Final Report

Team Alpha Centauri 5
Michael Eskew, Dylan Mansour, Robert Moody, Joe Schell
College of Charleston

Table of Contents

Introduction

Our team began as two separate teams who after having both lost group members we merged into one team, Alpha Centauri 5. Conveniently Both of our teams happened to switch from Miradi to OpenMRS, so the transition into one team was easy. Staring off both teams encountered issues building Miradi which led to the transition into OpenMRS. Being able to successfully build OpenMRS allowed us to move on. Through the course of this project we learned not only how to build an open source project, but we learned how to create an automated testing framework that allows for easier testing. Using that framework we were then able to further test the code by injecting faults that would break programs that we choose.

Chapter 1: Picking the project

Miradi

Staring out our group chose to use Miradi for our project. Miradi is Swahili for "project" or "goal" and this user-friendly program aims to help accomplish goals. This software, coded in Java, allows nature conservation practitioners to design, manage, monitor, and learn from their projects following a process laid out in the Open Standards for the Practice of Conservation. Written in Java, this software is adaptive to help aid Conservation projects. Miradi helps to assist in each stage in a project style, which is listed out below:

1. Conceptualize.
2. Plan Actions and Monitoring.
3. Implement Actions and Monitoring.
4. Analyze, Use, Adapt.
5. Capture and Share Learning.

However due to the community being largely dead and the shear size of the project, we were unable to build it. Realizing this we decided to switch.

OpenMRS

Having encountered issues with Miradi the team decided to work on OpenMRS instead. OpenMRS is an open source electronic medical record system. It allows users to create a customized medical record system without any programming knowledge. This customizability comes from OpenMRS not being dependent on a particular data collection form. Being able to successfully build this project, and noticing the active community involvement our group decided to stick with it.

Chapter 2: Planning the Testing Schedule

Testing Process

We decided Our testing process would begin by manually checking the results of methods utilizing the terminal. We would later implement an automatic testing framework.

Requirements Traceability

- MRS-1: The form shall require the patient to have a first and last name. Validated by: testName()
- MRS-2: The form shall require the patient to enter a valid age. Validated by: testAge()
- MRS-3: The form shall require the patient to fill out a valid gender. Validated by: testGender()
- MRS-4: The form shall require the patient to enter a valid date. Validated by: testDate()
- MRS-5: The form shall require the patient to enter an emergency contact. Validated by: testContact()

Tested Items

We began by picking 5 testcses to start with.

- testName() will test OpenMRS method setGivenName() and requirement MRS-1
- testAge() will test OpenMRS method setBirthDate() and requirement MRS-2
- testGender() will test OpenMRS method setGender() and requirement MRS-3
- testDate() will test OpenMRS method setDateCreated() and requirement MRS-4
- testContact() will test OpenMRS method getPersonB() and requirement MRS-5

Testing Schedule

Our team came up with the following testing schedule.

- September 29th : have five test cases developed
- October 7th : have eleven test cases developed
- October 14th : have seventeen test cases developed
- October 18th : have twenty-five test cases developed, develop an automatic testing framework

- October 22nd : have nine test cases implemented in the framework
- October 29th : have eighteen test cases implemented in the framework
- November 10th : have twenty-five test cases implemented in the framework
- November 17th : have five faults in testing code
- November 22nd : have Chapters 1 - 5 in our final report
- November 26th : have powerpoint and poster for the project prepared

Test Recording Procedures

We captured our testing results with testing reports. We then used a testing script to write output results to a HTML file.

Hardware and Software Requirements

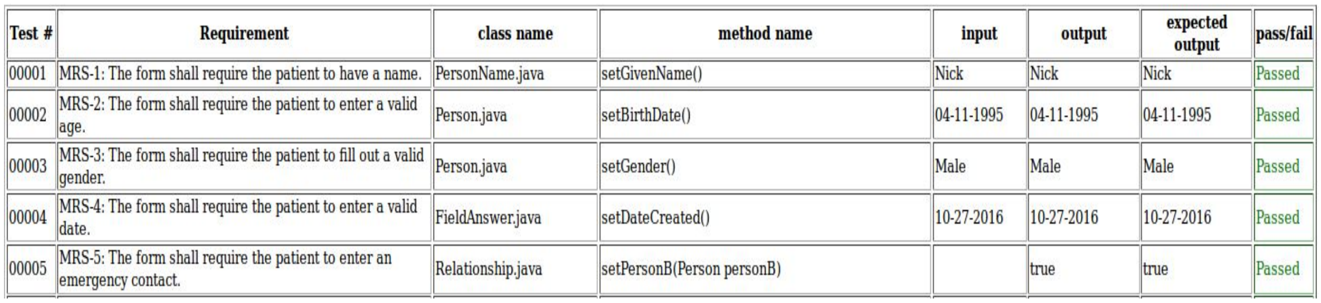
Required hardware were a laptop that is capable of running our software requirements: OpenMRS, MySQL, TomCat, Maven, and Eclipse IDE.

Constraints

Our biggest constraints were time and being able to all meet at once.

Chapter 3: Creating an Automated Testing Framework

Starting off we encountered problems with the driver running. We were able to get the driver to initially run inside a dummy environment. When it ran in the project environment it would fail. After fixing this issue we were able to move on with the testing of the first test cases. With the results looking as follows.



Test #	Requirement	class name	method name	input	output	expected output	pass/fail
00001	MRS-1: The form shall require the patient to have a name.	PersonName.java	setGivenName()	Nick	Nick	Nick	Passed
00002	MRS-2: The form shall require the patient to enter a valid age.	Person.java	setBirthDate()	04-11-1995	04-11-1995	04-11-1995	Passed
00003	MRS-3: The form shall require the patient to fill out a valid gender.	Person.java	setGender()	Male	Male	Male	Passed
00004	MRS-4: The form shall require the patient to enter a valid date.	FieldAnswer.java	setDateCreated()	10-27-2016	10-27-2016	10-27-2016	Passed
00005	MRS-5: The form shall require the patient to enter an emergency contact.	Relationship.java	setPersonB(Person personB)		true	true	Passed

Chapter 4: Testing the code

For our project, we chose OpenMRS, a java application for doctors that allows for ease when storing and tracking patient information. Once built our project can be run from command line by a single command.

How-To

1. Build OpenMRS
2. Clone the TestAutomation Repository
3. Open the command prompt and type the command from the TestAutomation directory ./scripts/runAllTests.sh

Framework

Our framework directory structure is:

- /TestAutomation
  - /project
  - /scripts
  - /testCases
  - /testCaseExecutables
  - /temp
  - /oracles
  - /docs
  - /reports

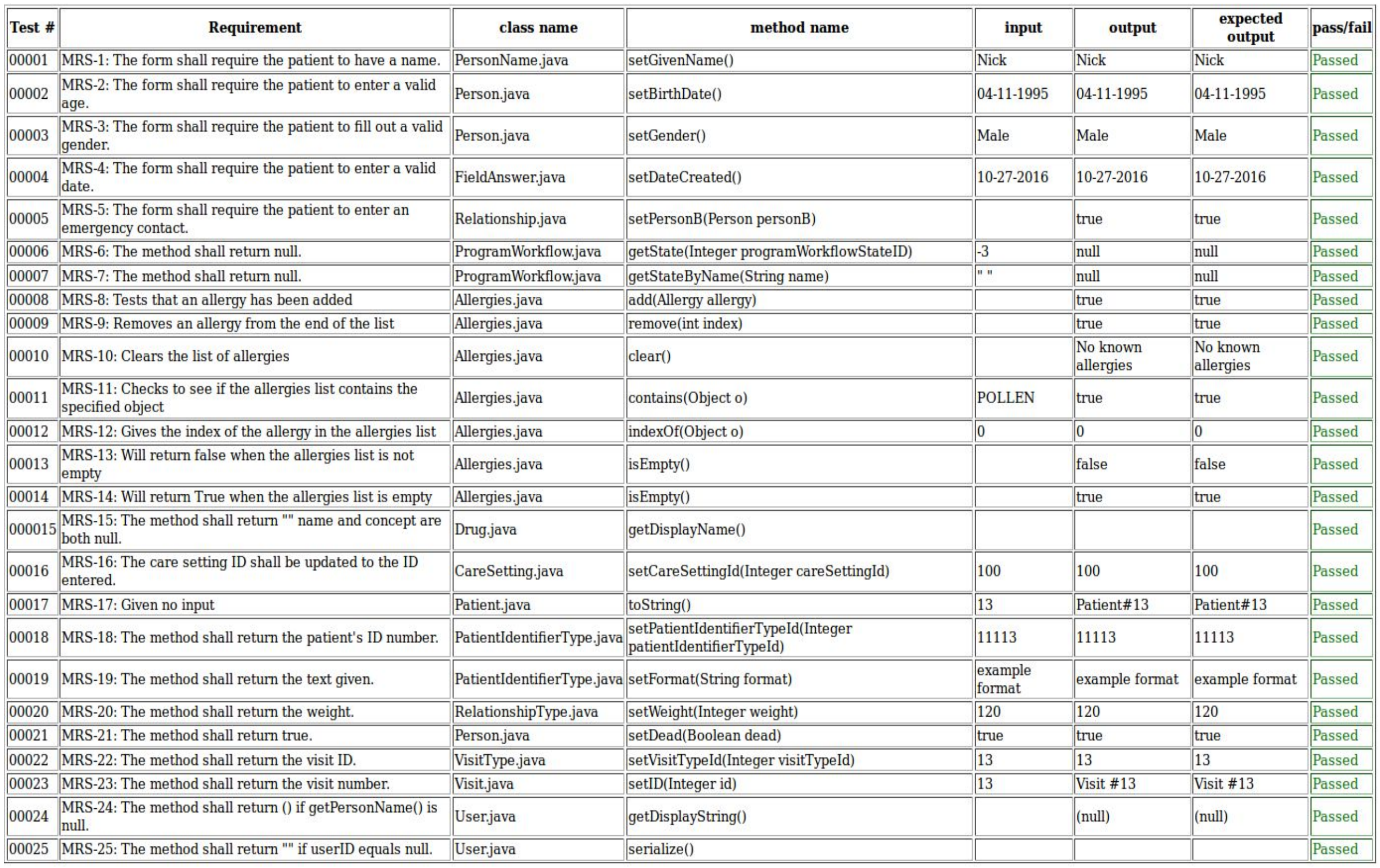
Test case structure

Using our automated testing framework required our test cases to follow a certain format so that they would run correctly. The structure for our test cases is as follows.

1. Test Case ID
2. Requirement being tested
3. Class being tested
4. Method(s) being tested
5. Test input(s)
6. Expected outcome(s)
7. Class path
8. Driver Name

Final testing

Having picked out all of the 25 test cases we were going to be testing we began checking to see if our driver would successfully run the new test cases. After our first few attempts a few of the test cases were failing. After looking into the issue we were able to fix the problems and successfully run all our test cases. Here is an image of the final results.



Test #	Requirement	class name	method name	input	output	expected output	pass/fail
00001	MRS-1: The form shall require the patient to have a name.	PersonName.java	setGivenName()	Nick	Nick	Nick	Passed
00002	MRS-2: The form shall require the patient to enter a valid age.	Person.java	setBirthDate()	04-11-1995	04-11-1995	04-11-1995	Passed
00003	MRS-3: The form shall require the patient to fill out a valid gender.	Person.java	setGender()	Male	Male	Male	Passed
00004	MRS-4: The form shall require the patient to enter a valid date.	FieldAnswer.java	setDateCreated()	10-27-2016	10-27-2016	10-27-2016	Passed
00005	MRS-5: The form shall require the patient to enter an emergency contact.	Relationship.java	setPersonB(Person personB)		true	true	Passed
00006	MRS-6: The method shall return null.	ProgramWorkflow.java	getState(Integer programWorkflowStateID)	-3	null	null	Passed
00007	MRS-7: The method shall return null.	ProgramWorkflow.java	getStateByName(String name)	" "	null	null	Passed
00008	MRS-8: Tests that an allergy has been added	Allergies.java	add(Allergy allergy)		true	true	Passed
00009	MRS-9: Removes an allergy from the end of the list	Allergies.java	remove(int index)		true	true	Passed
00010	MRS-10: Clears the list of allergies	Allergies.java	clear()		No known allergies	No known allergies	Passed
00011	MRS-11: Checks to see if the allergies list contains the specified object	Allergies.java	contains(Object o)	POLLEN	true	true	Passed
00012	MRS-12: Gives the index of the allergy in the allergies list	Allergies.java	indexOf(Object o)	0	0	0	Passed
00013	MRS-13: Will return false when the allergies list is not empty	Allergies.java	isEmpty()		false	false	Passed
00014	MRS-14: Will return True when the allergies list is empty	Allergies.java	isEmpty()		true	true	Passed
000015	MRS-15: The method shall return "" name and concept are both null.	Drug.java	getDisplayName()				Passed
00016	MRS-16: The care setting ID shall be updated to the ID entered.	CareSetting.java	setCareSettingId(Integer careSettingId)	100	100	100	Passed
00017	MRS-17: Given no input	Patient.java	toString()	13	Patient#13	Patient#13	Passed
00018	MRS-18: The method shall return the patient's ID number.	PatientIdentifierType.java	setPatientIdentifierTypeId(Integer patientIdentifierTypeId)	11113	11113	11113	Passed
00019	MRS-19: The method shall return the text given.	PatientIdentifierType.java	setFormat(String format)	example format	example format	example format	Passed
00020	MRS-20: The method shall return the weight.	RelationshipType.java	setWeight(Integer weight)	120	120	120	Passed
00021	MRS-21: The method shall return true.	Person.java	setDead(Boolean dead)	true	true	true	Passed
00022	MRS-22: The method shall return the visit ID.	VisitType.java	setVisitTypeId(Integer visitTypeId)	13	13	13	Passed
00023	MRS-23: The method shall return the visit number.	Visit.java	setID(Integer id)	13	Visit #13	Visit #13	Passed
00024	MRS-24: The method shall return () if getPersonName() is null.	User.java	getDisplayString()		(null)	(null)	Passed
00025	MRS-25: The method shall return "" if userID equals null.	User.java	serialize()				Passed

Chapter 5: fault injection

Picking which test cases to break

Having been able to successfully run all of our test cases we were able to begin thinking about fault injection. We began by picking which test cases we wanted to break. Looking into what methods depended on other methods, we discovered that 3 of our chosen methods to test depended on the add method which we had also chosen to test. Having realized this we simply injected a line of code into our add method that would break it. This line of code simply created a blank allergy object to be added instead of the intended allergy. This caused not only the add() method to not function properly but also the contains, indexOf(), and remove() methods. Having already broken four test cases we only had to decide on one more to break. We decided to go with the serialize() method. This method was chosen because unlike the add() method nothing else we were testing depended on the serialize() method. Breaking serialize was done by changing a (!=) to (==).

Chapter 6: Experiences and Knowledge Gained

Experiences

Overall the project was a great experience. Working as a group on this project allowed us to get to know each other better while furthering our knowledge in the process. The beginning was rough. Having to change projects due to being unable to build Miradi stressed everyone out and put us behind. However merging our two groups and switching to OpenMRS increased our confidence in our project.

We also gained experience using the Ubuntu linux distribution as well as experience using the command line and writing scripts to automate testing. For most of us, this project was the first time developing software in a team. We gained insight on how to organize and deal with team dynamics and work together in order to produce a functional piece of software.

What We Learned

This project taught us a number of things. The first being that some open source programs die out when the community stops working on it, as was the case with Miradi. With no one updating it we were unable to build it. We became aware of the importance of good documentation in a software project after switching to OpenMRS. We were able to learn how to pull code from github and build it in eclipse on a linux operating system which was made easy due to OpenMRS having excellent build files that told you exactly how to build it. We also learned a lot about using version control, in our case, github, to keep everyone up-to-date with the latest developments in our project.

After building the project in eclipse we were able to begin learning how to create an automated testing framework for software. Writing scripts, using a linux operating system, and even html were new to us and we had to learn about these things to complete the project. We gained knowledge in the bash scripting language and about writing drivers to test specific methods in Java. As we began searching for methods to test, we saw the importance of having testable code. When picking out what test cases to use test our driver with, we began to learn of the importance of dependencies. We didn't want all of our cases to be dependent on each other because it would have made fault injection impossible.

Finally and most importantly we learned how to work with others on a project like this. Something we will surely appreciate later in our careers.