ECSE 321 - Intro to Software Engineering
Design Specification Document - Deliverable 3

Harley Wiltzer
Camilo Garcia La Rotta
Jake Shnaidman
Robert Attard
Matthew Lesko

March 19, 2017

Contents

Part I

Unit Test Plan

1 Description

In this section we will present an extensive suite of test cases aimed to cover in the most efficient manner possible the Fantasy Basketball system we are to deliver. Following the paradigm of Test-Driven Development, we shall also implement the unit test cases. This will help keep the development of the system on track with the requirements imposed to the team.

2 Test Cases

All the classes to test will have the following pattern:

CLASS TO TEST/NOT TO TEST

- Reason to test/not to test
- attributes and related tests

2.1 Classes To Test

- **Course**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. Its attributes TA/Grader time budget define how many students can apply to the position. Less relative to the application, but still needed for the correct behaviour of the process is the fact that the student must have experience in the given course in order to apply for a position as TA/Grader for it.
  - **Test Cases:**
    - **className**: test for null, empty, only numerical and only spaces inputs.
    - **CDN**: test for non-unique, null, empty, not alphabetic and only spaces inputs.
    - **className**: test for null, empty, not numerical and only spaces inputs.
    - **graderTimeBudget/taTimeBudget**: test for null, empty, not numerical and negative inputs.
    - **Jobs**: test for retrieval of job list
- **Job**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. It encapsulates half of the persistence aspect of the app which is to publish and view job postings.
  - **Test Cases:**
    - **CDN**: test for non-existent, null, empty, only numerical and only spaces inputs.
    - **requirements**: test for null, empty, empty and only spaces inputs.
    - **position**: test for null inputs.
    - **salary**: test for null, empty, not numerical and negative inputs.
    - **day**: test for null, empty and weekend inputs.

    * **startTime/endTime**: test for null, empty and outside of working hours inputs.

- **Tutorial**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. Job postings are for either TA/Grader. In the first case, the availabilities of the TA must be compatible with those of the tutorial.
  - **Test Cases:**
    * test the same attributes as Course to ensure inheritance was correctly implemented by the UML

- **Laboratory**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. Job postings are for either TA/Grader. In the first case, the availabilities of the TA must be compatible with those of the laboratory.
  - **Test Cases:**
    * test the same attributes as Course to ensure inheritance was correctly implemented by the UML

- **Profile**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. The system requires profile identifiers to discern which methods and attributes each instance has access to.
  - **Test Cases:**
    * **id**:test for non-unique, null, empty, not alphabetic and only spaces inputs.
    * **username**: test for non-unique, null, empty, not alphanumeric and only spaces inputs.
    * **password**: test for invalid difficulty, null and empty inputs.
    * **firstName/lastName**: test for null, empty, not alphabetical and only spaces inputs.

- **Student**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. The student is the only instance of profile allowed to apply to a job posting.
  - **Test Cases:**
    * test the same attributes as Profile to ensure inheritance was correctly implemented by the UML
    * **experience**: test for null, empty and only spaces inputs.
    * **degree**: test for null inputs.
    * **Jobs**: test for retrieval of job list

- **Instructor**

- **Reason to test:** This class is intrinsic to the job posting and application transaction. The instructor is the only instance of profile allowed to post jobs.
- **Test Cases:**
  - test the same attributes as Profile to ensure inheritance was correctly implemented by the UML
  - **experience**: test for null, empty and only spaces inputs.
  - **degree**: test for null inputs.
  - **Jobs**: test for retrieval and modification of job list
  - **Application**: test for retrieval of application list
  - **course**: test for retrieval of course list

- **Application**
  - **Reason to test:** This class is necessary for the Student to apply for a job. Each application is associated to a student and a job posting; hence one needs to test for different Student attributes for different Jobs.
  - **Test Cases:**
    - **Test Apply for a Job:** test for error if hours are not compatible, test for error if job is null, and test for error if student is null.

- **Admin**
  - **Reason to test:** This class is intrinsic to the creation of a course, the creation of a Student and Instructor, and application transaction. Its attributes inherit from profile. They define information about the person that is registering an admin profile; hence it is needed to test for different attribute inputs and for successful creation of classes.
  - **Test Cases:**
    - test the same attributes as Profile to ensure inheritance was correctly implemented by the UML
    - **Application Transaction:** test for successful application transaction between Student and Job Posting
    - **Successfully Create Classes:** test to successfully create Instructor, and Student.

- **Application Manager**
  - **Reason to test:** This class acts as a container for Application and Job. It is necessary to test that it can correctly get and set Application and Job attributes from within the persistence layer.
  - **Test Cases:**
    - test to successfully retrieve correct attributes for Application and Job from within the persistence layer: Instantiate one ApplicationManager that sets attributes for two new instances of the Application and Job classes, then instantiate another ApplicationManager that gets the same attributes from the same Application and Job objects. assertEquals these attributes to ensure they were correctly saved.

- **Profile Manager**
  - **Reason to test:** This class acts as a container for Student, Instructor, and Admin. It is necessary to test that it can correctly get and set Student, Instructor and Admin attributes from within the persistence layer.
  - **Test Cases:**
    * test to successfully retrieve correct attributes for Student, Instructor, and Admin from within the persistence layer: Instantiate one ProfileManager that sets attributes for instances of the Student, Instructor, and Admin classes, then instantiate another ProfileManager that gets the same attributes from the same Student, Instructor, and Admin objects. assertEquals these attributes to ensure they were correctly saved.
- **Course Manager**
  - **Reason to test:** This class acts as a container for Course. It is necessary to test that it can correctly get and set Course attributes from within the persistence layer.
  - **Test Cases:**
    * test to successfully retrieve correct attributes for Course from within the persistence layer: Instantiate one CourseManager that sets attributes for a new Course object, then instantiate another CourseManager that gets the same attributes from the same Course object. assertEquals these attributes to ensure they were correctly saved.
- **Persistence**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. Without persistence capability the controllers have no data from which to derive the desired outcome
  - **Test Cases:**
    * creation, modification and deletion of Course, Job, Application and Profile instances.

2.2 Classes to Not Test

- **No Classes to Not Test**
  - **E**very class is tested

3 Techniques and Tools

The system is to be designed in a complete Test-Driven Development paradigm. Hence all test cases will be written before the actual controllers are implemented. Furthermore, the order in which the tests will be passed follows the same order as the requirements derived in the Requirements document of deliverable #1. This will aid the classification and prioritization of each bi-weekly runs' objectives. In parallel to this method, the team will rely heavily on code revisions by the

senior members of the team to ensure the code written is up to visual, performance and logical standards. In terms of frequency, individual nightly tests will be ran on all platforms of the system and bi-weekly builds and test will be done every Saturday to ensure the deliverable validates and verifies the given requirements. To facilitate the testing process, the team will use the following tools:

- **Unit Test Framework:** Automated, well documented and supported system to allow a standardized set of tests to be done regardless of the platform. JUnit and PHPUnit will be used.
- **Test Coverage Tool:** Ensure that the measure to which the tests and actions cover the source code is optimal by function, statement, branch and condition standards. Its outputs will be used during the code revision sessions. EclEmma and PHPUnit will be used for this purposes.

4 Coverage Statistics

TODOOOOOOO To the extent possible, include a quantification of your testing goal (e.g., N% statement coverage or M% branch coverage).

Furthermore, discuss how often your tests are executed, i.e., what triggers your tests to be run? Finally, describe any differences between your desktop/laptop app, mobile app, and web app when it comes to unit testing.

Part II

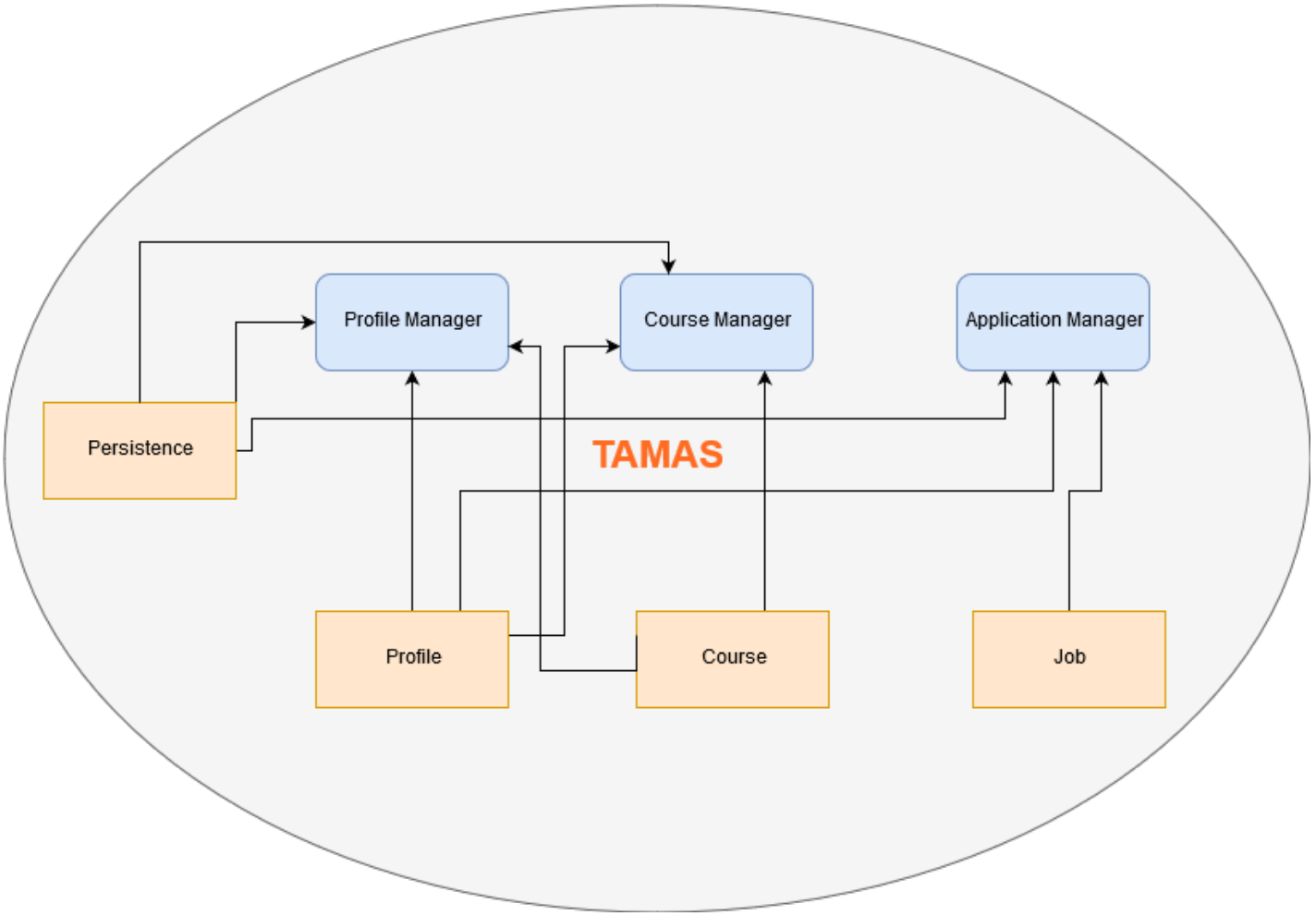
Integration Test Plan

5 Integration Strategy

The strategy of integration will follow a bottom-up approach. The system will first have all the unit tests done for each class. Then the manager classes will integrate the units together into discrete subsystems. Since course manager is going to be used on all platforms, it was decided to test this subsystem first. Using EclEmma, we aspire to have near 100% code coverage. This would imply that there should be near 100% statement and branch coverage. Since our mobile app is only for students, there is no need to test the course manager on the mobile application. Likewise, the web app will not need to test the application manager since the web app is just for teachers. All the tests that follow are done programmatically rather than through a user interface. For example, Applying for a job will involve just creating an instance of an application and attributing it a profile, a job, etc. The system test will test whether the user can create a form and apply through the mobile application, for example.

5.1 Subsystem Testing Diagram

The following diagram shows the subsystems and interactions that will be tested.



6 Integration Test descriptions

6.1 Subsystem Testing

- **Profile Persistence Subsystem Testing**
  - **Reason to test:** These tests will evaluate the relationship between the profiles the persistence layer, this is necessary to ensure that all profile data and associations are properly propagated throughout the system for later use.
  - **Situations:**
    - **Create a profile:** Test to ensure that a profile can properly be created in persistence.
      - Create profile using complete valid profile data and ensure that it is properly persisted and available to other classes.
      - Attempt to create a profile using invalid or incomplete input data, ensure that it is not propagated in the persistence layer.
      - Attempt to create a profile by giving no input data, ensure that this is not propagated in the persistence layer.
    - **Modify a profile:**
      - Modify a profile so that the new profile data is valid, this new profile data should persist.
      - Modify a profile so that the new data is invalid/incomplete, the system should not allow this so the persistence layer will remain unchanged.
    - **Delete a profile:**
      - Deleting a valid profile should remove it's data from the persistence layer.
      - Deleting an invalid/nonexistent profile should be rejected and have no effect on the persistence layer.

- **Course Persistence Subsystem Testing**
  - **Reason to test:** These tests will evaluate the relationship between the courses, the profiles and the persistence layer this is necessary to ensure that all course data and associations are properly propagated throughout the system.
  - **Situations:**
    - **Create a course**
      - Create a course using complete valid course data and ensure that it is properly persisted and available to other classes.
      - Attempt to create a course using invalid or incomplete input data, ensure that it is not propagated in the persistence layer.
      - Attempt to create a course by giving no input data, ensure that this is not propagated in the persistence layer.
    - **Modify a course**
      - Modify a course so that the new data is valid, this new course data should persist.

        - Modify a course so that the new data is invalid/incomplete, the system should not allow this so the persistence layer will remain unchanged.
    - **Delete a course**
        - Deleting a valid course should remove it's data from the persistence layer.
        - Deleting an invalid/nonexistent course should be rejected and have no effect on the persistence layer.

- **Application Persistence Subsystem Testing**
    - **Reason to test:** These tests will evaluate the relationship between the profiles and the jobs that make up the applications and the persistence layer.
    - **Situations:**
        - **Create an application**
            - Create an application using complete valid data and ensure that it is properly persisted and available to other classes.
            - Attempt to create an application using invalid or incomplete input data, ensure that it is not propagated in the persistence layer.
            - Attempt to create an application by giving no input data, ensure that this is not propagated in the persistence layer.
        - **Modify an application**
            - Modify an application so that the new data is valid, this new course data should persist.
            - Modify an application so that the new data is invalid/incomplete, the system should not allow this so the persistence layer will remain unchanged.
        - **Delete an application**
            - Deleting a valid application should remove it's data from the persistence layer.
            - Deleting an invalid/nonexistent application should be rejected and have no effect on the persistence layer.

Part III

System Test Plan

7 Description

The purpose of system testing is to validate the functionality of the system as a whole. System tests are black-box tests, meaning they are done without any concern regarding the code - that is to say, only the inputs and outputs are examined.

So as to examine the functionality of the system in an organized and systematic fashion, system tests will be designed according to the various *use cases* of the system. The use cases provide concrete details concerning how each functionality of the system should behave, and in particular how each functionality of the system should react to unexpected or faulty input.

The proposed system tests will be discussed below.

7.1 Administrator-Specific Actions

1. **Sending Student Enrollment Data and Course List**

   (a) Valid data entered

   - Enter appropriate data for student enrollment, as well as the course list, in the desktop application. Ensure that the proper XML files are generated.

   (b) Invalid student enrollment data entered

   - Enter faulty student enrollment data (by omitting names, for example) and enter correct course list data in the desktop application. Ensure that an error message about invalid student data is made present. Moreover, assert that the corresponding student enrollment XML has not been generated at all, and that the course list XML file has been generated appropriately.

   (c) Invalid course list data entered

   - Enter faulty course list data (by omitting required names, or by including non-integral course codes, for example) and enter correct student data in the desktop application. Ensure that an error message about invalid course list data is made present. Moreover, assert that the corresponding course list XML file has not been generated at all, and that the student enrollment XML file has been generated appropriately.

2. **Generating Initial Student Job Placements**

   (a) Valid XML Application file is present

   - Ensure that the appropriate XML file containing data about student applications is present. Use desktop application to generate default TA placement.
     i. Verify that an appropriate XML file containing data about the job offerings was generated.
     ii. Verify that initial placement confines to the budget for each department.
     iii. Verify that graduate students were selected with higher priority than undergraduate students.
     iv. Verify that the same student was selected for as many jobs as possible for the same course.

(b) XML Application file not found or is corrupted

- Place invalid XML file (containing syntax errors or type mismatches, for example) in the appropriate location, and attempt to generate default TA placement with the desktop application. Ensure that an error message about the invalidity of the XML file is output, and that no XML file about job offerings has been created.

3. **Accepting or Rejecting Instructor TA Modifications**

(a) Accepting a modification

- Ensure that a list of all instructor modifications is present in the *view proposed modifications* page. Select a modification for viewing, and select to accept the modification.
  i. Ensure that the original allocations XML file has been overwritten using the data from the modification XML file that is currently active.

(b) Rejecting a modification

- Ensure that a list of all instructor modifications is present in the *view proposed modifications* page. Select a modification for viewing, and choose to reject the modification.
  i. Ensure that the original allocations XML file has not been modified.
  ii. Ensure that the modifications XML file has been deleted.

4. **Send Job Offers**

(a) Modification XML files present

- Ensure that a TA placement modification XML file is present. Attempt sending job offers in the *view allocation* page.
  i. Ensure that no `offers.xml` file is generated.
  ii. Ensure that an error message informing the user that there are still outstanding modifications is output.

(b) No modification XML files are present

- Remove all modification XML files, and ensure that initial allocation XML file is present. Attempt sending job offers in the *view allocation* page.
  i. Ensure that the `offers.xml` file has been generated with appropriate job offer data corresponding to the TA allocation.

7.2 Instructor Use-Cases

1. **Publish Job Posting**

(a) No course data found

- Remove all course data, or delete the `courses.xml` file. Attempt to open the page containing the *publish job posting* form. Ensure that the page does not open, and an error message informing the user that no courses are available is output.

(b) Course and Instructor data present

- Ensure the presence of the `courses.xml` and `profiles.xml` files, and make sure at least one course is present. Then, open the *publish job posting* form and assert that for the indicated Instructor, all of its courses, and only courses that are being taught by that given professor, are available for selection in the Course list.

(c) Valid input entered, no previous XML file found

- Specify valid information corresponding to a job posting in the *publish job posting* form, and submit the data. Ensure that a XML file called `applications.xml` was generated and contains correct data concerning the job postings that were submitted.

(d) Valid input entered, with previous XML file in place

- With a previously-generated `applications.xml` in place, attempt submitting a job posting using the *publish job posting* form. Ensure that the XML file has been modified such that the new job postings have been appended after the previous ones (therefore, ensure no data was lost).

(e) Type mismatches or empty fields submitted

- Attempt submitting a job posting, leaving fields blank or including type mismatches, such as a non-integral input in the Salary field. Ensure that the `applications.xml` file was not modified (and if it did not previously exist, ensure that it has not been generated).

(f) Valid, but unsound data submitted

- Attempt submitting a job posting with unsound data, such as a start time that is later than an end time.
  i. Ensure that the appropriate error message is output.
  ii. Ensure that the `applications.xml` file was not modified (and if it did not previously exist, ensure that it has not been generated).

2. **Modify initial TA allocation**

(a) No initial allocation data found

- Remove XML file with initial allocation data. Attempt to open TA placement modification page. Ensure that the page does not open, and that an error message informing the user that no allocation has been made is output.

(b) Valid modifications entered

- Enter valid modifications in the modification form, and submit.
  i. Ensure that the initial allocation XML file has not been modified.
  ii. Ensure that a seperate XML file containing the data about the modifications has been created appropriately.

(c) Invalid modifications entered

- Enter modifications that have empty fields or type mismatches, for example. Attempt submitting modifications.
  i. Verify that an error message containing the details of the invalidity of the input has been shown.

ii. Ensure that the initial allocation XML file has not been modified.

iii. Ensure that no seperate XML files containing modification data have been generated.

3. **Evaluate TA/Grader**

   (a) No student XML data found or no TA's associated to current professor.

   - Remove all student data, or remove job data for a specific professor. Under such professor's account, attempt to open the *evaluate TA* page. Ensure that the page does not open, and that a message informing the user that no TA's are available is output.

   (b) TA data available

   - Open the *evaluate TA* page.
     i. Ensure that only the TA's that worked for courses taught by the selected professor are available for evaluation.
     ii. After submission, ensure that the Student that was evaluated had its `experience` field modified in the `profiles.xml` file.

7.3 Student Use-Cases

1. **Apply to Job Posting**

   (a) No Job data available

   - Remove `applications.xml` file, or ensure that it contains no Job data. Attempt opening the *apply to job posting* page. Ensure that the page does not open and that a message informing the user that Jobs have not been created yet is output.

   (b) Valid input entered

   - Ensure that Job data is available. Open *apply to job posting* page and select a Job to apply to. Ensure that all Jobs that were present in the XML data are available in the selection list. Ensure that after submission, the `applications.xml` file was modified appropriately, and that no data has been lost.

   (c) Invalid input submitted

   - Ensure that Job data is available. Open the *apply to job posting* page and select a Job to apply to. Leave Job field blank, and attempt submitting.
     i. Ensure that an error message informing the user that no Job was selected has been output.
     ii. Ensure that the `applications.xml` file has not been modified.

2. **Submit Personal Details**

   (a) Open *modify profile* page. Ensure that all current data for the current Student is present in the form. Modify data in the forms and submit. Ensure that the *profiles.xml* file has been modified appropriately.

3. **Accept or Reject Job Offerings**

(a) Open the *job offers* page. Ensure that only jobs offered to the current Student are listed. Attempt accepting a job offer and rejecting a job offer. Ensure that the `offers.xml` file has been modified accordingly.

8 Rationale

The system tests that were designed, and their classifications by main actors, were done to provide an organized scheme to verify that all functionalities of the system are covered by the system tests. Given the fact that system tests should be black-box, it only remains to determine if the desired use cases of the system are functioning according to the client's requirements.

To ensure that all requirements are met, the system tests were divided into categories specified by the actor that would use those functionalities. Section **7.1** shows system tests associated with the functionalities that only an Administrator would be responsible for. Then, section **7.2** shows system tests associated with the Instructor's actions. Finally, section **7.3** shows system tests associated with the Student's actions. This categorization made it much simpler to verify that all functionalities were tested, because it allowed functionalities to be listed off as the Use Cases defined in the first Deliverable.

It is important to note that all logic-related bugs concerning the fine details of the calculations carried out by the system should be already covered by the Unit Tests and Integration tests above. Hence, the goal of the System Test plan is to design a routine that tests the communications and interactions between subsystems that were tested in the Integration Tests. Such communications involve mainly the flow of information from the View classes all the way down to the Controller classes that store information about the domain entities in the persistence. All tests described in section **7** verify the proper functioning of the input parsing from the Views, and the output results of the persistence. Once again, everything that happens between the input and the output is in a *black-box*, and it is assumed to have been tested in the previous testing schemes.

Another important thing to consider when designing these System Tests is that the users will not always enter valid inputs, and illegal actions may be attempted. Therefore, testing the ideal inputs only is not a good idea, because it is equally important that the system handles erroneous or invalid input properly as well. To take this into account, each System Test Case was examined, and possible areas of invalid input were found. Each of those invalid input scenarios is accounted for in the System Test plan. Most importantly, it is imperative that submitting erroneous input does not affect the current persistence. Although each test case may experience significantly different errors, if the persistence is not modified under these conditions, the errors will not affect the *bigger picture* of the whole system. Therefore, to complete each test case, any possible invalid inputs are isolated, and it is verified that the persistence is not affected by them.

By carrying out the tests described in section **5**, it is ensured that all Use Cases have been verified, and all bad behavior on the user's end can be handled reliably. Therefore, assuming all tests succeed, it can be confided that all user requirements are functioning properly. As the project continues, new System Tests may be implemented to reflect changes in the user requirements.

9 Test Coverage

The goal of the System Test Plan described in sectio **7** is to achieve approximately 100% branch coverage. Statement coverage is likely to be very high with the compendium of the Integration and Unit tests. Given the black-box nature of system testing, the focus should be more on the fact that all branches of logic are tested rather than all statements individually.

Such test coverage statistics will be achieved by designing a state diagram of the system, and ensuring that each state, as well as each state transition, is covered by the tests in section **5**.

10 Detailed Descriptions of Two Test Cases

The following section will outline two system test cases in such great detail that they could be implemented without the authors' assistance. Due to the omnicapable power of the desktop application, the following two test cases will be described for testing on the desktop.

10.1 The Publish Job Posting Test Case

Setting Up

1. Remove prior XML files. If the system is being run from Eclipse, remove all files in the `output/` directory.
2. Open the main menu
3. Select "Publish Job Posting"
4. **Verify that the Publish Job Posting Window does not appear, but rather an error message saying that no instructors or courses are available is displayed.**

Creating Instructor Profiles

1. Open the main menu
2. Select "Register Profile"
3. Select the Instructor radio button
4. In the First Name field, type "Jim"
5. In the Last Name field, type "Lahey"
6. In the Username field, type "hwn1977"
7. In the Password field, type "pswd"
8. Click "Submit". A confirmation message saying "Instructor hwn1977 was created" should appear, and all fields are reset to blank.

9. Now, for First Name, Last Name, Username, and Password, enter "Randy", "Bobandy", "cheeseburger","pswd2" respectively. Finally, press submit and close the Register Profile window.

10. From the main menu, select "Publish Job Posting".

11. **Verify that the Publish Job Posting Window does not appear, but rather an error message saying that no courses are available is displayed.**

Creating the Courses

1. Open the main menu

2. Select "Create Course"

3. The names "Jim Lahey" and "Randy Bobandy" should now be seen in the list view at the top. Select Jim Lahey.

4. In the Course Name field, type "FACC100 - Trailer Park Supervision"

5. In the CDN field, type "1"

6. In the Grader Time Budget field, enter "19000.0"

7. In the TA Time budget field, enter "14000.0"

8. Click "Create Course". All data in the fields will be cleared.

9. With Jim Lahey still selected, enter the values "ECSE322 - Fixing Lawnmowers", "2", "20000.0,", "15000.0" into the course name, CDN, grader budget, and TA budget fields, respectively. Click submit.

10. Next, select Randy Bobandy from the list at the top of the page.

11. Enter the values "ECSE211 - Dealing with Abuse From an Old Drunk Man", "3", "13","8.75" into the Course Name, CDN, grader budget, and TA budget fields, respectively.

12. Click submit and close the Create Course menu.

At this stage, enough data has been created to test the Publish Job Posting Use Case. The XML files for profiles and courses should already be in the appropriate location.

Publishing a Job Posting

1. Open the main menu, and select "Publish Job Posting"

2. The names Jim Lahey and Randy Bobandy should be in the top-most list view. Select Jim Lahey.

3. **Verify that "FACC100 - Trailer Park Supervision" and "ECSE322 - Fixing Lawnmowers" (and no other courses) appear in the list beneath the instructor names.**

4. Select FACC100 - Trailer Park Supervision. Click "Publish Job".

5. **Verify that an appropriate error message is displayed.**

6. **Verify that, if there is already a `applications.xml` file, there is no mention of an application for Job with the current course name.**

7. Enter "10.5" in the salary field, and "Associate Trailer Park Supervision Experience" in the requirements field. Then, set the end time back by two hours so it is before the start time. Click "Publish Job".

8. **Verify that an appropriate error message is displayed.**

9. **Verify that, if there is already a `applications.xml` file, there is no mention of an Job for the course chosen.**

10. Adjust the end time to be after the start time. Press "Publish Job".

11. **Verify that the `applications.xml` file contains the data for a Job for this chosen course. If other Jobs were posted prior to this one, verify that their corresponding data remains in the XML file.**

12. Chose Randy Bobandy from the top-most list. **Verify that ECSE211 - Dealing with Abuse From an Old Drunk Man is the only option in the list below.**

13. Select the ECSE211 option. Click "Publish Job".

14. Repeat steps 6-11.

10.2 Applying for a Job

Setting up

1. Remove prior XML files. If the system is being run from Eclipse, remove all files in the `output/` directory.

2. Open the main menu

3. Select "Create Job Application"

4. **Verify that the Job Application window does not appear, but rather an error message saying that no jobs or students are available is displayed.**

5. Repeat the *Creating Instructor Profiles* and *Creating Courses* steps from section **8.1**.

6. Repeat steps 2-3.

7. Repeat the *Publishing a Job Posting* steps from section **8.1**, and repeat steps 2-3.

Creating Students

1. Open the main menu. Select "Register Profile"
2. Select the Student radio button.
3. Enter the values "Cosmo", "Kramer", "kman", "pennypacker" into the First Name, Last Name, Username, and Password fields, respectively.
4. Enter "Golfing" in the Skills textbox.
5. Click "Submit". A message saying "Student kman created" should be seen at the top of the window, and all fields should be reset.
6. Enter "George", "Costanza", "vandelay","bosco" into the First Name, Last Name, Username, and Password fields, respectively.
7. Enter "Baseball" in the Skills textbox.
8. Click "Submit". A message saying "Student vandelay created" should be seen at the top of the window. Close the window.

Creating A Job Application

1. From the main menu, select "Create a Job Application". The Job Application window should finally instantiate.
2. Two lists should be present, one containing the student names, and the other containing the job names. In future versions of the system, students will log in and will not need to select their name from a list. For this version, select any student from the list.
3. Without selecting a Job from the list, click "Apply".
4. **Verify that an error message saying that no Job was selected is present, and ensure that the `applications.xml` file has not been modified.**
5. Select a Job from the list. Click "Apply". A friendly message wishing the given student luck for his application to the selected Job will appear.
6. **Verify that the `applications.xml` file has included the job application.**
7. Repeat steps 5 and 6 with different combinations of students and Jobs.
8. **Verify that no data is lost in the `applications.xml` file when multiple applications are created. That is to say, make sure the file has persisted all applications and all courses after multiple applications were submitted.**