Module 14 Software Development 2

Module title	Software Development 2
Module NFQ level (only if an NFQ level can be demonstrated)	6
Module number/reference	BSCH-SD2
Parent programme(s)	Bachelor of Science (Honours) in Computing Science
Stage of parent programme	Stage 2
Semester (semester1/semester2 if applicable)	Semester 2
Module credit units (FET/HET/ECTS)	ECTS
Module credit number of units	10
List the teaching and learning modes	Direct, Blended
Entry requirements (statement of knowledge, skill and competence)	Learners must have achieved programme entry requirements.
Pre-requisite module titles	BSCH-SD1
Co-requisite module titles	None
Is this a capstone module? (Yes or No)	No
Specification of the qualifications (academic, pedagogical and professional/occupational) and experience required of staff (staff includes workplace personnel who are responsible for learners such as apprentices, trainees and learners in clinical placements)	Qualified to as least a Bachelor of Science (Honours) level in Computer Science or equivalent and with a Certificate in Training and Education (30 ECTS at level 9 on the NFQ) or equivalent.
Maximum number of learners per centre (or instance of the module)	60
Duration of the module	One Academic Semesters, 12weeks teaching
Average (over the duration of the module) of the contact hours per week	2.5
Module-specific physical resources and support required per centre (or instance of the module)	One class room with capacity for 60 learners along with one computer lab with capacity for 25 learners for each group of 25 learners

Analysis of required learning effort		
	Minimum ratio teacher / learner	**Hours**
Effort while in contact with staff		
Classroom and demonstrations	1:60	8
Monitoring and small-group teaching	1:25	12
Other (specify) Hackathon	1:60	10
Independent Learning		
Directed e-learning		
Independent Learning		120
Other hours (worksheets and assignments)		100
Work-based learning – learning effort		
Total Effort		250

Allocation of marks (within the module)					
	Continuous assessment	**Supervised project**	**Proctored practical examination**	**Proctored written examination**	**Total**
Percentage contribution	20%	80%			100%

Module aims and objectives

The Software Development 2 module builds on the work completed in Software Development 1. In that module the focus was on source code management using version control systems. In this module the focus is again on developing a large piece of work, using the learning from Software Development 1, but with a focus on software testing and testing suites. They get the opportunity to work on a large-scale project in a team dynamic. They are required to complete requirements analysis, produce complete a software application, host said software on a code repository, implement a testing suite, and to document the process.

They not only learn new technical skills such as software testing and requirements analysis but also work as part of a team to develop a software product.

Teaching in this module is conducted mainly between the team of learners and the lecturer. However, in the early stages of the process the faculty organise a number of relevant seminars. Topics for these will outline how to preform requirement analysis for the project, and how to systematically test the project to assure it is performing to specification.

The skills that the learners develop in this module benefit them as they progress through their degree and into their professional life.

Minimum intended module learning outcomes

On successful completion of this module, the learner will be able to:

1. Install, configure and utilize a testing framework for a software project
2. Use technical design and implementation skills
3. Produce comprehensive reports from software testing to feed into future feature cycle
4. Write coherently and present information in a systematic manner to the required academic level
5. Undertake a technical project and bring it to completion implementing a rigorous testing framework
6. Document the project life-cycle from specification to implementation

Rationale for inclusion of the module in the programme and its contribution to the overall MIPLOs

The module is the designed to expose the learners to a larger scale project than they have experienced so far, it accumulates the skill and knowledge that the learner has developed over the previous semester and combines that with a degree of independent learning to enable learners to specify, design, and build a system that accurately reflects a 2nd year standard of work. Appendix 1 of the programme document maps MIPLOs to the modules through which they are delivered.

Information provided to learners about the module

Learners receive a programme handbook to include module descriptor, module learning outcomes (MIMLO), class plan, assignment briefs, assessment strategy and reading materials.

Module content, organisation and structure

Software Testing

Introduction

- What is Testing?
- Unit Testing
- Integration testing
- Continuous testing
- Regression testing

Unit Testing

- Installing and configuring a testing framework

Integration Testing

- Commit early, commit often

Continuous Testing
Regression Testing
Project Specification
Project Timeline
A series of 4 two-hour seminars are held over the first 4 weeks of semester 1, where the usage of code management software is explained and demonstrated. In week 5, the Learners will take part in a hackathon event to create a basic prototype for their project. The learners are then given another week to review and refine their team's idea before it is approved by the faculty. The remaining time is dedicated to bringing their project to competition. In the final week of the semester, the teams will present their work to the faculty. During the project work period the teams with be required to create an initial iteration of the system, and based on milestone reviews perform two subsequent sets of bug reports and iterations.

Module teaching and learning (including formative assessment) strategy
The module is taught as a combination of seminars sessions and team meetings between the lecturer and each team of learner. The seminar sessions discuss and explain to learners the principles and challenges involved in correctly using code management software.

Assessment is split into 5 elements.

- Worksheets on testing (20%)
- 3 iterations of the project (20%)
- 3 test reports after each review (20%)
- 3 milestone reviews (20%)
- Project Documentation (20%)

Timetabling, learner effort and credit
The module is timetabled as four 2 hour lectures and a series of meetings with lecturer.

There are 28 contact hours made up of 4 lectures delivered over the first 4 weeks with classes taking place in a classroom and 7 team meetings held over the last 7 weeks of the semester. There is an 8-hour hackathon held in week 5. Between week 7 and week 12 there will be a weekly 2-hour meeting time taking place in a project room. The learner will need 97 hours of independent effort to further develop the project that is proposed.

The team believes that 125 hours of learner effort are required by learners to achieve the MIMLOs and justify the award of 5 ECTS credits at this stage of the programme.

Work-based learning and practice-placement
There is no work based learning or practice placement involved in the module.

E-learning
The college VLE is used to disseminate notes, advice, and online resources to support the learners. The learners are also given access to Lynda.com as a resource for reference.

Module physical resource requirements
Requirements are for a classroom for 60 learners equipped with a projector, and a work area / project lab to hold regular meetings.

Reading lists and other information resources

Reading lists and other information resources
Beck, K. (2014) *Test-driven Development by Example*. Boston: Addison-Wesley.

Secondary Reading
Khan, R. and Das, A. (2018) *Build Better Chatbots*. Berkeley: Apress.

Shevat, A. (2017) *Designing Bots: Creating Conversational Experiences*. Boston: O'Reilly.

Whitehead, R. (2001) *Leading a Software Development Team*. London: Addison Wesley.

https://developer.amazon.com [3]

Specifications for module staffing requirements
For each instance of the module, one lecturer qualified to at least Bachelor of Science (Honours) in Computer Science or equivalent, and with a Certificate in Training and Education (30 ECTS at level 9 on the NFQ) or equivalent.. Industry experience would be a benefit but is not a requirement.

Learners also benefit from the support of the programme director, programme administrator, learner representative and the Student Union and Counselling Service.

[3] Last accessed 24/07/2018

Module Assessment Strategy

The assignments constitute the overall grade achieved, and are based on each individual learner's work. The continuous assessments provide for ongoing feedback to the learner and relates to the module curriculum. The assessment of this module is a combination of individual and group based assessment.

- Worksheets on source control (20%)
- 3 iterations of the project (20%)
- 3 test reports after each review (20%)
- 3 milestone reviews (20%)
- Project Documentation (20%)

No.	Description	MIMLOs	Weighting
1	Worksheets on source control; the learner submits a series of worksheets to demonstrate knowledge in software testing. Individual assessment	1,2,3	20%
2	Initial iteration project; learners develop an initial prototype to test for future review. Group assessment	1,2,3,4,6	20%
3	Second iteration project; based on first review learner's product a testing report and update project features for future review. Group assessment.	1,2,3,4,6	20%
4	Final iteration project; based on first review learner's product a testing report and update project features for final demonstration. Group assessment.	1-6	20%
5	Project Documentation; Learners submit a comprehensive document that outlines the research taken for this project, and documents the implementation and testing process. Group assessment.	2,4,6	20%

All repeat work is capped at 40%.

Sample assessment materials

Note: All assignment briefs are subject to change in order to maintain current content.

Testing Worksheets
Worksheet 1 Introduction to Testing

Introduction:
In this worksheet you will be introduced to writing unit tests for the first time. Here you will practice how to write tests, run them, and write code to satisfy the tests. In the following worksheet we will integrate this into an SCM (Source Code Manager) to show the full process of how Test Driven Development works. It is deliberately structured such that you write your tests first followed by the code that should satisfy them. This is the standard way that testing works.
A well written test will show that the person writing the test understands how the method or item to be tested functions. If the test is correct then the implementation of the method or item should satisfy the test. The test should not be adjusted to satisfy the code.
For assessment you will be required to submit both your code and unit tests.
Tasks:
01) Create a JUnit test file called "MyMathTest.java" and a java file "MyMath.java". In your JUnit test add the following four stub methods

- testAdd()
- testSubtract()
- testMultiply()
- testDivide()

In the regular java file add in the following stub methods

- int MyAdd(int a, int b)
- int MySubtract(int a, int b)
- int MyMultiply(int a, int b)
- int MyDivide(int a, int b)

02) Write five test cases for each of the unit test methods that call the appropriate methods in the java file. The unit tests should still fail at this point as you are writing the unit test first
03) Write the four methods in MyMath.java you may only move onto the next implementation once each unit test has been satisfied.
04) Create a JUnit test file called "GeometryTest.java" and a java file "Geometry.java". in the JUnit test add the following four stub methods

- testAreaRect()
- testPerimeterRect()
- testVolumeSphere()
- testSurfaceAreaSphere()

In the regular java file add the following stub methods

- float areaRect(float width, float height)

- float perimeterRect(float width, float height)
- float volumeSphere(float radius)
- float surfaceAreaSphere(float radius)

05) Write four test cases for each test method that calls the appropriate method in the java file. You should not write code in the regular java file for now. As you are dealing with floating point numbers. Your margin for error (epsilon) is 0.1
06) Write the code for the methods in Geometry.java You may only move onto the next method once the unit test has been satisfied.

Worksheet 2 Introduction to Test Driven Development with Git

Introduction:
In this worksheet you will be introduced to how a Source Code Manager should be used and interacted with in combination with Test Driven Development. Combining the concept of Commit Early, Commit Often you will be required to write a unit test first then write the code that satisfies said unit test later. Once both are complete the unit test and code should be committed and pushed to the repository.
For assessment you will be required to submit a link to a gitlab project (college only gitlab) containing all of your work. Note that it is an absolute requirement of the worksheet that you commit after each completion of a unit test and implementation. You will lose marks if your repository does not reflect this.
All of the methods that you must implement below must use floating point values as input and output. Your epsilon for each method is 0.1. Each unit test must perform at least three tests on the method involved.
Tasks:
01) Create a fresh gitlab repository. Make sure to grant the lecturer access to the repository.
02) Create two files “Geometry3DTest.java” and “Geometry3D.java” and commit them to the repository.
03) Add a unit test for calculating the volume of a cuboid and add the appropriate implementation.
04) Add a unit test for calculating the surface area of a cuboid and add the appropriate implementation.
05) Add a unit test for calculating the volume of a square based pyramid and add the appropriate implementation.
06) Add a unit test for calculating the surface area of a square based pyramid and add the appropriate implementation.
07) Add a unit test for calculating the volume of a tetrahedron and add the appropriate implementation
08) Add a unit test for calculating the surface area of a tetrahedron and add the appropriate implementation.

Worksheet 3: Introduction to Integration testing

Introduction:

So far you've been introduced to Unit testing where testing has taken place on individual classes and individual methods. The next step after Unit testing is to perform integration testing where classes and modules are aggregated together to see if integration works as designed. For the purposes of this testing we will take the code written in the previous worksheet and turn it into an inheritance structure that is polymorphic. You will have a generic Shape class (abstract in nature) and the subclasses Cuboid, SquareBasedPyramid, and Tetrahedron. The shape class will declare abstract volume(), surfaceArea() and description() methods that return floats but nothing more. description() should return a string saying what this shape is e.g. "Cuboid", "Tetrahedron" etc.

You are required to write unit tests for the three subclasses and then write an integration test that tests the polymorphic functionality of the Shape class.

For assessment you are required to submit all of your written code. As before your epsilon for your floating point tests is 0.1 and you are required to write three tests for each unit test unless otherwise specified.

Tasks:

01) Create a Shape class and a ShapeTest unit test. For now write the abstract method stubs for the Shape class. You will not be able to write a unit test for the Shape class until you have at least one subclass to work with.

02) Create a Cuboid class that subclasses Shape and a CuboidTest unit test. Write the tests and code for each method. Do not move onto the next method until each unit test is satisfied.

03) Create a SquareBasedPyramid class that subclasses Shape and a SquareBasedPyramidTest unit test. Write the tests and code for each method. Do not move onto the next method until each unit test is satisfied.

04) Create a Tetrahedron class that subclasses Shape and a TetrahedronTest unit test. Write the tests and code for each method. Do not move onto the next method until each unit test is satisfied.

05) In ShapeTest write a single integration test that generates an array of 6 Shape objects. There should be two of each type of shape in the array in any permutation. Using the abstract methods alone (no introspection allowed) check that the right volume, surface area, and description is returned.

Project Specification

Based on your knowledge of chatbots and Amazon Alexa skills from the hackathon, develop an Alexa skill that interacts with a custom made chatbot. The goal of your chatbot is to plan your clothing requirements for a trip that will visit 5 locations in 3 days. The bot should be able to review the weather for each of these locations and suggest appropriate clothing for each location for the day that you will visit.

Your team must indicate which team member wrote each method and document its functionality. Each team needs to create a repository for the chatbot and add the lecturer to the project as a member.

There must be evidence of a testing framework for each feature of the chatbot.

There must be evidence that all members have both pulled code from the repository and committed changes to the project at each milestone review.

Milestone reviews
The work period between each milestone is 2 weeks. At the end of the period the team will present their code, pull / commit logs, and a short report to the lecturer for a code interview. Once the interviews have been completed, the team will progress onto the next features to implement. Based on each milestone review the team must produce a testing framework for the features in the next section of the lifecycle.

There will be a total of 3 review cycles before a final league event. Each review cycle will follow the same pattern, but the team must clearly indicate what changes have been made since the previous review.