Abdullahi Shidane   180104174

CS3270:

Testing and Reliable Software Engineering

Test Strategy

1.1 Why Testing the Admission System is important?

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is defect free.

Testing the university admission system is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the admission system. A properly tested university admission system ensures reliability, security, and high performance which further results in time saving, cost effectiveness and user satisfaction.

Testing the admission system ensures quality software is delivered that meets the requirements. For instance, one requirement of this system is that applicants must be able enter grades that have been achieved. This requirement could be tested with erroneous data to make sure that the systems only except valid grades. Not testing the system would mean the system might accept grades that do not exist such as ‘K’, which would cause a ‘failure’ meaning the requirements have not been met, and the system is not reliable. Moreover, Testing will make sure the admission system is secure, meaning students or anyone’s personal data held in the system would be safe from unauthorised access. Not testing will put individuals personal and sensitive data at risk.

One specific key technique is Boundary value analysis, this is a black box testing technique where tests are intended to include boundary values in a range. Reliability testing is also key as this ensures the reliability of the application; this is discussed more into depth under the ‘Reliability Testing’ header.

1.2 Why do we test and who is involved?

Although testing can take place throughout the duration of the creation of the admission system, in Software Development Life Cycle (SDLC), testing can be started from the requirements gathering phase and continued till the deployment of the software. Agile is iterative and incremental. This means that the testers test each increment of coding as soon as it is finished. An iteration might be as short as one week, or if a month. The team builds and tests a little bit of code, making sure it works correctly, and then moves on to the next piece that needs to be built. Programmers never get ahead of the testers because a story is not “done” until it has been tested, in this testing methodology the tester is involved with each iteration/ phase. (Crispin, L. & Gregory, J. (2009)). Figure 1 illustrates the iterative model of testing.

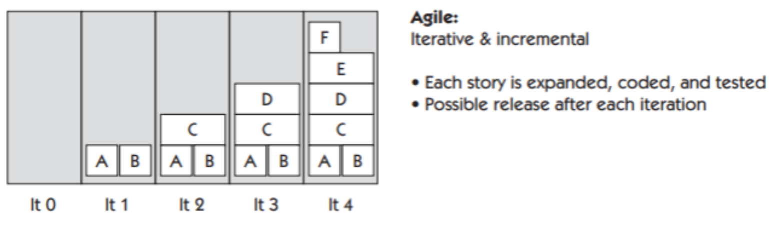
On the other hand, the testing phase happens after the creation of the full product is completed. Figure 2 illustrates where testing is conducted in the Waterfall methodology.

A planned User acceptance testing (UAT) is normally the final phase of the software testing process. Where White Box testing is Typically performed during the coding stage of a project.

The tester of a system should ideally be involved from the start of a project as the tester would normally decide whether a requirement is testable and measurable. By having a tester involved from the start of a project it could save money, time and resources as the tester would say whether a requirement is worth pursuing or whether it is a workable requirement.

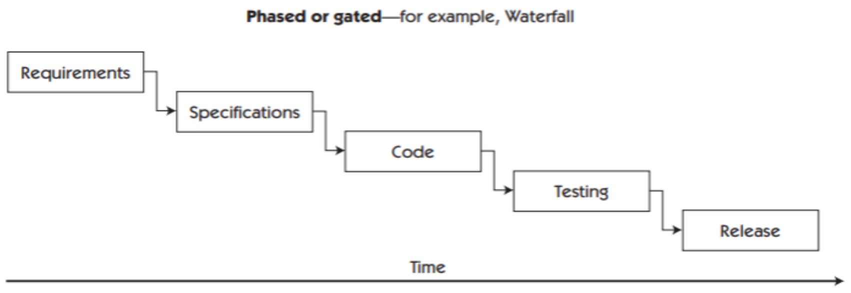
Different testing techniques are conducted by different types of people, UAT/ usability testing is normally conducted by potential users of the system, or various participants. On the other hand, white box testing techniques are conducted by developers, as this test design technique which considers the internal structure of a system or component. Moreover, Black box testing is conducted by testing professionals.

Figure 1 (Crispin, L. & Gregory, J. (2009)).



Agile:
Iterative & incremental
• Each story is expanded, coded, and tested
• Possible release after each iteration
A
B
C
D
E
F
It 0
It 1
It 2
It 3
It 4

Figure 2 (Crispin, L. & Gregory, J. (2009)).



1.3 How can development aid testing?

There are numerous ways on how development can aid testing. The most important concept is writing code that is “testable”.

Other concepts that can aid testing include:

Separation of concerns.

This refers to the separation of the admission systems code into distinct segments with least overlap in functionality as conceivable. The most essential factor is to minimise the interaction points to achieve high cohesion and low coupling.

Single Responsibility principle.

This implies each component or module must control only a specific section or functionality related to the components.

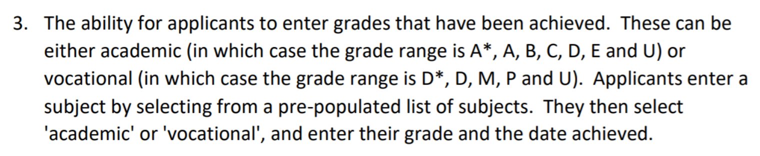
Principle of Least Knowledge.

Components in the Admission system must not know about the inner specifics of other components of the admission system. This will help testing the admission system by providing better information flow.

2 Black Box Testing And BDD

In this section of the report there is a discussion of black box testing strategies, supported by BDD scenarios and associated methods in Java programming language.

Feature being tested:



3. The ability for applicants to enter grades that have been achieved. These can be either academic (in which case the grade range is A*, A, B, C, D, E and U) or vocational (in which case the grade range is D*, D, M, P and U). Applicants enter a subject by selecting from a pre-populated list of subjects. They then select 'academic' or 'vocational', and enter their grade and the date achieved.

Black Box testing involves testing the software for functionality, it is used to find out the errors in data structure, faulty functions, interface errors, etc. Black box testing ignores internal mechanism of a system (Mumtaz Ahmad Khan 2011). Black box testing is also known as functional testing.

2.1 Boundary Value Analysis

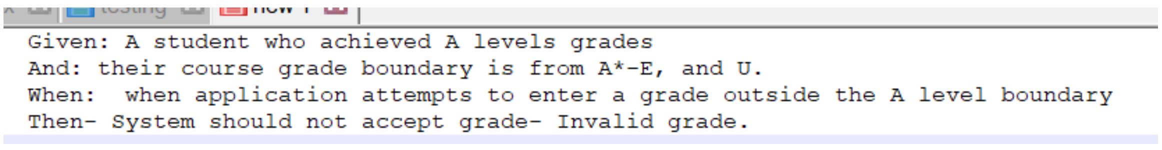
One black box testing technique is boundary value analysis. This is a technique that deems boundary values for inputs /outputs. Using BVA is useful to test this feature because BVA considers the boundaries, meaning it will check maximum and minimum grades that can be entered on the admission system. Using this technique will enable the admission system to become more reliable as boundaries commonly cause defects in many systems.

Worst case BVA is when we make many fault assumptions, for instance the values validity can alter based on other values, and we test with valid values only. in Worst case BVA we can make multiple false assumptions, meaning that it assumes dependency between variables. In this technique we may only test with valid data, however, we test with all combinations we would have selected with simple BVA. It would be beneficial to use worst case BVA as these tests are considered more comprehensive.

2.2 Behaviour-Driven Development

BDD scenarios will essentially be written descriptions of the admissions systems behaviour from the user's perspectives. Figure 3 illustrates a BDD scenario that is essential when testing this feature.

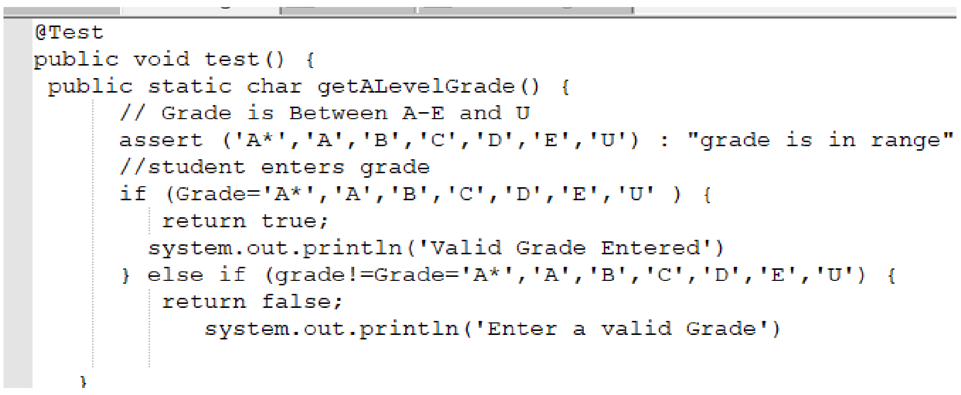
Figure 3



```
Given: A student who achieved A levels grades
And: their course grade boundary is from A*-E, and U.
When:  when application attempts to enter a grade outside the A level boundary
Then- System should not accept grade- Invalid grade.
```

The BDD scenarios were created to cut down on interpretation costs while still making requirements easy to understand. It breaks down requirements into different segments. This BDD scenario is needed to ensure the system only accepts valid inputs, i.e., the system should not accept a grade 'H' as this is out of the boundary. The reasons this test was written is to ensure the integrity and reliability of the admission system.

Figure 4



```
@Test
public void test() {
 public static char getALevelGrade() {
      // Grade is Between A-E and U
      assert ('A*','A','B','C','D','E','U') : "grade is in range"
      //student enters grade
      if (Grade='A*','A','B','C','D','E','U' ) {
         return true;
        system.out.println('Valid Grade Entered')
      } else if (grade!=Grade='A*','A','B','C','D','E','U') {
         return false;
            system.out.println('Enter a valid Grade')

   }
```

The image on figure 4 illustrates the java method that would illustrate the BDD scenario discussed.

Appendix 1 illustrates a complete set of test cases for worst-case BVA for this feature. The test cases test the admission systems grade boundaries, to ensure that invalid grades are not accepted. Firstly, valid grades are entered, for instance 'A', the expected outcome is that the grade is 'Valid'. Moreover, the expected outcome when a grade (L) that is not within the boundary is entered, is that the system should not accept this input. the table ensures that the system only expects valid data and no erroneous values.

3 White Box Testing

In this section of the report white box testing methodologies will be discussed, more specifically: Condition coverage and Test cases associated with the feature being tested.

White-box testing refers to test methods that rely on the internal structure of the software. White-box methods are based on executing or “covering” specific elements of the code. Ostrand, T. (2002). Meaning, this type of testing mainly focuses on the code rather than the general requirements. The code also informs the test cases.

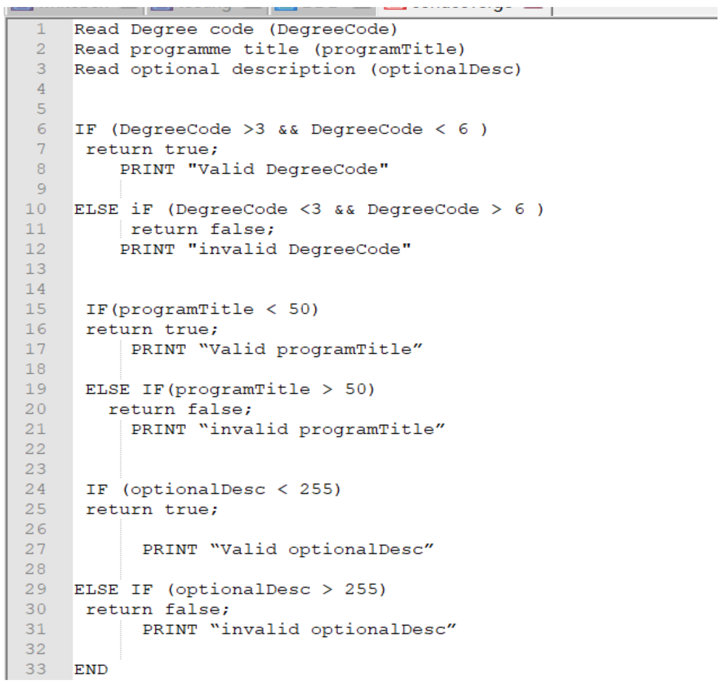
The Feature being tested using this technique:

1. The ability for admissions staff to create, edit and delete degree programme information. The information required is the degree code (3-6 characters), programme title (up to 50 characters) and an optional description (up to 255 characters).

Condition coverage (which is a white box testing technique), is appropriate for testing this feature because it will examine individual results for each logical condition. It will find out whether a condition is true or false. Moreover, condition coverage provides improved sensitivity to the control flow, compared to other methods like than decision coverage. Decision coverage disregards branches within a Boolean expression, which is a disadvantage. Using condition coverage, the smallest number of tests is n+1.

Figure 5 illustrates the pseudocode for condition coverage, the code will test degree code boundaries, program title boundaries, as well as the optional description. The pseudocode below will ensure that the degree information is only created when the conditions identified are met.

Figure 5



```
Read Degree code (DegreeCode)
Read programme title (programTitle)
Read optional description (optionalDesc)


IF (DegreeCode >3 && DegreeCode < 6 )
 return true;
    PRINT "Valid DegreeCode"

ELSE iF (DegreeCode <3 && DegreeCode > 6 )
     return false;
    PRINT "invalid DegreeCode"


 IF(programTitle < 50)
 return true;
     PRINT “Valid programTitle”

 ELSE IF(programTitle > 50)
   return false;
     PRINT “invalid programTitle”


 IF (optionalDesc < 255)
 return true;

      PRINT “Valid optionalDesc”

ELSE IF (optionalDesc > 255)
 return false;
      PRINT “invalid optionalDesc”

END
```

The test cases identified using this strategy can be found in appendix 2. In the table there 8 test cases identified, whereby each test case has an outcome of true or false. These determine the message that is printed, or essentially the determining factor whether an outcome is valid or invalid. This table uses a series of true and false outcomes, there needs to be two tests in each category (etc DegreeCode, programTitle, or optionalDesc) to display all the valid and invalid outcomes. There are 8 different possible combinations needed to create a valid degree programme information. The condition coverage table illustrates the conditions that need to be men to achieve a valid 'degree information'.

The test cases identified are based on the condition coverage pseudo code form figure 4. The purpose of the test cases on appendix 2 is to ensure all conditions/ requirements of the degree information is valid. For example, a programme title must up to 50 characters. The test cases ensure all tests are necessary are conducted and also further minimise gaps in coverage.

4 Quadrant 3

In this section of the report Scenario testing and Exploratory testing will be discussed.

The feature being tested in quadrant 3 is illustrated below. This feature will be tested in both quadrant 3 techniques: Scenario testing and Exploratory testing.

7. The ability for an applicant to enter personal statements of up to 3,000 characters after all grades have been verified or predicted. An applicant will create between one and six personal statements. They then select up to six institutions to which they wish to apply, select a personal statement to attach, then submit the application.

4.1 Scenario testing

Scenario Testing is a method in which actual scenarios are used for testing the software application instead of test case. A scenario is a hypothetical story, used to help a person think through a complex problem or system (Kraner, 2003). In this segment of the report, we will determine how scenario testing is used to test this feature of the admission system.

Scenario testing is conducted by testers, who test the system as if they were the end users. The testers contact participants or possible clients to discuss possible test scenarios.

A scenario is a hypothetical story, utilised to help testers test the system. (Kraner, 2003) described essential characteristics of a scenario, Kraner's article stated that the 5 major characteristics of a scenario is: "A scenario test has five key characteristics. It is (a) a story that is (b) motivating, (c) credible, (d) complex, and (e) easy to evaluate".

The list below illustrates a scenario for this feature:

1. grades are verified.
2. A student wants to submit a personal statement.
3. A student writes a personal statement up to 3000.
4. system checks word limits.
5. An applicant will create between one and six personal statements.
6. System checks whether their personal statement is <6 and >1.
7. Students then select up to six institutions to which they wish to apply-
8. system checks that students can only apply to 6 institutions.
9. students submit personal statements to up to six institutions to which they wish to apply, select a personal statement to attach, then submit the application.

According to (Kraner's, 2003), we also may need to identify all potential users of the system.

- Students submitting personal statements.

After using scenario testing, we need to consider disfavoured users and a list of all events.

- Personal statement is submitted.
- Applicant selects up to 6 institutions.
- System checks whether their personal statement is <6 and >1.

Risks of Scenario testing

Scenario testing can be a vigorous process and include many features. One drawback of this technique is that if the first feature in the scenario doesn't function, then the rest might not be able to run.

Another drawback of using this technique is that it sometimes requires a lot of documentation which makes this technique time consuming. Moreover, this testing technique is not intended for testing at early stages of developments nor to provide coverage of the system.

4.2 Exploratory testing

According to James Bach (2003), "exploratory testing is any testing to the extent that the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests". This implies testing is simultaneous learning which drives towards faster results. (guru99.com)

This testing technique is conducted by a software tester. This software tester will normally work closely to the developer of the system.

When using this technique to test the feature, there are several testing considerations that need to be considered. These include realistic inputs, the system states, code paths, user data and execution environment. In exploratory testing, we want to explore actions that could break the admission system and that are critical to our system. Exploratory testing is based on how the system is expected to behave. Below is a list of exploratory testing scenarios:

1. System should not allow users to enter more than 3000 – personal statements.
2. System should not all more than 6 personal statements
3. System me should not allow students to apply to more than 6 institutions.

Risks of Exploratory testing

A risk of using this technique to test the admission system is that it may be difficult to reproduce a defect again.

5 Quadrant 4

Quadrant 4 focus is on non-functional requirements, it is a technique that is technology facing with an objective of critiquing a product rather than supporting a team, for instance how the admission system will perform under a certain load.

In this section of the report Performance testing and Reliability testing will be discussed.

The feature being tested in quadrant 4 is illustrated below. This feature will be tested in both quadrant 4 techniques: Performance testing and Reliability testing.

1. The ability for admissions staff to create, edit and delete degree programme information. The information required is the degree code (3-6 characters), programme title (up to 50 characters) and an optional description (up to 255 characters).

5.1 Performance testing

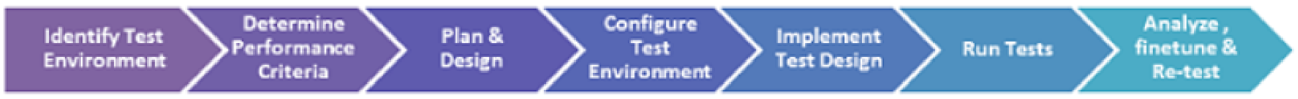
The main aim of Performance Testing is not to just discover some defects but to reduce possible performance bottlenecks and to set in place a baseline for subsequent performance testing, these factors generally refer to non-functional requirements, which this technique aims to test. Using Performance testing to test this feature will improve the admissions systems UX.

This type of testing is conducted by Test engineers or developers. Test engineers measure the quality of an application under different circumstances.

Performance tests will be conducted when the development of the admission system is completed. Tests are conducted by testers to ensure the admission system there are no performance-related bottlenecks, the performance testing process has 7 crucial stages as illustrated below on Figure 6.

Performance Testing Process

Figure 6: performance testing process


Identify Test Environment
Determine Performance Criteria
Plan & Design
Configure Test Environment
Implement Test Design
Run Tests
Analyze , finetune & Re-test

Performance tests to tests this specific feature of the admission system could be:

- The ability of multiple staff to create degree information at the same time.
- How many staff can use the system at a time?
- Can staff create degree information on any device.
- What is the response time of the admission system – when degree information is created, how long will it take for it to be uploaded.
- Data verification.

Risks of Performance testing

Risks of using this technique include network latency. Other drawbacks are there is no way to make behaviours and actions perfectly mimic the human actions.

5.2 Reliability Testing

Reliability refers to the chance that the system is operational without any failures in a particular environment at a given period. It is important for the university admission system to work reliably and accurately. Using this technique to test the University admission system, will ensure that the system is reliable and accurate. The ‘Failure Intensity Objective (FIO)’ can be used to measure the rate of failure over a period, this can be used as an indicator to find out how reliable the admission system is. This testing technique is usually performed by software testers.

Specifically for the feature above, we can test how reliable the ‘create’, ‘edit’, and ‘delete’ functionalities are. Let's focus on the delete functionality, here we can use FIO to measure how many failures can occur after a delete request. Similarly, we can also look at how many failures can occur when there is an ‘create degree information request’. Moreover, using these techniques, the mean time between failures can also be analysed, to gain a better understanding of how often failures occur within the admission system.

Risks of Reliability Testing

However, a drawback from using this technique is that there needs to be an acceptable rate of failure, for instance we need to decide whether 2 out 12 failure is acceptable or not. If an acceptable rate is not set, then we might not know whether the system is reliable or not.

References

Crispin, L. & Gregory, J. (2009) Agile testing: a practical guide for testers and agile teams. Boston: Addison-Wesley.

Mumtaz Ahmad Khan, Mohd. Sadiq, (2011) Analysis of Black Box Software Testing Techniques: A Case Study, Section of Electrical Engineering, University Polytechnic, Faculty of Engineering and Technology,

Kaner (2003), An introduction to Scenario Testing, Florida Institute of Technology, April 2013, June 2003.

James Bach (2003), Exploratory Testing Explained.

https://www.guru99.com/exploratory-testing.html

Ostrand, T. (2002). White-Box Testing. In Encyclopaedia of Software Engineering, J.J. Marciniak (Ed.). https://doi.org/10.1002/0471028959.sof378

Appendices

Appendix 1: Worst Case BVA

Test cases	Grade entered.	Expected Output
1	A	“Valid Grade Entered”.
2	L	“Invalid grade Entered”.
3	J	“Invalid grade Entered”.
4	C	“Valid Grade Entered”.
5	L	“Invalid Grade Entered”.
6	B	“Valid Grade Entered”.
7	D	“Valid Grade Entered”.
8	E	“Valid Grade Entered”.
9	K	““Invalid Grade Entered”.
10	U	“Valid Grade Entered”.
11	A*	“Valid Grade Entered”.

12	H	“Invalid Grade Entered”.

Appendix 2

F= false

T= true

Test Case	**Condition: (DegreeCode >3 && DegreeCode < 6)**	**Condition: programTitle < 50**	**Condition: optionalDesc < 255**	**Outcome**
1	T	T	T	PRINT: Valid DegreeCode Valid programTitle Valid optionalDesc
2	T	F	T	PRINT: Valid DegreeCode Invalid programTitle Valid optionalDesc
	T	T	F	PRINT: Valid DegreeCode Valid programTitle Invalid optionalDesc
	T	F	F	PRINT: Valid DegreeCode invalid programTitle invalid optionalDesc
	F	T	T	PRINT: invalid DegreeCode Valid programTitle Valid optionalDesc
	F	F	T	PRINT: Valid DegreeCode Valid programTitle Valid optionalDesc
	F	T	F	PRINT: invalid DegreeCode Valid programTitle invalid optionalDesc
	F	F	F	PRINT: invalid DegreeCode invalid programTitle invalid optionalDesc