A Testing Tool for Introductory Programming Courses

Thesis C Seminar

Kyu-Sang Kim
z5208931

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

Term 3, 2022

Contents

Autotest - Introduction

- Autotest is a general code testing tool developed by Andrew Taylor
- First introduced to UNSW in COMP2041 2015 S2
- Currently used to perform automated marking for most UNSW introductory programming courses which has become mandatory to reduce staff workload with increasing enrolment numbers
  - **COMP1521:** 715 (2017) $\rightarrow$ 1633 (2021)
- Written in Python (+ some Shell and Perl script interfaces)
- Has been in use for many sessions with different courses which has led to an abundance of example test cases for a multitude of assignment types

Autotest - Issues

- Has some **design deficiencies** and accumulated non-trivial **technical debt** since introduction:
  - Single-thread design does not utilise all potential processing power that is available to autotest
  - Vulnerable to unintended or malicious actions when autotesting user-provided programs
  - Design is highly coupled with low cohesion making improvements significantly more difficult to implement
  - Documentation in regard to crucial autotest components were lacking or non-existent which made it difficult to understand
- A refactoring or re-write of the autotest package would provide great benefit to the education of UNSW CSE students

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

Contents

Design Considerations

The design requirements of the solution emphasised the following properties in accordance with the thesis goals:

- *Accessible* - Users of the new software package should have the easiest possible experience in integrating automatic testing and grading into their courses
- *Familiar* - Users who have previously utilised Andrew Taylor autotest should not feel the new software package to be completely independent of the former
- *Performant/Efficient* - The new software package should have the same, if not better performance than the original Andrew Taylor autotest overall

Design Considerations - continued

- *Maintainable* - The new software package should be adequately documented and support ease of maintainability and possible extension of features via the appropriate architectural decisions
- *Secure* - The new software package should provide reasonable security measures to ensure that any damage by the package on the host system is minimised as much as possible

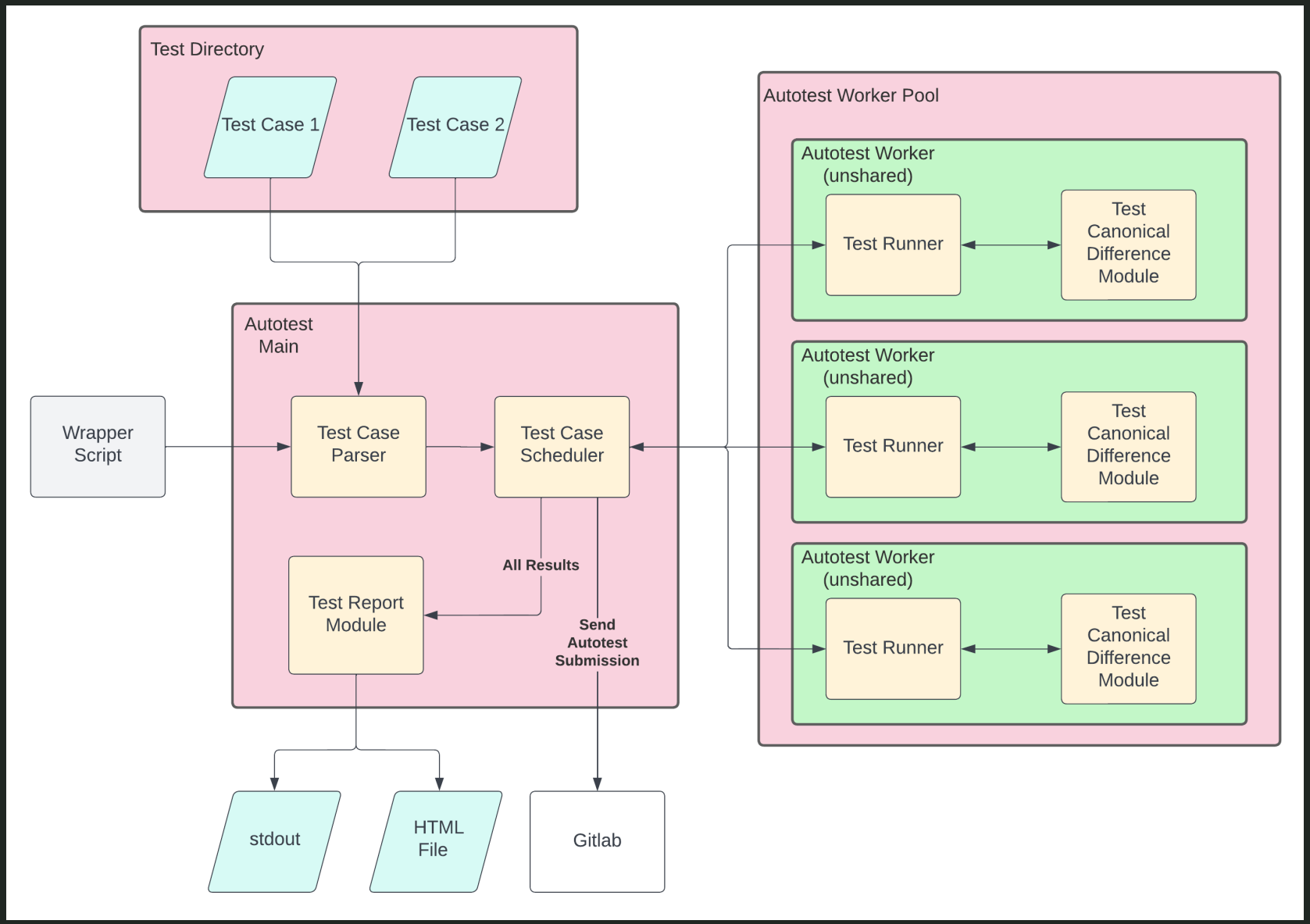
Exclusions

- *Novelty* - We assume novelty to be out of scope as introducing a novel user experience is in conflict with the aforementioned design requirements
- Users of the existing Andrew Taylor autotest may elect to not utilise the new software package if the transition requires significant work

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 easier swapping of components with minimal to no code changes other than import statements
- Maintenance and Improvements can be tested without a complete overhaul of autotest
- 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
- Facilitates information transfer between modules
- Exposed only to the abstraction of each module
- Direct code only for optional "administrative" tasks
  - Upload submission to UNSW GitLab
  - Upload autotest results to UNSW CSE

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 time and correctness concerns

Status: Salvaged/Ported legacy + adaptor

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, execution instructions etc.
- Used by the runner to execute the test itself
- Easily expanded for new features and improvements

Status: Completed - Improvements (further decoupling etc.)

Test Case Scheduler Module

Purpose: Test runner pool manager and test allocator

- Utilises given parameters to initialise and setup 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
- Delivers the most performance benefit by enabling parallel execution of tests which decreases testing time when additional CPU threads are available compared to legacy that will only ever utilise one thread

Status: Completed

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 novel pure-python container implementation to enable compartmentalised execution of a given test
- Manages container environment setup and cleanup for each test
- Expected to deliver on security needs as containerisation minimises the damage to the host system in the event unintended or malicious actions are performed by a test case

Status: Completed - Improvements (further configuration etc.)

Why do we need containerisation?

- 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
- We can fix this by having a **autotest worker** execute tests in an isolated container to eliminate most risks of damage on the host file system
- In the event the container is "escaped", we would like to have containerisation **without the need for root access (rootless)** to limit the damage associated with inheriting root permissions

Why a novel container implementation?

In most cases, writing your own container implementation is a terrible idea

For this project, a suitable existing container engine could not be found:

- Docker, Bubblewrap etc. have too much overhead and are too powerful for the job
- Rootless container engines exist but they are considered to be inferior and task-specific support/documentation has been minimal
- Various pure-python container projects rely on the *setns(2)* system call which requires root permissions

Most of these container engine projects wrap around the *unshare(2)* system call to deliver capabilities

Novel container details

In the interest of time, only an overview of sandbox will be presented
Specific details will be available in the final report

- Written entirely in Python and easily used via *with* statement
- Rootless container engine with no visible alternatives on Github
- Provides *chroot* functionality with additional isolation provided by namespaces via *unshare(2)*
- Container file system is mostly isolated from host with configuration options available to expose additional directories as both read-only or read-write
- Sandbox namespace isolation is irreversible for a specific Python interpreter process due to it's rootless nature as *setns(2)* requires root

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 output 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: Semi-Completed - Coupled with Test Definition but decoupled as much as possible in a semi-modular fashion (isolated into it's own function)

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: Dropped - Easily implemented into the Core module (suitable implementation area already marked)

Contents

Demonstration

Let's see the new autotest (interim name: lemontest) in action on the following compared to legacy:

- **COMP1521 C Exercise:** float_2048
- **COMP1521 MIPS Exercise:** ackermann
- **COMP1521 MIPS Assignment:** connect_four (no comparison to legacy as it is not available)
- **Local pytest on standard tests:** Ideally automated but GitHub Actions did not support proper execution of lemontest

Evaluation - Test Environment

- The server used for evaluating preliminary results is the UNSW CSE *nw-k17-login1* server with 4 available CPU threads
- Test cases are identical with only the introduction of lemontest exclusive parameters such as *worker_count* and other parameters that are necessary for test execution
- The preliminary evaluation is only in respect to the float_2048 C exercise with more formalised results to come in the report
- Performance gains from using lemontest depend on the hardware and the resources available to the executing user (little performance gain expected from parallelisation if only one thread is available)

Evaluation - Error-free Programs

For testing of programs that compile correctly and have no runtime issues for the float_2048 C exercise (averaged over 10 executions):

- A testing time **reduction** of **58.5%** is observed with lemontest over 11 test cases **(51.305s vs 123.831s)**
- A testing time **reduction** of **12.8%** is observed with lemontest over 1 test case **(26.034s vs 29.875s)**
- The less than expected performance gain for both cases when compared to the legacy autotest implementation is attributed to overhead introduced with lemontest via containerisation of tests
- It is noted from Shrey Somaiya, “Can we gain performance by disabling containerisation?”, a testing time **increase** of **7.1%** was observed when disabling containerisation for lemontest **(54.985s vs 51.305s)**

Evaluation - Runtime Error Programs

For testing of programs that compile correctly but have runtime errors for the float_2048 C exercise:

- The massive increase in testing time has been attributed to an issue with how DCC compiled binaries work on CSE
- The issue itself has been identified but it seems to only occur on CSE and was not able to be reproduced locally
- DCC issues are out of scope for this thesis so a reasonable evaluation on performance could not be performed for programs with runtime errors

Contents

Analysis & Contributions

- Transition testing ready implementation of new autotest package
- User experience for course staff is very similar to legacy autotest with minimal changes required for test cases **(2-3 lines on average per exercise)**
- User experience for students is very similar to legacy autotest with slight output differences reflecting the isolated and parallelised execution of each test case
- Designed for ease of understanding and to support feature extensibility
- Provides better overall performance in the general use case with additional computing resources

Analysis & Contributions - continued

- Unexpected setbacks led to some less-used features being unimplemented but feature parity has been achieved for most use cases
- Documentation is more extensive than legacy autotest
  - Significant commenting of code to assist ease of understanding
  - Additional supporting documentation that will assist new courses to consider integrating autotest to reduce manual marking fatigue and minimise errors

Near Future: Rest of Thesis C

- Complete documentation of the new autotest package
- Clean up code and fix any outstanding known bugs
- Implement any extension or low-priority features if time permits
- Write the thesis report, with further evaluation and analysis

Foreseeable Future: To Complete

- Feature improvements to containerisation
  - cgroups v2 support
  - File mounting (only directories supported at the current time)
  - Customised mount targets for both directories and files
  - Improved containerisation roll-back functionality (currently impossible)
- Further decoupling of canonical difference module from the test definition class
- Verify fixes and performance results once DCC issue has been fixed in regard to delayed process exit
- Transition UNSW CSE to the new autotest package for further testing within a course like COMP1521 such that any hidden issues can be identified and rectified as necessary

Distant Future

- Overhaul of the legacy test case parser module
- Overhaul of the subprocess protocol for execution of tests (unlikely to have much improvement)
- Autotest platform upgraded to become cloud-based such that autotest is accessible to all students on the internet rather than requiring direct access to CSE servers
- Propagation of autotest package to other educational institutions (legacy autotest is rumoured to have been shared with Western Sydney University)

Summary

We have covered:

- Andrew Taylor autotest and it's issues
- The thesis statement
- Design considerations made for the new autotest
- Design approach and core architecture for the new autotest
- Modules that combine to provide functionality for the new autotest
- Demonstration and preliminary evaluation of the new autotest
- Analysis and contributions of the new autotest
- What's left for thesis C and the future

Thank you for attending! *Questions?*