Requirements Based Test Cases for GRADS

Revision 0.2
12/13/14

Group 15

Daniel Belling
David Cavalari
Christine Hallstrom
Guobao Sun

Document Revision History

Rev	Date	Author	Change Description
0.1	12/7/2014	Daniel Belling, David Cavalari, Christine Hallstrom, Guobao Sun	Created document
0.2	12/13/2014	Daniel Belling, David Cavalari, Christine Hallstrom, Guobao Sun	Improved version

This document is originating from Neil Bitzenhofer and DataCard Corporation.

Table of Contents

1. Test Requirements 3
1.1 Objective 3
1.2 Definitions and Acronyms 3
1.3 Traceability Matrix..... 3 - 5
2. Test Cases 5
2.1 Description 5
2.2 Test Cases Section..... 5
Test Case 1A: Load Users ……..…….5
Test Case 1B: Load Users Exception ……5
Test Case 2A: Load Courses ……..…..……...6
Test Case 2B: Load Courses Exception ……..6
Test Case 3A: Load Records ……..……...6
Test Case 3B: Load Record Exception ……...…..7
Test Case 4A: Set User ……..……7
Test Case 4B: Set User Exception ……8
Test Case 5: Get User ……....8
Test Case 6A: Get Student ID's - GPC User ……8
Test Case 6B: Get Student ID's Student User ……..……..9
Test Case 7A: Get Transcript - GPC User ……..……....9
Test Case 7B: Get Transcript - Student User ……..……....9
Test Case 7C: Get Transcript - Student User with Incorrect Permissions ……...…..10
Test Case 7D: Get Transcript - Invalid User ……...10
Test Case 8A: Update Transcript - GPC User ……....11
Test Case 8B: Update Transcript - Student User (Incorrect Permissions) ……11
Test Case 9A: Add Note - GPC User ……..…….....12
Test Case 9B: Add Note - Student User (Incorrect Permissions) ……...12
Test Case 10A: Generate Progress Summary - PHD - GPC User ……..……....12
Test Case 10B: Generate Progress Summary - MS_A - GPC User ……..……..13
Test Case 10C: Generate Progress Summary - MS_B - GPC User ……..……..13

Test Case 10D: Generate Progress Summary - MS_C - GPC User ……..……..14
Test Case 10E: Generate Progress Summary - PHD - Student User ……..……14
Test Case 10F: Generate Progress Summary - MS_A - Student User ……..…14
Test Case 10G: Generate Progress Summary - MS_B - Student User ……..…….15
Test Case 10H: Generate Progress Summary - MS_C - Student User ……..……15
Test Case 11A: Check Course Impact - Single Course ……16
Test Case 11B: Check Course Impact - Multiple Courses ……16
Test Case 11C: Check Course Impact - Incorrect Permissions ……...17
Test Case 11D: Check Course Impact - No Change ……17
Test Case 11E: Check Course Impact - Course Which Has No Impact ……18
Test Case 12A: Check Empty Transcript - PHD - GPC User ……...18
Test Case 12B: Check Empty Transcript - MS_A - GPC User ……19
Test Case 12C: Check Empty Transcript - MS_B - GPC User ……19
Test Case 12D: Check Empty Transcript - MS_C - GPC User ……20

3. Testing Overview and Analysis ……20

1. Test Requirements

1.1 Objective

This section lists all hardware and software test requirements as determined by the GRADS Requirements Document and the GRADS Design Document, incorporating any changes made to the requirements throughout the design process.

1.2 Definitions and Acronyms

Below are the definitions of the following technical terms and acronyms used in this document:

TM	Traceability Matrix

1.3 Traceability Matrix

The purpose of a Traceability Matrix is to show which test cases verify which requirements. A possible format for a Traceability Matrix is as follows:

Requirement \ Test Case	Req # 1	Req # 2	Req # 3	Req # 4	Req # 5	Req # 6	Req # 7
Test Case 1A	X						
Test Case 1B	X						
Test Case 2A				X			
Test Case 2B				X			
Test Case 3A			X				
Test Case 3B			X				
Test Case 4A	X						
Test Case 4B	X						
Test Case 5	X						
Test Case 6A		X					
Test Case 6B		X					
Test Case 7A			X				
Test Case 7B			X				
Test Case 7C			X				
Test Case 7D			X				
Test Case 8A					X		
Test Case 8B					X		
Test Case 9A					X		
Test Case 9B					X		

Test Case 10A				X			X
Test Case 10B				X			X
Test Case 10C				X			X
Test Case 10D				X			X
Test Case 10E				X			X
Test Case 10F				X			X
Test Case 10G				X			X
Test Case 10H				X			X
Test Case 11A						X	
Test Case 11B						X	
Test Case 11C						X	
Test Case 11D						X	
Test Case 11E						X	
Test Case 12A				X			X
Test Case 12B				X			X
Test Case 12C				X			X
Test Case 12D				X			X

2. Test Cases

2.1 Description

In this section, we present a group of automated system-level tests derived from white box testing of the GRADS system. A brief summary of each scenario is given for each test as well as the desired input and output in hopes that our testing approach is reproducible outside of the JUnit framework.

2.2 Test Cases

Test Case 1A **Load Users**

Description:	Verifies that GRADS correctly loads data from a text file containing user data into the users attribute contained within the GRADS class.
Test Inputs:	Name of a text file containing valid user data.
Expected Results:	Instance of users attribute in GRAD class is not null.
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class.
Test Steps:	1. Load a valid users.txt file into the system. 2. Verify that the list of users contained in the GRADS class is not null.

Test Case 1B **Load Users Exception**

Description:	Verifies that GRADS correctly returns an ExCantLoadUsers exception when loadUsers is called with a user file that does not exist.
Test Inputs:	Invalid name of a text file.
Expected Results:	ExCantLoadUsers exception.
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class.
Test Steps:	1. Call loadUsers() with an invalid file name. 2. Verify that an ExCantLoadUsers exception occurs.

Test Case 2A **Load Courses**

Description:	Verifies that GRADS correctly loads data from a text file containing course data into the courses attribute contained within the GRADS class.
Test Inputs:	Name of a text file containing valid course data.

Expected Results:	Instance of courses attribute in GRAD class is not null.
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class.
Test Steps:	1. Load a valid courses.txt file into the system. 2. Verify that the list of courses contained in the GRADS class is not null.

Test Case 2B **Load Courses Exception**

Description:	Verifies that GRADS correctly returns an ExCantLoadCourses exception when loadCourses is called with a course file that does not exist.
Test Inputs:	Invalid name of a text file.
Expected Results:	ExCantLoadCourses exception.
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class.
Test Steps:	1. Call loadCourses() with an invalid file name. 2. Verify that an ExCantLoadCourses exception occurs.

Test Case 3A **Load Records**

Description:	Verifies that GRADS correctly loads data from a text file containing student record data into the records attribute contained within the GRADS class.
Test Inputs:	Name of a text file containing valid student record data.
Expected Results:	Instance of records attribute in GRAD class is not null.
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class.
Test Steps:	1. Load a valid students.txt file into the system. 2. Verify that the list of records contained in the GRADS class is not null.

Test Case 3B **Load Records Exception**

Description:	Verifies that GRADS correctly returns an ExCantLoadRecords exception when loadRecords is called with a student records file that does not exist.
Test Inputs:	Invalid name of a text file.
Expected Results:	ExCantLoadRecords exception.
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class.
Test Steps:	1. Call loadRecords() with an invalid file name. 2. Verify that an ExCantLoadRecords exception occurs.

Test Case 4A **Set User**

Description:	Verifies that GRADS correctly sets the currentUser attribute of the GRADS class when setUser() is called.
Test Inputs:	Valid user ID “tolas9999”
Expected Results:	currentUser is set to user “tolas9999”
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call loadUsers() with valid user file containing data for “tolas9999”
Test Steps:	1. Call setUser() with user “tolas9999” 2. Verify that currentUser has value “tolas9999”

Test Case 4B **Set User Exception**

Description:	Verifies that GRADS correctly returns an ExIdNotFound exception when setUser() is called with an invalid user ID
Test Inputs:	Invalid user ID
Expected Results:	ExIdNotFound exception is thrown
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call loadUsers() with valid user file that does not contain data for input user ID
Test Steps:	1. Call setUser() with invalid user ID 2. Verify that ExIdNotFound exception is thrown

Test Case 5 **Get User**

Description:	Verifies that GRADS correctly gets the value of the currentUser attribute of the GRADS class when getUser() is called.
Test Inputs:	Valid user ID “smith0001”
Expected Results:	“smith0001” is returned as the current user
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call loadUsers() with valid user file containing data for “smith0001”. Set current user to “smith0001”.
Test Steps:	1. Call getUser() 2. Verify that “smith0001” is returned as the current user

Test Case 6A **Get Student IDs - GPC User**

Description:	Verifies that GRADS correctly retrieves the list of student IDs when getStudentIDs is called by a GPC.
Test Inputs:	None
Expected Results:	List returned is not null
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call loadUsers() with valid user file and call loadRecords with a valid records file. Set current user to GPC.
Test Steps:	1. Call getStudentIDs() 2. Verify that the list returned is not empty.

Test Case 6B **Get Student IDs - Student User**

Description:	Verifies that GRADS correctly returns an ExIncorrectPermissions exception when a student user tries to call getStudentIDs()
Test Inputs:	None
Expected Results:	ExIncorrectPermissions exception is thrown
Dependencies:	None

Initialization:	Initialize a new instance of the GRADS class. Call loadUsers() with valid user file and call loadRecords with a valid records file. Set current user to student.
Test Steps:	1. Call getStudentIDs() 2. Verify that an ExIncorrectPermissions exception is thrown.

Test Case 7A **Get Transcript - GPC User**

Description:	Verifies that GRADS correctly returns a transcript for a student when a GPC user calls getTranscript().
Test Inputs:	Valid user ID: “tolas9999” Valid (existent) records file: newStudentsPHD.txt
Expected Results:	The transcript that is returned is the same as the transcript in file “group15PHD_PASSED”
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user file and call loadRecords() with a valid records file. Set current user to GPC.
Test Steps:	1. Call getTranscript() with a valid student ID. 2. Create a JSON instance of the valid transcript file. 3. Assert the files are equal.

Test Case 7B **Get Transcript - Student User**

Description:	Verifies that GRADS correctly returns a transcript for a student who requests their own transcript by calling getTranscript().
Test Inputs:	Valid user ID: “nguy0621”
Expected Results:	The transcript that is returned is the same as the transcript in file “transcriptNguy0621.txt”
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user file and call loadrecords() with a valid records file. Set current user to student.
Test Steps:	1. Call getTranscript() with current student’s ID. 2. Create a JSON instance of the valid transcript file. 3. Assert the files are equal.

Test Case 7C **Get Transcript - Student User with Incorrect Permissions**

Description:	Verifies that an ExIncorrectPermissions exception is thrown when a student user tries to retrieve another student's transcript using getTranscript().
Test Inputs:	Valid user ID: "nguy0621"
Expected Results:	exIncorrectPermissions exception.
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user file and call loadRecords() with a valid records file. Set current user to student.
Test Steps:	1. Call getTranscript() with a student ID which differs from the current user. 2. Verify exIncorrectPermissions is thrown.

Test Case 7D **Get Transcript - Invalid User**

Description:	Verifies that an ExIdNotFound exception is thrown when a user tries to retrieve a transcript for a student that does not exist.
Test Inputs:	Valid user ID: "nguy0621"
Expected Results:	ExIdNotFound exception.
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user.
Test Steps:	1. Call getTranscript with an invalid student ID. 2. Verify that ExIdNotFound is thrown.

Test Case 8A **Update Transcript - GPC User**

Description:	Verifies that a GPC user is able to update a student's transcript.
Test Inputs:	Valid user ID: "tolas9999" Valid (existent) records file: newStudentsPHD.txt
Expected Results:	An updated StudentRecord which differs from the original StudentRecord.
Dependencies:	None

Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user and call loadRecords() with a valid records file.
Test Steps:	1. Call getTranscript() with a valid studentID. 2. Create a JSON instance of the records file. Assert that is equal to the generated transcript. 3. Call updateTranscript() with a new student record. 4. Create a JSON instance of the updated records file. Assert that it is equal to the updated transcript.

Test Case 8B **Update Transcript - Student User (Incorrect Permissions)**

Description:	Verifies that a student is not able to update their transcript, and that an ExIncorrectPermissions exception is thrown when they try to do so.
Test Inputs:	Valid user ID: “nguy0621”
Expected Results:	ExIncorrectPermissions exception.
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser with valid user. Call getTranscript() for the current user.
Test Steps:	1. Update coursesTaken with “test1234” 2. Call updateTranscript with new courseTaken. 3. Verify that ExIncorrectPermissions is thrown.

Test Case 9A **Add Note - GPC User**

Description:	Verifies that a GPC user can successfully add a note to a student’s record.
Test Inputs:	Valid user ID: “tolas9999” Valid (existent) records file: newStudentdsPHD.txt
Expected Results:	An updated transcript with a new note is produced.
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with a valid GPC user and call loadRecords() with a valid records file.
Test Steps:	1. Call addNote() with a valid user ID currently in the students list. 2. Create a JSON instance of the source record. Verify that it is the same as the generated note.

3. Update the students transcript with the note.
4. Assertthat updated transcript contains the note.

Test Case 9B **Add Note - Student User (Incorrect Permissions)**

Description:	Verifies that a student user cannot add a note to their record, and that an ExIncorrectPermissions exception is thrown when they try to do so.
Test Inputs:	Valid user ID: “nguy0621”
Expected Results:	exIncorrectPermissions exception.
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with a valid student user and call loadRecords() with a valid records file.
Test Steps:	1. Call addNote() with a valid note 2. Verify that ExIncorrectPermissions exception is thrown.

Test Case 10A **Generate Progress Summary - PHD - GPC User**

Description:	Verifies that a GPC user can successfully generate a progress summary for a PHD student.
Test Inputs:	Valid user ID: “tolas9999” Valid source file: “newStudentsPHD.txt”
Expected Results:	A ProgressSummary is generated for “group15PHD_PASSED”
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user ID and call loadRecords() with a valid records file.
Test Steps:	1. Call generateProgressSummary() with valid student id. 2. Assert that returned ProgressSummary matches the file progressSummaryPHD_PASSED.txt.

Test Case 10B **Generate Progress Summary - MS_A - GPC User**

Description:	Verifies that a GPC user can successfully generate a progress summary for an MS_A student.
Test Inputs:	Valid user ID: “tolas9999” Valid source file: “newStudentsMS_A.txt”

Expected Results:	A ProgressSummary is generated for “group15MS_A_PASSED”
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user ID and call loadRecords() with a valid records file.
Test Steps:	1. Call generateProgressSummary with valid student ID. 2. Assert that the returned ProgressSummary matches the file progressSummaryMS_A_PASSED.txt.

Test Case 10C **Generate Progress Summary - MS_B - GPC User**

Description:	Verifies that a GPC user can successfully generate a progress summary for an MS_B student.
Test Inputs:	Valid user ID: “tolas9999” Valid source file: “newStudentsMS_B.txt”
Expected Results:	A ProgressSummary is generated for “group15MS_B_PASSED”
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user ID and call loadRecords() with a valid records file.
Test Steps:	1. Call generateProgressSummary() with given source file. 2. Assert that the returned ProgressSummary matches the file progressSummaryMS_B_PASSED.txt.

Test Case 10D **Generate Progress Summary - MS_C - GPC User**

Description:	Verifies that a GPC user can successfully generate a progress summary for an MS_C student.
Test Inputs:	Valid user ID: “tolas9999” Valid source file: “newStudentsMS_C.txt”
Expected Results:	A ProgressSummary is generated for “group15MS_C_PASSED”
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user ID and call loadRecords() with a valid records file.
Test Steps:	1. Call generateProgressSummary() with the given studentID. 2. Assert that the returned ProgressSummary matches the file progressSummaryMS_C_PASSED.txt.

Test Case 10E **Generate Progress Summary - PHD - Student User**

Description:	Verifies that a student user can successfully generate a progress summary for a PHD student.
Test Inputs:	Valid student userid “nguy0621” Expected output file ProgressNguy0621.txt
Expected Results:	GRADS will return a ProgressSummary matching the input file
Dependencies:	None
Initialization:	Create a new instance of GRADS and load the database.
Test Steps:	1. Set user to “nguy0621”. 2. Call generateProgressSummary 3. Assert that generated ProgressSummary matches input file.

Test Case 10F **Generate Progress Summary - MS_A - Student User**

Description:	Verifies that a student user can successfully generate a progress summary for an MS_A student.
Test Inputs:	Student ID group15MS_A_PASSED progressSummaryMS_A_PASSED.txt newUsers.txt
Expected Results:	GRADS will generate a progress summary for the given user.
Dependencies:	None
Initialization:	Initialize a new instance of GRADS. Call loadUsers with newUsers.txt and loadRecords with newStudentsMS_A.txt
Test Steps:	1. Set current user to group15MS_A_PASSED 2. Call generateProgressSummary 3. Verify that GRADS returns a progress summary which matches the file progressSummaryMS_A_PASSED.txt

Test Case 10G **Generate Progress Summary - MS_B - Student User**

Description:	Verifies that a student user can successfully generate a progress summary for an MS_B student.
Test Inputs:	Student ID group15MS_B_PASSED progressSummaryMS_B_PASSED.txt

	newUsers.txt
Expected Results:	GRADS will generate a progress summary for the given user.
Dependencies:	None
Initialization:	Initialize a new instance of GRADS. Call loadUsers with newUsers.txt and loadRecords with newStudentsMS_B.txt
Test Steps:	1. Set current user to group15MS_B_PASSED 2. Call generateProgressSummary 3. Verify that GRADS returns a progress summary which matches the file progressSummaryMS_B_PASSED.txt

Test Case 10H **Generate Progress Summary - MS_C - Student User**

Description:	Verifies that a student user can successfully generate a progress summary for an MS_C student.
Test Inputs:	Student ID group15MS_C_PASSED progressSummaryMS_C_PASSED.txt newUsers.txt
Expected Results:	GRADS will generate a progress summary for the given user.
Dependencies:	None
Initialization:	Initialize a new instance of GRADS. Call loadUsers with newUsers.txt and loadRecords with newStudentsMS_C.txt
Test Steps:	1. Set current user to group15MS_C_PASSED 2. Call generateProgressSummary 3. Verify that GRADS returns a progress summary which matches the file progressSummaryMS_C_PASSED.txt

TEST CASE 11 **For test cases 11A-11E, since there is no direct equality check for ProgressSummary, the individual attributes will be checked for equality one at a time. Determining the equality of two lists of requirement check results will be accomplished by iterating through the two lists and verifying that the same name/passed pairs exist in each list.**

Test Case 11A **Check Course Impact (Single Course)**

Description: Determine that a CourseTaken will impact a student's ProgressSummary.

Test Inputs: Valid user ID: “nguy0621”
Valid source file: “transcriptNguy0621.txt”

Expected Results: The ProgressSummary returned from checking the course’s impact will be different from the original ProgressSummary

Dependencies: None
Initialization: Initialize a new instance of GRADS and load data.

Test Steps:
1. Set user to nguy0621
2. Generate list of courses taken identical to the list for the current user
3.Create a new course and add it to the new list
4. Create a ProgressSummary for current user. Then call simulateCourses() using the new course list, and verify that the two summaries are different.

Test Case 11B **Check Course Impact (Multiple Courses)**
Description: Determine that multiple CourseTaken(s) will impact a students ProgressSummary.

Test Inputs: Valid user ID: “nguy0621”
Valid source file: “transcriptNguy0621.txt”

Expected Results: ProgressSummary returned from simulateCourses() will differ from the ProgressSummary returned from generateProgressSummary().

Dependencies: None
Initialization: Initialize a new instance of GRADS and load all data.

Test Steps:
1. Set current user to nguy0621
2. Create a list of courses taken that matches the current student’s courses taken
3. Generate a new course and add it to the new list
4. Call generateProgressSummary and verify that the returned ProgressSummary differs from that returned by simulateCourses called with the new course

Test Case 11C **Check Course Impact (Incorrect Permissions)**

Description: Verify that only a student and a GPC can check a course’s impact on his/her (student) Progress Summary. Ensure ownership privileges.

Test Inputs: Valid user ID: “nguy0621”
Valid source file: “transcriptNguy0621.txt”

Expected Results: GRADS will throw an exception

Dependencies: None
Initialization: Initialize a new instance of the GRADS class and load all data

Test Steps:
1. Set current user to nguy0621
2. Create a new list of courses taken equal to the current user's list of courses taken
3. Call simulateCourses() and verify that GRADS throws an exception.

Test Case 11D **Check Course Impact (No Change)**

Description: Verify that checking a course's impact works properly with no new course added.

Test Inputs: Valid user ID: "tolas9999"
Valid source file: "transcriptNguy0621.txt"

Expected Results: Calling the simulateCourses() method with a given list of courses taken will return the same result as generateProgressSummary() for the same list of courses.

Dependencies: None
Initialization: Initialize a new instance of the GRADS class and load all data

Test Steps:
1. Set user to tolas9999
2. Create new list of courses taken equal to the list of courses taken for nguy0621
3. Assert that the ProgressSummary for the user ngyu0621 by calling generateProgressSummary() matches the ProgressSummary returned from simulateCourses() with the new course list (which should also remain unchanged).

Test Case 11E **Check Course Impact (Course Which Has No Impact)**

Description: Verify that checking a course's impact works properly with no new course added.

Test Inputs: Valid user ID: "nguy0621"
Valid source file: "transcriptNguy0621.txt"

Expected Results: Calling the simulateCourses() method with a given list of courses taken will return the same result as generateProgressSummary() for a list of courses containing an additional course which does not affect the student's progress toward graduation.

Dependencies: None
Initialization: Initialize a new instance of the GRADS class and load all data

Test Steps:
1. Set user to nguy0621
2. Create new list of courses taken equal to the list of courses taken for nguy0621. Add a course to this list. Since the student has already passed, this course should not affect the progress summary.
3. Assert that the ProgressSummary for the user ngyu0621 by calling generateProgressSummary() matches the ProgressSummary returned from simulateCourses() with the new course list.

Test Case 12A **Check Empty Transcript - PHD - GPC User**

Description:	Verifies that a GPC user can successfully generate a progress summary for a PHD student with empty transcript.
Test Inputs:	Valid user ID: “tolas9999” Valid source file: “newStudentsPHD.txt”
Expected Results:	An empty ProgressSummary is generated for “group15PHD_EMPTY”
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user ID and call loadRecords() with a valid records file.
Test Steps:	3. Call generateProgressSummary() with valid student id. 4. Assert that returned ProgressSummary matches the file progressSummaryPHD_EMPTY.txt.

Test Case 12B **Check Empty Transcript - MS_A - GPC User**

Description:	Verifies that a GPC user can successfully generate a progress summary for a MS_A student with empty transcript.
Test Inputs:	Valid user ID: “tolas9999” Valid source file: “newStudentsMS_A.txt”
Expected Results:	An empty ProgressSummary is generated for “group15MS_A_EMPTY”
Dependencies:	None

Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user ID and call loadRecords() with a valid records file.
Test Steps:	5. Call generateProgressSummary() with valid student id. 6. Assert that returned ProgressSummary matches the file progressSummaryMS_A_EMPTY.txt.

Test Case 12C **Check Empty Transcript - MS_B - GPC User**

Description:	Verifies that a GPC user can successfully generate a progress summary for a MS_B student with empty transcript.
Test Inputs:	Valid user ID: “tolas9999” Valid source file: “newStudentsMS_B.txt”
Expected Results:	An empty ProgressSummary is generated for “group15MS_B_EMPTY”
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user ID and call loadRecords() with a valid records file.
Test Steps:	7. Call generateProgressSummary() with valid student id. 8. Assert that returned ProgressSummary matches the file progressSummaryMS_B_EMPTY.txt.

Test Case 12D **Check Empty Transcript - MS_C - GPC User**

Description:	Verifies that a GPC user can successfully generate a progress summary for a MS_C student with empty transcript.
Test Inputs:	Valid user ID: “tolas9999” Valid source file: “newStudentsMS_C.txt”
Expected Results:	An empty ProgressSummary is generated for “group15MS_C_EMPTY”
Dependencies:	None
Initialization:	Initialize a new instance of the GRADS class. Call setUser() with valid user ID and call loadRecords() with a valid records file.
Test Steps:	9. Call generateProgressSummary() with valid student id. 10. Assert that returned ProgressSummary matches the file progressSummaryMS_C_EMPTY.txt.

3. Testing Overview & Analysis

Which portions of our code were difficult to cover.

Testing GRADS proved to be deceptively difficult for a seemingly straightforward system implementation. Our Progress Summary generation methods were particularly difficult to test. At a minimum we needed to test generating summaries for both student and GPC users for each of the four student paths (4 * 2 = 8) test cases In our testing environment, we created tests for summaries in which a corresponding student had met all the requirements. To achieve more comprehensive coverage within the Progress Summary methods, we might have opted to include tests in which a student had met; no milestones, some milestones, or some intermediate sized subset of milestones. Given the fairly extensive breadth of our testing scope; we felt that achieving this was not a time effective endeavor.

Also, there were times where we discovered during testing that our data was inadequate. In particular, while writing tests for the simulateCourses() method, the test kept mysteriously showing that the ProgressReport generated with the extended course list was identical to the original. The most obvious place to look for the error was in the code, but it eventually became clear that since the test data being used was for a student who had already met all of the requirements, any course at all would have no impact.

There were also, as always, subtle bugs in the tests themselves. Again, during the ProgressSummary test, we had been verifying that the initial ProgressSummary was not equal to the simulated ProgressSummary. At first, this seemed like a reasonable approach, and the tests all passed. But eventually, while we were working on another test, it became clear that these were false positives, and that the equality test was just correctly identifying that the two ProgressSummarys were not the same object, rather than checking the actual data within.

Why 100% statement coverage may not be possible with system level tests.

Achieving 100% statement coverage for our GRADS system was not realistic, as tests written to achieve additional coverage above our coverage of ~80% ran into a region of diminishing returns. Testing every possible combination of requirement checks for each of the four possible student paths was not realistic given our time frame of approximately two additional weeks after implementation.

Additionally, we included several exceptions to handle possible fatal errors of the GRADS system. These included testing for insufficient space within the system, possible null values, or invalid values for enumerated types - which, in an ideal world, should be impossible. Supplemental private methods were also included in our system for the same reason. Above all, these were included for defensive programming purposes; and achieving coverage of these conditions was not practical.

Given a longer deadline, we may have asymptotically approached complete coverage; but exhaustive testing of every course, student, and requirement input combination was not practical for our GRADS implementation.

Further, 100% statement coverage would not be guaranteed to find all faults in any case, and at a certain point, reviewing the code manually line-by-line, perhaps by someone other than the original coder, seems at least as beneficial as an extra percentage point of coverage with our system tests.