Simon Fraser University
School of Computing Science
Cmpt 275 Project

Date: Wednesday, August 5, 2015

Team name: Fast and Furious 9 (Group 9)

Project Deliverable #: 7

Project Deliverable Name: Test Cases Document

Phase leader(s): Joshua Campbell

Team members and Student #:

- Ching Lam – 301139713
- Connor MacLeod – 301148273
- David Chow – 301196333
- German Villarreal – 301183384
- Janice Mardjuki – 301152558
- Joshua Campbell – 301266191
- Madison Jones – 301196793
- Robert Cornall – 301176349
- Samnang Sok – 301156625

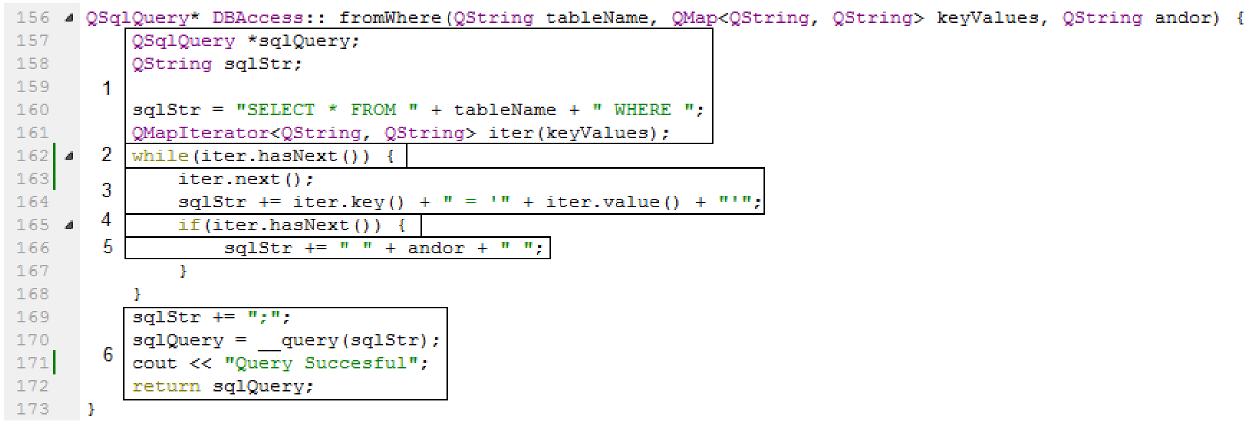
Grade:

Revision History

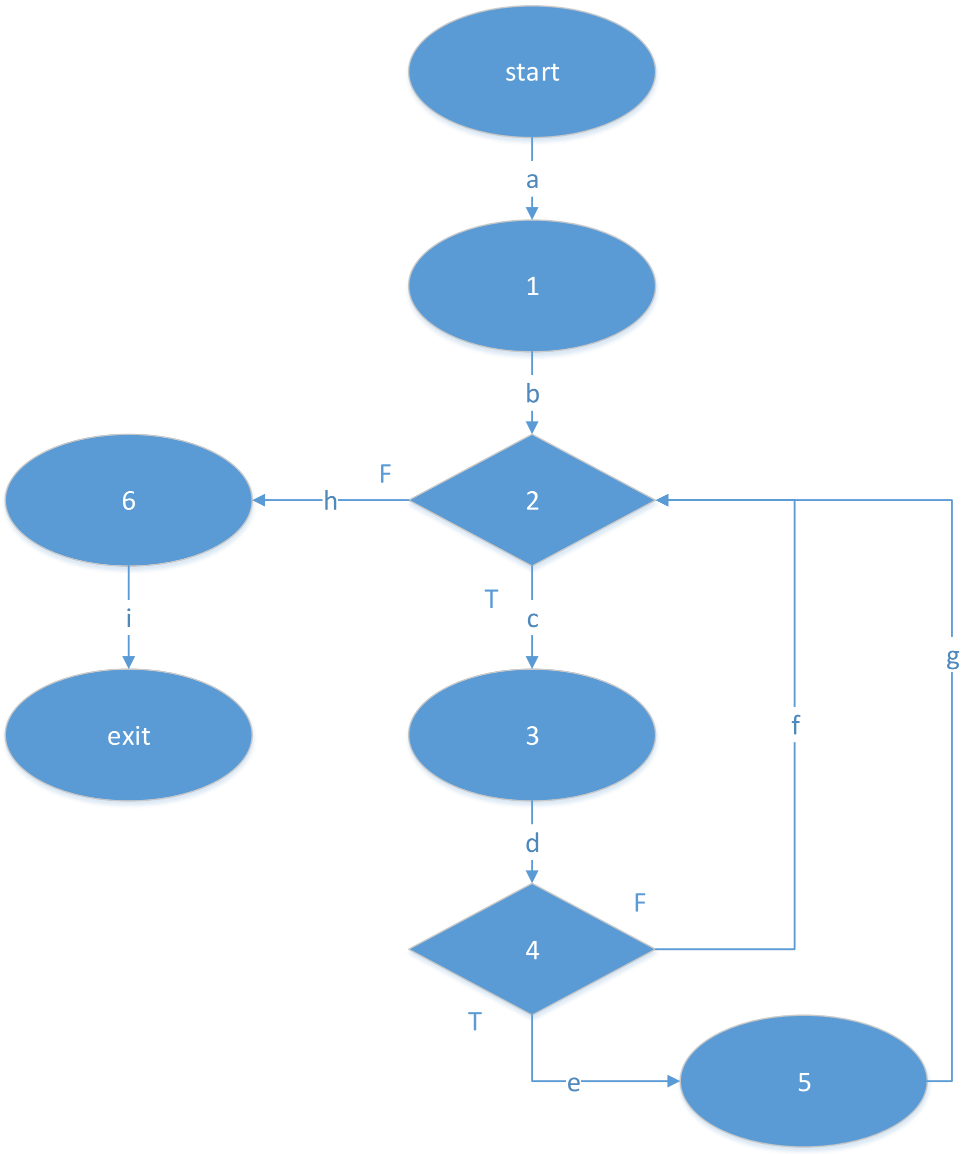
Revision	Status	Publication/Revision Date	By:
1.0	Created	Tuesday, July 21, 2015	Maddy Jones
1.1	Added Black Box Test Case	Thursday, July 23, 2015	Maddy Jones Rob Cornall
1.2	Added Diagrams and Method Description to Glass Box Test Case	Thursday, July 23, 2015	David Chow
1.3	Added Glass Box Test Case & Coverage, Revised Glass Box Diagrams	Saturday, July 25, 2015	Janice Mardjuki David Chow
2.0	Added Getting Started Section	Tuesday, August 4, 2015	Janice Mardjuki Maddy Jones
2.1	Added Post Mortem and updated Getting Started	Wednesday, August 5, 2015	Maddy Jones Connor MacLeod
2.2	Added Known Bugs section and updated Getting Started	Wednesday, August 5, 2015	Josh Campbell David Chow

Table of Contents

Glass Box Test Plan



```
   QSqlQuery* DBAccess:: fromWhere(QString tableName, QMap<QString, QString> keyValues, QString andor) {
       QSqlQuery *sqlQuery;
       QString sqlStr;
   1
       sqlStr = "SELECT * FROM " + tableName + " WHERE ";
       QMapIterator<QString, QString> iter(keyValues);
   2   while(iter.hasNext()) {
           iter.next();
   3       sqlStr += iter.key() + " = '" + iter.value() + "'";
   4       if(iter.hasNext()) {
   5           sqlStr += " " + andor + " ";
           }
       }
       sqlStr += ";";
       sqlQuery = __query(sqlStr);
   6   cout << "Query Succesful";
       return sqlQuery;
   }
```



Method: QSqlQuery *_fromWhere(QString tableName, QMap<QString, QString> keyValues, QString andor)
Purpose of the method: generates a simple Select-From-Where SQL statement based on input parameters and then runs the query
Parameters:

- "tableName" is the name of the table to query
- "keyValues" holds attribute-value pairs as selection criteria (ex. "id", "12345")
- "andor" should be either "AND" or "OR" to determine whether the selection criteria are all joined by conjunctions or disjunctions
    - ex.1 ...WHERE 'id' = '12345' AND 'name' = 'Bob' AND...
    - ex. 2 ... WHERE 'id' = '12345' OR 'name' = 'Bob' OR...

Statement coverage is necessary to test each line of code at least once.
This test case is useful because each line of code is run at least once.
Test case id - 1S
start-1-2-3-4-5-2-6-exit
keyValues must be non-empty

Branch coverage is necessary because we have an if statement.
This test case is useful because it tests both possibilities of the if statement.
Test case id - 1B
start-a-b-c-d-e-g-c-d-f-h-i-exit
keyValues has 2 attribute-value pairs
(pairs can have any value)

Path coverage is necessary because we have a while loop.
This test case is useful because it tests the case where keyValues is empty.
Test case id - 1P -> do while loop 0 times
start-1-2-6-exit

This test case is useful because it tests the case where keyValues has one value.
Test case id - 2P -> do while loop 1 time
start-1-2-3-4-2-6-exit
(start-1-2-3-4-5-2-6-exit is not possible because the while condition is the same as the if condition)

This test case is useful because it tests the case where keyValues has several values.
Test case id - 3P -> do while loop (many) 3 times
start-1-2-3-4-5-2-3-4-5-2-3-4-2-6-exit

Description of One Test Case:
Test id -> 2P
Test purpose: Unit test function _fromWhere(QString, QMap<QString, QString>, QString) (in the DBAccess class) using Glass Box testing strategy (statement coverage: start-1-2-3-4-2-6-exit)
Requirement # 1.1.1
Inputs: the name of a table in the database "tableName" which has value = "Activity", a QMap "keyValues" with two key-value pairs: keyValues["activityID"] = "1" and keyValues["courseID"] = "1" (2 non-empty keys), and a string with value = "AND"
Testing procedure:

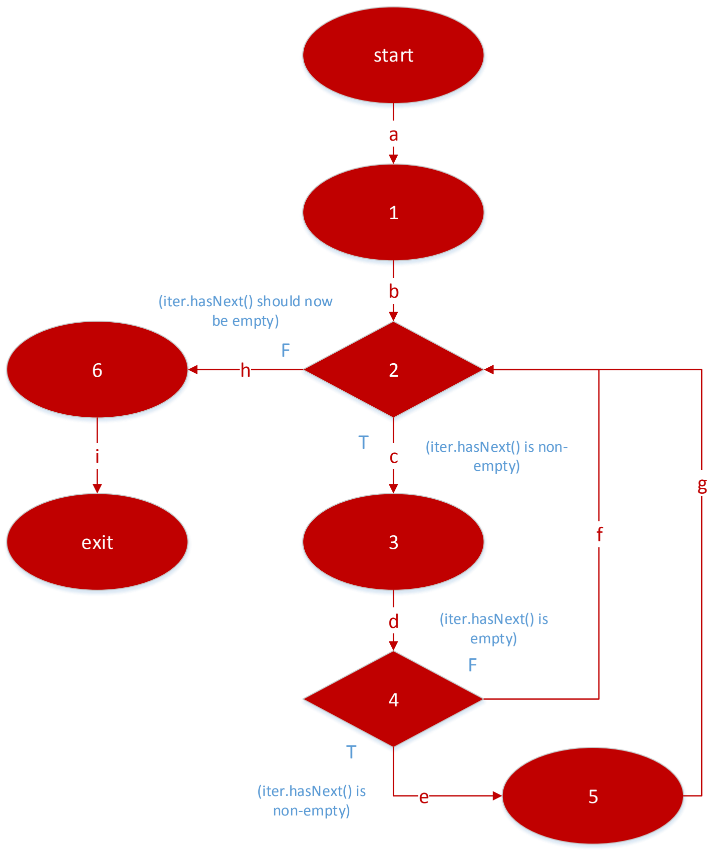
- open database to view entries in the Activity table
- login as Instructor
- select the course with courseID = 1 which has the activity with activityID = 1
- when the list of activities for the course is generated, then _fromWhere(...) is called
- close the program to see output

Expected behaviours and results:

- "Query Successful" in Qt creator's Application Output window

Actual behaviours and results:

- "Query Successful" in Qt creator's Application Output window


start
a
1
b
(iter.hasNext() should now be empty)
F
h
6
2
T
c
(iter.hasNext() is non-empty)
i
exit
3
f
g
d
(iter.hasNext() is empty)
F
4
T
(iter.hasNext() is non-empty)
e
5

(paths and nodes covered in the test case are colored in red)

Black Box Test Plan

Method: checkAndContinue()
Purpose of the Method: change/reset a user's password when they login for the first time or after being unblocked by the system admin
Parameters: "Password" and "Confirm Password"
Preconditions: system admin has created a new account or unblocked an account (will set resetPassword flag to true) and the user has successfully logged in with his or her temporary password

Equivalence Classes (preconditions):

- Valid: resetPassword == 1
- Invalid: resetPassword == 0

Equivalence Classes (inputs):

- Valid: "Password" is equal to "Confirm Password" and "Password" length >= 4 and "Password" length <= 40
- Invalid: "Password" length < 4 or "Password" length > 40 or "Password" is not equal to "Confirm Password"

How to choose representative values:

- Consider the flag resetPassword
    - Two possible states: true or false
        - Use the cases false (flag not set), true (new account), and true (unblocked account)
- Consider "Password" length, "Confirm Password" length and "Password" is equal to "Confirm Password"
    - For "Password" length and "Confirm Password" length choose one value from each equivalence class (2, 12, 45)
        - For each value, choose a matching set of passwords and a non-matching set
    - For "Password" length and "Confirm Password" length choose boundary values (3, 4, 40, 41)
        - For each value, choose a matching set of passwords and a non-matching set

Describe how to use the representative values to construct a set of test cases:

- Test with all valid variables and preconditions (reset flag is true and passwords match and are of valid length)
- Test with all valid variables and the invalid precondition
    - resetPassword flag false
- Test with all valid preconditions and one invalid variable
    - passwords of differing lengths within the valid range
    - passwords matching, but too short
    - passwords matching, but too long
- Test with all valid preconditions and matching passwords of boundary lengths

- Test with pairs of invalid variables
    - Passwords not matching and password lengths not matching but within the valid range
    - Passwords the same length but not matching and too short
    - Passwords the same length but not matching and too long
- Test with all invalid variables and preconditions
    - Passwords different lengths and not matching and outside range and resetPassword flag is false

Single Test Case: all valid preconditions and one invalid variable (passwords matching, but too short)

Describe Test Case:

- Test ID: 1
- Test purpose: Unit test StreamlinedGradingSystem class resetpasswordscreen method checkAndContinue( ) using black box testing strategy
- Requirement: #6.2.4.2 version 2.0
- Inputs: resetPassword is true; "password": "cat"; "confirm password": "cat"
- Testing procedure:
    - Create an account object (includes a temporary password)
    - Check the database to verify resetPassword flag is true
    - Log in to new account using temporary password (displays reset password screen)
    - Type "cat" into the "Password" field
    - Type "cat" into the "Confirm Password" field
- Evaluation:
    - Click "Continue"
- Expected behaviours and results:
    - Dialog box with message "Password is too short"
- Actual behaviours and results:
    - Dialog box with message "Password is too short"

Post Mortem

Successfully Managing a Group Project

For many of us, this was the first time working in a group of this size. As a result, we learned a lot about managing a project and working as part of a team. Each of us took a turn acting as phase manager as well as managing another aspect of the project. This gave us an opportunity to experience managing a project as well as working under various management styles. When it was our turn to manage an aspect of a project, we learned how to coordinate a group of people and assign tasks appropriately. Some people took a very active management approach, whereas others managed more passively. We all learned from our own management style as well as our fellow group members'. Unfortunately, there were a few instances where multiple people were unknowingly working on the same task. From this, we learned how to coordinate better. We made lists of all the requirements and who was assigned to each. All of this knowledge will undoubtedly prove to be valuable as we move from a classroom setting to real world opportunities. We also learned a lot about time estimation. The most important thing we learned is that time estimation for a group is not the same as time estimation for an individual. For example, if you expect something will take an hour to complete and you're working by yourself, taking a few extra minutes is fine. However, when a group meeting is not completed in the given amount of time, people are often unable stay and finish due to prior commitments. Luckily, our group meetings were managed well and we rarely encountered this situation. However, without fail, each phase has taken more time than we had originally budgeted. Regardless of how early we got started, we continued working until the last minute. This is not surprising because, like Parkinson's Law states, "work expands so as to fill the time available for its completion". Phase one obviously took the least time to complete and after that, it seems like each phase has taken longer than the one before it. This is not surprising considering each phase builds on the previous ones, so we often needed to revisit and revise things that were done in previous phases in order to complete the current phase. Deliverable five took significantly more time than the previous phases because of the number of different documents required. Also, not surprisingly, deliverable seven was not as time consuming due to it being so short. Another reason for the latter phases' increased duration is the addition of programming. Implementing this project has taken more time and effort than the documenting it, although it certainly would've taken even longer without the documentation to reference. Nevertheless, it is evident that designing a Streamlined Grading System in theory is much easier than actually implementing it. One of the most notable examples of this was the implementation of .pdf viewing. It only took a few minutes to document this feature in the early phases, but implementing took over fifteen hours. The most valuable thing we learned about time estimation was to always budget more time than we thought we needed.

Difficulties with Testing

This has been a rather large project compared to most of the projects our group members have worked on previously. As a result, testing needed to be more thorough than many of us were used to. We learned about different techniques for testing, many of which were new to a lot of our group members. We found that some testing strategies that we had used before were not ideal for a larger project such as this. Because we had many people working on it at once, we needed to be able to test part of our project without other sections being completed. For example, testing the UI before any of the functionality has been implemented. In class we learned about formal testing procedures. Specifically, we learned of “black box” and “glass box” testing. We experienced how to formally document test cases in both of these formats. For many of us, this type of formal testing was completely new. Looking at and documenting detailed test cases helped us to identify possible sources of errors in our own code without needing to go through the formal process in its entirety. Still, the preferred testing strategy seemed to be to code and then try to break the code. The lack of structure inherent in this strategy may not have been ideal, but it was ultimately effective. We often found that testing one person’s code relied on the correct implementation of someone else’s. Identifying the source of a problem became difficult when the desired outcome was dependant on many different parts coded by different people. Furthermore, testing the system often relied on data that was constantly changing in the database. For example, a user we had created (“11111” with password “11111” and all roles) was often used for testing purposes. One test involved changing the password to “22222”. The next person to try to log in to this account was obviously unable to, which they may have incorrectly attributed to something other than a simple password change. Similarly, when multiple people tried to access the 11111 account at the same time, the application would crash. This may have incorrectly appeared to be the result of something more complicated. We learned that some problems could be attributed to incorrectly configured libraries or incompatible software versions. Some members of our group had never used a debugger before, so we learned how to use this tool to make testing easier. Ultimately, we learned that there are a lot of things that can go wrong when it comes to testing, but there are also a lot of strategies for assessing and fixing problems. Throughout the course of this project we became much more proficient at recognizing possible sources of errors and identifying ways to check for solutions.

The Importance of Teamwork

This project has taught us a lot about working in a team setting. From the beginning, most of us were meting each other for the first time and we didn't know what to expect. Our group members were diverse in their personalities as well as their backgrounds. As a result, we saw a variety of management styles throughout the course of our project. We had to be able to adapt to these different personalities and management styles. Differences were especially noticeable during team meetings. Some members of our group felt very comfortable sharing their opinions, while others were more comfortable sitting back and observing the discussion. Some people were more certain of their opinions, while others were quick to abandon their idea for another. It wasn't always easy to make sure everyone had an opportunity to contribute, but we learned from this and in the end, our group discussions were very productive. However, our discussions didn't always stay completely on topic. In these situations, it was important for someone, often our meeting manager, to keep the discussion focused. Another problem our team faced was arranging times to meet. Everyone had other commitments that they needed to schedule around, such as work, other classes, or in one case, the birth of a child. Whether it was coordinating meeting times, assigning roles, or just exchanging ideas, we found that working in this type of team setting required much better communication than previous classes or projects. To aid this need for communication, we used a variety of tools. We discovered that the most effective communication tools for our group were Skype, Google docs, and Facebook. We were able to instant message via Facebook if we needed to ask for help with part of the project. We were also able to keep relevant links and information centralized with a Facebook group. When working on many of the deliverables, using a Google doc helped us collaborate without worrying about conflicts in SVN. Finally, Skype allowed us to discuss solutions without needing to meet face to face. Working in a team setting for this project forced us to collectively decide on appropriate means of communication for different situations. The communication skills that we learned during this project will be important for us in the future when we start working on larger projects with more people. Another challenge we faced when working in this group was assigning tasks to people with a variety of experience and skill levels. In order to successfully complete this project, we had to utilise everyone's strengths. Some people had more experience leading a team and others had stronger programming backgrounds. We learned how to assign roles to the people who could contribute most. Ultimately, our team did a good job of assigning people roles that fit their strengths.

Getting Started

1. Software Requirement
Before running Streamline Grading System, these software requirements are needed:

- Microsoft XP or higher with latest update
- Microsoft SQL Server 2008 Management Studio
- Qt 5.4.1 (or later) and C++ Platform installed
- MinGW installed
- Connected to SFU server (cypress.csil.sfu.ca), so SGS only can be accessed in CSIL computer or remote desktop to leto.csil.ca

2. Hardware Requirement
Before running Streamline Grading System, these hardware requirements are needed:

- Monitor, Keyboard, and Mouse
- RAM of at least 2GB
- 500MB disk space available (1GB recommended)
- Intel i3 processor or higher
- Internet connection (at least 10 Mbps)

3. Installation

- Before installation
    - Make sure your computer meets the Hardware and Software Requirements
- Installation of Streamline Grading System (SGS)
    - Download Streamlined Grading System by saving the .zip file on your local disk
    - Extract the .zip file that you just downloaded to your folder of choice
    - Go to the folder where you just extracted the file and double click the "StreamlinedGradingSystem.exe" file
    - (note: all required .dll files and .exe files should already be in the folder for the system to run)

4. Installation of the application for the UAT

- Extract the Streamlined Grading System archive with the source code to the folder
- Run Qt Creator and open the “StreamlinedGradingSystem.pro” file located in the “src” folder
- On the “Configure Project” screen, select a kit which contains “QT 5.4.1 MinGW 32bit” in its name (or a later release of QT) and click the “Configure Project” button
- Located at the bottom-left of the Qt Creator interface, above the “Run” icon (green play button), change the build from “debug” to “release”
- Under the “Build” menu, launch configuration “Run qmake”
- Click on “Build All” from the “Build” menu
- Create a new folder to hold a working copy of the system
- In the folder where the archive was extracted (the same directory as the “src” folder), a new folder with a name that is (or is similar to) “build-StreamlinedGradingSystem-

Desktop_Qt_5_4_1_MinGW_32bit-Release/" will have been created. Enter this directory.

- Click on the "release" folder
- Copy the "StreamlinedGradingSystem.exe" into the new folder to hold the working copy of the system
- Go to the "lib" folder and copy all .dll files into the new folder which will hold the working copy of the system
- In the new folder, clicking on the "StreamlinedGradingSystem.exe" will run the Streamlined Grading System program

Known Bugs

EssayGradingScreen:

- If path to solution or submission file is invalid or does not exist, the program will crash
- If you select an annotation from the annotation list, the width and height have a tendency to change.