A Testing Tool for Introductory Programming Courses Thesis A Seminar

Kyu-Sang Kim z5208931

Supervised by Andrew Taylor (UNSW)
Assessed by John Shephard (UNSW)

Term 1, 2022

Contents

Introduction

Motivation - Student Enrolments in Introductory Courses Motivation - Andrew Taylor autotest Motivation - Technical Debt Thesis Statement

- 2 Literature Review
 - Background Code Marking in Introductory Courses
 Background Automated Testing & Marking Approache
 Existing Work Andrew Taylor autotest
 Existing Work Harvard Uni CS50 check50
 Existing Work Google gtest
- 3 Design
 Design Requirement
 Proposed Design
- 4 Schedule & Conclusion Schedule
 - Summary References



Student numbers enrolling in introductory programming courses at UNSW have significantly increased in recent years



Student numbers enrolling in introductory programming courses at UNSW have significantly increased in recent years

Enrolments/Year	2017	2018	2019	2020	2021
COMP1511	1351	1655	1874	1905	2381
COMP1521	715	1136	1352	1417	1633
COMP2521	378	1019	1389	1445	1551

Table 1: Course enrolments from UNSW Class Timetable (April 2022)

Student numbers enrolling in introductory programming courses at UNSW have significantly increased in recent years

Enrolments/Year	2017	2018	2019	2020	2021
COMP1511	1351	1655	1874	1905	2381
COMP1521	715	1136	1352	1417	1633
COMP2521	378	1019	1389	1445	1551

Table 1: Course enrolments from UNSW Class Timetable (April 2022)

The trend in increasing student enrolments for these courses present two main challenges for course staff:

Student numbers enrolling in introductory programming courses at UNSW have significantly increased in recent years

Enrolments/Year	2017	2018	2019	2020	2021
COMP1511	1351	1655	1874	1905	2381
COMP1521	715	1136	1352	1417	1633
COMP2521	378	1019	1389	1445	1551

Table 1: Course enrolments from UNSW Class Timetable (April 2022)

The trend in increasing student enrolments for these courses present two main challenges for course staff:

• How can courses feasibly start/continue utilising practical assessments without automation tools as the staff to student ratio decreases?

Student numbers enrolling in introductory programming courses at UNSW have significantly increased in recent years

Enrolments/Year	2017	2018	2019	2020	2021
COMP1511	1351	1655	1874	1905	2381
COMP1521	715	1136	1352	1417	1633
COMP2521	378	1019	1389	1445	1551

Table 1: Course enrolments from UNSW Class Timetable (April 2022)

The trend in increasing student enrolments for these courses present two main challenges for course staff:

- 1 How can courses feasibly start/continue utilising practical assessments without automation tools as the staff to student ratio decreases?
- ② How much of the marking process can be automated via tools and will the automation tools remain viable as assessments change?

Written by Andrew Taylor in 2015 (Python rewrite)



- 📵 Written by Andrew Taylor in 2015 (Python rewrite)
- First introduced to UNSW in COMP2041 2015 S2



- Written by Andrew Taylor in 2015 (Python rewrite)
- First introduced to UNSW in COMP2041 2015 S2
- 3 Sufficiently capable and in current use to perform automated marking for most UNSW introductory programming courses

- 🕦 Written by Andrew Taylor in 2015 (Python rewrite)
- First introduced to UNSW in COMP2041 2015 S2
- 3 Sufficiently capable and in current use to perform automated marking for most UNSW introductory programming courses
- 4 Has some design deficiencies and accumulated non-trivial technical debt since introduction

- 🕠 Written by Andrew Taylor in 2015 (Python rewrite)
- First introduced to UNSW in COMP2041 2015 S2
- 3 Sufficiently capable and in current use to perform automated marking for most UNSW introductory programming courses
- 4 Has some design deficiencies and accumulated non-trivial technical debt since introduction
- 5 Deeper exploration of architecture and implementation details in Background section of Seminar

<ロ> <回> <個> < 量> < 量> < 量> < 量 < 9 < @

Technical Debt

We approach technical debt as outlined in the framework defined within An Exploration of Technical Debt¹

¹Tom Edith, Aybüke Aurum and Richard Vidgen, 'An Exploration of Technical Debt' (2013) 86(6) *The Journal of systems and software 1498*

²Cunningham, Ward, 'The WyCash Portfolio Management System' in Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (addendum) (ACM, 1992) 29

Technical Debt

- We approach technical debt as outlined in the framework defined within An Exploration of Technical Debt¹
- 2 Cunningham², who introduced the concept of technical debt, described how "shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite"

¹Tom Edith, Aybüke Aurum and Richard Vidgen, 'An Exploration of Technical Debt' (2013) 86(6) *The Journal of systems and software 1498*

²Cunningham, Ward, 'The WyCash Portfolio Management System' in Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (addendum) (ACM, 1992) 29

Technical Debt

- We approach technical debt as outlined in the framework defined within An Exploration of Technical Debt¹
- 2 Cunningham², who introduced the concept of technical debt, described how "shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite"
- 3 How can we reconcile existing technical debt and ensure debt remains manageable in the future?

¹Tom Edith, Aybüke Aurum and Richard Vidgen, 'An Exploration of Technical Debt' (2013) 86(6) *The Journal of systems and software 1498*

²Cunningham, Ward, 'The WyCash Portfolio Management System' in Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (addendum) (ACM, 1992) 29

Thesis Statement

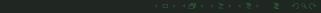
A user-friendly and maintainable general code testing tool is important to streamline the administration of introductory programming courses



Thesis Statement

A user-friendly and maintainable general code testing tool is important to streamline the administration of introductory programming courses

We will:



Thesis Statement

A user-friendly and maintainable general code testing tool is important to streamline the administration of introductory programming courses

We will:

 Develop an extensible and easy to use software package which parses and runs pre-written tests on submitted code

Thesis Statement

A user-friendly and maintainable general code testing tool is important to streamline the administration of introductory programming courses

We will:

- Develop an extensible and easy to use software package which parses and runs pre-written tests on submitted code
- Remediate known flaws in the existing autotest package

Thesis Statement

A user-friendly and maintainable general code testing tool is important to streamline the administration of introductory programming courses

We will:

- Develop an extensible and easy to use software package which parses and runs pre-written tests on submitted code
- Remediate known flaws in the existing autotest package
- Maintain backwards compatibility with legacy tests written for the existing autotest package

Thesis Statement

A user-friendly and maintainable general code testing tool is important to streamline the administration of introductory programming courses

We will:

- Develop an extensible and easy to use software package which parses and runs pre-written tests on submitted code
- Remediate known flaws in the existing autotest package
- Maintain backwards compatibility with legacy tests written for the existing autotest package
- Perform proving and performance tests on the new software package

Thesis Statement

A user-friendly and maintainable general code testing tool is important to streamline the administration of introductory programming courses

We will:

- Develop an extensible and easy to use software package which parses and runs pre-written tests on submitted code
- Remediate known flaws in the existing autotest package
- Maintain backwards compatibility with legacy tests written for the existing autotest package
- Perform proving and performance tests on the new software package
- 5 Deprecate and replace the existing autotest used for introductory programming courses at UNSW CSE

Contents

Introduction

Motivation - Student Enrolments in Introductory Course Motivation - Andrew Taylor autotest Motivation - Technical Debt Thesis Statement

2 Literature Review

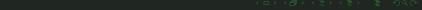
Background - Code Marking in Introductory Courses
Background - Automated Testing & Marking Approaches
Existing Work - Andrew Taylor autotest
Existing Work - Harvard Uni CS50 check50
Existing Work - Google gtest
Existing Work - JUnit

Design

Design Requirements
Proposed Design

4) Schedule & Conclusion

Schedule Summary References



Code Marking in Introductory Courses

We assume that code marking refers to the determination of whether submitted code when run conforms to some behaviour that is outlined in a specification or any similar resource

³missing source

⁴my experiences marking => do last ditch survey?

Code Marking in Introductory Courses

- We assume that *code marking* refers to the determination of whether submitted code when run conforms to some behaviour that is outlined in a specification or any similar resource
- What are some potential issues with manual code marking?



³missing source

⁴my experiences marking => do last ditch survey?

Code Marking in Introductory Courses

- We assume that code marking refers to the determination of whether submitted code when run conforms to some behaviour that is outlined in a specification or any similar resource
- What are some potential issues with manual code marking?
 - Tedious and repetitive work potentially prone to mistakes³

³missing source

⁴my experiences marking => do last ditch survey?

Code Marking in Introductory Courses

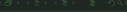
- We assume that code marking refers to the determination of whether submitted code when run conforms to some behaviour that is outlined in a specification or any similar resource
- What are some potential issues with manual code marking?
 - Tedious and repetitive work potentially prone to mistakes³
 - Potential inconsistencies when distributed between separate markers⁴

³missing source

⁴my experiences marking => do last ditch survey?

Code Marking in Introductory Courses

- We assume that code marking refers to the determination of whether submitted code when run conforms to some behaviour that is outlined in a specification or any similar resource
- What are some potential issues with manual code marking?
 - Tedious and repetitive work potentially prone to mistakes³
 - Potential inconsistencies when distributed between separate markers⁴
 - Increasing marking load per course staff member due to increasing student numbers can exacerbate the aforementioned issues



³missing source

⁴my experiences marking => do last ditch survey?

Code Marking in Introductory Courses

- We assume that code marking refers to the determination of whether submitted code when run conforms to some behaviour that is outlined in a specification or any similar resource
- What are some potential issues with manual code marking?
 - Tedious and repetitive work potentially prone to mistakes³
 - Potential inconsistencies when distributed between separate markers⁴
 - Increasing marking load per course staff member due to increasing student numbers can exacerbate the aforementioned issues
- How can we eliminate or reduce these issues?

³missing source

⁴my experiences marking => do last ditch survey?

1 Main goal: Autonomously verify the correctness of student submitted code in relation to pre-defined expected behaviour



- **1 Main goal:** Autonomously verify the correctness of student submitted code in relation to pre-defined expected behaviour
- 2 Main benefit: Automation of performance/correctness testing of student submitted code
 - Time spent on manual marking can be utilised in other tasks



- **1 Main goal:** Autonomously verify the correctness of student submitted code in relation to pre-defined expected behaviour
 - 2 Main benefit: Automation of performance/correctness testing of student submitted code
 - Time spent on manual marking can be utilised in other tasks
 - Any fix from incorrect marking tests can be quickly rolled out to all students

- Main goal: Autonomously verify the correctness of student submitted code in relation to pre-defined expected behaviour
- 2 Main benefit: Automation of performance/correctness testing of student submitted code
 - Time spent on manual marking can be utilised in other tasks
 - Any fix from incorrect marking tests can be quickly rolled out to all students
 - Economically beneficial as costs to pay staff for manual marking is minimised

1 How can we methodically determine that "submitted code matches expected behaviour"?



- 1 How can we methodically determine that "submitted code matches expected behaviour"?
 - External Program Side-effect Comparison



- How can we methodically determine that "submitted code matches expected behaviour"?
 - External Program Side-effect Comparison
 - Internal Program Unit Testing



Automated Testing & Marking Approaches

- How can we methodically determine that "submitted code matches expected behaviour"?
 - External Program Side-effect Comparison
 - Internal Program Unit Testing
 - Combination of above



Automated Testing & Marking Approaches

- 1 How can we methodically determine that "submitted code matches expected behaviour"?
 - External Program Side-effect Comparison
 - Internal Program Unit Testing
 - Combination of above
- What are the specifics of these methodologies and what are some existing automated testing frameworks?



External Program Side-effect Comparison

Assume program side-effects to be observable output from a tested program which has been stored on external resources such as the console via stdout or files that store logs/program state



- Assume program side-effects to be observable output from a tested program which has been stored on external resources such as the console via stdout or files that store logs/program state
- This output can be externally compared with those pre-generated by a sample solution to determine whether expected behaviour has been achieved

(ロ) (個) (重) (重) (重) のQの

External Program Side-effect Comparison

- Assume program side-effects to be observable output from a tested program which has been stored on external resources such as the console via stdout or files that store logs/program state
- 2 This output can be externally compared with those pre-generated by a sample solution to determine whether expected behaviour has been achieved
- 3 Comparisons on side-effects after program execution allows for easier testing of complex programs with dependencies on multiple different components at the cost of storage space

(ロ) (固) (重) (重) 重 の(の

External Program Side-effect Comparison

- Assume program side-effects to be observable output from a tested program which has been stored on external resources such as the console via stdout or files that store logs/program state
- This output can be externally compared with those pre-generated by a sample solution to determine whether expected behaviour has been achieved
- 3 Comparisons on side-effects after program execution allows for easier testing of complex programs with dependencies on multiple different components at the cost of storage space

Need graph outlining process of generating and comparing side-effects in side column

Tarva 1 0000 11 /00

Internal Program Unit Testing

Assume unit testing refers to the testing of individual discrete code components within the program by comparing internal program state to a known expected state pre-generated by a sample solution



Internal Program Unit Testing

- Assume unit testing refers to the testing of individual discrete code components within the program by comparing internal program state to a known expected state pre-generated by a sample solution
- 2 Executing and collecting the results of the internal unit tests for the program can be used to determine whether expected behaviour has been achieved

- Assume unit testing refers to the testing of individual discrete code components within the program by comparing internal program state to a known expected state pre-generated by a sample solution
- Executing and collecting the results of the internal unit tests for the program can be used to determine whether expected behaviour has been achieved
- 3 Testing internal code components allow for more thorough determination of program correctness at the cost of difficulty in testing more complex programs which may not be able to share the same unit testing framework

Kyu-Sang Kim Thesis A Seminar Term 1, 2022 12/

terature Review Design Schedule & Conclu

Internal Program Unit Testing

- Assume unit testing refers to the testing of individual discrete code components within the program by comparing internal program state to a known expected state pre-generated by a sample solution
- Executing and collecting the results of the internal unit tests for the program can be used to determine whether expected behaviour has been achieved
- Testing internal code components allow for more thorough determination of program correctness at the cost of difficulty in testing more complex programs which may not be able to share the same unit testing framework

Need graph outlining process of unit testing in side column here

Kyu-Sang Kim Thesis A Seminar Term 1, 2022 12/3

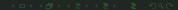
Author(s) & Introduction: Andrew Taylor - 2015



- **1** Author(s) & Introduction: Andrew Taylor 2015
- Testing Methodology: External Program Side-effect Comparison



- Author(s) & Introduction: Andrew Taylor 2015
- Testing Methodology: External Program Side-effect Comparison
- 3 Language: Python (+ some Shell and Perl script interfaces)

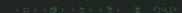


troduction Literature Review Design Schedule & Conclusion

Andrew Taylor autotest

- **1 Author(s) & Introduction:** Andrew Taylor 2015
- Testing Methodology: External Program Side-effect Comparison
- 3 Language: Python (+ some Shell and Perl script interfaces)
- Maintained/Support: Not regularly maintained outside of rare UNSW CSE student engagement

- Author(s) & Introduction: Andrew Taylor 2015
- Testing Methodology: External Program Side-effect Comparison
- Stanguage: Python (+ some Shell and Perl script interfaces)
- Maintained/Support: Not regularly maintained outside of rare UNSW CSE student engagement
- **6** Current UNSW CSE automated general code testing tool utilised for lab and assessment marking in most introductory programming courses and at times, some higher level courses (COMP Level 2+)



Andrew Taylor autotest implementation details

Example test case goes here

Graph of autotest architecture goes here



Andrew Taylor autotest pros & cons

1 Pros:

- Proven to be mostly reliable at UNSW CSE
- Can support any programming language assuming autotest can detect and compare side-effects
- Extensive parameters exposed to manage test execution environment
- Test results on failure provide meaningful information on the differences between actual and expected output



Andrew Taylor autotest pros & cons

Pros:

- Proven to be mostly reliable at UNSW CSE
- Can support any programming language assuming autotest can detect and compare side-effects
- Extensive parameters exposed to manage test execution environment
- Test results on failure provide meaningful information on the differences between actual and expected output

2 Cons:

- High levels of technical debt due to outdated use of technology and architecture
- Significant lack of documentation
- Difficult for first-time users to create tests and manage test execution environment



Harvard Uni CS50 check50

1 Author(s) & Introduction: Chad Sharp - v2.0 released in 2017 (no earlier records)



Harvard Uni CS50 check50

- **1 Author(s) & Introduction:** Chad Sharp v2.0 released in 2017 (no earlier records)
- Testing Methodology: External Program Side-effect Comparison



Harvard Uni CS50 check50

- **1 Author(s) & Introduction:** Chad Sharp v2.0 released in 2017 (no earlier records)
- Testing Methodology: External Program Side-effect Comparison
- 3 Language: Python

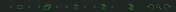


ntroduction Literature Review Design Schedule & Conclusion

Q Q_ Q

Harvard Uni CS50 check50

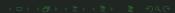
- Author(s) & Introduction: Chad Sharp v2.0 released in 2017 (no earlier records)
- Testing Methodology: External Program Side-effect Comparison
- 3 Language: Python
- Maintained/Support: Regularly maintained with active development (in private repository)



troduction Literature Review Design Schedule & Conclusion

Harvard Uni CS50 check50

- Author(s) & Introduction: Chad Sharp v2.0 released in 2017 (no earlier records)
- Testing Methodology: External Program Side-effect Comparison
- 3 Language: Python
- Maintained/Support: Regularly maintained with active development (in private repository)
- 5 Current Harvard University CS50: Introduction to Computer Science automated general code testing tool for checking correctness of practical lab exercises



Harvard Uni CS50 check50 implementation details

Example test case goes here

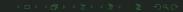
Graph of autotest architecture goes here



Harvard Uni CS50 check50 pros & cons

1 Pros:

- Tests are very simple to create as a chain of functions
- Documentation is very extensive
- Testing results can be rendered to HTML via a module for easier viewing
- Supports running of tests on both local and remote machines (PaaS Support)
- Concurrent running of tests is supported



Harvard Uni CS50 check50 pros & cons

Pros:

- Tests are very simple to create as a chain of functions
- Documentation is very extensive
- Testing results can be rendered to HTML via a module for easier viewing
- Supports running of tests on both local and remote machines (PaaS Support)
- Concurrent running of tests is supported

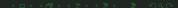
2 Cons:

- As a result of design choice for simplicity, testing of complex programs can be challenging (may require Harnessing)
- No official support or implementations for programming languages outside of C and Python (Flask Supported)

(ロ) (型) (型) (型) (型) の(の)

Google gtest

1 Author(s) & Introduction: Google - v1.0.0 released in 2008



Google gtest

- **1** Author(s) & Introduction: Google v1.0.0 released in 2008
- **Testing Methodology:** Internal Program Unit Testing (Technically Supports External Program Side-effect Comparison)

troduction Literature Review Design Schedule & Conclusion

Google gtest

- **1 Author(s) & Introduction:** Google v1.0.0 released in 2008
- Testing Methodology: Internal Program Unit Testing (Technically Supports External Program Side-effect Comparison)
- 3 Language: C++



troduction Literature Review Design Schedule & Conclusion

Google gtest

- **1 Author(s) & Introduction:** Google v1.0.0 released in 2008
- Testing Methodology: Internal Program Unit Testing (Technically Supports External Program Side-effect Comparison)
- 3 Language: C++
- Maintained/Support: Regularly maintained by Google and Open Source Community

Kyu-Sang Kim Thesis A Seminar

Google gtest

- **Author(s) & Introduction:** Google v1.0.0 released in 2008
- **Testing Methodology:** Internal Program Unit Testing (Technically Supports External Program Side-effect Comparison)
- Language: C++
- Maintained/Support: Regularly maintained by Google and Open Source Community
- Originally created by Google for internal use but has become one of the most popular C++ unit testing frameworks



Google gtest implementation details

Example test case goes here

Graph of gtest process goes here



Google gtest pros & cons

- 1 Pros:
 - Pro 1 goes here



Google gtest pros & cons

- Pros:
 - Pro 1 goes here
- 2 Cons:
 - Con 1 goes here



troduction Literature Review Design Schedule & Conclusion

JUnit

1 Author(s) & Introduction: Kent Beck, Erich Gamma, David Saff, Kris Vasudevan - Initial Prototype in 1997⁵

⁵Martin Fowler, "Bliki: Xunit", martinfowler.com (Webpage, 2022) https://martinfowler.com/bliki/Xunit.html

troduction Literature Review Design Schedule & Conclusion

JUnit

- **1 Author(s) & Introduction:** Kent Beck, Erich Gamma, David Saff, Kris Vasudevan Initial Prototype in 1997⁵
- **Testing Methodology:** Internal Program Unit Testing (Technically Supports External Program Side-effect Comparison)

⁵Martin Fowler, "Bliki: Xunit", *martinfowler.com* (Webpage, 2022) https://martinfowler.com/bliki/Xunit.html

JUnit

1 Author(s) & Introduction: Kent Beck, Erich Gamma, David Saff, Kris Vasudevan - Initial Prototype in 1997⁵

- Testing Methodology: Internal Program Unit Testing (Technically Supports External Program Side-effect Comparison)
- 3 Language: Java

⁵Martin Fowler, "Bliki: Xunit", *martinfowler.com* (Webpage, 2022) https://martinfowler.com/bliki/Xunit.html

JUnit

- Author(s) & Introduction: Kent Beck, Erich Gamma, David Saff, Kris Vasudevan - Initial Prototype in 1997⁵
- Testing Methodology: Internal Program Unit Testing (Technically Supports External Program Side-effect Comparison)
- 3 Language: Java
- Maintained/Support: Regularly maintained by JUnit team and Open Source Community

Kyu-Sang Kim Thesis A Seminar Term 1, 2022 22/32

⁵Martin Fowler, "Bliki: Xunit", *martinfowler.com* (Webpage, 2022) https://martinfowler.com/bliki/Xunit.html

JUnit

- Author(s) & Introduction: Kent Beck, Erich Gamma, David Saff, Kris Vasudevan - Initial Prototype in 1997⁵
- Testing Methodology: Internal Program Unit Testing (Technically Supports External Program Side-effect Comparison)
- Language: Java
- Maintained/Support: Regularly maintained by JUnit team and Open Source Community
- **5** A very popular open source unit testing framework for Java applications within the xUnit family of testing frameworks

Kyu-Sang Kim Thesis A Seminar Term 1, 2022 22/32

⁵Martin Fowler, "Bliki: Xunit", *martinfowler.com* (Webpage, 2022) https://martinfowler.com/bliki/Xunit.html

JUnit implementation details

Example test case goes here

Graph of pytest process goes here

JUnit pros & cons

- 1 Pros:
 - Pro 1 goes here



JUnit pros & cons

- 1 Pros:
 - Pro 1 goes here
- 2 Cons:
 - Con 1 goes here



Contents



Motivation - Student Enrolments in Introductory Course Motivation - Andrew Taylor autotest Motivation - Technical Debt

2 Literature Review

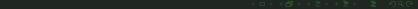
Background - Code Marking in Introductory Courses
Background - Automated Testing & Marking Approache
Existing Work - Andrew Taylor autotest
Existing Work - Harvard Uni CS50 check50
Existing Work - Google gtest

3 Design

Design Requirements Proposed Design

4 Schedule & Conclusion

Summary



Design Requirements

The design requirements for the solution will follow along



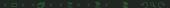
Exclusions

Blah Blah Blah



Proposed Design

Blah Blah Blah



Contents

- Introduction
 - Motivation Student Enrolments in Introductory Course Motivation - Andrew Taylor autotest Motivation - Technical Debt
- 2 Literature Review
 - Background Code Marking in Introductory Courses
 Background Automated Testing & Marking Approache
 Existing Work Andrew Taylor autotest
 Existing Work Harvard Uni CS50 check50
 Existing Work Google gtest
 Existing Work IUnit
- 3 Design
 Design Requirement
 Proposed Design
- 4 Schedule & Conclusion Schedule Summary References



Schedule

- Rest of thesis A:
 - Continue inspection of Andrew Taylor autotest implementation



- Rest of thesis A:
 - Continue inspection of Andrew Taylor autotest implementation
 - Complete Draft Design Document



- Rest of thesis A:
 - Continue inspection of Andrew Taylor autotest implementation
 - Complete Draft Design Document
 - Collect feedback on Design Document and make adjustments as necessary



- Rest of thesis A:
 - Continue inspection of Andrew Taylor autotest implementation
 - Complete Draft Design Document
 - Collect feedback on Design Document and make adjustments as necessary
- 2 Thesis B:
 - Implement Core Main Module



Schedule

- Rest of thesis A:
 - Continue inspection of Andrew Taylor autotest implementation
 - Complete Draft Design Document
 - Collect feedback on Design Document and make adjustments as necessary
- 2 Thesis B:
 - Implement Core Main Module
 - Implement Core Testcase Parser Module

Schedule

- Rest of thesis A:
 - Continue inspection of Andrew Taylor autotest implementation
 - Complete Draft Design Document
 - Collect feedback on Design Document and make adjustments as necessary
- 2 Thesis B:
 - Implement Core Main Module
 - Implement Core Testcase Parser Module
 - Implement Core Testcase Runner Module

- Rest of thesis A:
 - Continue inspection of Andrew Taylor autotest implementation
 - Complete Draft Design Document
 - Collect feedback on Design Document and make adjustments as necessary
- 2) Thesis B:
 - Implement Core Main Module
 - Implement Core Testcase Parser Module
 - Implement Core Testcase Runner Module
 - Run correctness and performance testing on Parser and Runner

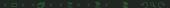
- Rest of thesis A:
 - Continue inspection of Andrew Taylor autotest implementation
 - Complete Draft Design Document
 - Collect feedback on Design Document and make adjustments as necessary
- 2) Thesis B:
 - Implement Core Main Module
 - Implement Core Testcase Parser Module
 - Implement Core Testcase Runner Module
 - Run correctness and performance testing on Parser and Runner
- 3 Thesis C:
 - Implement Core Testcase Program Correctness Module

- Rest of thesis A:
 - Continue inspection of Andrew Taylor autotest implementation
 - Complete Draft Design Document
 - Collect feedback on Design Document and make adjustments as necessary
- 2) Thesis B:
 - Implement Core Main Module
 - Implement Core Testcase Parser Module
 - Implement Core Testcase Runner Module
 - Run correctness and performance testing on Parser and Runner
- 3 Thesis C:
 - Implement Core Testcase Program Correctness Module
 - Implement any extensions that have been deemed necessary by the Design Document

- Rest of thesis A:
 - Continue inspection of Andrew Taylor autotest implementation
 - Complete Draft Design Document
 - Collect feedback on Design Document and make adjustments as necessary
- 2) Thesis B:
 - Implement Core Main Module
 - Implement Core Testcase Parser Module
 - Implement Core Testcase Runner Module
 - Run correctness and performance testing on Parser and Runner
- 3 Thesis C:
 - Implement Core Testcase Program Correctness Module
 - Implement any extensions that have been deemed necessary by the Design Document
 - Run correctness and performance testing on complete package and make final adjustments

Summary

Blah Blah Blah



References

- 1 Tom Edith, Aybüke Aurum and Richard Vidgen, 'An Exploration of Technical Debt' (2013) 86(6) The Journal of systems and software 1498
- 2 Cunningham, Ward, 'The WyCash Portfolio Management System' in Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (addendum) (ACM, 1992) 29
- Martin Fowler, "Bliki: Xunit", martinfowler.com (Webpage, 2022) https://martinfowler.com/bliki/Xunit.html