE: While this extensive lab document has provided detailed instructions on what should be done, we do not intend to mention each minor task to be done in the lab doc. As a Software Test Engineer in training, you need to make your own judgements, and may need to make your own assumptions, if and when needed. When you do so, ensure stating your judgements and assumptions in the corresponding section of your lab report. |

## Setting up your project’s bug repository in the Zoho bug tracking software

We will use the Zoho bug tracker software ([www.zoho.com/bugtracker](http://www.zoho.com/bugtracker)) in this lab. It provides a free account for one single project. To create more projects in one account, paid registration is needed and we will not use that feature. But you need to pay close attention to the tool’s limited-time free license, details below:

**The last time we checked it in their page** [**www.zoho.com/bugtracker/pricing.html**](http://www.zoho.com/bugtracker/pricing.html)**, it said: “10 days free trial”. That period starts from the time you create your account. When that free trial period is over, we believe the tool will close your account and you cannot access your bugs and data anymore. Thus, make sure to take your bug-report screenshots from your repository (to be included in your lab report) before the expiry of your free license.**

Zoho is only used for lab0 of our course, so you will not need it anymore after lab0.

To learn about this tool, you can watch an introduction video of the tool: [youtu.be/ZzXneqMTrrs](https://youtu.be/ZzXneqMTrrs):

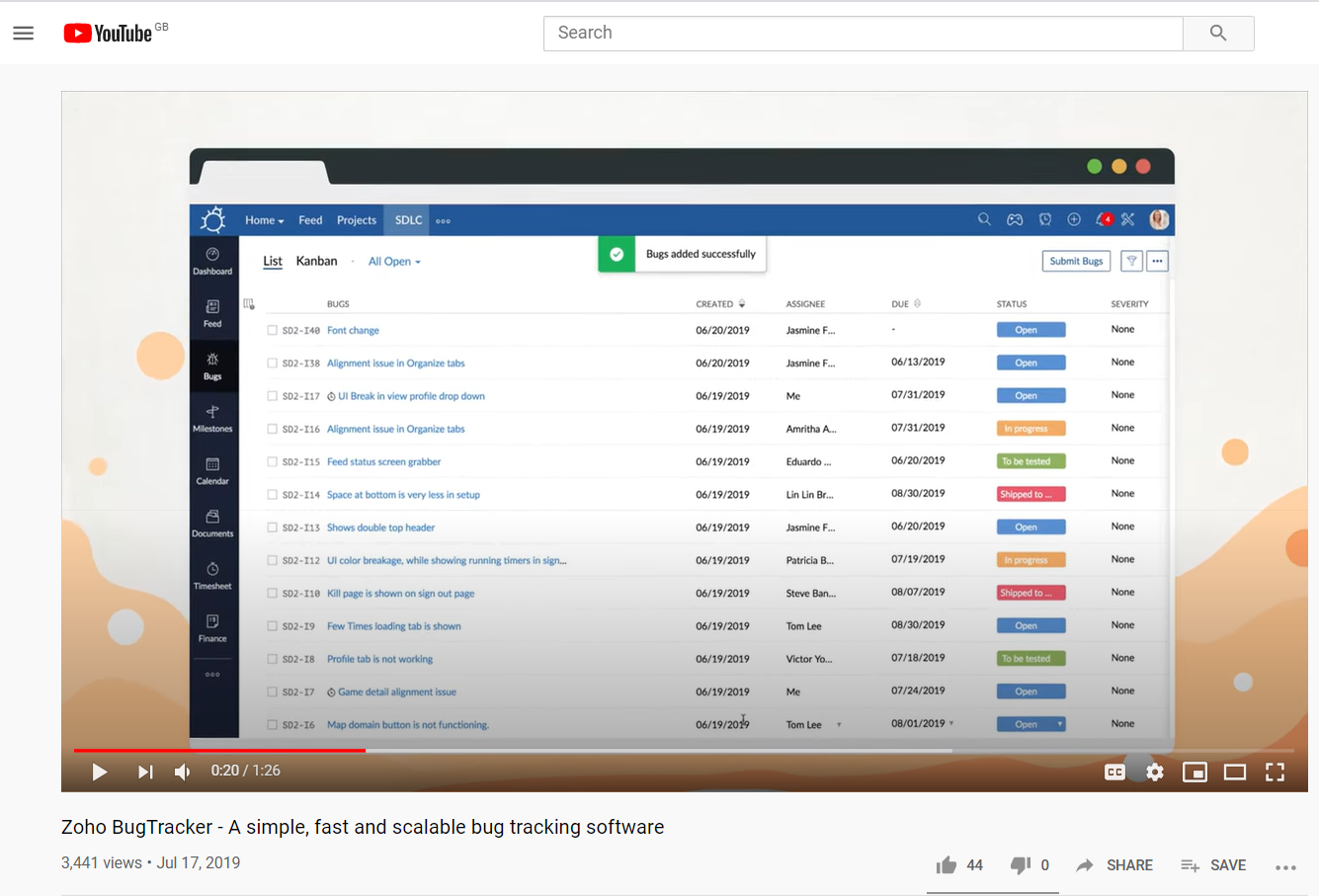


Figure 5- An introduction video ([youtu.be/ZzXneqMTrrs](https://youtu.be/ZzXneqMTrrs)) of the Zoho issue tracking software

### First, create an “account”, a “portal” and a “project” in the issue tracking software (Zoho)

1. The first step for you is to create a user “account” for each student, then a **SINGLE “portal”, and then a SINGLE “project” for the team** in Zoho. Note that these three terms are not the same in the context of Zoho. Take your time in the system to fully understand these concepts. A page in the Zoho help pages mentions[[2]](#footnote-2):

*“In Zoho Projects, only one portal can be created using an e-mail id. However, the user with the same e-mail id can be added as either Admin or Manager or Employee or Contractor or Client in any number of portals.”*

1. You need to sign up for a new account: You can either use your QUB email or any other email (such as Gmail) for user registration.
2. One of the students should create A PORTAL and A PROJECT in her/his account, as the group’s single portal and project. Then, s/he should give access to the other student, as discussed below.
3. **Important notes:** 
   * Each student of a team should have a separate user account in the Zoho system
   * BUT you need to create one single “portal” in Zoho, corresponding to your group, and name it by using the following naming convention.
   * Make sure to include your group ID in the “company” and “portal” names, using the naming format, shown in Figure 6 below: *year* and then *your group number*. For example, if the current year is 2025 and your group number is G12, the “portal” name should be 2025G12. Following this naming conversion by all students is important for the course teaching staff to find your profile if they need to work, from among 100’s of student projects, in Zoho, in different years. Be careful that, once a portal is created, it is impossible to rename it in Zoho; thus, follow these steps carefully.
   * Note: If your portal name and the set-up details discussed above do not follow the above notes, that can result in some loss of your lab mark. Again, do not forget that you need to create and use only ONE SINGLE PORTAL for your group.
   * There are three different concepts in Zoho: Company, Portal and Project. A company can have several portals and, a portal can have several projects. Thus, ensure learning the terminology carefully.
   * Essentially, one of the students should create A PORTAL and A PROJECT in her/his account, as the group’s single portal and project. Then, s/he should give access to the other student, as discussed below.
   * Both student accounts shall be included in the portal (as administrators), so that you can both work in that environment (add bugs, view /edit them, etc.)

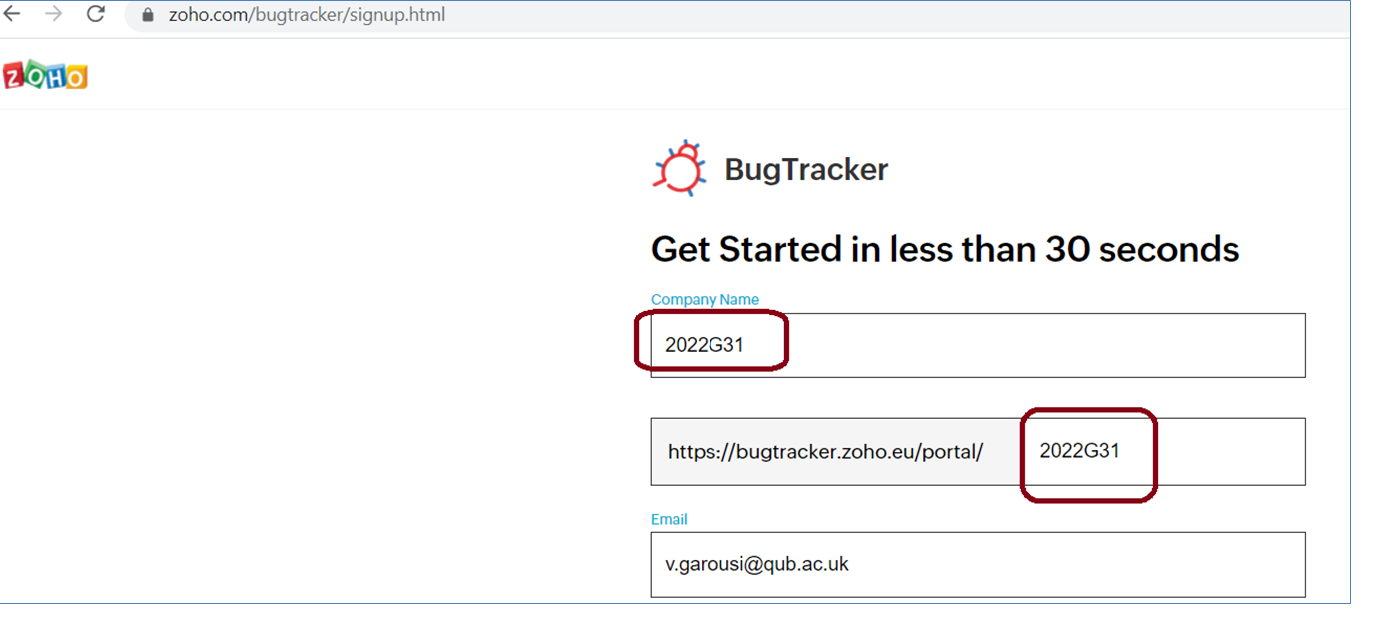


Figure 6- Creating an “account”, a “company”, and a “portal” in Zoho issue tracking software, and the format to name the “company” and “portal” names

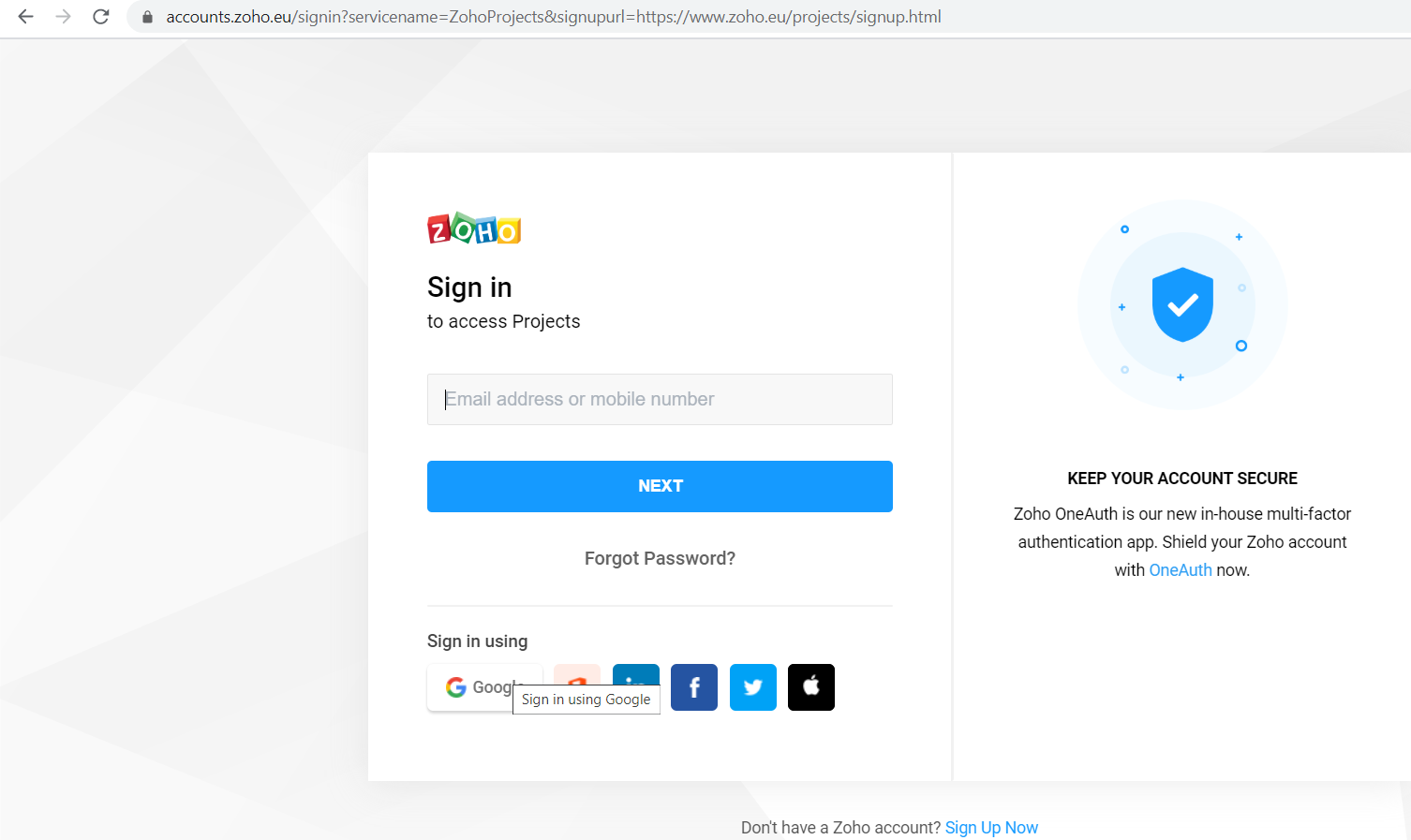


Figure 7- Sign-in to Zoho issue tracking software

1. Zoho used to have a feature to add more portals under one “company” / user account (Figure 8), but it seems they have removed that feature from the free accounts. Thus, if for whatever reasons, you made mistakes in creating and naming your new “portal” (as described above), you may have to create another Zoho user account with another email address.

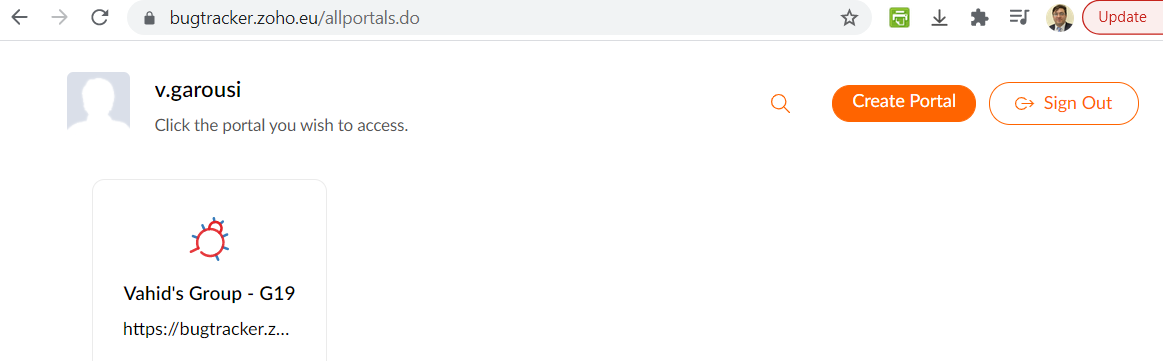
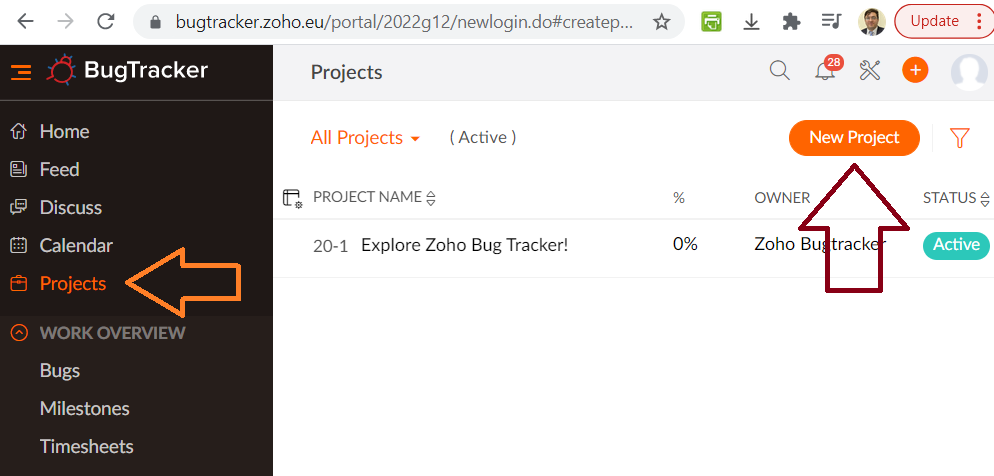


Figure 8- Creating new “portals” in the Zoho software

1. Go to “Projects” page (using the left menu) and create a project for this lab. You should name the “project” using the name format provided in Figure 9. For example, for group12, the project name will be: *Lab0-Testing ATM-G12*.
   * **Note: To prevent inconsistencies, not following the name format below can result in loss of marks.**



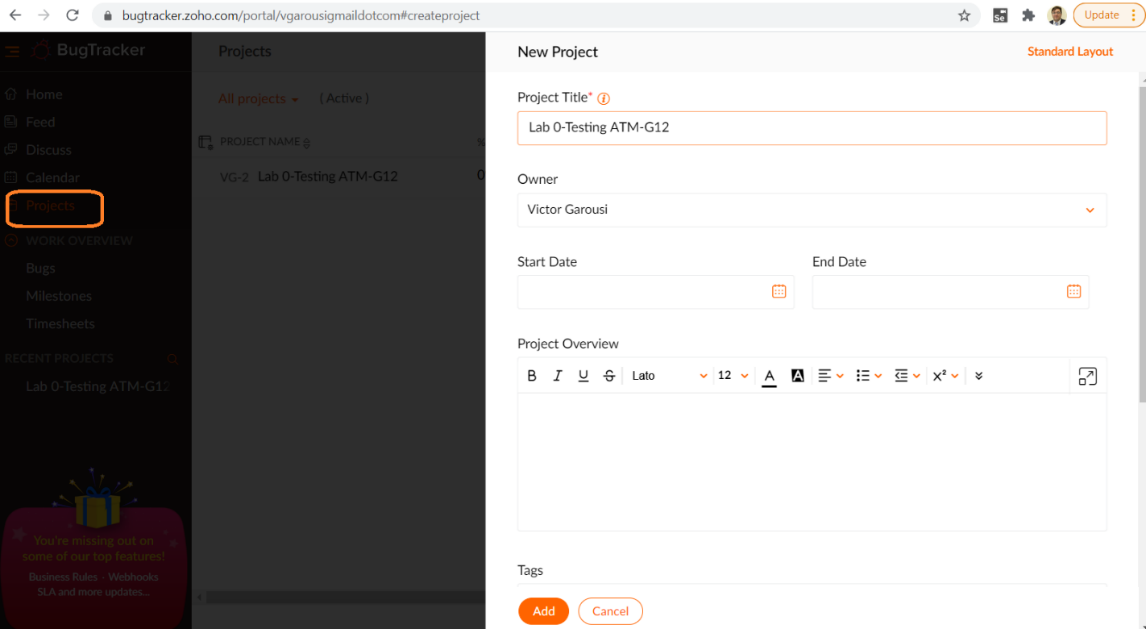


Figure 9- Creating a “project” in the Zoho software

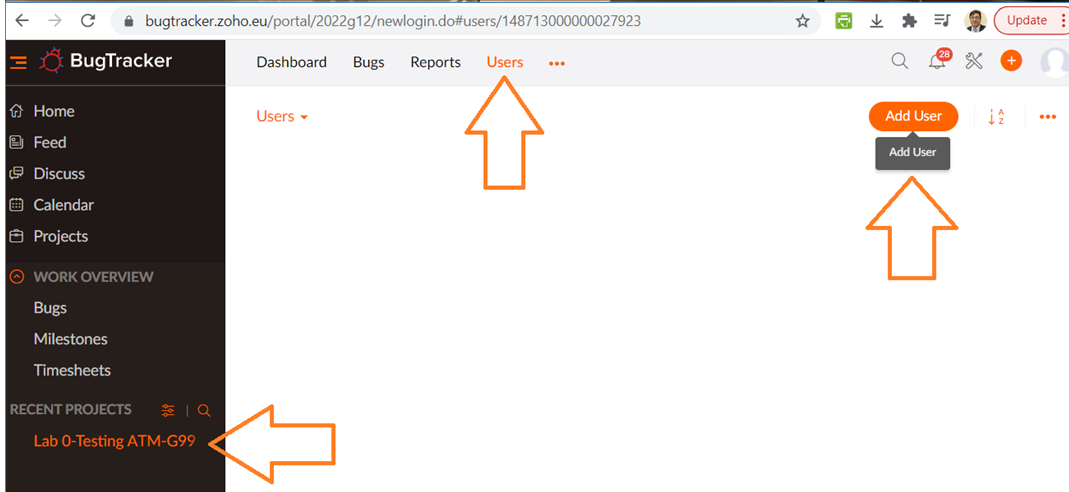
1. Click on your new project and go inside its page
2. Familiarize yourselves with the Zoho web application, its features and GUI

A short video on how to do the initial setup of your Zoho portal, following all the above steps, is provided here: [youtube.com/watch?v=FOX7\_q-WtJc](https://www.youtube.com/watch?v=FOX7_q-WtJc)

### Giving full access to both student accounts, to see the new “project” in Zoho

1. As shown below, the student who created the Zoho profile, should go to “Users” page in his/her Zoho profile and should then add the email address (Zoho user) of the other student in the team, as an Admin/Manager to the project.

* **Important:** For the free Zoho accounts that we are using, Zoho has a limitation on the maximum number of other users to be on a project. Thus, you need to carefully add the users since some of the user actions (adding/removing users) cannot be undone, and you will later get error messages from Zoho about not being able to add more users.



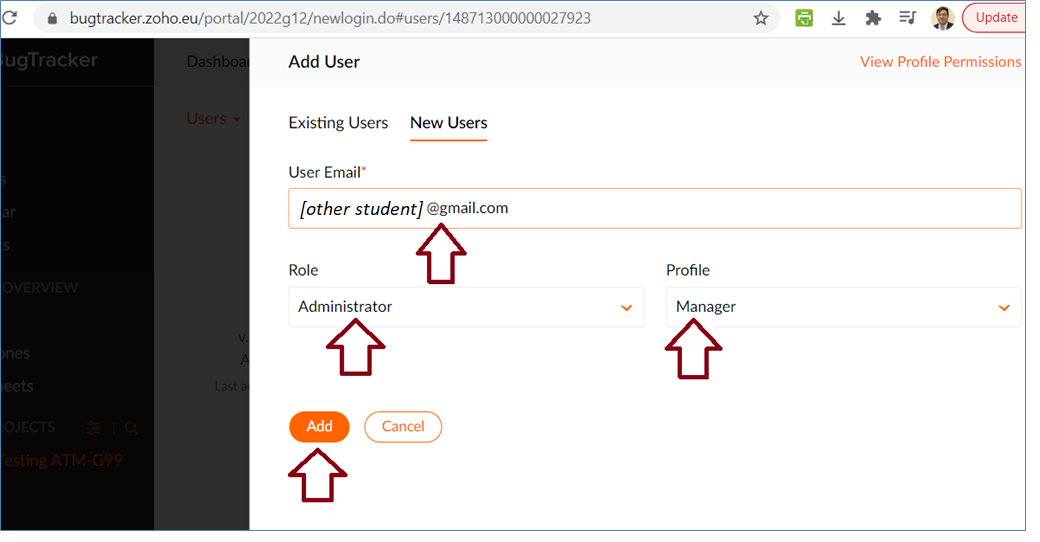


Figure 10- Adding access to your project, for other users, in the Zoho issue tracking software

### Submitting an example bug in the issue (bug) tracking system

1. Each student should submit an example dummy bug, as shown below in Figure 11. **Make sure you use** the following template in ALL your bug reports and to include all the important information, as follows:

* The function being tested (e.g., Login use-case)
* The initial state of the system (e.g., System is on and is idle, i.e., not already serving a customer)
* Steps to reproduce the defect (e.g., Insert a card, then enter the correct card number and PIN)
* What was the expected outcome? (e.g., the system successfully accepts the customer, and shows the banking menu)
* What was the actual outcome?

**IMPORTANT POINT: Initial state should be one phrase only, while the Steps section should include all the step-wise details**

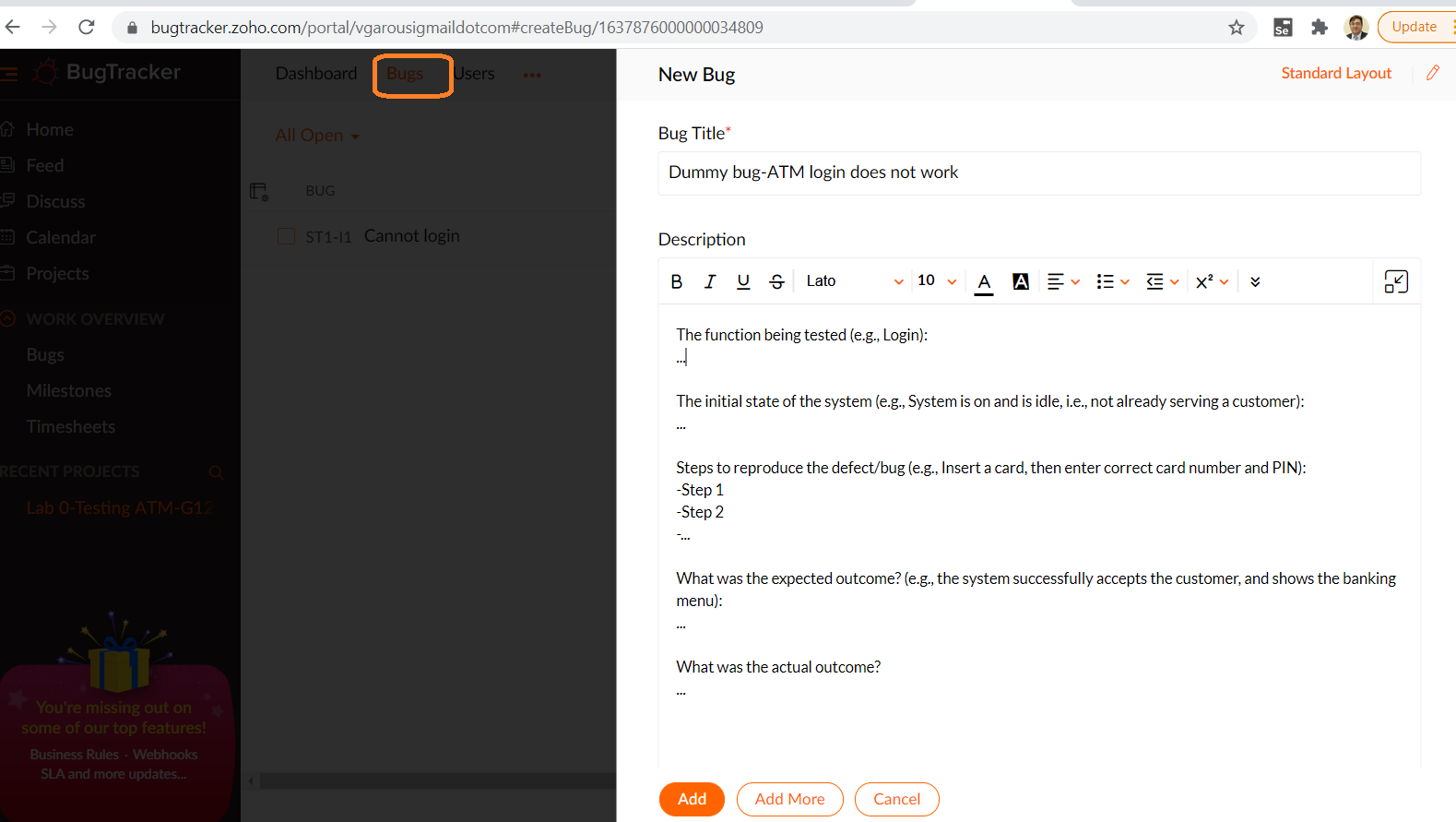


Figure 11- Entering a new bug

1. Make sure to familiarize yourself with different fields of the new bug (defect), as those fields are important for different team members (test engineers, software developers and managers) during the lifecycle of a defect. You can for example, “assign” the bug to a team member (as seen above), meaning that you essentially assign the bug to her/him to be fixed. But for the purpose of this lab, to keep things simple, we will not use the bug assignment feature.

### Install and try the mobile app of the bug tracking system on your mobile phone (optional)

1. Zoho Bug Tracker also has a mobile app: [www.zoho.com/bugtracker/mobile.html](http://www.zoho.com/bugtracker/mobile.html). See the screenshot below in Figure 12

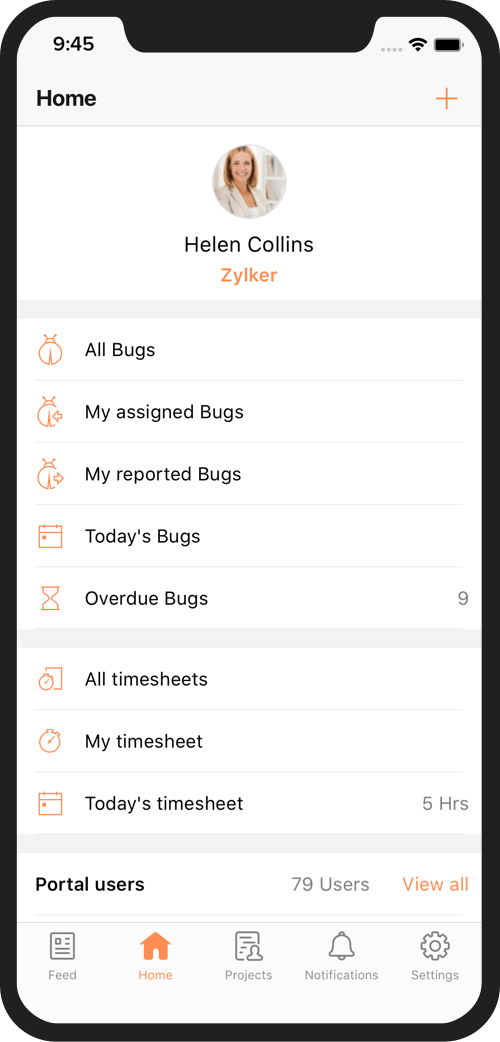


Figure 12- Mobile app of Zoho

1. If you wish, you may install and try the mobile app as well, in addition to the system’s web interface. So that in future, when you work in the software industry as a test engineer, you will have the experience of working with such apps.

## Exploratory testing

**This section shall be done individually (by each student of each group).**

1. In order to perform any type of testing, we need to first review and learn the software requirements of the SUT. Read over the requirements for the ATM simulation system as outlined in Appendix B before continuing with the rest of this section.

### Writing your exploratory test plans

1. Before beginning executing your exploratory testing, each team member needs to write an exploratory test plan for how they intend to test the system
   * There are many ways to plan any testing activity, including exploratory testing
   * You can see many hints online about this: <https://www.google.com/search?q=planning+exploratory+testing>
   * We provide a “minimalistic” approach to plan exploratory testing in the Appendix C, that we want all students to use and follow in this lab
   * Each lab report should contain two test plans, one by each student. See the lab report template to see how to present them.

### Execute your devised exploratory test plan and report defects

1. Carry out the exploratory test plan that you have devised for your team. Recall that this phase of exploratory testing should be done individually. Each student needs to perform exploratory testing independently, using her/his own test plan (designed in the previous phase), and record defects using their own account in Zoho.
2. The list of defects and also the names of bug reporter in the issue tracking system will be reviewed by the module instructor and/or the TAs after the lab to assess each student’s activity. While performing your testing, record the following information for each test case.

While performing the tests, if any of the actual results differ from the expected results, report that as a defect in the issue tracking system. Reporting defects as they are found follows the heuristic of reporting defects promptly[[3]](#footnote-3) (before forgetting the defect and its detailed conditions).

### Peer reviewing of the bug reports, filed during exploratory testing

1. Once you and your lab partner have each completed the exploratory testing phase, perform a search in the issue tracking system to list all the defects reported by your partner.
   * To do this, you can either use the search feature, or the filter feature shown below, and then choose your partner’s name.

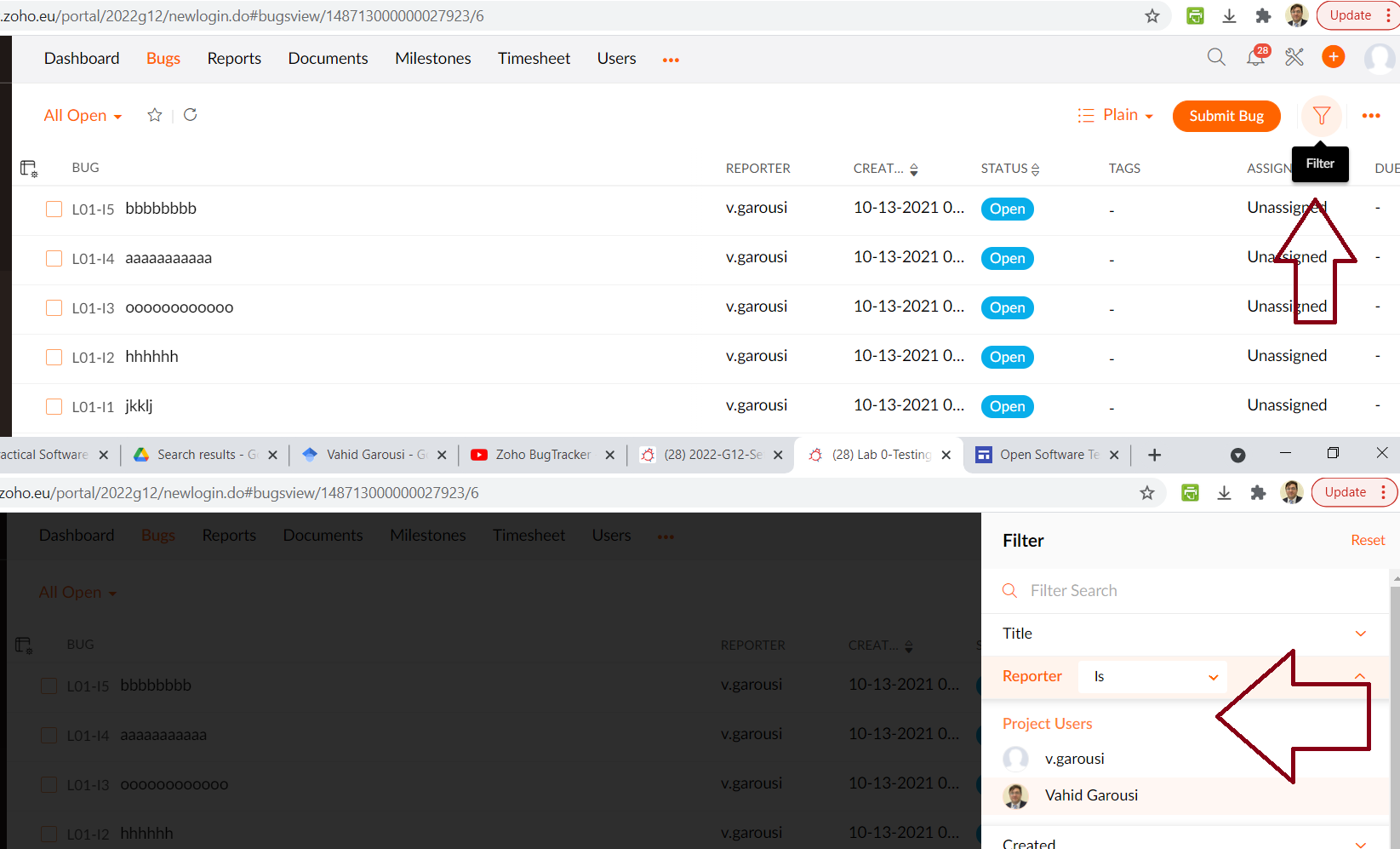


Figure 13- Filtering bugs in Zoho

* + Now, each student should peer review at least half of the defect reports that her/his partner has entered[[4]](#footnote-4). To do that, use the checklist in Table 1 below that we have provided for peer reviewing a given defect report. If there are any issue w.r.t. any of these eight criteria, add a comment for that bug report in the bug management system.

Table 1-Checklist for reviewing a given defect report

|  |  |
| --- | --- |
| 1-Does the bug report clearly mention: The function being tested? | Yes  No |
| 2-Does the bug report clearly mention: the initial state of the system, before the bug is observed? | Yes  No |
| 3-Does the bug report clearly mention: Steps to reproduce the defect? | Yes  No |
| 4-Does the bug report clearly mention: the expected outcome / output? | Yes  No |
| 5-Does the bug report clearly mention: the actual outcome/ output? | Yes  No |
| 6: Is the nature of bug report based on precise evaluation or speculation? Speculation could be when the report includes personal opinions or anecdotes that seem out of place. | Precise evaluation  Speculation |
| 7: Does the report seem too long? Too short? Does it seem to have a lot of unnecessary steps? | Reasonable length  Too long / too short |
| 8: Is the bug report written precisely or overly general? “Open a file and you will see the issue!” – What file? What kind of file? | Precise  Overly general |

* + If you see issues in any of the bug reports of your partner, using the above checklist, you should add your suggestions to the comment area of that bug in the bug repository system, for your partner to review later on. To keep your effort manageable, the partner does NOT need to improve the bug report based on the peer-review comments, but all team members should review those feedbacks to ensure knowing their mistakes and for improving their learning.
  + When you review a set of bug reports (let’s say *n* reports), and want to provide the details of your work in lab report, to be organized, place a tabular form of the above checklist in your lab report, like the dummy table including several dummy rows, shown in Table 2.

Table 2-Showing the review checklist for a set of defect reports

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Peer-review checklist criteria | | | | | | | |  |
| **Bug ID** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **Comment entered in the bug repository?** |
| 324 | √ | √ | √ | √ | √ | √ | √ | x | Yes |
| 591 | √ | √ | √ | √ | √ | √ | √ | √ | No need |
| 324 | √ | x | x | √ | √ | √ | x | √ | Yes |
| … |  |  |  |  |  |  |  |  |  |

* + In the peer review section of your lab report, each group member needs to mention the number of bug reports reviewed. Also each group member needs to provide in the report a few examples of those issues identified while peer reviewing. You do not need to provide in the lab report all the issues (that you have found).
  + Finally, in the lab report, provide a brief discussion on group’s learning (What did you learn from it?) and overall observation on peer-reviewing of bug reports by the group.

### Identifying duplicate defects (found by both students during exploratory testing)

1. Coordinate with your partner to mark any defects which both of you have found and reported during exploratory testing, and mark them as duplicates. Obviously, it is not a mistake if you both have filed a lot of duplicate defects. It is actually a good sign to find lots of duplicate bugs, after independent testing by each tester in the group, since it shows that, via careful testing, you both have found the same issues in the software.
   * To mark two given defects as duplicates, open one of the two duplicate defect reports for editing. In the “Status” drop-down menu of this defect report, mark it as a “Duplicate” bug. It is almost certain that the bug status list in your profile area in Zoho may not contain all these bug status items (as per the defect lifecycle in Appendix A). Thus, you need to add further status values for bugs in Zoho, which is quite easy to do. To help you to do this, Dr. Garousi has provided a short tutorial video in YouTube: [youtube.com/watch?v=IFR8h9Uwl0E](https://www.youtube.com/watch?v=IFR8h9Uwl0E) (a snapshot of the video is shown below).

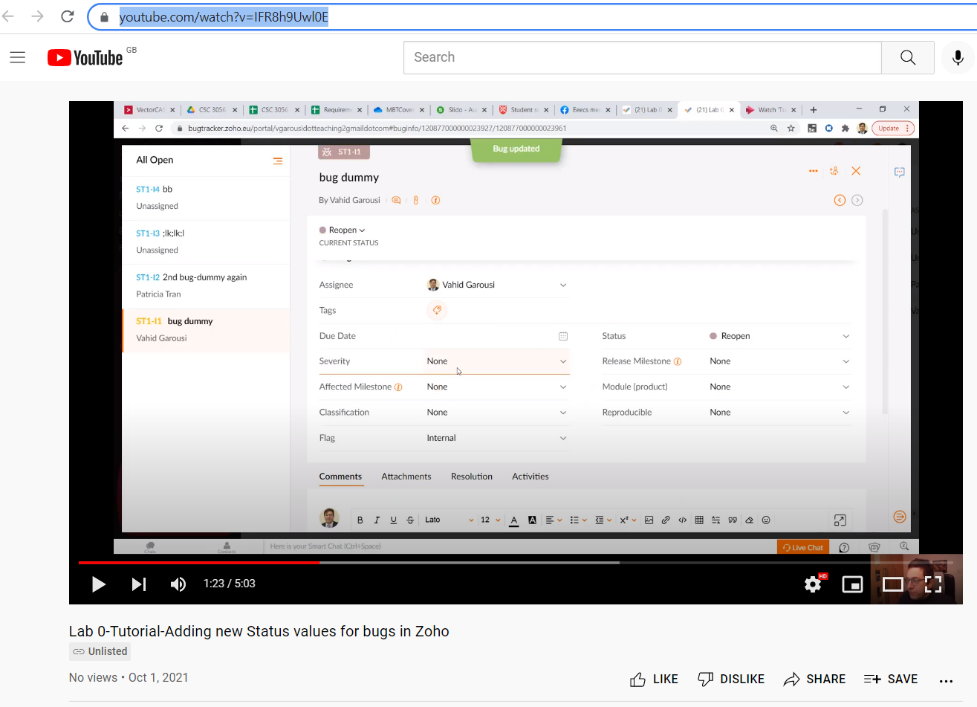


Figure 14- Tutorial video: Adding new status values for bugs in Zoho

* + When you mark a given bug as a duplicate bug, some bug tracking software systems allow you to enter the defect number of the other (equivalent) bug. If this feature is not available, you need to add a comment to both bug reports to indicate that they are equivalents, for example, if bug numbers #912 and #782 are duplicate:
    - In the comment box of Bug #912, we add: *“This bug report is duplicate (equivalent) with Bug #782”*.
    - In the comment box of Bug #782, we add: *“This bug report is duplicate (equivalent) with Bug #912”*.
  + Doing the above would ensure that each defect has only one defect report associated with it.

## Manual scripted 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. Note that it does not matter which student reports the defects, as it is a group effort.

1. For executing manual scripted testing, you should follow the provided test suite in Appendix C. 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 Section 2.3. In order to differentiate between defects found during this stage and the previous stage, in the summary field type “MFT”, stands for: Manual Functional Testing, before the summary of the defect. Do not report defects which have already been found by your group during the exploratory 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.
   * Note: During manual scripted testing, you may find some bugs which you had already found during exploratory testing (in Section 2.4). You should not see this as an ineffective (duplicate) approach; since in real projects, both manual scripted testing and exploratory testing are often done by test engineers to ensure preventing any bugs from leaking to the field (post-release of software).
   * Important: Keep track of the exact number of defects found and reported by each student in Zoho, for each test approach; as you will need to report the exact numbers in your lab report.
2. At any time, if you want to seethe bug report of MFT phase, you can perform a search in the issue tracking system for defects containing “MFT”, like shown below in Figure 15. This will produce a list of the newly added defect reports.

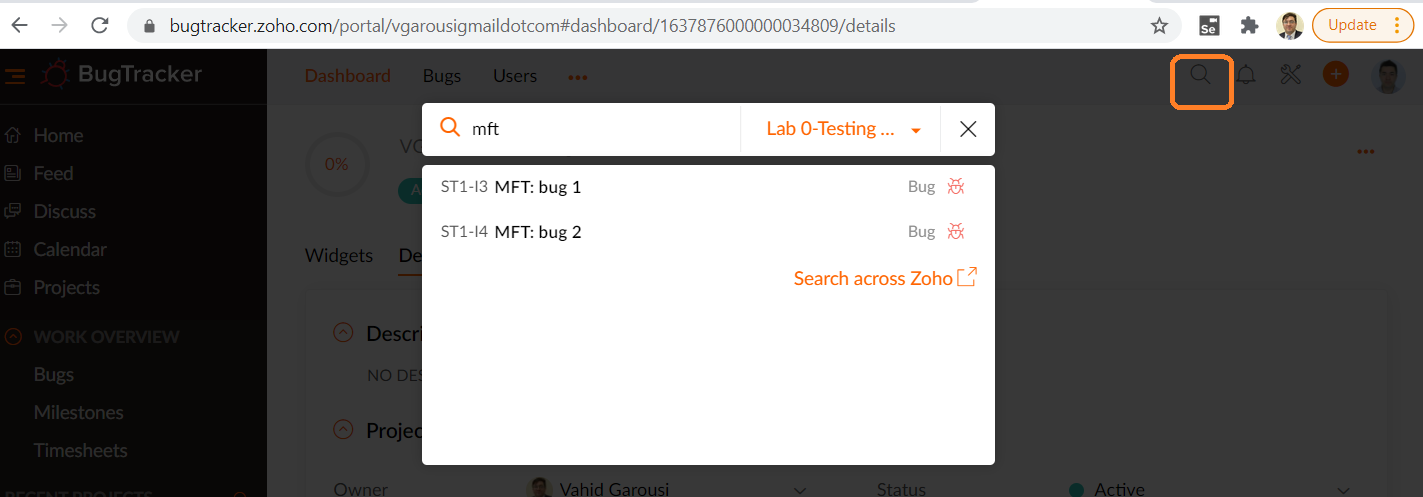


Figure 15- Searching for bugs

## Regression testing (verification of defect fixes)

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.

1. Download the updated version (version 1.1) of the ATM simulation system from the website. This version of the system has been partially fixed by imaginary developers based on the defect reports previously existing.
2. In the bug tracking system, configure your project so that your SUT has an additional version (1.1); in addition to the version 1.0 that you tested in all previous steps. In Zoho software in particular, this can be done as shown in the following short tutorial video: [youtube.com/watch?v=FY1\_GUvBfOs](https://www.youtube.com/watch?v=FY1_GUvBfOs) (watch it).

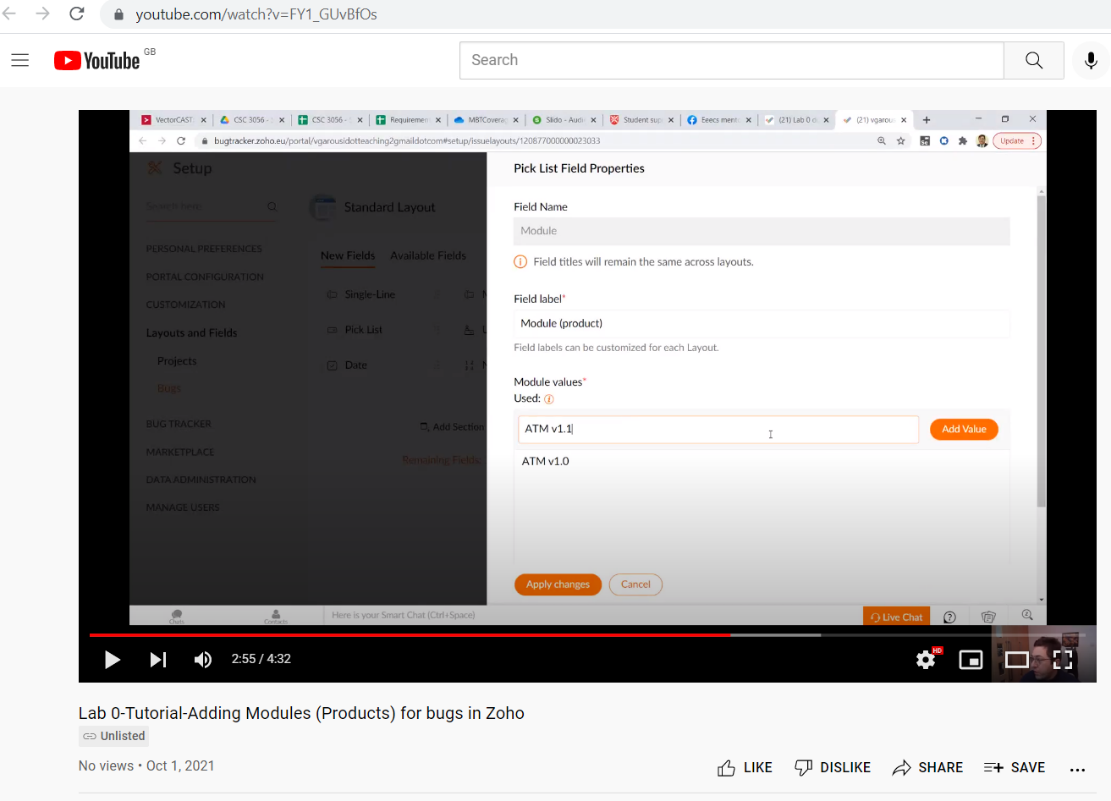


Figure 16- Tutorial video: Adding additional “modules” (product versions) in Zoho

1. Navigate to the product editing page in the issue tracking system, and
2. Perform a search in the issue tracking system for all defects reported by your group for version 1.0 of the ATM system product.
3. Retest each of these defects to determine which have been fixed and which have not. To do this, you should follow the defect lifecycle as shown in Appendix A.
   * 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. For the case of Duplicate bug reports (assume we have two), note that one of them would have status = Open and the 2nd copy one will be Duplicate ... Thus, you can set the "Open" bug report to *Resolved*
   * Retest each of these defects to determine the ones that have been fixed. 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*”.
   * To clarify the above steps: you need to first set status of all bug reports to *Resolved*, and then to either to *Verified* or *Reopened*, based on whether they have been really fixed or not. That work-flow is shown in the defect lifecycle in Appendix A. (it means that no *Resolved* defect report should be left, at the end)
   * Review the defect lifecycle shown in Appendix A with your group member to ensure full understanding of what you have done and also other status nodes in that defect lifecycle.
4. Execute the steps of “Manual Scripted Testing” (above) once again, looking only for new defects that have occurred (injected by developers) in the system. 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.
   * Important: Keep track of the exact number of defects found and reported by each student in Zoho, for each test approach; as you will need to report the exact numbers in your lab report.

## Summary of the lab work and learning

Within your group, you should each have become familiar with the main features of the issue tracking system, 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 exploratory testing, two iterations of manual functional testing and an iteration of regression testing.

# Deliverables and Grading

## Defect reports– to be recorded in Zoho

Students will be graded on their defect reports as saved in the issue tracking system, and as provided in the lab report.

## Lab report

* Students should use the template Word file “*Lab 0-Report Template*” provided in the provided lab artifacts package.
* Specific instructions for completing the lab report have been provided in the template Word file “*Lab 0-Report Template*”
* One single lab report should be completed by students together, i.e., one lab report for each group. Only one student of each group should submit the report in Canvas.
* Do not convert the Word to PDF file. Students should upload the lab report in Word DOCX format (not PDF).
* Although in the lab documents, we do not provide explicit minimum textual requirements for each of the lab report sections below (for example, minimum number of Words, etc.), each section should provide enough discussions / information. Otherwise, mark deductions will be made by the teaching team for sections which are seen “too short” in the report.

Note: Failure to follow any of the above rules will result in loss of grades.

## Marking scheme

For the detailed grading criteria (marking scheme), see the file in Canvas.

# Acknowledgements

This lab is part of a software-testing laboratory courseware available under a Creative Commons license. The laboratory courseware has been used by 50+ testing educators world-wide. More details can be found in the courseware’s website:

[sites.google.com/view/software-testing-labs/](https://sites.google.com/view/software-testing-labs/)

The courseware has been made a reality thanks to many people, including Yuri Shewchuk, Michael Godwin, Christian Wiederseiner, Riley Kotchorek and Negar Koochakzadeh for their helps in developing and testing these exercises.

# References

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

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

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

[4] Wikipedia, "Exploratory testing," Internet: <http://en.wikipedia.org/wiki/Exploratory_testing> [October 24, 2011]

[5] Wikipedia, " Scrum (development)," Internet: <http://en.wikipedia.org/wiki/Scrum_(development)> [Nov 5, 2011]

# Appendix A – Defect lifecycle: Different states/status modes: from “new” to “closed”

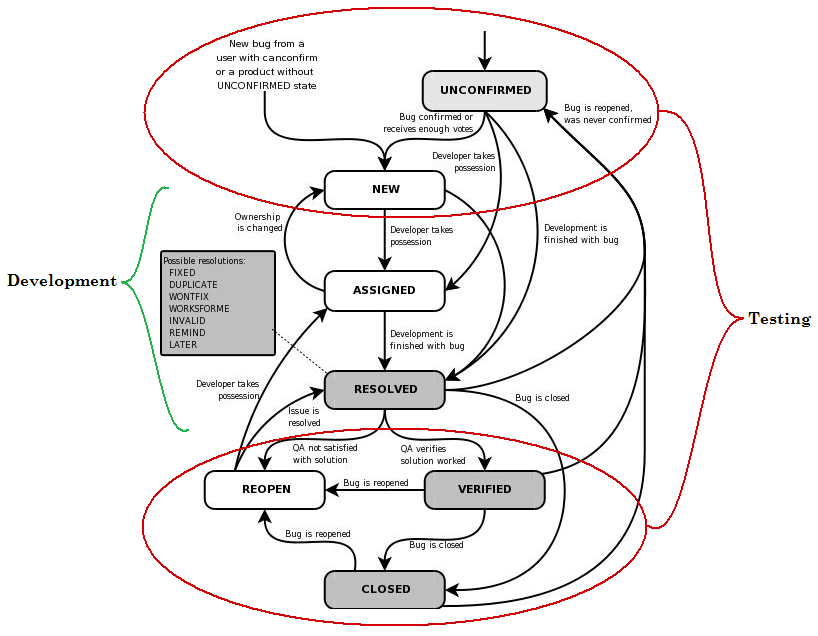


Figure 17-

# Appendix B – Requirements / specifications for the ATM simulation system

## 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 service 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, it will *never* contain a PIN.

## Use-case diagram of the SUT

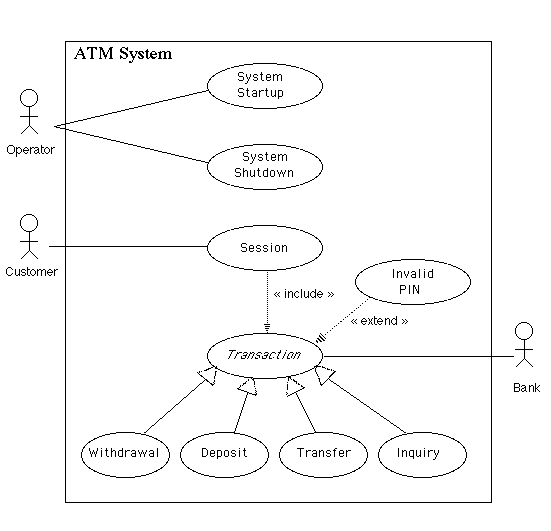


Figure 18-

# Appendix C- A minimalistic approach to plan exploratory testing

**1. Define a High-Level Objective**

* **What to Test:**  
  Identify the key feature, module, or risk area to explore (e.g., "Test the login process").
* **Why to Test:**  
  Define the goal, such as finding usability issues, ensuring stability, or verifying integration.

**2. Create Test Charters**

* **One-Line Charters:**  
  Write concise charters that guide the session, such as:
  + "Explore the withdrawal process for edge cases."
  + "Investigate error messages in failed transactions."
  + "Test receipt generation for different transaction types."

**3. Timebox the Testing**

* Allocate **30–60 minutes** per session.  
  Example: "Spend 20-minutes testing the account balance inquiry."

**4. Prepare Essentials**

* **Environment Setup:**  
  Ensure the system under test (SUT) is accessible and configured.
* **Test Data:**  
  Create basic test data (valid and invalid inputs).

**5. Perform and Record**

* **Free-Form Testing:**  
  Explore the system based on the charter without rigid scripts.
* **Minimal Documentation:**  
  Record findings in simple notes:
  + Observed issue (e.g., "Error message unclear for invalid PIN").
  + Steps to reproduce if needed.
  + Screenshots or logs if helpful.

**6. Focus on Reporting**

* At the end of each session, summarize:
  + What was tested.
  + Issues found (if any).
  + Areas left unexplored for follow-up.

**Example Minimal Plan**

1. **Feature:** Login
   * **Charter:** Explore valid and invalid PIN inputs.
   * **Timebox:** 30 minutes.
2. **Feature:** Withdrawals
   * **Charter:** Test withdrawals with sufficient and insufficient funds.
   * **Timebox:** 30 minutes.

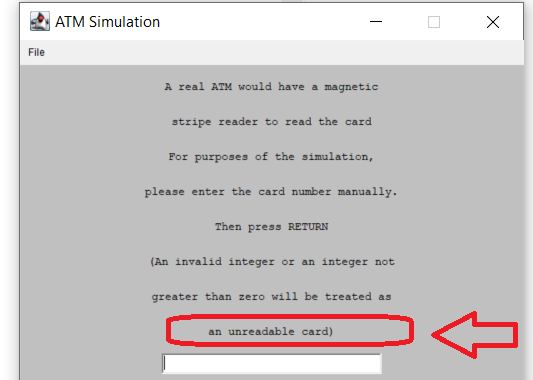
**Why This Works**

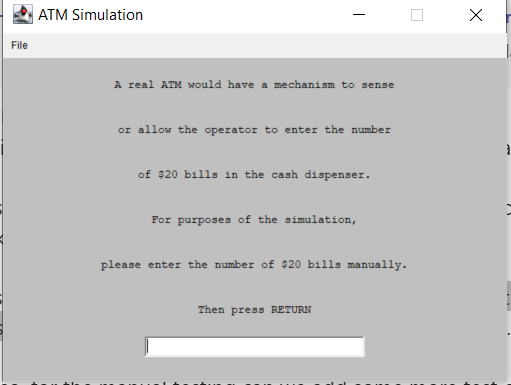
This minimalistic approach avoids over-planning, keeps testers flexible, and focuses efforts where they matter most. It’s ideal for time-constrained or iterative projects.

# Appendix D - Functional Test Suite for Manual Scripted Testing

| **Test Case #** | **Use Case** | **Function Being Tested** | **Initial System State** | **Input** | **Expected Output** |
| --- | --- | --- | --- | --- | --- |
|  | **System Startup** | System is started when the switch is turned "on" | System is off | Activate the "on" switch | System requests initial cash amount |
|  | **System Startup** | System accepts initial cash amount | System is requesting cash amount | Enter a legitimate amount | System is on |
|  | **System Shutdown** | System is shut down when the switch is turned "off" | System is on and not servicing a customer | Activate the "off" switch | System is off |
|  | **Session** | System reads a customer's ATM card | System is on and not servicing a customer | Insert a readable card | Card is accepted; System asks for entry of PIN |
|  | **Session** | System rejects an unreadable card | System is on and not servicing a customer | Insert an unreadable card[[5]](#footnote-5) | Card is ejected; System displays an error screen; System is ready to start a new session |
|  | **Session** | System accepts customer's PIN | System is asking for entry of PIN | Enter a PIN | System displays a menu of transaction types |
|  | **Session** | System allows customer to perform a transaction | System is displaying menu of transaction types | Perform a transaction | System asks whether customer wants another transaction |
|  | **Session** | System allows multiple transactions in one session | System is asking whether customer wants another transaction | Answer yes | System displays a menu of transaction types |
|  | **Session** | Session ends when customer chooses not to do another transaction | System is asking whether customer wants another transaction | Answer no | System ejects card and is ready to start a new session |
|  | **Withdrawal** | System asks customer to choose an account to withdraw from | Menu of transaction types is being displayed | Choose Withdrawal transaction | System displays a menu of account types |
|  | **Withdrawal** | System asks customer to choose a dollar amount to withdraw | Menu of account types is being displayed | Choose checking account | System displays a menu of possible withdrawal amounts |
|  | **Withdrawal** | System performs a legitimate withdrawal transaction properly | System is displaying the menu of withdrawal amounts | Choose an amount that the ATM machine currently has (in its storage[[6]](#footnote-6)) and which is not greater than the account balance | System dispenses this amount of cash; System prints a correct receipt showing amount and correct updated balance; System records transaction correctly in the log (showing both message to the bank and approval back) |
|  | **Withdrawal** | System verifies that it has sufficient cash on hand to fulfill the request | System has been started up with less than the maximum withdrawal amount in cash on hand; System is requesting a withdrawal amount | Choose an amount greater than what the system currently has | System displays an appropriate message and asks customer to choose a different amount |
|  | **Withdrawal** | System verifies that customer's balance is sufficient to fulfill the request | System is requesting a withdrawal amount | Choose an amount that the ATM machine currently has (in its storage) but which is greater than the account balance | System displays an appropriate message and offers customer the option of choosing to do another transaction or not. |
|  | **Withdrawal** | A withdrawal transaction can be cancelled by the customer any time prior to choosing the dollar amount | System is displaying menu of account types | Press "Cancel" key | System displays an appropriate message and offers customer the option of choosing to do another transaction or not. |
|  | **Withdrawal** | A withdrawal transaction can be cancelled by the customer any time prior to choosing the dollar amount | System is displaying menu of dollar amounts | Press "Cancel" key | System displays an appropriate message and offers customer the option of choosing to do another transaction or not. |
|  | **Deposit** | System asks customer to choose an account to deposit to | Menu of transaction types is being displayed | Choose Deposit transaction | System displays a menu of account types |
|  | **Deposit** | System asks customer to enter a dollar amount to deposit | Menu of account types is being displayed | Choose checking account | System displays a request for the customer to type a dollar amount |
|  | **Deposit** | System asks customer to insert an envelope | System is displaying a request for the customer to type a dollar amount | Enter a legitimate dollar amount | System requests that customer insert an envelope |
|  | **Deposit** | System performs a legitimate deposit transaction properly | System is requesting that customer insert an envelope | Insert an envelope | System accepts envelope; System prints a correct receipt showing amount and correct updated balance; System records transaction correctly in the log (showing message to the bank, approval back, and acceptance of the envelope) |
|  | **Deposit** | A deposit transaction can be cancelled by the customer any time prior to inserting an envelope | System is displaying menu of account types | Press "Cancel" key | System displays an appropriate message and offers customer the option of choosing to do another transaction or not. |
|  | **Deposit** | A deposit transaction can be cancelled by the customer any time prior to inserting an envelope | System is requesting customer to enter a dollar amount | Press "Cancel" key | System displays an appropriate message and offers customer the option of choosing to do another transaction or not. |
|  | **Deposit** | A deposit transaction can be cancelled by the customer any time prior to inserting an envelope | System is requesting customer to insert an envelope | Press "Cancel" key | System displays an appropriate message and offers customer the option of choosing to do another transaction or not. |
|  | **Transfer** | System asks customer to choose an account to transfer from | Menu of transaction types is being displayed | Choose Transfer transaction | System displays a menu of account types specifying transfer from |
|  | **Transfer** | System asks customer to choose an account to transfer to | Menu of account types to transfer from is being displayed | Choose checking account | System displays a menu of account types specifying transfer to |
|  | **Transfer** | System asks customer to enter a dollar amount to transfer | Menu of account types to transfer to is being displayed | Choose savings account | System displays a request for the customer to type a dollar amount |
|  | **Transfer** | System performs a legitimate transfer transaction properly | System is displaying a request for the customer to type a dollar amount | Enter a legitimate dollar amount | System prints a correct receipt showing amount and correct updated balance; System records transaction correctly in the log (showing both message to the bank and approval back) |
|  | **Transfer** | A transfer transaction can be cancelled by the customer any time prior to entering dollar amount | System is displaying menu of account types specifying transfer from | Press "Cancel" key | System displays an appropriate message and offers customer the option of choosing to do another transaction or not. |
|  | **Transfer** | A transfer transaction can be cancelled by the customer any time prior to entering dollar amount | System is displaying menu of account types specifying transfer to | Press "Cancel" key | System displays an appropriate message and offers customer the option of choosing to do another transaction or not. |
|  | **Transfer** | A transfer transaction can be cancelled by the customer any time prior to entering dollar amount | System is requesting customer to enter a dollar amount | Press "Cancel" key | System displays an appropriate message and offers customer the option of choosing to do another transaction or not. |
|  | **Inquiry** | System asks customer to choose an account to inquire about | Menu of transaction types is being displayed | Choose Inquiry transaction | System displays a menu of account types |
|  | **Inquiry** | System performs a legitimate inquiry transaction properly | System is displaying menu of account types | Choose checking account | System prints a correct receipt showing correct balance; System records transaction correctly in the log (showing both message to the bank and approval back) |
|  | **Inquiry** | An inquiry transaction can be cancelled by the customer any time prior to choosing an account | System is displaying menu of account types | Press "Cancel" key | System displays an appropriate message and offers customer the option of choosing to do another transaction or not. |
|  | **Invalid PIN Extension** | Customer is asked to reenter PIN |  | Enter an incorrect PIN; | Customer is asked to re-enter PIN |
|  | **Invalid PIN Extension** | Incorrect re-entry of PIN is not accepted | Request to re-enter PIN is being displayed | Enter incorrect PIN the first time, then incorrect PIN again the second time | An appropriate message is displayed and re-entry of the PIN is again requested |
|  | **Invalid PIN Extension** | Correct re-entry of PIN on the second try is accepted | Request to re-enter PIN is being displayed | Enter incorrect PIN the first time, then correct PIN the second time | System displays a menu of transaction types |
|  | **Invalid PIN Extension** | Correct re-entry of PIN on the third try is accepted | Request to re-enter PIN is being displayed | Enter incorrect PIN the first time and second times, then correct PIN the third time | System displays a menu of transaction types |
|  | **Invalid PIN Extension** | Three incorrect re-entries of PIN result in retaining card and aborting transaction |  | Enter incorrect PIN three times | An appropriate message is displayed; Card is retained by machine; Session is terminated |

1. [en.wikipedia.org/wiki/Exploratory\_testing](https://en.wikipedia.org/wiki/Exploratory_testing) [↑](#footnote-ref-1)
2. <https://help.zoho.com/portal/en/community/topic/how-to-create-a-new-portal> [↑](#footnote-ref-2)
3. “*Lesson 67: Report defects promptly*” – from the book “Lessons Learned in Software Testing” [↑](#footnote-ref-3)
4. “*Lesson 90: Review each other’s bug reports*” – from the book “Lessons Learned in Software Testing” [↑](#footnote-ref-4)
5. For “unreadable” cards, see the following screen, shown when one clicks on “Click to insert card”:

    [↑](#footnote-ref-5)
6. When you turn the ATM “on”, you specify the number of bills to be stored in its storage:

    [↑](#footnote-ref-6)