A Testing Tool for Introductory Programming Courses

Thesis B Seminar

Kyu-Sang Kim
z5208931

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

Term 2, 2022

Contents

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:

1. 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:

1. Develop an extensible and easy to use software package which parses and runs pre-written tests on submitted code
2. Implement development procedures that minimise both current and future technical debt

Thesis Statement

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

We will:

1. Develop an extensible and easy to use software package which parses and runs pre-written tests on submitted code
2. Implement development procedures that minimise both current and future technical debt
3. 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:

1. Develop an extensible and easy to use software package which parses and runs pre-written tests on submitted code
2. Implement development procedures that minimise both current and future technical debt
3. Remediate known flaws in the existing autotest package
4. 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:

1. Develop an extensible and easy to use software package which parses and runs pre-written tests on submitted code
2. Implement development procedures that minimise both current and future technical debt
3. Remediate known flaws in the existing autotest package
4. Maintain backwards compatibility with legacy tests written for the existing autotest package
5. 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:

1. Develop an extensible and easy to use software package which parses and runs pre-written tests on submitted code
2. Implement development procedures that minimise both current and future technical debt
3. Remediate known flaws in the existing autotest package
4. Maintain backwards compatibility with legacy tests written for the existing autotest package
5. Perform proving and performance tests on the new software package
6. Deprecate and replace the existing autotest used for introductory programming courses at UNSW CSE

Schedule

The original plan was set to the following:

1. Thesis B:
   - Implement Core Main Module

Schedule

The original plan was set to the following:

1. Thesis B:
   - Implement Core Main Module
   - Implement Core Testcase Parser Module

Schedule

The original plan was set to the following:

1. Thesis B:
   - Implement Core Main Module
   - Implement Core Testcase Parser Module
   - Implement Core Testcase Runner Module

Schedule

The original plan was set to the following:

1. 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

Schedule

The original plan was set to the following:

1. 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
2. Thesis C:
   - Implement Core Testcase Program Correctness Module

Schedule

The original plan was set to the following:

1. 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
2. Thesis C:
   - Implement Core Testcase Program Correctness Module
   - Implement any extensions that have been deemed necessary

Schedule

The original plan was set to the following:

1. 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
2. Thesis C:
   - Implement Core Testcase Program Correctness Module
   - Implement any extensions that have been deemed necessary
   - Run correctness and performance testing on complete package and make final adjustments

Schedule

The original plan was set to the following:

1. 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
2. Thesis C:
   - Implement Core Testcase Program Correctness Module
   - Implement any extensions that have been deemed necessary
   - Run correctness and performance testing on complete package and make final adjustments

This presentation will discuss work done for Thesis B.

Contents

Summary

The majority of thesis B was on enabling development.
This involved:

- Designing the core architecture

Summary

The majority of thesis B was on enabling development.
This involved:

- Designing the core architecture
- Porting over legacy code to modules for the new architecture

Summary

The majority of thesis B was on enabling development.
This involved:

- Designing the core architecture
- Porting over legacy code to modules for the new architecture
- Designing and implementing the new Test definition

Summary

The majority of thesis B was on enabling development.
This involved:

- Designing the core architecture
- Porting over legacy code to modules for the new architecture
- Designing and implementing the new Test definition
- Designing and implementing the new Thesis B modules

Summary

The majority of thesis B was on enabling development.
This involved:

- Designing the core architecture
- Porting over legacy code to modules for the new architecture
- Designing and implementing the new Test definition
- Designing and implementing the new Thesis B modules
- Running performance and correctness tests on legacy, ported and new versions of the same modules

Summary

The majority of thesis B was on enabling development.
This involved:

- Designing the core architecture
- Porting over legacy code to modules for the new architecture
- Designing and implementing the new Test definition
- Designing and implementing the new Thesis B modules
- Running performance and correctness tests on legacy, ported and new versions of the same modules
- Setting up an automated testing and styling infrastructure

Modular Design

- A modular design has been selected for the new autotest

Modular Design

- A modular design has been selected for the new autotest
- Modules are an established design pattern that fits the purpose of autotest

Modular Design

- A modular design has been selected for the new autotest
- Modules are an established design pattern that fits the purpose of autotest
- Enables components of high complexity to be rendered mostly if not completely separate from the internal implementations of other components

Modular Design

- A modular design has been selected for the new autotest
- Modules are an established design pattern that fits the purpose of autotest
- Enables components of high complexity to be rendered mostly if not completely separate from the internal implementations of other components
- Each component will be inclined to do only **one** task well

Modular Design

- A modular design has been selected for the new autotest
- Modules are an established design pattern that fits the purpose of autotest
- Enables components of high complexity to be rendered mostly if not completely separate from the internal implementations of other components
- Each component will be inclined to do only **one** task well
- Enables easier swapping of components with minimal to no code changes other than import statements

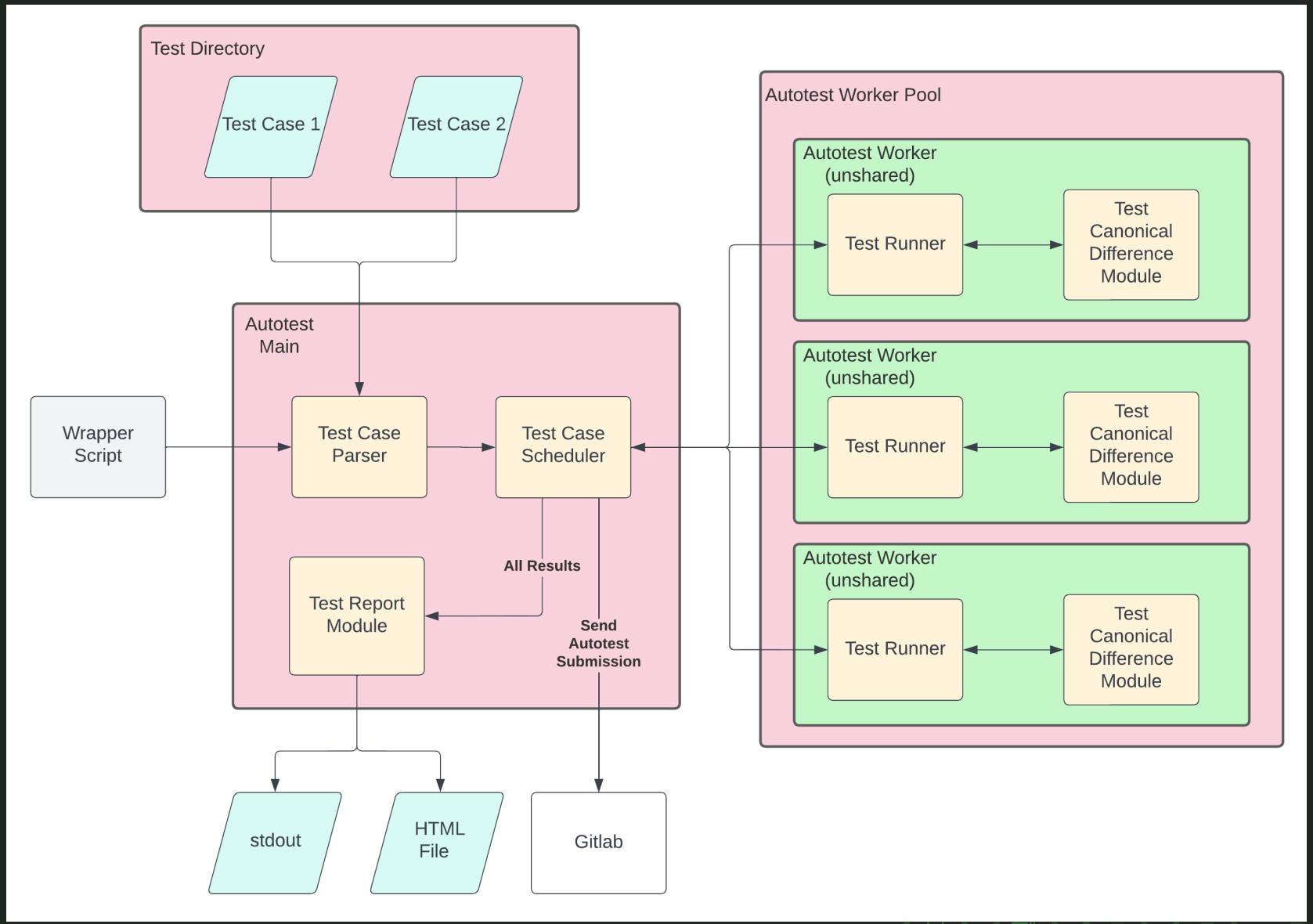
Modular Design

- A modular design has been selected for the new autotest
- Modules are an established design pattern that fits the purpose of autotest
- Enables components of high complexity to be rendered mostly if not completely separate from the internal implementations of other components
- Each component will be inclined to do only **one** task well
- Enables easier swapping of components with minimal to no code changes other than import statements
- Maintenance and Improvements can be tested without an overhaul

Modular Design

- A modular design has been selected for the new autotest
- Modules are an established design pattern that fits the purpose of autotest
- Enables components of high complexity to be rendered mostly if not completely separate from the internal implementations of other components
- Each component will be inclined to do only **one** task well
- Enables easier swapping of components with minimal to no code changes other than import statements
- Maintenance and Improvements can be tested without an overhaul
- Abstract Class design allows for easier novel design and implementation of pre-existing modules

Architecture Diagram



Core Module

Purpose: The "Main" Program

- Coordinates execution of modules

Core Module

Purpose: The "Main" Program

- Coordinates execution of modules
- Facilitates information transfer between modules

Core Module

Purpose: The "Main" Program

- Coordinates execution of modules
- Facilitates information transfer between modules
- Exposed only to the abstraction of each module

Core Module

Purpose: The “Main” Program

- Coordinates execution of modules
- Facilitates information transfer between modules
- Exposed only to the abstraction of each module
- Direct code only for optional “administrative” tasks

Parser Module

Purpose: Argument and Test Case Parser

- Ingests autotest parameters for processing into usable information

Parser Module

Purpose: Argument and Test Case Parser

- Ingests autotest parameters for processing into usable information
- Retrieves relevant autotest test cases and processes them into test definitions

Parser Module

Purpose: Argument and Test Case Parser

- Ingests autotest parameters for processing into usable information
- Retrieves relevant autotest test cases and processes them into test definitions
- Minimised exposure to test definition

Parser Module

Purpose: Argument and Test Case Parser

- Ingests autotest parameters for processing into usable information
- Retrieves relevant autotest test cases and processes them into test definitions
- Minimised exposure to test definition
- Most complex component in the entire package

Parser Module

Purpose: Argument and Test Case Parser

- Ingests autotest parameters for processing into usable information
- Retrieves relevant autotest test cases and processes them into test definitions
- Minimised exposure to test definition
- Most complex component in the entire package

Changes:

- Legacy parser code carries too much technical debt for a refactor (circular dependency hell etc.)

Parser Module

Purpose: Argument and Test Case Parser

- Ingests autotest parameters for processing into usable information
- Retrieves relevant autotest test cases and processes them into test definitions
- Minimised exposure to test definition
- Most complex component in the entire package

Changes:

- Legacy parser code carries too much technical debt for a refactor (circular dependency hell etc.)
- Writing a new one is unrealistic due to possible correctness concerns

Parser Module

Purpose: Argument and Test Case Parser

- Ingests autotest parameters for processing into usable information
- Retrieves relevant autotest test cases and processes them into test definitions
- Minimised exposure to test definition
- Most complex component in the entire package

Changes:

- Legacy parser code carries too much technical debt for a refactor (circular dependency hell etc.)
- Writing a new one is unrealistic due to possible correctness concerns
- **Scope Change - Salvaged/Ported legacy + adaptor**

Parser Module

Purpose: Argument and Test Case Parser

- Ingests autotest parameters for processing into usable information
- Retrieves relevant autotest test cases and processes them into test definitions
- Minimised exposure to test definition
- Most complex component in the entire package

Changes:

- Legacy parser code carries too much technical debt for a refactor (circular dependency hell etc.)
- Writing a new one is unrealistic due to possible correctness concerns
- **Scope Change - Salvaged/Ported legacy + adaptor**

Status: Completed

Test Case Definition

Purpose: Defines a test and the information contained within it

- Used by the parser to generate the test definitions to be scheduled and executed

Test Case Definition

Purpose: Defines a test and the information contained within it

- Used by the parser to generate the test definitions to be scheduled and executed
- Contains all information involving each test such as required files, pre-checks etc.

Test Case Definition

Purpose: Defines a test and the information contained within it

- Used by the parser to generate the test definitions to be scheduled and executed
- Contains all information involving each test such as required files, pre-checks etc.
- Easily expanded for new features per design

Test Case Definition

Purpose: Defines a test and the information contained within it

- Used by the parser to generate the test definitions to be scheduled and executed
- Contains all information involving each test such as required files, pre-checks etc.
- Easily expanded for new features per design

Status: Semi-Completed - Awaiting Possible Modifications

Test Case Scheduler Module

Purpose: Test runner pool manager and test allocator

- Ingests parameters to initialise a container pool that enables concurrent execution of tests

Test Case Scheduler Module

Purpose: Test runner pool manager and test allocator

- Ingests parameters to initialise a container pool that enables concurrent execution of tests
- Schedules ingested test definitions to free workers in container pool via producer-consumer pattern with synchronisation primitives

Test Case Scheduler Module

Purpose: Test runner pool manager and test allocator

- Ingests parameters to initialise a container pool that enables concurrent execution of tests
- Schedules ingested test definitions to free workers in container pool via producer-consumer pattern with synchronisation primitives
- Manages worker instances and cleanup on completion/process signal

Test Case Scheduler Module

Purpose: Test runner pool manager and test allocator

- Ingests parameters to initialise a container pool that enables concurrent execution of tests
- Schedules ingested test definitions to free workers in container pool via producer-consumer pattern with synchronisation primitives
- Manages worker instances and cleanup on completion/process signal
- Expected to deliver the most performance benefit compared to legacy by eliminating Single Instruction, Single Data (SISD) limitation (AKA single-thread, single-process)

Test Case Scheduler Module

Purpose: Test runner pool manager and test allocator

- Ingests parameters to initialise a container pool that enables concurrent execution of tests
- Schedules ingested test definitions to free workers in container pool via producer-consumer pattern with synchronisation primitives
- Manages worker instances and cleanup on completion/process signal
- Expected to deliver the most performance benefit compared to legacy by eliminating Single Instruction, Single Data (SISD) limitation (AKA single-thread, single-process)

Status: In Progress

Test Case Runner Module

Purpose: Container management and test runner

- Initialises a worker which accepts test definitions until no more can be found or signal sent

Test Case Runner Module

Purpose: Container management and test runner

- Initialises a worker which accepts test definitions until no more can be found or signal sent
- Utilises *unshare* to implement containerised execution of given test

Test Case Runner Module

Purpose: Container management and test runner

- Initialises a worker which accepts test definitions until no more can be found or signal sent
- Utilises *unshare* to implement containerised execution of given test
- Manages container environment setup and cleanup for each test

Test Case Runner Module

Purpose: Container management and test runner

- Initialises a worker which accepts test definitions until no more can be found or signal sent
- Utilises *unshare* to implement containerised execution of given test
- Manages container environment setup and cleanup for each test
- Expected to deliver on security needs

Test Case Runner Module

Purpose: Container management and test runner

- Initialises a worker which accepts test definitions until no more can be found or signal sent
- Utilises *unshare* to implement containerised execution of given test
- Manages container environment setup and cleanup for each test
- Expected to deliver on security needs

Changes:

- **Scope Change - Compartmentalisation is now included in scope**

Test Case Runner Module

Purpose: Container management and test runner

- Initialises a worker which accepts test definitions until no more can be found or signal sent
- Utilises *unshare* to implement containerised execution of given test
- Manages container environment setup and cleanup for each test
- Expected to deliver on security needs

Changes:

- **Scope Change - Compartmentalisation is now included in scope**

Status: In Progress

Why *unshare*?

Purpose: *unshare* is a Linux kernel system call introduced in 2008 that allows a process to *disassociate* some components of it's inherited process context

Why *unshare*?

Purpose: *unshare* is a Linux kernel system call introduced in 2008 that allows a process to *disassociate* some components of it's inherited process context

In the case of autotest, *unshare* allows processes to *disassociate* a *user namespace* and replace it with a subordinate that cannot "see" the original

Why *unshare*?

Purpose: *unshare* is a Linux kernel system call introduced in 2008 that allows a process to *disassociate* some components of it's inherited process context

In the case of autotest, *unshare* allows processes to *disassociate* a *user namespace* and replace it with a subordinate that cannot "see" the original

The same can also be done for network access, file system mounts etc.

Why *unshare*?

What does *unshare* solve?

Why *unshare*?

What does *unshare* solve?

- Many courses run autotest with their class account (cs1521) when marking which has led to concerns of malicious programs inheriting an overly privileged user id

Why *unshare*?

What does *unshare* solve?

- Many courses run autotest with their class account (cs1521) when marking which has led to concerns of malicious programs inheriting an overly privileged user id
- With *unshare*, a very lightweight fakeroot container can be created for an **autotest worker** to execute tests to eliminate most risks of damage on the host file system **without needing root access (rootless)**

Why *unshare*?

What does *unshare* solve?

- Many courses run autotest with their class account (cs1521) when marking which has led to concerns of malicious programs inheriting an overly privileged user id
- With *unshare*, a very lightweight fakeroot container can be created for an **autotest worker** to execute tests to eliminate most risks of damage on the host file system **without needing root access (rootless)**

Why not alternatives?

Why *unshare*?

What does *unshare* solve?

- Many courses run autotest with their class account (cs1521) when marking which has led to concerns of malicious programs inheriting an overly privileged user id
- With *unshare*, a very lightweight fakeroot container can be created for an **autotest worker** to execute tests to eliminate most risks of damage on the host file system **without needing root access (rootless)**

Why not alternatives?

- Most of the Linux container engines available (**Docker**, **Bubblewrap** etc.) have too much overhead and are far too powerful for the job

Why *unshare*?

What does *unshare* solve?

- Many courses run autotest with their class account (cs1521) when marking which has led to concerns of malicious programs inheriting an overly privileged user id
- With *unshare*, a very lightweight fakeroot container can be created for an **autotest worker** to execute tests to eliminate most risks of damage on the host file system **without needing root access (rootless)**

Why not alternatives?

- Most of the Linux container engines available (**Docker**, **Bubblewrap** etc.) have too much overhead and are far too powerful for the job
- Although rootless versions of some engines exist, it is considered to be "inferior" and task-specific support/documentation has been minimal

Why *unshare*?

What does *unshare* solve?

- Many courses run autotest with their class account (cs1521) when marking which has led to concerns of malicious programs inheriting an overly privileged user id
- With *unshare*, a very lightweight fakeroot container can be created for an **autotest worker** to execute tests to eliminate most risks of damage on the host file system **without needing root access (rootless)**

Why not alternatives?

- Most of the Linux container engines available (**Docker**, **Bubblewrap** etc.) have too much overhead and are far too powerful for the job
- Although rootless versions of some engines exist, it is considered to be "inferior" and task-specific support/documentation has been minimal
- Most engines wrap around *unshare* to deliver capabilities

Test Case Canonical Difference Module

Purpose: Test correctness checker and test case report generation

- Compares output of executed test to expected output

Test Case Canonical Difference Module

Purpose: Test correctness checker and test case report generation

- Compares output of executed test to expected output
- Reports success when output matches expected

Test Case Canonical Difference Module

Purpose: Test correctness checker and test case report generation

- Compares output of executed test to expected output
- Reports success when output matches expected
- Reports failure when out does not match expected with a report on differences and test reproduction information

Test Case Canonical Difference Module

Purpose: Test correctness checker and test case report generation

- Compares output of executed test to expected output
- Reports success when output matches expected
- Reports failure when out does not match expected with a report on differences and test reproduction information
- Should report similar if not better information than what legacy currently provides

Test Case Canonical Difference Module

Purpose: Test correctness checker and test case report generation

- Compares output of executed test to expected output
- Reports success when output matches expected
- Reports failure when out does not match expected with a report on differences and test reproduction information
- Should report similar if not better information than what legacy currently provides

Status: Planned

Test Case Report Module

Purpose: Optional HTML report generation

- Converts autotest output to more readable HTML form

Test Case Report Module

Purpose: Optional HTML report generation

- Converts autotest output to more readable HTML form
- Assists in course administration with forums by making output much more easier to copy by students

Test Case Report Module

Purpose: Optional HTML report generation

- Converts autotest output to more readable HTML form
- Assists in course administration with forums by making output much more easier to copy by students

Status: Planned as extension - could be dropped

Demo

- Main module is regulating how the parser and test case scheduler modules interact
- Legacy Parser has been cut down, revamped and ported over to the new architecture
- Observe that tests are being parsed correctly and from the correct directory

What was supposed to happen

If we had implemented all of Thesis B as originally planned:

- Main module initialises the test scheduler with parsed parameters

What was supposed to happen

If we had implemented all of Thesis B as originally planned:

- Main module initialises the test scheduler with parsed parameters
- Test scheduler spins up a parameterised number of autotest workers

What was supposed to happen

If we had implemented all of Thesis B as originally planned:

- Main module initialises the test scheduler with parsed parameters
- Test scheduler spins up a parameterised number of autotest workers
- Each autotest worker will create an execution environment with *unshare* and await tests

What was supposed to happen

If we had implemented all of Thesis B as originally planned:

- Main module initialises the test scheduler with parsed parameters
- Test scheduler spins up a parameterised number of autotest workers
- Each autotest worker will create an execution environment with *unshare* and await tests
- Main module queues tests in an array to be consumed by free workers

What was supposed to happen

If we had implemented all of Thesis B as originally planned:

- Main module initialises the test scheduler with parsed parameters
- Test scheduler spins up a parameterised number of autotest workers
- Each autotest worker will create an execution environment with *unshare* and await tests
- Main module queues tests in an array to be consumed by free workers
- Autotest worker consumes test by setting up any pre-test requirements, executing the test in *unshare* environment, cleaning up and populating test instance with produced output

What was supposed to happen

If we had implemented all of Thesis B as originally planned:

- Main module initialises the test scheduler with parsed parameters
- Test scheduler spins up a parameterised number of autotest workers
- Each autotest worker will create an execution environment with *unshare* and await tests
- Main module queues tests in an array to be consumed by free workers
- Autotest worker consumes test by setting up any pre-test requirements, executing the test in *unshare* environment, cleaning up and populating test instance with produced output
- Autotest worker sends back populated test back to Main module

What was supposed to happen

If we had implemented all of Thesis B as originally planned:

- Main module initialises the test scheduler with parsed parameters
- Test scheduler spins up a parameterised number of autotest workers
- Each autotest worker will create an execution environment with *unshare* and await tests
- Main module queues tests in an array to be consumed by free workers
- Autotest worker consumes test by setting up any pre-test requirements, executing the test in *unshare* environment, cleaning up and populating test instance with produced output
- Autotest worker sends back populated test back to Main module
- Main module makes basic output comparison to determine if test has passed or failed before printing result to stdout

Contents

What went wrong?

- Initial goal to refactor the test parser was more challenging than originally anticipated

What went wrong?

- Initial goal to refactor the test parser was more challenging than originally anticipated
- Too much time was spent on the parser before scope was revised

What went wrong?

- Initial goal to refactor the test parser was more challenging than originally anticipated
- Too much time was spent on the parser before scope was revised
- My own expectations on the time I thought I had available for Thesis B were not met

What will be done?

- Scope Revisions

What will be done?

- Scope Revisions
- Higher allocation of weekly time to Thesis C

Scope Revision

To make thesis C more realistic with the remaining time, the following changes have been made:

- Removal of the following clause from the thesis statement:
  - Deprecate and replace the existing autotest used for introductory programming courses at UNSW CSE

Scope Revision

To make thesis C more realistic with the remaining time, the following changes have been made:

- Removal of the following clause from the thesis statement:
  - Deprecate and replace the existing autotest used for introductory programming courses at UNSW CSE
- Refactoring of the legacy parser is to be considered “out of scope” for this thesis

Thesis C Plan

- Implement remaining work for Test Definition, Test Scheduler and Autotest Worker

Thesis C Plan

- Implement remaining work for Test Definition, Test Scheduler and Autotest Worker
- Finalise location and concrete implementation of Canonical Difference module

Thesis C Plan

- Implement remaining work for Test Definition, Test Scheduler and Autotest Worker
- Finalise location and concrete implementation of Canonical Difference module
- Assess whether making the optional Report module will be viable within the remaining time

Thesis C Plan

- Implement remaining work for Test Definition, Test Scheduler and Autotest Worker
- Finalise location and concrete implementation of Canonical Difference module
- Assess whether making the optional Report module will be viable within the remaining time
- Perform correctness and performance testing for the final report

Thesis C Plan

- Implement remaining work for Test Definition, Test Scheduler and Autotest Worker
- Finalise location and concrete implementation of Canonical Difference module
- Assess whether making the optional Report module will be viable within the remaining time
- Perform correctness and performance testing for the final report
- Finalise Thesis C demo plan and slides

Thesis C Plan

- Implement remaining work for Test Definition, Test Scheduler and Autotest Worker
- Finalise location and concrete implementation of Canonical Difference module
- Assess whether making the optional Report module will be viable within the remaining time
- Perform correctness and performance testing for the final report
- Finalise Thesis C demo plan and slides
- Finalise and submit Thesis C report

Thesis C Plan

- Implement remaining work for Test Definition, Test Scheduler and Autotest Worker
- Finalise location and concrete implementation of Canonical Difference module
- Assess whether making the optional Report module will be viable within the remaining time
- Perform correctness and performance testing for the final report
- Finalise Thesis C demo plan and slides
- Finalise and submit Thesis C report

Thank you for attending! *Questions?*