Assignment 2 Brief

1. Assessment Specification

ASSIGNMENT TITLE	A Case Study of Test-driven Development
LEARNING OUTCOMES ASSESSED	**K3:** Critically evaluate the practice of test-driven development. **S3:** Write program code using the functional-programming paradigm. **S5:** Design, develop and use unit tests, in the context of test-driven development.
CONTRIBUTION TO MODULE	40%
DATE SET	Monday 11th January
SUBMISSION DEADLINE	11pm Friday 14th May 2021
METHOD OF SUBMISSION	NOW Dropbox Folder
DATE OF FEEDBACK	After the Summer Examination Board

METHOD OF FEEDBACK	Electronic via NOW; Verbal feedback available on request

2. Assessment Requirements

This assignment is a case study of the software-engineering practice of *test-driven development* (TDD). This case study will involve developing program code in the functional-programming paradigm, using the test-driven approach. You are also expected to critically evaluate the test-driven development practice, and to reflect on your experiences.

2.1 Deliverables

You are required to produce Haskell source files and a written report.

The source files should contain your developed code and the accompanying tests. You should not submit any files related to configuring a Cabal or Stack project, or using an integrated development environment.

The report should contain your:

1. explanation and critical evaluation of test-driven development;
2. narrative account of applying the test-driven development process;
3. reflections on the tools and practices used.

The report should indicate which specific testing tools you have used.

You should ***not*** include a complete copy of the source code in the report, though you may include selected extracts to use as examples as part of the narrative account. The report should be approximately 3000 to 5000 words in length, and is required to be at most 6000 words (excluding title page, contents page, reference list, and code extracts).

2.2 Submission Instructions

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

- A single **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 is required to be submitted separately from the archive file. (This is necessary because the Turnitin similarity checker does not analyse files within an archive.)

Do not submit any other files relating to developing and compiling your code. In particular, you should not submit any generated files, and if you have used an integrated development environment to develop your code, then you should not submit any files specific to that environment.

3. Assessed Tasks

Task 1: Explaining Test-driven Development

Explain the practice of test-driven development, and critically evaluate

this practice.

For the higher grades, this evaluation should include:

- analysis of evidence from the research literature;
- your own opinions based on your own experiences.

Task 2: Applying Test-driven Development

Apply the test-driven development approach to the development of functional-style Haskell code.

Clarifications:

- In order to convey your ability to follow the TDD process, **your report should contain a narrative account of selected parts of the development, including multiple examples of each step of the TDD process** (though you do not need to describe the entire development in detail).
- You are free to choose any program/algorithm/library to develop; some suggestions are given in Section 4.

Task 3: Using Automated Testing Tools

Utilise Haskell libraries that provide automated unit-testing tools.

For the higher grades:

- this should involve ***property-based*** testing tools.

Task 4: Reflecting on Experiences

Reflect on your own experiences of using test-driven development, functional programming, and automated testing tools for this case study.

This should involve considering:

- what went well and what did not;
- what were the likely reasons for this;
- whether and how you would use the practices/tools in future.

4. Suggested Subjects

For this assignment you are free to choose any program/algorithm/library to develop, though you should ensure that it is challenging enough to allow you to meet the assessment criteria (see Section 5). As a general guideline, the problem being solved should be complex enough that it can be broken down into multiple functions each requiring multiple tests during the test-driven development process.

This section provides two suggested subjects for your development. The first, a dictionary implementation using a binary search tree, is sufficiently challenging to meet the highest grade criteria. The second, an income tax calculator, is simpler, but sufficiently challenging to meet the ***pass*** criteria.

4.1 Binary Search Tree

A dictionary data structure contains a collection of *key/item pairs*, such that each

key occurs at-most once, but the same *item* may occur several times. Key/item pairs are known as *entries*.

Four basic dictionary operations are:

- create an empty dictionary;
- insert an entry;
- lookup the item by specifying a key;
- produce a list of all entries.

More advanced dictionary operations are:

- remove an entry by specifying the key;
- remove all entries that satisfy a specified predicate function (like the final C++ exercise in Week 3 [/d2l/common/dialogs/quickLink/quickLink.d2l?ou=711571&type=content&rcode=ntu-2098621]).

Suggestion: Develop a Haskell module that provides the dictionary operations, encapsulating an internal implementation that uses a binary search tree (BST).

To be of the appropriate complexity for this assignment:

- you should develop the details of the BST implementation yourself, rather than using an existing library;
- the BST does not need to be self balancing.

4.2 Income Tax Calculator

Annual income tax in the United Kingdom is calculated as follows.

Each individual has a *personal allowance*, which is an amount of income that is exempt from income tax. A person's *taxable income* is their annual income minus their personal allowance (to a minimum of zero). The default personal allowance in 2020/2021 is £12,500 [6]. However, for higher earners:

> *"Your Personal Allowance goes down by £1 for every £2 that your [annual] income is above £100,000. This means your allowance is zero if your [annual] income is £125,000 or above."* [6]

Once an individual's taxable income is determined, then income tax is applied as follows [6]:

Band Name	**Taxable income**	**Tax rate**
Basic rate	£0 to £37,500	20%
Higher rate	£37,501 to £150,000	40%
Additional rate	over £150,000	45%

Each tax rate only applies to income within that band, not to all earned income. For example, a taxable income of £60,000 is taxed 20% on the first £37,500, and taxed 40% on the remaining £22,500 (which results in £16,500 of tax overall).

Suggestion: Develop Haskell functions for computing an individual's personal allowance and income tax from their annual income.

Advice:

- You might find it helpful to generate test data using an online tax calculator (e.g. [1]).

5. Assessment Criteria

Understanding of TDD — Weighting 20%

Class	Criteria
1st	Demonstrates the ability to rigorously apply the TDD process.
2:1	Demonstrates the ability to mostly apply the TDD process, but with some minor omissions, deviations, or aspects for improvement.
2:2	Demonstrates the ability to apply some key aspects of the TDD process, but with significant omissions.
3rd	Describes the full TDD process correctly.
Fail	Description of the TDD process has some omissions / inaccuracies.
Zero	No understanding of the TDD process conveyed.

Critical evaluation of TDD — Weighting 20%

Class	Criteria
1st	Critical evaluation of TDD, supported and informed by research data **AND** own experiences.
2:1	Critical evaluation of TDD, supported and informed by the findings of others **AND** own experiences.
2:2	Describes benefits and detriments of TDD, either taken from the findings of others **OR** based on own experiences.
3rd	Describes benefits and detriments of TDD, either taken from the opinions of others **OR** based on own opinions.
Fail	Some discussion of TDD, but missing either the benefits or the detriments.
Zero	No discussion of TDD benefits or detriments.

Use of research literature — Weighting 10%

Class	Criteria
Exceptional 1st	Draws from at least six relevant peer-reviewed research studies, at least three of which were published in the last 10 years.

Class	Criteria
Mid 1st	Draws from at least four relevant peer-reviewed research studies, at least two of which were published in the last 10 years.
2:1	Draws from multiple relevant sources, at least one of which is peer-reviewed research literature published in the last 10 years, and at least four are formal publications (peer-reviewed literature, textbooks, or company literature).
2:2	Draws from multiple relevant sources, including some formal (but not peer-reviewed) publications such as textbooks or company literature.
3rd	Draws from informal sources such as blogs and wikis.
Fail	Reference list given without citations, or citations used without accompanying reference list.
Zero	No citing or referencing.

Scale of Haskell code — Weighting 20%

Class	Criteria

Class	Criteria
1st	Comparable in complexity to a dictionary module encapsulating an implementation using a BST, and providing polymorphic operations for lookup, insertion, listing of entries, removal, and removal by predicate.
2:1	Comparable in complexity to a dictionary module implemented using a BST, providing polymorphic lookup and insertion, listing of entries, and some cases of the removal algorithm.
2:2	Comparable in complexity to a BST, providing lookup, insertion, and listing of entries.
3rd	Comparable in complexity to an income-tax calculator that takes into account tax bands and personal allowance.
Fail	Limited complexity **OR** or trivial adaptations of pre-existing code **OR** code does not compile **OR** code contains significant run-time errors.
Zero	Submitted Haskell code is limited to auto-generated placeholder code **OR** to code copied from a tutorial without any adaptations.

Use of testing tools — Weighting 20%

Class	Criteria

Class	Criteria
1st	Expertise with property-based testing tools demonstrated, creating numerous well-designed tests, including appropriate tests involving user-defined data types. **AND** use of features or frameworks beyond those taught.
2:1	Competence with property-based testing tools demonstrated, creating several tests that serve a useful purpose, including appropriate tests involving user-defined data types.
2:2	Use of a property-based testing tool demonstrated, though the property-based tests created may have serious flaws. **OR** competence with a unit-testing tool demonstrated, creating numerous well-designed unit tests.
3rd	Competence with a unit-testing tool demonstrated, creating several unit tests that serve a useful purpose.
Fail	Some use of a testing tool, but the resulting tests are either incomplete, inappropriate, or are a trivial adaptation of pre-existing code. **OR** tests do not compile.
Zero	No test code **OR** code is limited to auto-generated placeholder code **OR** code is copied from a tutorial without any adaptations.

Reflection on own experiences — Weighting 10%

Class	Criteria
1st	Valid, clear and justified critical reflections on testing tools, functional programming and TDD.
2:1	Mostly valid, clear and justified reflections on testing tools, functional programming and TDD.
2:2	Some valid reflections on testing tools, functional programming and TDD, but may be unclear and/or unjustified in several ways.
3rd	Some relevant reflections on testing tools, functional programming and TDD.
Fail	Lacks relevant reflections on at least one of either testing tools, functional programming, or TDD.
Zero	No relevant reflections.

6. Resources that may be Useful

The short textbook by Beck [2] is dedicated to the practice of test-driven development. Part I provides a straightforward introduction to TDD, and numerous examples of applying the process, and then Part III discusses general principles and advice (you may want to skip Part II as it is more complex). If you'd prefer to read a some short overview of test-driven development, then I

recommend the introductory article by Jeffries and Melnik [10].

A convenient way to set up Haskell development infrastructure on your own personal machine is by using the Stack build system [14]. For learning the Haskell language, there are four introductory textbooks on the module reading list [9, 11, 12, 15]. You are advised to dedicate some time to reading at least one of them.

The standard unit-testing tool for Haskell is HUnit [8]. Popular property-based testing tools include QuickCheck [5] and SmallCheck [13]. Unit tests and/or property-based tests defined using these tools can be managed by a testing framework, such as Tasty [4], Hspec [7], or test-framework [3].

The textbook by Thompson [15] makes use of QuickCheck throughout, and Section 19.6 explains how to generate test data for user-defined data types. Section 4.8 also briefly introduces the HUnit tool. Chapter 11 of the textbook by O'Sullivan et al. [12] is dedicated to QuickCheck and property-based testing principles, and includes an introduction to generating QuickCheck test data for user-defined data types.

7. 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.

8. Feedback Opportunities

Verbal formative feedback on your progress is available during the laboratory sessions. You will receive written feedback on coursework submission, together with your awarded grade, after the Board of Examiners meet to determine your results. Further verbal feedback on your submission is available on request.

9. 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 automated testing tools and the functional-programming paradigm.

References

[1] Income Tax Calculator. URL: https://www.incometaxcalculator.org.uk/. [https://www.incometaxcalculator.org.uk/.] [Accessed 02/12/20]

[2] Kent Beck. Test Driven Development: By Example. Addison-Wesley, 2002. ISBN 978-0321146533. [https://rl.talis.com/3/ntu/items/8E3B6F37-E6CA-70A3-D301-1E7626BA129D.html]

[3] 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 [https://hackage.haskell.org/package/test-framework] [Accessed 26/07/19]

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

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

[6] Government Digital Service. Income tax rates and personal allowances.

URL: https://www.gov.uk/income-tax-rates [https://www.gov.uk/income-tax-rates]. [Accessed 02/12/20]

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

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

[9] Graham Hutton. Programming in Haskell. Cambridge University Press, 2nd edition, 2016. [https://rl.talis.com/3/ntu/items/B757E5A2-C5B9-439D-FD8D-894EAAC47DE4.html]

[10] Ron Jeffries and Grigori Melnik. TDD: The art of fearless programming. IEEE Software, 24(3): 24–30, 2007. DOI: 10.1109/MS.2007.75 [https://rl.talis.com/3/ntu/items/3D05B80B-F07F-2C99-FF6F-9B42CACDCE5B.html]

[11] Miran Lipovača. Learn You a Haskell for Great Good! No Starch Press, 2011. [https://rl.talis.com/3/ntu/items/DFBBEEF5-E72B-5CF6-A6CD-87E429256686.html]

[12] Bryan O'Sullivan, John Goerzen, and Don Stewart. Real World Haskell. O'Reilly, 2008. [https://rl.talis.com/3/ntu/items/62F81CEC-7896-17FB-9FE4-9A7C11FE54D9.html]

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

26/07/19]

[14] Stack contributors. The Haskell Tool Stack. https://docs.haskellstack.org/en/stable/README/ [https://docs.haskellstack.org/en/stable/README/]. [Accessed 07/01/20]

[15] Simon Thompson. Haskell: The Craft of Functional Programming. Addison-Wesley, 3rd edition, 2011. [https://rl.talis.com/3/ntu/items/79F5F8AD-17F8-1509-FEDB-FAD8457A3728.html]

Go to top