NOTTINGHAM TRENT UNIVERSITY

School of Science and Technology

COURSEWORK ASSESSMENT SPECIFICATION

- Details of module and team.
- What learning outcomes are assessed?
- What are my deadlines and how much does this assessment contribute to my module grade?

- What am I required to do in the assessment?
- What are my assessment criteria? (What do I have to achieve for each grade?)
- Can I get formative feedback before submitting? If so, how?
- What extra support could I look for myself?

- How and when do I submit this assessment?
- How and when will I get summative feedback?
- What skills might this work evidence to employers?

MODULE CODE	SOFT30161
MODULE TITLE	Advanced Software Engineering
MODULE LEADER	Dr. Neil Sculthorpe
COURSEWORK TITLE	Functional Test-driven Development
LEARNING OUTCOMES ASSESSED	Critically evaluate the practice of test-driven development. Write program code using the functional-programming paradigm. Design, develop and use unit tests, in the context of test-driven development.
CONTRIBUTION TO MODULE	40%
DATE SET	Monday 13th January 2020
DATE OF SUBMISSION	11pm Thursday 7th May 2020
METHOD OF SUBMISSION	NOW Dropbox Folder (Turnitin Enabled)
DATE OF FEEDBACK	Tuesday 2nd June 2020
METHOD OF FEEDBACK	Electronic via NOW; Oral feedback available on request

Late Submissions and NECs

Work handed in up to five working days late will be given a maximum grade of Low Third, whilst work that arrives more than five working days will be given a grade of Zero. Work will only be accepted without these caps if a *Notification of Extenuating Circumstances* (NEC) is submitted and upheld. NECs should be submitted through the following online portal:
https://ntu.ac.uk/current_students/resources/student_handbook/appeals/index.html

Plagiarism, Collusion and Turnitin

The University views **plagiarism and collusion** as serious academic irregularities. Penalties range from capped marks to dismissal from the course and termination of studies.

The Student Handbook has a section on Academic Irregularities, which outlines the penalties and states that **plagiarism** includes:

> "The incorporation of material (**including text, graph, diagrams, videos etc.**) derived from the work (published or unpublished) of another, by unacknowledged quotation, paraphrased imitation or other device in any work submitted for progression towards or for the completion of an award, which in any way suggests that it is the student's own original work. Such work may include printed material in textbooks, journals and material accessible electronically for example from web pages."

Whereas **collusion** includes:

> "Unauthorised and unacknowledged copying or use of material prepared by another person for use in submitted work. This may be with or without their consent or agreement to the copying or use of their work."

If copied with the agreement of the other candidate, both parties are considered guilty of Academic Irregularity. To help you avoid plagiarism and collusion, you are permitted to submit your report once to a separate drop box entitled 'Draft report', to view both the matching score and to look at what areas are affected. It is then down to you to make any changes needed. Note that Turnitin cannot say if something has been plagiarised or not. Instead it highlights matches between your text and other Turnitin content. For help, do not contact the module leader, but instead make use of the Library's Plagiarism and Turnitin Support.

I Assessment Requirements

This assignment involves employing the software-engineering practice of *test-driven development* to develop program code in the functional-programming paradigm. You are also expected to critically evaluate the test-driven development practice, and to reflect on your experiences.

I.1 Tasks

You are required to:

1. Explain the practice of *test-driven development*, and critically evaluate this practice. This evaluation can include your own impressions and experiences, and, for the higher grades, references to evidence from the research literature.
2. Apply the test-driven development approach to the development of functional-style code written in the Haskell programming language. You are free to choose any algorithm/application to develop, though it should be complex enough that it can be broken down into multiple functions each requiring multiple tests during the test-driven development process.
3. Utilise Haskell libraries that provide unit-testing tools. For the higher grades, this should involve property-based testing tools.
4. Reflect on your own experiences of using test-driven development, functional programming, and testing tools for this assignment. This should involve considering what went well and what did not, whether you think the reasons for this were due to the nature of the practice/tools or your own approach/familiarity, and whether and how you would use the practice/tools in future.

I.2 Submission Requirements

You are required to produce Haskell source files and a written report. The source files should contain your developed code and the accompanying tests. The report should contain your explanation, evaluation and reflections, as well as a narrative account demonstrating your application of the test-driven development process. You should **not** include a complete copy of the source code in the report, though you may include representative extracts to use as examples as part of the narrative account. The report should also briefly introduce any testing tools that you have used.

The report should be approximately 5 to 10 pages in length, and is required to be at most 12 pages (excluding title page and references). You may structure your report however you see fit.

You should submit the following files to the NOW dropbox folder:

- A **PDF document** containing your report.
- A compressed archive file (e.g. ZIP or TAR) containing the set of Haskell source files that you have developed.

Note that your PDF report must be submitted separately from the archive file.[1]

[1] This is necessary because the Turnitin similarity checker does not analyse files within an archive.

II Assessment Criteria

Criteria	1st	2:1	2:2	3rd	Fail
Critical evaluation of test-driven development.	Critical evaluation of test-driven development, drawing on a substantial amount of relevant research to inform and support arguments.	Critical evaluation of test-driven development, but supporting references are few, or mostly limited to informal sources such as blogs.	Accurate description of test-driven development, but evaluation is limited.	Mostly accurate description of test-driven development, but without meaningful evaluation.	Some relevant discussion of test-driven development, but with serious inaccuracies or misconceptions.
Application of test-driven development.	Rigorous adherence to the test-driven development practice.	Mostly adheres to the test-driven development practice.	Partially adheres to the test-driven development practice, but some aspects are frequently missing or incorrect.	*Test-first* development is demonstrated, but not the distinguishing aspects of *test-driven* development.	The test-driven development practice is not demonstrated.
Writing functional code.	Excellent use of Haskell to write purposeful functional-style code, going beyond what has been taught.	Competent use of Haskell for writing purposeful functional-style code.	Successful use of Haskell for writing purposeful code, but not consistently in the functional style.	Some purposeful Haskell code has been produced, but it mostly does not make use of the functional style.	Some Haskell code has been produced, but it is either trivial, or lacks purpose.
Use of testing tools.	Expertise with property-based testing tools demonstrated, using features or frameworks beyond those taught, and including appropriate tests involving user-defined data types.	Strong skills with property-based testing tools demonstrated, including appropriate tests involving user-defined data types.	Competent and appropriate use of a property-based testing tool demonstrated.	Competent and appropriate use of a unit-testing tool demonstrated.	Some use of a testing tool, but the resulting tests are either incomplete, inappropriate, or trivial adaptations of pre-existing code.
Reflection on own work.	Insightful, clear, and well justified critical reflection.	Meaningful and clear reflection, with some justification.	Reasonable reflection with some justification, but lacks clarity.	Some relevant reflection, but either lacks justification, is very unclear, or is partially incoherent.	Any reflection is mostly irrelevant to the tools and practices being assessed.

III Resources that may be Useful

A straightforward way to set up Haskell development infrastructure on your own personal machine is to install the Haskell Platform [10]. For learning the Haskell language, there are four introductory textbooks on the module resource list [6, 7, 8, 11]. You are advised to dedicate some time to reading at least one of them.

The standard unit-testing tool for Haskell is HUnit [5]. Popular property-based testing tools include QuickCheck [3] and SmallCheck [9]. Unit tests and/or property-based tests defined using these tools can be managed by a testing framework, such as Tasty [2], Hspec [4], or test-framework [1].

The textbook by Thompson [11] makes use of QuickCheck throughout, and Section 4.8 briefly introduces the HUnit tool. Chapter 11 of the textbook by O'Sullivan et al. [8] is dedicated to QuickCheck and property-based testing principles, and includes an introduction to generating QuickCheck test data for user-defined data types.

IV Feedback Opportunities

Oral formative feedback on your progress is available during the laboratory sessions. You will receive written feedback on coursework submission, together with your awarded grade, within three weeks of the submission deadline. Further oral feedback on your submission is available on request.

V Moderation

The clarity of this specification, and the appropriateness of the assessment criteria, have been checked by two members of the Department of Computing and Technology. The grades awarded by the assessor will be reviewed by another member of the Department to check for consistency and fairness.

VI Referencing, Plagiarism and Collusion

This coursework assessment is an individual assignment. This means that the report that you submit must be authored entirely by yourself.

If any submitted code is *adapted from* code written by another person, then this must be clearly identified and an appropriate reference given. The reference must identify precisely where the code is to be found so that it can be examined when assessing your work. For example, a reference to a website must identify the specific webpage on which the code is found, whereas a reference to a textbook must identify the specific subsection and pages on which the code is found.

Failure to identify and correctly reference such material is **plagiarism**. This includes code written by another student — such code, if not publicly available, can be referenced as a 'personal communication', and a copy of the original code submitted as an additional file with your submission. If any code in your submission is adapted from another piece of code of which you are a *joint* author, then you should state this, provide a copy of the original code, and make clear what adaptations you have made individually. You should not jointly develop program code or unit tests for this assignment — doing so is **collusion**.

However, note that sharing reference materials, discussion of software-engineering practices, and mutual experimentation with software-engineering tools, are perfectly acceptable and encouraged.

VII Aspects for Professional Development

This assignment will develop your knowledge and understanding of the test-driven development practice, as well as giving you practical experience. You will also gain familiarity with testing tools and the functional-programming paradigm.

References

[1] Max Bolingbroke. test-framework: Framework for running and organising tests, with HUnit and QuickCheck support. Hackage, 2008. URL https://hackage.haskell.org/package/test-framework. [Accessed 26/07/19].

[2] Roman Cheplyaka. tasty: Modern and extensible testing framework. Hackage, 2013. URL https://hackage.haskell.org/package/tasty. [Accessed 26/07/19].

[3] Koen Claessen. QuickCheck: Automatic testing of Haskell programs. Hackage, 2006. URL https://hackage.haskell.org/package/QuickCheck. [Accessed 26/07/19].

[4] Simon Hengel, Trystan Spangler, and Greg Weber. hspec: A testing framework for Haskell. Hackage, 2011. URL https://hackage.haskell.org/package/hspec. [Accessed 26/07/19].

[5] Dean Herington. HUnit: A unit testing framework for Haskell. Hackage, 2006. URL https://hackage.haskell.org/package/HUnit. [Accessed 26/07/19].

[6] Graham Hutton. *Programming in Haskell*. Cambridge University Press, 2nd edition, 2016.

[7] Miran Lipovača. *Learn You a Haskell for Great Good!* No Starch Press, 2011. URL http://learnyouahaskell.com/.

[8] Bryan O'Sullivan, John Goerzen, and Don Stewart. *Real World Haskell*. O'Reilly, 2008. URL http://book.realworldhaskell.org/.

[9] Colin Runciman and Roman Cheplyaka. smallcheck: A property-based testing library. Hackage, 2008. URL https://hackage.haskell.org/package/smallcheck. [Accessed 26/07/19].

[10] Haskell Platform Infrastructure Team. Haskell platform, 2018. URL https://www.haskell.org/platform/. [Accessed 26/07/19].

[11] Simon Thompson. *Haskell: The Craft of Functional Programming*. Addison-Wesley, 3rd edition, 2011.