Module 8 Software Development 1

Module title	Software Development 1
Module NFQ level (only if an NFQ level can be demonstrated)	n/a
Module number/reference	BSCH-SD1
Parent programme(s)	Bachelor of Science (Honours) in Computing Science
Stage of parent programme	Stage 1
Semester (semester1/semester2 if applicable)	Semester 2
Module credit units (FET/HET/ECTS)	ECTS
Module credit number of units	5
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	None
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.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 Semester, 12 weeks teaching
Average (over the duration of the module) of the contact hours per week	2
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	12
Monitoring and small-group teaching	1:25	10
Other (specify)(Hackathon)	1:60	8
Independent Learning		
Directed e-learning		
Independent Learning		38
Other hours (worksheets and assignments)		57
Work-based learning – learning effort		
Total Effort		125

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

In the Software Development 1 module the learners complete a large piece of work, encompassing both independent learning and development. They get the opportunity to work on a large-scale project in a team dynamic. They are required to produce complete a software application, host said software on a code repository and to document the process.

They not only learn new technical skills such as code management but also learn how to develop a software product while working as part of a team. This module focusses on code management using version control systems such as Git and GitHub.

Teaching in this module is conducted mainly through 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 the correct usage of code repositories such as GitLab and BitBucket, as well as industry expectations when working with code management software in a team of developers.

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 source control system for a software project
2. Use technical design and implementation skills
3. Apply industry standard etiquette when using source control in a team environment
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
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 1st year standard of work. The Learners are also expected to begin using code management tools in their project and throughout the remaining duration of their programme. 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
Code Management
Introduction
- What is a Version Control System (VCS)?
- What is Git and where did it come from?
- Alternatives to Git
- Cloud-based solutions (Github, Gitlab, BitBucket etc)

GIT
- Installing and configuring Git
- Creating a repository
- Adding content to repository
- Accessing repository remotely

Daily Git use

- Commit early, commit often
- Branch / merge actions
- Push / pull updates

GitLab

- Source control as a project management tool
- Task management / work division
- Bug reporting

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 completion. In the final week of the semester, the teams present their work to the faculty. During the project work period the teams will 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 source control (20%)
- 3 iterations of the project (20%)
- 3 bug 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 workshops, an 8-hour hackathon and a series of meetings with lecturer. The number of 5 ECTS credits assigned to this module is our assessment of the amount of learner effort required.

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 40 hours of independent effort to further develop the skills and knowledge gained through the contact hours. An additional 57 hours are set aside of the learners to work on 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.

Recommended Text
Hethey, J. M. (2013) *GitLab Repository Management*. Birmingham: Packt Publishing

Secondary Reading
Loeliger, J. and McCullough, M. (2012) *Version Control with Git: Powerful tools and techniques for collaborative software development*. 2nd edition. Cambridge: O'Reilly Media.

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

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.

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.

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

Git Worksheets

Worksheet 1 Introduction to Git

Introduction:

As per the lecture notes you will need to setup access to your gitlab account on http://gitlab.griffith.ie/ this requires that you have a fully working learner email address. If you do not have such an address or it is not setup you will need to get that sorted before you can access gitlab. It will not accept any other form of email address. For assessment you will be required to document screenshots of the commands you have performed. I will also require access to your repositories to check that they are working correctly. Granting access is covered in the lecture notes.

Tasks:

01) Create a repository called "Worksheet1" within gitlab. Using the clone command provided use the command line (as shown in lectures) to clone the initially empty repository.

02) Write the Hello World program you covered in your Computer Programming lectures inside your repository. Run the status command from the command line.

03) Add the new file to repository, do a local commit, followed by a push to place the written file on the gitlab server.

04) Modify the hello world program to take in two integers from the command line, add them together and print the results to console. Once this is working do a commit like in 03) above

05) Use the git log and diff commands to show the differences between the first and second commits you have done

06) modify your code again to take in a third integer from the command line and add that to the result. Again perform a similar task as in 03 to commit these changes to the repository.

Worksheet 2 Introduction to branching

Introduction:

In this worksheet you will be introduced to the branching system within git. The main idea behind branching that you will have seen in lectures is that it enables development of experimental features or large architectural changes in a separate space from the mainline or master development. In this worksheet you will cover the basics of making branches, we will cover merging at a later stage as there are some complexities that arise when merging

For assessment you will be required to document screenshots of the commands you have performed. I will also require access to your repositories to check that they are working correctly. Granting access is covered in the lecture notes.

Tasks:

01) Create a repository called “Worksheet2” within gitlab. Using the clone command provided use the command line (as shown in lectures) to clone the initially empty repository.
02) Write a program that will take in two integers from the command line and will multiply them together. It should write a message to the console stating that “The area of the rectangle is:” followed by the result of the multiplication. Commit this to the repository.
03) Using the commands as shown in the lecture notes create a branch called “experiment1”.
04) Modify the code to take in a third integer from the command line and multiply all three values together. It should write a message to the console stating that “The volume of the cuboid is:” followed by the result of the multiplication. Commit this to the repository.
05) As shown in the lecture notes switch from the “experiment1” branch to the “master” branch. Inspect your code. Write down what you think has happened.
06) Switch back to the “experiment1” branch again. Inspect your code. Write down what you think has happened.

<u>Worksheet 3: Commit Early, Commit Often and rolling back code</u>

Introduction:
In this worksheet you will be introduced to the software development construct of “Commit Early, Commit Often”. As was covered in the notes the idea is that you write and complete small pieces of code (usually a method) and you commit them to the repository. As part of the commiting process you will be required to follow the commit-push series of commands to ensure you are not causing a code conflict. A later worksheet will expand on this to use your development group and working in parallel. You will also be introduced to the ability to revert code back to a previous commit this is useful in two situations

- You’ve made changes to the code that are not working and you cannot get back to a workable state
- The development path that was tried was unsuccessful and needs to be quickly removed from the code by reverting to a time before this path was started

In both cases you will need to find the commit that you need and get git to revert to that state.
For assessment you will be required to document screenshots of the commands you have performed. I will also require access to your repositories to check that they are working correctly. Granting access is covered in the lecture notes. After each step you must commit and push to the repository.
Tasks:

01) Create a repository called “Worksheet3” within gitlab. Using the clone command provided use the command line (as shown in lectures) to clone the initially empty repository.
02) Write a program that will take three integers from the command line. It should print out the values entered.
03) Add a method to the program that takes in those three integers and calculates and returns the volume of a cuboid. Print that value to console.
04) Add a method to the program that takes in those three integers and calculates the surface area of a cuboid (2*w*h + 2*w*d + 2*h*d). Print that value to console.
05) Add a method that totals up the length of the edges of the cuboid (4*w + 4*h + 4*d) and returns that value. Print that value to console.
06) Run the git log command and post a full screenshot of the entire log. There should be at least 4 entries in the log.
07) Make arbitrary mistakes in your code such that it will not compile then run the git reset command. What happens?
08) Checkout your second commit that is visible in the log. What has happened to your code?
09) Checkout the HEAD of your repository. What has happened to the code?
04) Modify the code to take in a third integer from the command line and multiply all three values together. It should write a message to the console stating that “The volume of the cuboid is:” followed by the result of the multiplication. Commit this to the repository.
05) As shown in the lecture notes switch from the “experiment1” branch to the “master” branch. Inspect your code. Write down what you think has happened.
06) Switch back to the “experiment1” branch again. Inspect your code. Write down what you think has happened.

Project Specification

Based on your knowledge of Robocode from the hackathon, develop a robot that will out-perform your classmate’s robot in a league event.
Your robot should have response patterns (at minimum) for the following methods:

- onBulletHit(BulletHitEvent event)
- onHitByBullet(HitByBulletEvent event)
- onHitRobot(HitRobotEvent event)
- onHitWall(HitWallEvent event)

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

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, there will be another round of the class league. Each robot will earn points for is performance. Based on this event the team must produce and log a bug report for the features that need to be address in the next review.

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