Extensible Protocol Development and Load Testing suite

Abstract

The problem being discussed is an investigation into whether extensible/more generic load testing systems can be created with a high throughput (using multi-threading) and user-decided configurability; without lacking any features of some currently available commercial load-testers. It is established through implementation and testing that a user-made protocol and communication plugin, with an XML testcase, using multiple threads can indeed be made. While the nice features of the more commercial solutions *are* lacking, the core of the solution is in place and other features can be added at a later date.

Table of Contents

[Introduction 4](#__RefHeading___Toc58313_1576207282)

[Research 4](#__RefHeading___Toc91753_641238142)

[Requirements & Acceptance Tests 11](#__RefHeading___Toc64587_2741984178)

[Acceptance Tests and Requirements Overview 11](#__RefHeading___Toc51762_1576207282)

[Requirements 13](#__RefHeading___Toc51764_1576207282)

[Themes 13](#__RefHeading___Toc51766_1576207282)

[Epics 13](#__RefHeading___Toc51768_1576207282)

[THEME 1 Epics (Plugin Manager) 13](#__RefHeading___Toc51770_1576207282)

[THEME 2 Epics (Threading Facility) 14](#__RefHeading___Toc51772_1576207282)

[THEME 3 Epics (Service Handler) 14](#__RefHeading___Toc51774_1576207282)

[THEME 4 Epics (API) 15](#__RefHeading___Toc51776_1576207282)

[THEME 5 Epics (Testcase Analyser) 15](#__RefHeading___Toc51778_1576207282)

[THEME 6 Epics (Test Runner) 15](#__RefHeading___Toc51780_1576207282)

[THEME 7 Epics (Logger) 15](#__RefHeading___Toc51782_1576207282)

[THEME 8 Epics (Database) 16](#__RefHeading___Toc51784_1576207282)

[THEME 9 Epics (Extra Server Comms) 16](#__RefHeading___Toc51786_1576207282)

[User Stories 16](#__RefHeading___Toc51788_1576207282)

[Acceptance Tests 24](#__RefHeading___Toc84682_2741984178)

[Methodology 38](#__RefHeading___Toc64589_2741984178)

[Development Methodology 38](#__RefHeading___Toc51846_1576207282)

[Designs 40](#__RefHeading___Toc91757_641238142)

[Current Designs 40](#__RefHeading___Toc84688_2741984178)

[What runs where and how (thread/tasks) 40](#__RefHeading___Toc51848_1576207282)

[Thread Pool Queue 40](#__RefHeading___Toc51850_1576207282)

[Logger 41](#__RefHeading___Toc51852_1576207282)

[Test Case Analyser 41](#__RefHeading___Toc51854_1576207282)

[Database 42](#__RefHeading___Toc51856_1576207282)

[TestHandler 43](#__RefHeading___Toc51858_1576207282)

[Implementation 44](#__RefHeading___Toc51860_1576207282)

[Log 44](#__RefHeading___Toc51862_1576207282)

[Setting Up Gtest/Gmock 44](#__RefHeading___Toc51866_1576207282)

[Project structure 44](#__RefHeading___Toc51868_1576207282)

[Moving From Raw Pointers To Special Pointers 44](#__RefHeading___Toc51870_1576207282)

[Generic Thread Pool 45](#__RefHeading___Toc51872_1576207282)

[Network Communication 45](#__RefHeading___Toc51874_1576207282)

[Logger and Testing 46](#__RefHeading___Toc51876_1576207282)

[Time Keeping/Timer 47](#__RefHeading___Toc51878_1576207282)

[Plugin Architecture 47](#__RefHeading___Toc51880_1576207282)

[Parsing Complications 48](#__RefHeading___Toc51884_1576207282)

[Running a test 49](#__RefHeading___Toc51886_1576207282)

[First run 49](#__RefHeading___Toc51888_1576207282)

[API 49](#__RefHeading___Toc51890_1576207282)

[Testing 50](#__RefHeading___Toc84696_2741984178)

[Unit Tests 50](#__RefHeading___Toc51892_1576207282)

[Acceptance Tests 50](#__RefHeading___Toc51894_1576207282)

[Evaluation 52](#__RefHeading___Toc64593_2741984178)

[What has been shown 52](#__RefHeading___Toc84700_2741984178)

[What is still outstanding 52](#__RefHeading___Toc84702_2741984178)

[Conclusion 52](#__RefHeading___Toc64595_2741984178)

[Future enhancements 52](#__RefHeading___Toc84704_2741984178)

[Into the future 52](#__RefHeading___Toc84708_2741984178)

[References 53](#__RefHeading___Toc91759_641238142)

# Introduction

There is a real need for IP traffic simulators with a more realistic flow and simpler interface than are currently used systems. In addition to this, a program that will support whatever protocol the developers wish to test against using a plugin system. It is important as the current software out there does not have particularly configurable flow, have complex interfaces, cannot be automated or have a very small subset of protocols with which they can work. Regarding the content of the project, any licensing issues will be avoided by using as few third-party libraries as possible; the ones that are used will be under FOSS licensing. Ethically, the project could help to trivialise Denial-Of-Services attacks by providing a platform by which unsavoury people could launch high traffic attacks against networks. Even using the victim’s preferred protocol to communicate and using the software to change and rotate the requests to perhaps help to mask what they are doing.

# Research

Simulating the internet proper is near impossible, due to its sheer scale, impracticality and the number of nodes needed (Sumeet Kumar and Kathleen 2017). But it is certainly possible to create software that facilitates the entire load testing process (Yan et al., 2014). It helps developers to understand their application in extreme circumstances (PR Newswire Association LLC, 2014).

Simulation has played a vital role in system verification for a long time; however, it requires a great deal of time and resources. But its usefulness and importance in software engineering has been noted in detail by many authors (Sun & Männistö, 2012). Test tools can be considered an addendum to normal development and perform a task related to but separate to direct development (Na and Huaichang, 2015). Load testing is vital to proving that web services (and other load-based systems) meet the demands of its users, both in traffic and behaviour (Draheim et al., 2006) and is as important to the testing of an application as unit testing and integration testing (Jiand & Hassan, 2015). It can help to locate issues in the code that might not appear under a smaller load. These errors are called "Load sensitive". The rate at which transactions are conducted constitutes load. Workload (described by characteristics and intensity), environment and, high-level metrics all define a load test and its results intensity refers to arrival rate and throughput of data (Malik, 2010). Sending this to a target for test purposes constitutes a Load Test (Jiang, 2015).

Stress testing is similar to workload, however, the express intention is to break the system, and to verify how it recovers (Bayan and Cangussu, 2006)(Jiand & Hassan, 2015). All of these come together in a test report (Malik, 2010).

Load testing is performed in some degree on implemented systems as opposed to the design or architecture of the system in question (Jiand & Hassan, 2015). The purpose of a load test is to help guarantee if a given system's performance is acceptable under peak conditions. This means checking response times and resource usage and helping developers, testers, managers and potential customers decide whether the system's performance is acceptable or if more work needs to be done to bring it to within acceptable limits. While all this happens there are still many other factors to consider (Zhang et al., 2011).

As focus shifts towards offering software as a service with offsite hosting (a.k.a. “The Cloud”), and datacentres, more and more emphasis must be put on testing these systems that require incredible high throughputs due to the sheer number of people accessing them and the complex actions that they perform. However, this also extends to making sure that a load tester can be provided/offered as-a-service (aaS) (Shojaee et al., 2015). Even classic server arrangements require load testing to some degree to guarantee their service is robust and reliable (Cico & Dika, 2014).

Mercury Interactive's LoadRunner was once the load testing market leader (Draheim et al., 2006). DITG, Harpoon, fudp, 2Hping, LoadUI, ApacheJmeter, and, curlLoader are currently available open source traffic generators, curlLoader has been noted for its flexibility (Bhatia et al., 2014) (Yan et al., 2014). Testmaker is another WS load tester that allows concurrency testing on Amazon EC2 and joins the open source load testers along with SoapUI and SOATest (The latter being noted for its growing popularity) (Grehan, 2005) (Yan et al., 2014) (Shojaee et al., 2015).

IBM Rational Performance Tester, SURGE and AlertSite (Hasenleithner and Ziegler, 2003) (Yan et al., 2014) are some closed source alternatives with proper support systems in place, AertSite even offers aftercare support to better help businesses understand and solve issues brought up from load testing. Yet another tool, Iometer, can be used to gauge hardware usage and test performance of storage systems without complicated testbeds (Baltazar, 1998), which provides some direct insight into the hardware side of load testing. Tests often need to be parameterised and modified, this can often be quite hard to do and means more advanced tools are needed (Draheim et al., 2006). Load test analysis methodology has been performed before (Malik, 2010) and it’s been shown that models can even be computer generated based on behaviour observation (Bayan and Cangussu, 2006). Current tools support simple test cases with a fixed sequence of actions (Draheim et al., 2006) with a limited number of scenarios (Bhatia et al., 2014)

As software shifts towards being offered “as-a-service” (-aas or Xaas). Offering some sort of load tester as a service would be wise and allow for future proofing. A Load tester will likely contain four major components: Test Receiver, Test Manager, Middleware Manager and a Test Runner:

* Test Receiver: Receives tests to run from the tester, can also monitor tests
* Test Manager: Manages queues of tests and dispatches them, gathering and merging test results
* Test Runner: Invokes the tests. Analyses the validity of results
* Middleware Manager: provides and manages test runners for use within the system

(Yan et al., 2014)

Giving the user the option of adjusting the size and rate of sending is beneficial. Existing approaches consist of increasing the query size, the number of queries or, the rate at which queries are made (Zhang et al., 2011). XML is also already used by large swathes of developers and engineers including many protocols themselves (Garg and Lavahte, 2017). JSON is not as widely supported regarding protocols and can fall over when it encounters special characters (however this is purely implementation dependent).

Some solutions do not provide workflow testing or even true load testing. They can also make documentation more difficult (Garg and Lavahte, 2017). Cross-platform running would be a boon to making sure that the most amount of people possible are reached (Garg and Lavahte, 2017).

Other software already offers many different testing approaches and services for maximum beneficial interaction (Garg and Lavahte, 2017). Allowing for scripting support of any kind would be very beneficial in making sure that the process can be automated with minimal user interaction and to better fit into an AGILE workflow (Shojaee et al., 2015). There are several limitations to load testing currently: Cost-effectiveness, increasing input size may be very costly and could force the system to continually perform the same computation (Zhang et al., 2011).

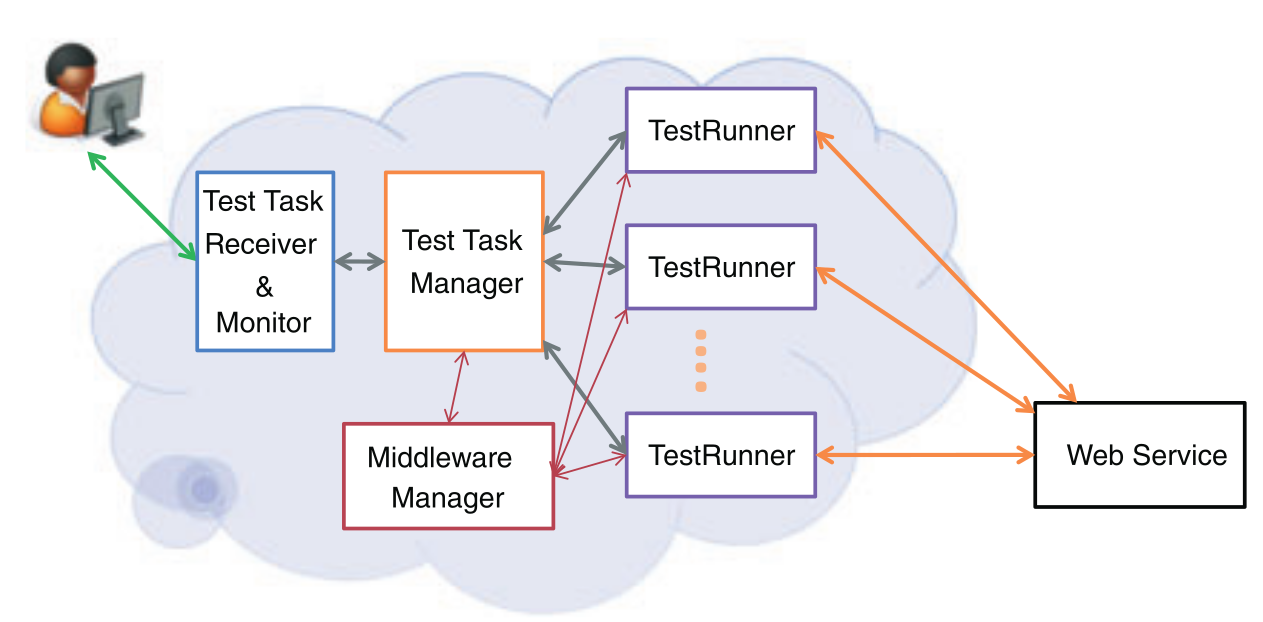


Figure 1: Theoretical layout of a web-service (Draheima and Weber, 2005)

Artificial traffic generation is often the only practical way to really verify the running of a service. Hardware traffic generation often leads to greater packet drop at the target, but it is faster than a software solution however, it also comes at a greater cost. Also, generating precise traffic allows for much greater control (Bhatia et al., 2014).

Load testing should be performed regularly to make sure that resources are correctly provisioned (Draheim et al., 2006). Different Modes need to be offered for proper testing; Static, step and maximal testing are three such modes. Static runs for a specific load. Step runs for measuring usability under a load span. Maximal load to determine upper limits (Yan et al., 2014). Load testing can turn up a variety of problems including memory leaks, error keywords, deadlocks, unhealthy system states and throughput problems (Jiand & Hassan, 2015).

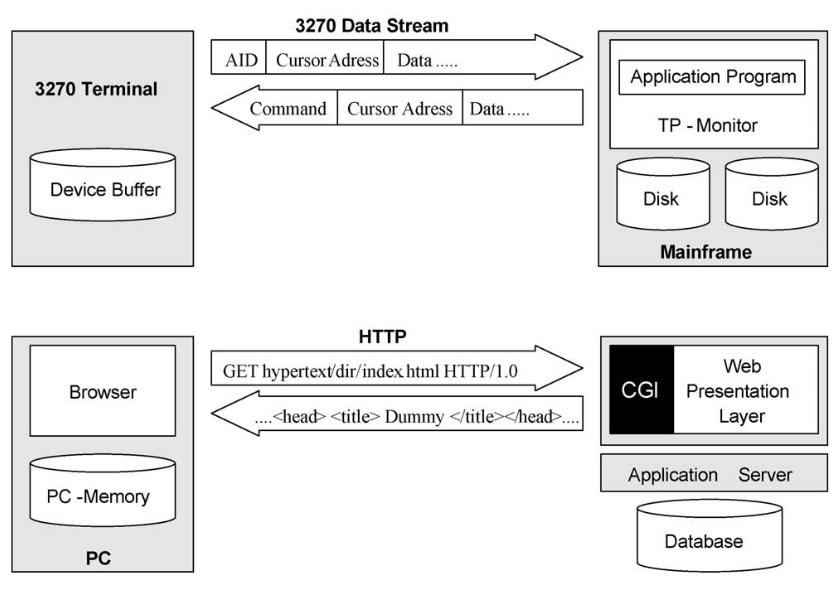
Testing must be performed even on the smallest servers and services. Without testing significant amounts of resources could be lost to minor bugs cropping up and taking out the entire system when they could have been found during a simple load testing phase during development/setup (Koh et al., 2018). Large scale systems especially so must be tested to make sure that they can still service the millions of simultaneous requests every second (Jiang, 2015). As the demand on servers increase, and as more things become cloud-based, it will become increasingly vital to test these systems to verify they still maintain maximum throughput. Especially as users attempt to store and access files on these services (Baltazar, 1998).

Load testing works best with lots of different entities fully flexing a system (Chapuis & Garbinato, 2017). AGILE development works best with higher amounts of automation and as deterministic results as possible (Garg and Lavahte, 2017). However different runs of the same test may produce different results anyway (non-deterministic) so it would be good to limit these situations as much as possible (Hwang et al., 2004). No matter the system being tested a “client/server” model will be used for all interactions (Hwang et al., 2004).

Manually testing web services would be incredibly laborious, slow and expensive (Garg and Lavahte, 2017). Real user data can (and should) be used to model traffic. However, such models are less versatile than stochastic ones. Script driven approaches are common; visual editors massively increase usability (Draheim et al., 2006).

Reconfigurability is required for a variety of different connections for different purposes and clients and tests. Low cost and low resource would be a significant boon, however using the resources as efficiently as possible is key. Traffic load co-ordination would also help to improve results and efficiency with traffic aggregation to make sure all traffic acts on a single interface at the target. Being able to monitor traffic is also a good feature to have (Bhatia et al., 2014).

Another structure would involve input identification, controller tuning and a controller. Input Identification involves finding inputs that affect the resource of interest. Controller tuning involves figuring out the internal parameters of the chosen controller. The controller itself drives the testcases to what load level is wanted. (Bayan and Cangussu, 2006).

  
Figure 2: Examples of ultra-thin client based submit/response style systems (Draheima and Weber, 2005).

Pseudo-random generators are potentially very good for the purposes of this program (Akhshani et al., 2017). While on one hand, we need different tests to be identical every run to guarantee that the tests are valid, it can offer some more specialised testing for tests that might run for an entire day, possibly detecting memory leaks and other load sensitive bugs that might only appear under such conditions. The algorithm for constructing a random number generator based on quantum chaotic maps can also be used to create a good chaos model from which to work. Only using user activity graphs instead, to better try to model peak times. (Akhshani et al., 2017) User interaction can be modelled as a bipartite state transition diagram (Draheim et al., 2006).

Geographically distributed instances would improve the results per test task. Large amounts of concurrency within each instance would be good as the loads can be very large, and the traffic could be very high (Yan et al., 2014).

Even though the systems themselves would need testing; load testing from a cloud-based platform would be entirely possible (and preferable in some situations where the location of requests is important) (Shojaee et al., 2015). Large amounts of concurrency within each instance would be good as the loads can be very large, and the traffic could be very high (Yan et al., 2014). Being able to scale to simulate potentially millions of requests would be incredibly beneficial to the testing of the software (Jiand & Hassan, 2015).

It is important that the user designs loads well, but more importantly that the software interprets them accurately, this is also a practice left up to the user in creating the plugin (Jiand & Hassan, 2015), however, the tests must be interpreted and returned accurately for reading.

Current load testers have a very limited number of virtual users and testing different configurations is difficult and time-consuming. Having the ability to scale the software up depending on the users' needs would be highly desirable (Shojaee et al., 2015).

Making sure messages are routed to the correct threads while the system is running is very important as messages can be interleaved by the underlying asynchronous operations (Koh et al., 2018). Making sure that the software can run on a variety of system's will help to maximise its customer base. This can best be achieved by making sure that only the standard libraries are adhered to mostly (Baltazar, 1998).

There will be three steps to load testing for the user when it comes to using the software after the plugins have been created: designing it, executing it and, analysing the results (Jiang, 2015). Anomalies should be able to be detected by analysis of the performance metrics and execution logs (Jiand & Hassan, 2015).

Having a variety of ways to check on and use the software would be very useful. This can include a full UI, a task monitor, and a configuration manager. An API for interaction would also be very useful as it would allow different ways of interacting with the system for different purposes. Being able to extend the program beyond its basic function would be an incredibly useful feature and is already in place in many modern systems such as Robot (Na and Huaichang, 2015).

Other great features to have that are present in other systems are custom load testing “Scripts”, specific agents, environment switching, test debugging and performance monitoring. Maximising what tools it can integrate would be a great help to supporting the AGILE workflow too, with integration with systems such as Maven, Hudson, Navigator, Junit, ApacheAnt (Garg and Lavahte, 2017).

# Requirements & Acceptance Tests

## Acceptance Tests and Requirements Overview

The requirements ID-list below is directly related to each user story as given in the following User Stories section. Here each requirement is given a priority, and to list what acceptance tests it is covered by. The acceptance tests are given in the section following that again.

**Key: S = The Highest Priority, A, B, C, D = The Lowest Priority**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **ID** | **Priority** | **Date Created** | **Acceptance Tests** |
| ***Theme 1: Plugin Manager*** | | | | |
| **1** | 1.1.1 | S | 23/10/2018 | PD\_1 |
| **2** | 1.1.2 | B | 23/10/2018 | PD\_2 |
| ***Theme 2: Threading Facility*** | | | | |
| **3** | 2.1.1 | S | 23/10/2018 | PD\_3, PD\_4 |
| **4** | 2.1.2 | S | 23/10/2018 | PD\_4 |
| **5** | 2.2.1 | S | 23/10/2018 | PD\_5 |
| **6** | 2.2.2 | S | 23/10/2018 | PD\_5 |
| **7** | 2.2.3 | S | 23/10/2018 | PD\_5 |
| **8** | 2.2.4 | S | 23/10/2018 | PD\_5 |
| **9** | 2.3.1 | A | 23/10/2018 | PD\_6 |
| ***Theme 3: Service Handler*** | | | | |
| **10** | 3.1.1 | S | 23/10/2018 | PD\_7 |
| **11** | 3.1.2 | S | 23/10/2018 | PD\_7, PD\_8 |
| **12** | 3.2.1 | S | 23/10/2018 | PD\_9 |
| **13** | 3.2.2 | A | 23/10/2018 | PD\_5, PD\_10 |
| **14** | 3.2.3 | B | 23/10/2018 | PD\_5, PD\_10 |
| **15** | 3.2.4 | B | 23/10/2018 | PD\_11 |
| **16** | 3.2.5 | C | 23/10/2018 | PD\_12 |
| **17** | 3.2.6 | C | 23/10/2018 | PD\_13 |
| **18** | 3.3.1 | S | 23/10/2018 | PD\_5 |
| **19** | 3.3.2 | B | 23/10/2018 | PD\_5 |
| **20** | 3.4.1 | S | 23/10/2018 | PD\_8 |
| **21** | 3.5.1 | S | 23/10/2018 | PD\_8 |
| **22** | 3.6.1 | S | 23/10/2018 | PD\_8 |
| **23** | 3.7.1 | A | 23/10/2018 | PD\_14 |
| **24** | 3.8.1 | A | 23/10/2018 | PD\_16 |
| ***Theme 4: API*** | | | | |
| **25** | 4.1.1 | S | 23/10/2018 | PD\_1, PD\_2, PD\_3, PD\_4, PD\_5, PD\_6, PD\_7, PD\_9, PD\_10, PD\_11, PD\_13, PD\_15, PD\_16 |
| **26** | 4.1.2 | B | 23/10/2018 | PD\_12 |
| **27** | 4.2.1 | B | 23/10/2018 | PD\_5, PD\_17, PD\_18 |
| **28** | 4.2.2 | B | 23/10/2018 | PD\_5, PD\_19 |
| ***Theme 5: Testcase Analyser*** | | | | |
| **29** | 5.1.1 | S | 26/10/2018 | PD\_3, PD\_4, PD\_9 |
| **30** | 5.2.1 | A | 26/10/2018 | PD\_3, PD\_4, PD\_20 |
| **31** | 5.3.1 | S | 26/10/2018 | PD\_3, PD\_4, PD\_21, PD\_22 |
| **32** | 5.3.2 | S | 26/10/2018 | PD\_9, PD\_23, PD\_24 |
| **33** | 5.3.3 | S | 26/10/2018 | PD\_9, PD\_25, PD\_28 |
| **34** | 5.3.4 | S | 26/10/2018 | PD\_9, PD\_26, PD\_27, PD\_28 |
| **35** | 5.4.1 | S | 26/10/2018 | PD\_9 |
| ***Theme 6: Test Runner*** | | | | |
| **36** | 6.1.1 | S | 26/10/2018 | PD\_28 |
| **37** | 6.2.1 | C | 26/10/2018 | PD\_29 |
| ***Theme 7: Logger*** | | | | |
| **38** | 7.1.1 | A | 26/10/2018 | PD\_31 |
| **39** | 7.1.2 | A | 26/10/2018 | PD\_31 |
| **40** | 7.1.3 | A | 26/10/2018 | PD\_31 |
| **41** | 7.1.4 | A | 26/10/2018 | PD\_31 |
| **42** | 7.2.1 | S | 26/10/2018 | PD\_31 |
| **43** | 7.3.1 | S | 26/10/2018 | PD\_32 |
| ***Theme 8: Database*** | | | | |
| **44** | 8.1.1 | C | 26/10/2018 | PD\_29, PD\_34 |
| **45** | 8.1.2 | B | 26/10/2018 | PD\_12, PD\_29, PD\_34 |
| **46** | 8.1.3 | B | 26/10/2018 | PD\_29, PD\_34 |
| **47** | 8.2.1 | B | 26/10/2018 | PD\_33, PD\_34 |
| **48** | 8.3.1 | C | 26/10/2018 | PD\_12, PD\_34 |
| **~~49~~** | ~~8.3.2~~ | ~~C~~ | ~~26/10/2018~~ | ~~REMOVED~~ |
| **50** | 8.3.3 | C | 26/10/2018 | PD\_34 |
| ***Theme 9: Extra Server Comms*** | | | | |
| **51** | 9.1.1 | B | 05/11/2018 | PD\_35 |
| **52** | 9.1.2 | C | 05/11/2018 | PD\_35 |
| **53** | 9.1.3 | C | 05/11/2018 | PD\_36 |
| **54** | 9.1.4 | C | 05/11/2018 | PD\_37 |
| ***Left Over Requirements*** | | | | |
| **55** | 3.7.2 | A | 19/11/2018 | PD\_15 |
| **56** | 6.3.1 | S | 13/01/2018 | PD\_30 |
| **57** | 6.3.2 | A | 13/01/2018 | PD\_30 |
| **58** | 6.3.3 | S | 13/01/2018 | PD\_23 |
| **59** | 6.3.4 | S | 13/01/2018 | PD\_30 |

## Requirements

### Themes

|  |  |
| --- | --- |
| ***THEME 1*** | |
| *I need a* | Plugin Manager |
| *For* | Loading different plugins to change the program function |

|  |  |
| --- | --- |
| ***THEME 2*** | |
| *I need a* | Threading Facility |
| *For* | Improved traffic modelling |

|  |  |
| --- | --- |
| ***THEME 3*** | |
| *I need a* | Service Handler |
| *For* | Uninterrupted use |

|  |  |
| --- | --- |
| ***THEME 4*** | |
| *I need a* | API |
| *For* | Allowing program expansion |

|  |  |
| --- | --- |
| ***THEME 5*** | |
| *I need a* | Testcase analyser |
| *For* | Reading testcases |

|  |  |
| --- | --- |
| ***THEME 6*** | |
| *I need a* | Test Runner |
| *For* | Running testcases |

|  |  |
| --- | --- |
| ***THEME 7*** | |
| *I need a* | Logger |
| *For* | Logging |

|  |  |
| --- | --- |
| ***THEME 8*** | |
| *I need a* | Database |
| *For* | Tracking results and configuration |

|  |  |
| --- | --- |
| ***THEME 9*** | |
| *I need a* | Extra server communication |
| *For* | More realistic traffic from more locations |

### Epics

### THEME 1 Epics (Plugin Manager)

|  |  |
| --- | --- |
| ***EPIC 1.1*** | |
| *The* | Plugin Manager |
| *Needs to* | Dynamically load plugins |

### THEME 2 Epics (Threading Facility)

|  |  |
| --- | --- |
| ***EPIC 2.2*** | |
| *The* | Thread Facility |
| *Needs to* | Manage the maximum number of threads |

|  |  |
| --- | --- |
| ***EPIC 2.2*** | |
| *The* | Thread Facility |
| *Needs to* | Spin up new threads |

|  |  |
| --- | --- |
| ***EPIC 2.3*** | |
| *The* | Thread Facility |
| *Needs to* | Have a generic thread handler |

### THEME 3 Epics (Service Handler)

|  |  |
| --- | --- |
| ***EPIC 3.1*** | |
| *The* | Service Handler |
| *Needs to* | Keep the program running |

|  |  |
| --- | --- |
| ***EPIC 3.2*** | |
| *The* | Service Handler |
| *Needs to* | Receive commands from the user |

|  |  |
| --- | --- |
| ***EPIC 3.3*** | |
| *The* | Service Handler |
| *Needs to* | Spin up new Jobs |

|  |  |
| --- | --- |
| ***EPIC 3.4*** | |
| *The* | Service Handler |
| *Needs to* | Work with the OS service handler |

|  |  |
| --- | --- |
| ***EPIC 3.5*** | |
| *The* | Service Handler |
| *Needs to* | Not interrupt running tests |

|  |  |
| --- | --- |
| ***EPIC 3.6*** | |
| *The* | Service Handler |
| *Needs to* | Handle program start-up |

|  |  |
| --- | --- |
| ***EPIC 3.7*** | |
| *The* | Service Handler |
| *Needs to* | Handle program first start-up |

|  |  |
| --- | --- |
| ***EPIC 3.8*** | |
| *The* | Service Handler |
| *Needs to* | Handle program Shutdown |

### THEME 4 Epics (API)

|  |  |
| --- | --- |
| ***EPIC 4.1*** | |
| *The* | API |
| *Needs to* | Allow the program to be controlled |

|  |  |
| --- | --- |
| ***EPIC 4.2*** | |
| *The* | API |
| *Needs to* | Allow data to be requested |

### THEME 5 Epics (Testcase Analyser)

|  |  |
| --- | --- |
| ***EPIC 5.1*** | |
| *The* | Testcase Analyser |
| *Needs to* | Read in serialised testcases |

|  |  |
| --- | --- |
| ***EPIC 5.2*** | |
| *The* | Testcase Analyser |
| *Needs to* | De-serialise testcases |

|  |  |
| --- | --- |
| ***EPIC 5.3*** | |
| *The* | Testcase Analyser |
| *Needs to* | Configure jobs based on testcase |

|  |  |
| --- | --- |
| ***EPIC 5.4*** | |
| *The* | Testcase Analyser |
| *Needs to* | Have a generic format to appease many protocols |

### THEME 6 Epics (Test Runner)

|  |  |
| --- | --- |
| ***EPIC 6.1*** | |
| *The* | Test Runner |
| *Needs to* | Send data to a target |

|  |  |
| --- | --- |
| ***EPIC 6.2*** | |
| *The* | Test Runner |
| *Needs to* | Receive Data from a target |

|  |  |
| --- | --- |
| ***EPIC 6.3*** | |
| *The* | Test Runner |
| *Needs to* | Configure plugins |

### THEME 7 Epics (Logger)

|  |  |
| --- | --- |
| ***EPIC 7.1*** | |
| *The* | Logger |
| *Needs to* | Log what is happening |

|  |  |
| --- | --- |
| ***EPIC 7.2*** | |
| *The* | Logger |
| *Needs to* | Be used by whatever requires it |

|  |  |
| --- | --- |
| ***EPIC 7.3*** | |
| *The* | Logger |
| *Needs to* | Not slow down operation/slow it down very little |

### THEME 8 Epics (Database)

|  |  |
| --- | --- |
| ***EPIC 8.1*** | |
| *The* | Database |
| *Needs to* | Be queried by the program |

|  |  |
| --- | --- |
| ***EPIC 8.2*** | |
| *The* | Database |
| *Needs to* | Store results |

|  |  |
| --- | --- |
| ***EPIC 8.3*** | |
| *The* | Database |
| *Needs to* | Be Accessible via the API |

### THEME 9 Epics (Extra Server Comms)

|  |  |
| --- | --- |
| ***EPIC 9.1*** | |
| *The* | Service |
| *Needs to* | Communicate with other instances |

|  |  |
| --- | --- |
| ***EPIC 9.2*** | |
| *The* | Service |
| *Needs to* |  |

### User Stories

#### Theme 1, Epic 1

|  |  |
| --- | --- |
| ***USER STORY 1.1.1*** | |
| *As a* | Tester |
| *I want to* | Dynamically load protocol plugins |
| *So that I can* | Build traffic based on a protocol message builder |

| ***USER STORY 1.1.2*** | |
| --- | --- |
| *As a* | Tester |
| *I want to* | Dynamically load communication plugins |
| *So that I can* | Change the target of built messages |

#### Theme 2, Epic 1

|  |  |
| --- | --- |
| ***USER STORY 2.1.1*** | |
| *As a* | Tester |
| *I want to* | Configure the number of threads to use per testcase |
| *So that I can* | Not worry about the default and run different kinds of tests |

|  |  |
| --- | --- |
| ***USER STORY 2.1.2*** | |
| *As a* | Sysadmin |
| *I want to* | Apply a hard limit to the number of threads to use |
| *So that I can* | Limit the resources the program will use |

#### Theme 2, Epic 2

|  |  |
| --- | --- |
| ***USER STORY 2.2.1*** | |
| *As a* | Tester |
| *I want to* | Interact with the software without interrupting anything |
| *So that I can* | Not disturb any running tests |

|  |  |
| --- | --- |
| ***USER STORY 2.2.2*** | |
| *As a* | Sysadmin |
| *I want to* | Interact with the software without interrupting anything |
| *So that I can* | Not disturb any running tests |

|  |  |
| --- | --- |
| ***USER STORY 2.2.3*** | |
| *As a* | Developer |
| *I want to* | Interact with the software without interrupting anything |
| *So that I can* | Not disturb any running tests |

|  |  |
| --- | --- |
| ***USER STORY 2.2.4*** | |
| *As a* | UX Designer |
| *I want to* | Interact with the software without interrupting anything |
| *So that I can* | Not disturb any running tests |

#### Theme 2, Epic 3

|  |  |
| --- | --- |
| ***USER STORY 2.3.1*** | |
| *As a* | Maintainer |
| *I want to* | Worry about less code |
| *So that I can* | Get more work done on the code base |

#### Theme 3, Epic 1

|  |  |
| --- | --- |
| ***USER STORY 3.1.1*** | |
| *As a* | Sysadmin |
| *I want to* | Not worry about having to start the software repeatedly |
| *So that I can* | Be bothered less to start the service |

|  |  |
| --- | --- |
| ***USER STORY 3.1.2*** | |
| *As a* | Sysadmin |
| *I want to* | Start the program as a service |
| *So that I can* | Run it in the background |

#### Theme 3, Epic 2

|  |  |
| --- | --- |
| ***USER STORY 3.2.1*** | |
| *As a* | Tester |
| *I want to* | Start a Test |
| *So that I can* | Evaluate a protocol or how a server handles load |

|  |  |
| --- | --- |
| ***USER STORY 3.2.2*** | |
| *As a* | Tester |
| *I want to* | Stop a test |
| *So that I can* | Cancel any currently running tests for whatever reason |

|  |  |
| --- | --- |
| ***USER STORY 3.2.3*** | |
| *As a* | Tester |
| *I want to* | Query running tests |
| *So that I can* | Check on their progress |

|  |  |
| --- | --- |
| ***USER STORY 3.2.4*** | |
| *As a* | Sysadmin |
| *I want to* | Query program running stats |
| *So that I can* | To verify how well it’s running |

|  |  |
| --- | --- |
| ***USER STORY 3.2.5*** | |
| *As a* | UX Designer |
| *I want to* | Be able to access the API |
| *So that I can* | Create graphical front ends |

|  |  |
| --- | --- |
| ***USER STORY 3.2.6*** | |
| *As a* | Developer |
| *I want to* | Load plugins |
| *So that I can* | Verify they can be loaded correctly |

#### Theme 3, Epic 3

|  |  |
| --- | --- |
| ***USER STORY 3.3.1*** | |
| *As a* | Tester |
| *I want to* | Multi-task |
| *So that I can* | Not interrupt any other currently running tasks |

|  |  |
| --- | --- |
| ***USER STORY 3.3.2*** | |
| *As a* | UX Designer |
| *I want to* | Interact with the software without touching the normal running |
| *So that I can* | Create more seamless and less intrusive Front-ends |

#### Theme 3, Epic 4

|  |  |
| --- | --- |
| ***USER STORY 3.4.1*** | |
| *As a* | Sysadmin |
| *I want to* | Start the service standardly |
| *So that I can* | Trust the OS to handle it better |

#### Theme 3, Epic 5

|  |  |
| --- | --- |
| ***USER STORY 3.5.1*** | |
| *As a* | Tester |
| *I want to* | Not interrupt the program’s normal running |
| *So that I can* | Guarantee more accurate results |

#### Theme 3, Epic 6

|  |  |
| --- | --- |
| ***USER STORY 3.6.1*** | |
| *As a* | Sysadmin |
| *I want to* | Start the service and it looks after itself |
| *So that I can* | Worry about other things |

#### Theme 3, Epic 7

|  |  |
| --- | --- |
| ***USER STORY 3.7.1*** | |
| *As a* | Sysadmin |
| *I want to* | Not have to prepare for the program’s first launch |
| *So that I can* | Have an easier time with setting it up |

|  |  |
| --- | --- |
| ***USER STORY 3.7.2*** | |
| *As a* | Sysadmin |
| *I want to* | Have the program clean up when removed/uninstalled |
| *So that I can* | Worry less about floating files |

#### Theme 3, Epic 8

|  |  |
| --- | --- |
| ***USER STORY 3.8.1*** | |
| *As a* | Sysadmin |
| *I want to* | Kill the program |
| *So that I can* | End any strange behaviour |

#### Theme 4, Epic 1

|  |  |
| --- | --- |
| ***USER STORY 4.1.1*** | |
| *As a* | Tester |
| *I want to* | Control the program using scripts |
| *So that I can* | Automate testing for different scenarios |

|  |  |
| --- | --- |
| ***USER STORY 4.1.2*** | |
| *As a* | UX Designer |
| *I want to* | Have a way of controlling the software using an API |
| *So that I can* | Create graphical front ends for controlling the program |

#### Theme 4, Epic 2

|  |  |
| --- | --- |
| ***USER STORY 4.2.1*** | |
| *As a* | Tester |
| *I want to* | Fetch data about currently running tests |
| *So that I can* | Verify the results |

|  |  |
| --- | --- |
| ***USER STORY 4.2.2*** | |
| *As a* | UX Designer |
| *I want to* | Fetch data from the software |
| *So that I can* | Create graphical front ends for viewing the data |

#### Theme 5, Epic 1

|  |  |
| --- | --- |
| ***USER STORY 5.1.1*** | |
| *As a* | Tester |
| *I want to* | Send testcases to the software for running |
| *So that I can* | Run test cases |

#### Theme 5, Epic 2

|  |  |
| --- | --- |
| ***USER STORY 5.2.1*** | |
| *As a* | Tester |
| *I want to* | Be told where my testcase is syntactically or lexically wrong |
| *So that I can* | Fix it more easily |

#### Theme 5, Epic 3

|  |  |
| --- | --- |
| ***USER STORY 5.3.1*** | |
| *As a* | Tester |
| *I want to* | Configure the number of threads to use from within the testcase |
| *So that I can* | I don’t have to edit the program configuration directly |

|  |  |
| --- | --- |
| ***USER STORY 5.3.2*** | |
| *As a* | Tester |
| *I want to* | Configure what protocol is to be loaded from within the testcase |
| *So that I can* | I don’t have to edit the program configuration directly |

|  |  |
| --- | --- |
| ***USER STORY 5.3.3*** | |
| *As a* | Tester |
| *I want to* | Configure the rate that traffic is sent from within the test case |
| *So that I can* | I don’t have to edit the program configuration directly |

|  |  |
| --- | --- |
| ***USER STORY 5.3.4*** | |
| *As a* | Tester |
| *I want to* | Configure whether chaos should be applied to the traffic |
| *So that I can* | I don’t have to edit the program configuration directly |

#### Theme 5, Epic 4

|  |  |
| --- | --- |
| ***USER STORY 5.4.1*** | |
| *As a* | Tester |
| *I want to* | Write testcases for many protocols |
| *So that I can* | Test many aspects of a server |

#### Theme 6, Epic 1

|  |  |
| --- | --- |
| ***USER STORY 6.1.1*** | |
| *As a* | Tester |
| *I want to* | Send data to a target |
| *So that I can* | Measure it’s load or test a protocol |

#### Theme 6, Epic 2

|  |  |
| --- | --- |
| ***USER STORY 6.2.1*** | |
| *As a* | Tester |
| *I want to* | Have the data results recorded |
| *So that I can* | Verify everything is working |

#### Theme 6, Epic 3

|  |  |
| --- | --- |
| ***USER STORY 6.3.1*** | |
| *As a* | Tester |
| *I want to* | Have the test runner configure itself automatically |
| *So that I can* | Make fewer API calls |

|  |  |
| --- | --- |
| ***USER STORY 6.3.2*** | |
| *As a* | Tester |
| *I want to* | Select the branching method |
| *So that I can* | Configure the software based on present resources |

|  |  |
| --- | --- |
| ***USER STORY 6.3.3*** | |
| *As a* | Tester |
| *I want to* | Know the test runner will read what protocol it needs |
| *So that I can* | Make fewer API calls |

|  |  |
| --- | --- |
| ***USER STORY 6.3.4*** | |
| *As a* | Tester |
| *I want to* | Know the test runner will read what Comms handler it needs |
| *So that I can* | Make fewer API calls |

#### Theme 7, Epic 1

|  |  |
| --- | --- |
| ***USER STORY 7.1.1*** | |
| *As a* | Sysadmin |
| *I want to* | Read logs |
| *So that I can* | Verify the program is working correctly |

|  |  |
| --- | --- |
| ***USER STORY 7.1.2*** | |
| *As a* | Tester |
| *I want to* | Read logs |
| *So that I can* | Verify tests are running |

|  |  |
| --- | --- |
| ***USER STORY 7.1.3*** | |
| *As a* | Developer |
| *I want to* | Read logs |
| *So that I can* | Verify tests are completed |

|  |  |
| --- | --- |
| ***USER STORY 7.1.4*** | |
| *As a* | Manager |
| *I want to* | Read logs |
| *So that I can* | Verify tests are performing as expected |

#### Theme 7, Epic 2

|  |  |
| --- | --- |
| ***USER STORY 7.2.1*** | |
| *As a* | Sysadmin |
| *I want to* | Receive reports from all aspects of the system |
| *So that I can* | Get a better picture of what is going on |

#### Theme 7, Epic 3

|  |  |
| --- | --- |
| ***USER STORY 7.3.1*** | |
| *As a* | Tester |
| *I want to* | Know that the logger won’t hinder the system too much |
| *So that I can* | Guarantee more accurate results |

#### Theme 8, Epic 1

|  |  |
| --- | --- |
| ***USER STORY 8.1.1*** | |
| *As a* | Tester |
| *I want to* | Read data from the database |
| *So that I can* | Look over past results |

|  |  |
| --- | --- |
| ***USER STORY 8.1.2*** | |
| *As a* | UX Designer |
| *I want to* | Read data from the database |
| *So that I can* | Create data visualisations |

|  |  |
| --- | --- |
| ***USER STORY 8.1.3*** | |
| *As a* | Manager |
| *I want to* | Read data from the database |
| *So that I can* | Make sure everything is going well |

#### Theme 8, Epic 2

|  |  |
| --- | --- |
| ***USER STORY 8.2.1*** | |
| *As a* | Tester |
| *I want to* | Compare previous runs |
| *So that I can* | Draw conclusions as to how the target is dealing with things |

#### Theme 8, Epic 3

|  |  |
| --- | --- |
| ***USER STORY 8.3.1*** | |
| *As a* | UX Designer |
| *I want to* | Access the database |
| *So that I can* | Create data visualisation front ends |

|  |  |
| --- | --- |
| ***USER STORY 8.3.3*** | |
| *As a* | Tester |
| *I want to* | Access the database via a command line |
| *So that I can* | Create automated verification scripts |

#### Theme 9, Epic 1

|  |  |
| --- | --- |
| ***USER STORY 9.1.1*** | |
| *As a* | Tester |
| *I want to* | Run tests from many locations |
| *So that I can* | Get a better and more accurate idea of how something handles load |

|  |  |
| --- | --- |
| ***USER STORY 9.1.2*** | |
| *As a* | Tester |
| *I want to* | Run tests from many locations |
| *So that I can* | Perform larger scale tests to further improve how load is handled |

|  |  |
| --- | --- |
| ***USER STORY 9.1.3*** | |
| *As a* | Tester |
| *I want to* | Verify results from multiple instances |
| *So that I can* | See how well the software performed |

| ***USER STORY 9.1.4*** | |
| --- | --- |
| *As a* | Tester |
| *I want to* | Worry only about using a single instance |
| *So that I can* | Trust the service will handle using other instances appropriately |

## Acceptance Tests

**Test Name**: PD\_1  
**Requirement(s) Tested**: 1, 25  
**Outline**: Load a Protocol plugin  
**Pre-requisites**: Service is running, a protocol plugin is available for loading, CLI open  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Enter the command to load protocol plugin | Service accepts command without issue |

**Test Name**: PD\_2  
**Requirement(s) Tested**: 2, 25  
**Outline**: Load a communication plugin  
**Pre-requisites**: Service is running, a communication plugin is available for loading, CLI open  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Enter the command to load communication plugin | Service accepts command without issue |

**Test Name**: PD\_3  
**Requirement(s) Tested**: 3, 25, 29, 30, 31  
**Outline**: Program reads the maximum threads from the testcase file  
**Pre-requisites**: Service is running, a testcase is ready to read, CLI open  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Enter the command to load a test | Service accepts command without issue |
| *2* | Read log | The log shows that maximum threads have been set |

**Test Name**: PD\_4  
**Requirement(s) Tested**: 3, 4, 25, 29, 30, 31  
**Outline**: Program reads the maximum threads from the testcase, applies hard limit when too high  
**Pre-requisites**: Service is running, a testcase is ready to read, CLI open  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Enter the command to load a test | Service accepts command without issue |
| *2* | Read log | The log shows that maximum threads have been set (lower than asked for but *at* the limit) |

**Test Name**: PD\_5  
**Requirement(s) Tested**: 5, 6, 7, 8, 14, 15, 18, 19, 25, 27, 28  
**Outline**: While a test is running, measure performance changes when making other queries against the system  
**Pre-requisites**: Service is running, a testcase is running, CLI is open, the log is being read  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Enter different commands to interact with the system | The system accepts the command |
| *2* | Do this several times in a row | The system happily performs each task |
| *3* | When the command completes, check logs | Logs should show no discernible difference in running time or efficiency |

**Test Name**: PD\_6  
**Requirement(s) Tested**: 9, 25  
**Outline**: When making changes regarding the threading facility, few files need changing, the ones that exist are relatively easy to edit  
**Pre-requisites**: Have source code, change needs to be made  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Look at the area that needs changing | Only a single class needs editing |

**Test Name**: PD\_7  
**Requirement(s) Tested**: 10, 11, 25  
**Outline**: The service needs to be somewhat self-recoverable, and should be handled by sysvinit/system.  
**Pre-requisites**: Device off, software installed on device and setup as a service/daemon  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Turn system on |  |
| *2* | Check running services | ProtDev should be listed |

**Test Name**: PD\_8  
**Requirement(s) Tested**: 11, 20, 21, 22  
**Outline**: The service should be started exclusively as a service/daemon  
**Pre-requisites**: Service is not running  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Attempt to run a ProtDev executable directly | The program should error out requiring it to be run as a service |

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Attempt to run ProtDev via the system's init infrastructure | The program should happily start running and the init systems logs should show a running service |

**Test Name**: PD\_9  
**Requirement(s) Tested**: 12, 25, 29, 32, 33, 34, 35, 36  
**Outline**: Initiate a test by passing in a testcase, it sends data to the target  
**Pre-requisites**: Service is running, protocol available, testcase written and correct, CLI open  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Send test case to program | The program accepts it without issue |
| *2* | Check logs | A log file for the test has been created, showing statistics and options provided along with any unexpected issues from the service |
| *3* | Check target | Target should show interaction with the service |

**Test Name**: PD\_10  
**Requirement(s) Tested**: 13, 14, 25  
**Outline**: Stop a test while it’s running  
**Pre-requisites**: Service is running, a test is running  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Enter a command to view running tests | A list is shown displaying tests IDs and their description |
| *2* | Enter a command referencing the test ID to stop it | The program accepts the command and begins to shut down the test cleanly |

**Test Name**: PD\_11  
**Requirement(s) Tested**: 15, 25  
**Outline**: As the program is running, query the program for its current statistics  
**Pre-requisites**:   
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Enter a command querying the current statistics for the programs running | The program outputs current stats, including any running threads and other such business |

**Test Name**: PD\_12  
**Requirement(s) Tested**: 16, 26, 45  
**Outline**: The API allows a UX designer access to the system for data or control purposes  
**Pre-requisites**: API is written and public  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Check the documentation for API | API is present and made public for easy access |

**Test Name**: PD\_13  
**Requirement(s) Tested**: 17, 25  
**Outline**: Developer loads a plugin, program performs self-checks to verify the plugin is functioning correctly  
**Pre-requisites**: Service is running, the plugin is written and ready to load  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Enter a command to solo load a plugin | The program accepts command without issue and loads the plugin |
| *2* | Wait | Program outputs that the plugin is working/valid |

**Test Name**: PD\_14  
**Requirement(s) Tested**: 23  
**Outline**: When the program is first launched performs a series of steps that validate its environment and rectify any issues that may be present  
**Pre-requisites**: Service hasn’t been started on this device before/clean installation.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Run either setup script or start the program | Program announces it’s the first time it’s being setup |
| *2* | Wait | The program will go through a checklist of items and setup its environment |
| *3* | Wait | Service will be running and will have set up its own directories with its required libs, logs and other such locations |

**Test Name**: PD\_15  
**Requirement(s) Tested**: 25, 55  
**Outline**: While the program is installed perform an uninstallation.  
**Pre-requisites**: Service is installed and configured on the system.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Enter the command to remove the service | The program begins exiting gracefully |
| *2* | Wait | The uninstallation process will verify what the user wishes to do with user-edited directories (plugins, logs, configs) |
| *3* | Enter choices | The program will carry out the choices (remove or keep and tar) |
| *4* | Wait | The program will no longer have a significant presence on the device |

**Test Name**: PD\_16  
**Requirement(s) Tested**: 24, 25  
**Outline**: While the program is running/performing tests, perform a hard yet As-Graceful-As-Possible (AGAP) exit.  
**Pre-requisites**: Program is running.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Send the Programs to kill command on the CLI | The program will begin attempting to kill all its child threads as quickly as possible while attempting to maintain some level of graceful exit |
| *2* | Verify resources returned and program killed | The program should no longer be present, and any resources returned to the system |

**Test Name**: PD\_17  
**Requirement(s) Tested**: 27  
**Outline**: Perform an API call to find out about running tests, no tests are running.  
**Pre-requisites**: Program is running, no tests are running.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Perform an API call | A list of running tests is displayed, the list is empty. |

**Test Name**: PD\_18  
**Requirement(s) Tested**: 27  
**Outline**: Perform an API call to find out about running tests. At least a single test is running.  
**Pre-requisites**: Program is running, at least one test is running.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Perform an API call | A list of running tests is displayed, the tests are displayed |

**Test Name**: PD\_19  
**Requirement(s) Tested**: 28  
**Outline**: Perform an API call to get data about the running software.  
**Pre-requisites**: Software is running.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Perform an API call to get Software running information | A collection of data representing |

**Test Name**: PD\_20  
**Requirement(s) Tested**: 30  
**Outline:** The software tells the user where the syntax is incorrect.  
**Pre-requisites**: Software is running, a syntactically incorrect test case is ready to be read in.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Send test case to software | The software accepts the testcase |
| *2* | Wait | The software will declare that the testcase is syntactically incorrect and print the line/location where it is wrong |

**Test Name**: PD\_21  
**Requirement(s) Tested**: 31  
**Outline**: Verify the correct number of threads have been created (the exact number wanted).  
**Pre-requisites**: Software is running and the software’s current number of threads is known and there’s enough space for more threads to be created.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Start a test | The test starts |
| *2* | Check the number of running threads | There will be the exact number of extra threads requested in the test thread. |

**Test Name**: PD\_22  
**Requirement(s) Tested**: 31  
**Outline**: Verify the software limits the number of newly created threads appropriately.  
**Pre-requisites**: Software is running and the software's current number of threads is known and there are too many threads to create the full roster required by the new test.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Start a test | The test starts |
| *2* | Check the number of running threads | The number of threads will be limited to whatever the programs max is and logs its decision |

**Test Name**: PD\_23  
**Requirement(s) Tested**: 32, 58  
**Outline**: Verify the software correctly loads the protocol as defined within the testcase.  
**Pre-requisites**: Software is running and there is a protocol ready for the software to read.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Start a test | The test starts |
| *2* | Wait | The software will continue running |

**Test Name**: PD\_24  
**Requirement(s) Tested**: 32  
**Outline**: Verify the software reports protocol loading failures appropriately.  
**Pre-requisites**: Software is running, a testcase with an incorrect protocol is listed.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Start a test | The test starts |
| *2* | Wait | The software will state/write to the log that the desired protocol is missing/can’t be found/doesn’t exist |

**Test Name**: PD\_25  
**Requirement(s) Tested**: 33  
**Outline**: Verify the software roughly matches the desired traffic rate.  
**Pre-requisites**: Software is running, a testcase is written.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Start a test | The test starts |
| *2* | Check the target for interaction | The target is logging transactions at roughly the desired rate |

**Test Name**: PD\_26  
**Requirement(s) Tested**: 34  
**Outline**: Verify that some amount of noticeable chaos is applied to the traffic.  
**Pre-requisites**: Software is running, a testcase is written.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Start a test | The test starts |
| *2* | Check the target for interaction over an extended period | The target is logging some wild differences in received traffic |

**Test Name**: PD\_27  
**Requirement(s) Tested**: 34  
**Outline**: Verify that the chaos can indeed be 100% turned off.  
**Pre-requisites**: Software is running, a testcase is written.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Start a test | The test starts |
| *2* | Check the target of interaction over an extended period | The target is logging no wildly mad differences in received traffic |

**Test Name**: PD\_28  
**Requirement(s) Tested**: 36  
**Outline**: While a test is running, verify that data is correctly being sent to the target.  
**Pre-requisites**: Software is running, a test is running.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Check the target for test data/traffic | The target is observed to be receiving test data/traffic |

**Test Name**: PD\_29  
**Requirement(s) Tested**: 37, 44, 45, 46  
**Outline**: Record the interpreted result codes.  
**Pre-requisites**: Software is running, the test is running, the database is up.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Check the database for result codes against action taken | There are result codes listed with the taken action |
| *2* | Check the target for the traffic | The target’s transaction logs match with the database’s |

**Test Name**: PD\_30  
**Requirement(s) Tested**: 56, 57, 59  
**Outline**: While a test is running, verify the settings used.  
**Pre-requisites**: Software is running, a test is running.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Request test running data from the software | The program will return all currently running tests and show that the |

**Test Name**: PD\_31  
**Requirement(s) Tested**: 38, 39, 40, 41, 42  
**Outline**: Logs are kept and recorded for all aspects of the system and these logs can be read by an operator.  
**Pre-requisites**: System is running.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Open the logs directory | It contains a collection of logs that are being added to |

**Test Name**: PD\_32  
**Requirement(s) Tested**: 43  
**Outline**: The log should be a very minor hindrance to the traffic generation, verify that it is not.  
**Pre-requisites**: Software running.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Run a test with the logger disabled | The Software will perform as asked |
| *2* | When the last test is finished, tun another with the logger enabled | The software will perform more or less identically to the last run |

**Test Name**: PD\_33  
**Requirement(s) Tested**: 47  
**Outline**: Previous runs should be recorded for comparison.  
**Pre-requisites**: System is running, multiple previous tests have been run.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Check database for previous runs | The database shows previous runs |

**Test Name:** PD\_34 **Requirement(s) Tested**: 50  
**Outline**: The database should be accessible, i.e. created.  
**Pre-requisites**: Software has at least run before.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Try to access the database | The database has been created and can, therefore, be accessed |

**Test Name**: PD\_35  
**Requirement(s) Tested**: 51, 52  
**Outline**: The software should have knowledge of many instances for more complex runs and handle test load balancing from any currently used instance.  
**Pre-requisites**: Software has separate running instances that can communicate.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Check database for knowledge of other instances | The database shows that the instances are talking and have knowledge of each other |
| *2* | Start a test configured for Wide-Area testing | The current instance will contact other instances with the desired configuration for performing tests with |

**Test Name**: PD\_36  
**Requirement(s) Tested**: 53  
**Outline**: The software should be able to access results regardless of what node the user might currently be sat on.  
**Pre-requisites**: Multiple, communicating, instances are currently running and all have databases.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Request data from one instance regarding another instance | The requested instance’s data is displayed/returned |

**Test Name**: PD\_37  
**Requirement(s) Tested**: 54  
**Outline**: When using the software, the user should be largely agnostic about other running instances and the software itself should handle everything.  
**Pre-requisites**: Multiple, communicating, instances are currently running and all have databases.  
**Method**:

|  |  |  |
| --- | --- | --- |
| ***STEP*** | *Action* | *Expected Observation* |
| *1* | Perform any task that might incur the use of a separate instance | The software will gladly handle the request, and unless there is a major/fatal issue, the user will not be aware of what is going happening |

# Methodology

## Development Methodology

Due to the nature of the project, traditional/team-based methodologies may not be totally appropriate, or there may even be methodologies that can be used specifically *for* solo teams (Pagotto et al., 2016). However, it has been suggested that using older, more standard, methodologies are best[[1]](#footnote-2)[[2]](#footnote-3) with only minor changes.

Initially, the plan for development was to follow an *iterative* model after the initial libraries had been identified. As each library is discovered to be a requirement it is to be designed, implemented and then tested. Testing is to take on a Test-Driven-Development approach, the code is written *exclusively* to make the unit tests pass.

However, an iterative model does not lend itself to management and organisation by design. As such using a stricter (and specific) AGILE methodology would make a great deal more sense as it does help to make sure that teams (or solo developers) can self organise better and more importantly can adapt better to any changes in design or requirements that may come up at a later date. The two methodologies under consideration are SCRUM and Kanban.

SCRUM methodology decides the work-flow of a cross-functional, self-organising team to perform all steps (designing, building and testing) of the development at once all the time. This is done by completing prioritised tasks pulled from a “product backlog” that describes all required tasks to complete the project. The tasks are pulled into a sprint (“timeboxed iteration”) backlog and completed by the team over the course of a sprint (typically a 2 to 4-week development-cycle). At the end of the sprint, the team is to evaluate how they did and how many (and what) tasks are to be pulled into the new sprint backlog. This process is typically managed by an honorific "SCRUM Master" that will often change sprint to sprint. Daily standups are a mandatory part of SCRUM (Rubin, 2013).

Kanban Functions similarly to scrum however it does not have a concept of Sprints. Instead, tasks are completed in a LEAN/“just-in-time” fashion. As one “channel” (the vertical component to a kanban board) fills up the previous task cannot continue. A channel can only contain so many tasks. If a new task is wanted in the channel, then some tasks must be cleared. This gives a constant push through on the system that pulls more and more tasks and forces other tasks to be completed, instead of their priority simply being demoted. Tasks are still completed with a priority fashion, however, they must ALL be finished (Ahmad et al., 2018).

Scrum is very, very, geared towards having meetings with the team and making sure each other know exactly where everyone stands to make sure nothing is blocked by some other task. Kanban will serve as a great way to make sure tasks are not put off until later “sprints”. As a solo team, there is no need to be quite that organised. Completing tasks “just-in-time” will mean a need for less second opinions and more just performing.

For that reason, Kanban will be the methodology that will be followed for this project.

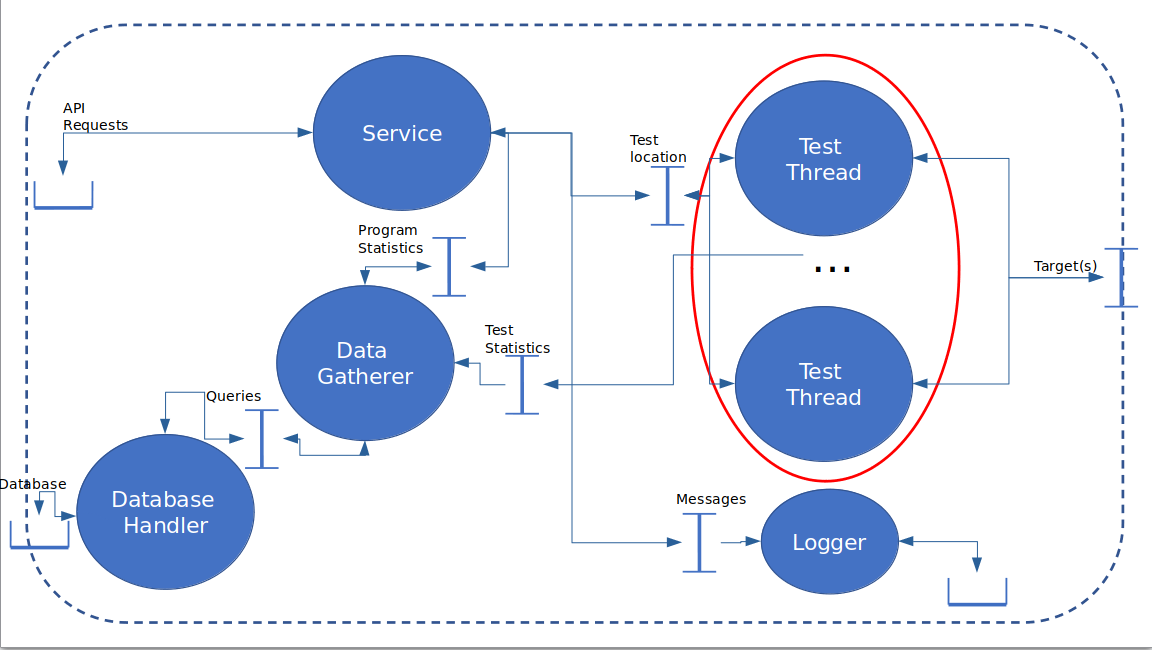
# Designs

## Current Designs

### What runs where and how (thread/tasks)

After deciding that it should run as a daemon, the next step was to Figure out the best way of handling commands and state changes. Along with making sure that data can be passed between all relevant points of the program to where it is needed.

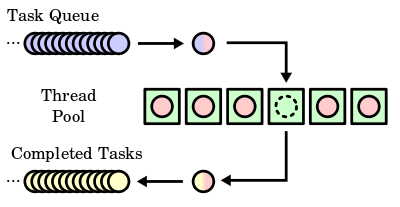
The original ACP diagram to model each concurrently running thread had to be updated to A) use the ACP diagram correctly, B) reflect newly gained knowledge regarding the API and, C) a change of opinion in how the program should be interfaced with.



### Thread Pool Queue

When this project was first started the initial plan was to have threads spawn wherever they were required (with no attention being paid to tracking them, other than in the instances that they existed). After some research and advice, the "Thread Pool Queue Pattern" (TPQ) was discovered. This allowed the software to have a central thread handler that simply dished out work to threads as they were required. The threads being put to sleep upon a job being completed.

Another very important component of the TPQ was that it needed to be more or less totally agnostic to the work it was doing to cut down on maintenance; so that a separate thread spawner wasn’t needed for every possible method/task that it would be

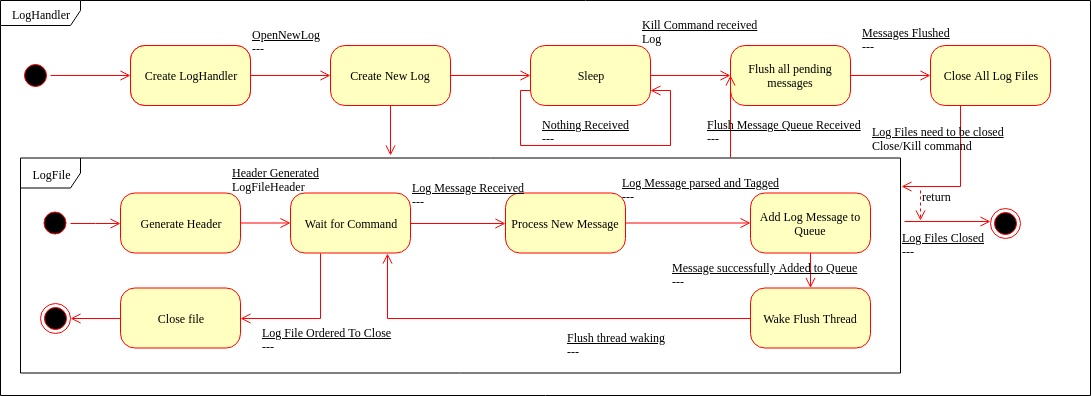


3: Picture from Wikipedia (<https://upload.wikimedia.org/wikipedia/commons/thumb/0/0c/Thread_pool.svg/600px-Thread_pool.svg.png>)

### Logger

The most important thing the logger needs to do is to A) log things and to B) not slow down the program. To this end, it was decided that the logger should run as a separate task. With each component only writing to a log message queue that is then gradually flushed to the file itself.

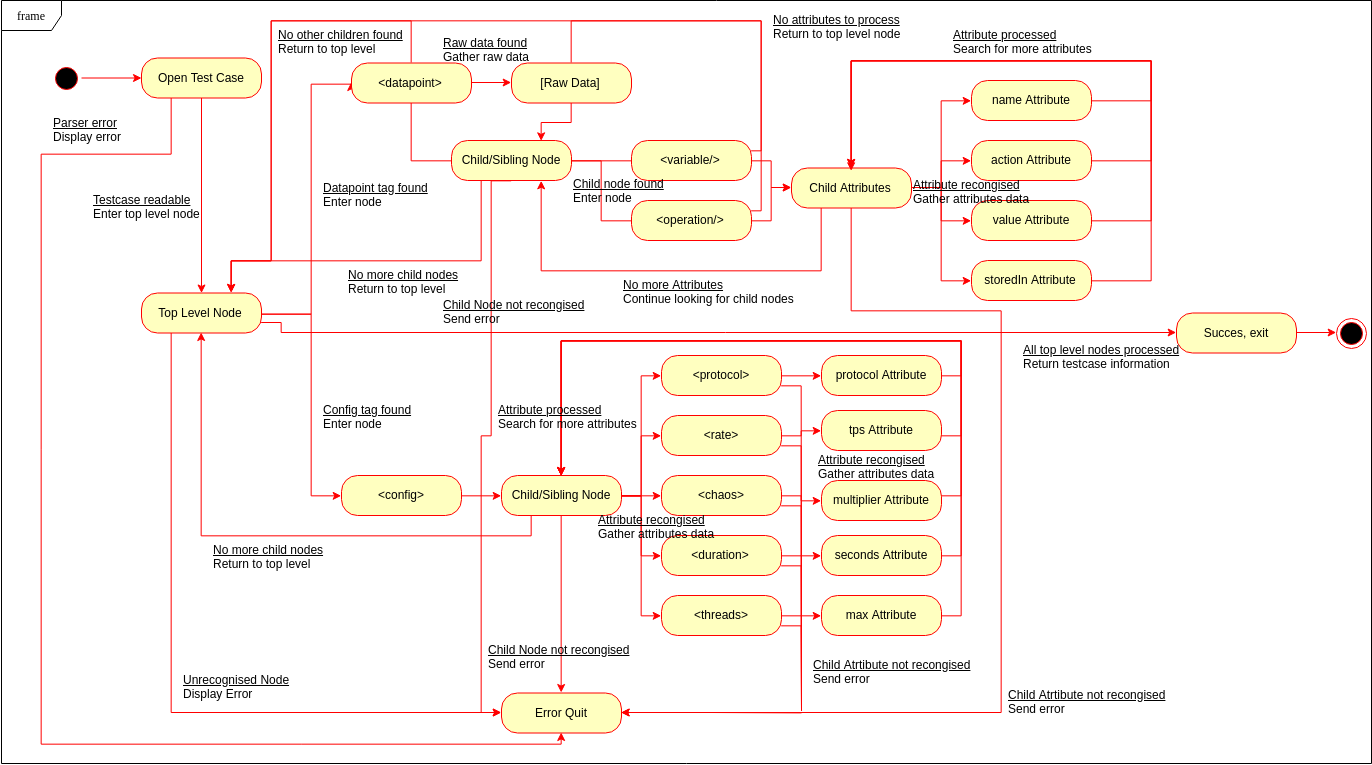
This method should help in making sure that the program does not end up bogged down in writing to the disk (a much slower task than simply using CPU). Though, processes dealing with disk writes are prioritised over processes vying for CPU usage [VERIFY].



### Test Case Analyser

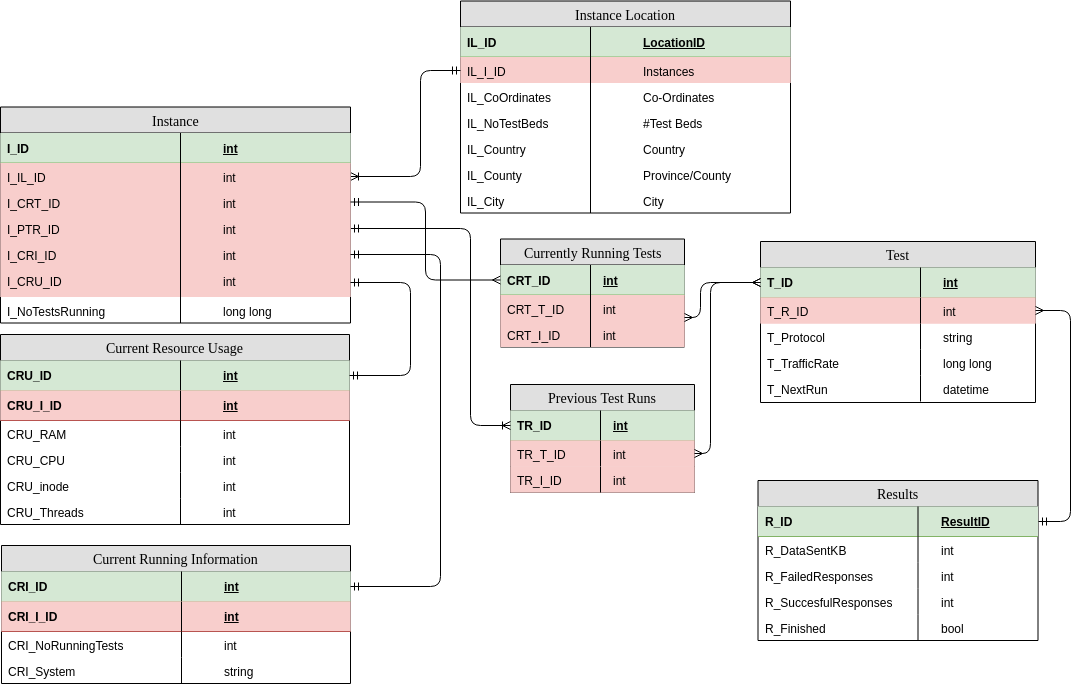
The test case analyser is one of the most important parts of the system. To that end making sure that its design is clear, robust and totally thorough is vital. Testcases will be written in a "human-readable format"; in this case, XML. That means a parser will be needed so that the testcase can be turned into something the program itself can understand.

Once the document is parsed, its data needs to be stored in some way that can easily be accessed by the plugin and the other parts of the program that might require it.



### Database

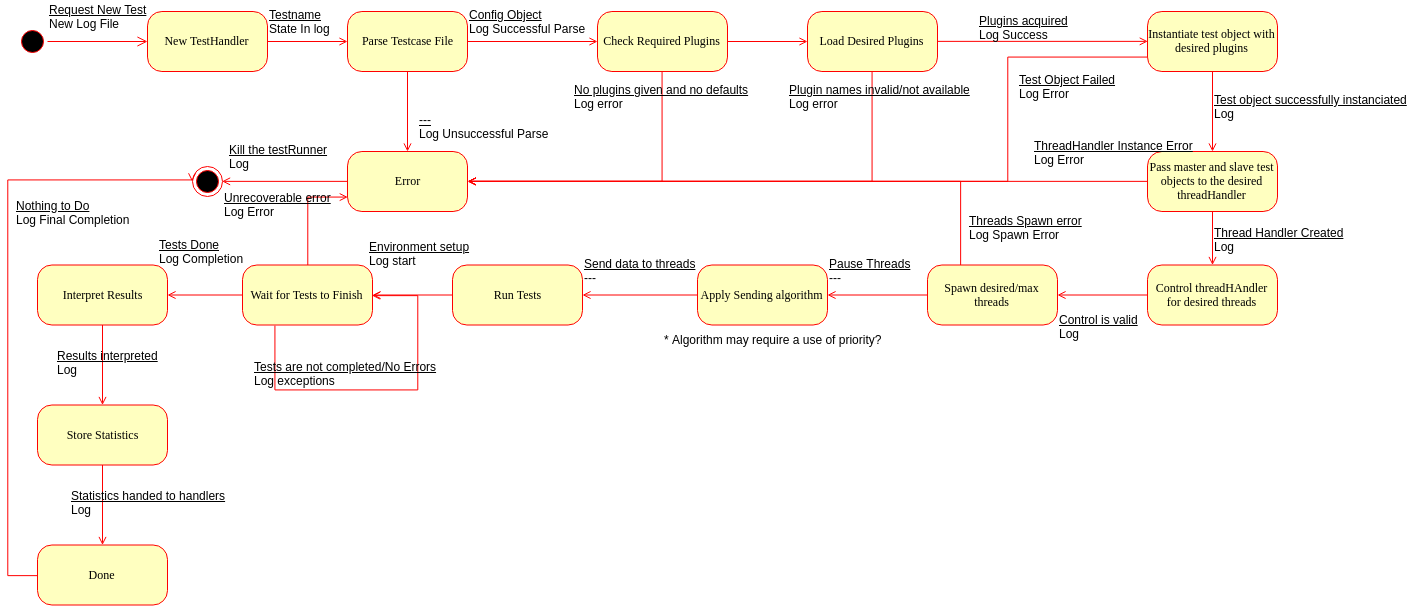
The database needs some sort of design, even if it will not be implemented. An Entity Relationship Diagram (ERD) was created to model what the database would/should like if it were to be created.



### TestHandler

Designing the test handler became a problem. One way would be to use the thread handler that had been so carefully created earlier in the project, as this would allow total control over the number of threads spawned. Using a thread branching method such as OpenMP would also be a very good way of performing this task. However, a third and potentially better way presented itself, especially regarding performance and maximum load testing: using a homogenous thread system (such as Open MPI).

Having at least 3 avenues to pursue means either more thought needs to be put into how the software should behave *or* the branching method can be created as a plugin, extending the usability and configurability even farther. And whilst the specifics of this is very much an implementation detail, abstracting everything *out* to allow the functionality to be switched will be an important design detail.



# Implementation

## Log

### Setting Up Gtest/Gmock

As soon as the project was started, it was decided that unit tests were a necessity. It would help in tracking down problems now, and it meant that small errors were corrected as soon as the library was built, not when the entire system fails and there’s no way to find out where.

After experimenting with building cppunit and Boost.Test, it was realised that mocking would also be a very useful tool to have, and neither of those projects supported it. Eventually, GoogleTest and GoogleMock were settled upon.

This made testing a great deal easier as it meant that there were a large user base and a strong customer base behind it. It also meant the unit test and mocking capabilities were built "into" each other and making sure the test and mocking frameworks worked together comfortably (and uniformly across files).

The GitHub project also provided some CMake targets that automatically downloaded, built, and linked them with the unit tests. Meaning that the dependency was self-resolving (requiring the more prudent users, who wish to compile the unit tests, need only an internet connection).

### Project structure

Originally, the project structure was very flat. The top-level directory contained a single CMakeLists.txt and the main.cpp (for ProtDev itself). With each library/component of the program in a folder in there. Eventually, the structure currently in use was found as per[[3]](#footnote-4). Instead of every single package needing to be referenced in the top-level CMakeLists.txt Every single library is its own self-contained project that is exported to the larger CMake Project. It also allows for more implementation hiding between libraries. And it helps to keep all executable files much tidier (along with executable targets being a great deal simpler to define).

### Moving From Raw Pointers To Special Pointers

A new feature of C++11 and up is the addition of *special pointers*. These special pointers are reference counted (thus are deleted automatically when nothing is referencing them) and allow finer control over how they should behave as “pointers”. When it was realised that modern C++ advocates *against* the use of traditional raw pointers, it was set about updating what had been written to conform to the current standard.

### Generic Thread Pool

The most significant feature of ProtDev is its ability to handle multiple threads sending data to a target. For increased control and higher bandwidth generally. The original plan was to have threads handled on an as-required process. Whatever object required them would just be handed one. After some more deliberation, discussion and research, the Thread-Pool Queue pattern was discovered and implemented.

This method allowed the maximum number of threads to be limited from a central area. However, a thread pool for every possible return type a thread may have. One method would be to just do that, and create a new handler for each return type, another would be to do away with type safety and use a void\* return type.

C++11 has seen a significant change in C++ and how it is to be used and implemented. This includes the “auto” specifier and other functions dedicated to more generic programming that tie in with C++’s <*template*> functionality and to push more work onto the compiler to catch more potential run-time issues.

Auto only works if the compiler can figure out what the type will be upon the instantiation of the variable; however, using the *decltype* specifier from C++11 along with C++14’s ability to deduce a return type depending on template parameters, a totally generic method can be written that decides the types on compile-time, thus saving lines of code and maintenance effort, ultimately less code smells.

With all the tools available, in terms of the modern C++ standard, a single thread pool was created. Cutting maintenance drastically and creating a good robust solution that can be used across the system without too much specificity. This will also help with system configuration as it means the maximum threads can be defined more specifically.

### Network Communication

The C++ standard library, though incredibly powerful and featureful, still leaves areas to be desired. Though C++17 is supposed to be addressing these, there’s no proper documentation on it yet (nor is it fully implemented in the compiler in use, GCC). The basic communication handler will have to be written using the old BSD Sockets library (as they were in C on Linux). This also means that functions (as opposed to methods) need to be used, which can cause issues with the googletest/googlemock unit testing framework.

It was decided that the absolute first step in this would be to create a class wrapper around the BSD sockets library and pass that in as a parameter to the class. This way a class for mocking outputs/inputs can be written that can do things in a more OO way.

To verify results, a simple hellogoodbyeworld program is being written. The idea behind it is that it can be run and then ProtDev against it. It’ll run a very simple protocol that simply answers back when spoken to.

### Logger and Testing

The first thing that was implemented was the logger service. This was to better understand some aspects of the problem and to delve directly into C++’s new threading features. The logger has gone through many iterations. The original plan for the logger consisted of having the logs written to using a single thread that sat in a full “log handler”. Each log file object would be passed into whatever component needed it and it would then write to that. The log handler would then loop over each log file and flush its messages to the stream chosen for that log file.

The original logger library was deleted in favour of re-writing it with fresh eyes and more experience into its current state. A great deal of chaff was skimmed out and more significance was placed on using C++ features such as std::condition\_variable; a kind of mutex that allows users to notify other threads with the same variable that they may continue, sleeping them in between calls and waits.

It was also decided that a logger could instead be setup to use more generic streams and repurposed for outputting to *stdin* or *stderr* as well as the file streams they were intended for. The best way of implementing this was by using a strategy pattern that allows the component to *choose how* it wants its output to be displayed*.*

To try saving on global singletons some research was done into the best way of implementing this. Eventually, it was discovered how best to use a singleton[[4]](#footnote-5) that allows us to print to the log whenever, wherever, without having to pass loggers around the entire system.

### Time Keeping/Timer

The timeTicker class has gone through many iterations using C++’s own std::chrono library and using timeval/timespec. Eventually, a solution using both timeval *and* timespec was created (each for different precisions). Std::chrono was not used as it works better when the time does not need to be outputted into any sort of string format (i.e. human readable).

### Plugin Architecture

Plugins are a major feature of the project. As such it is very important that the loader is robust, well tested and, easy to use.

When performing the original legwork, in the form of isolated “spikes” (self-contained programs that were used for testing) that were used to verify that different parts of the software were at all possible or simply to see how new libraries work, tests were done using the *dlfcn.h* header. And originally the tests were successful, however, when applied to a more complex project that required passing C++ objects between the boundary things quickly decayed and it was found that casting between void\* and function pointers in C++ are not allowed (if it were to be done correctly to the standard).

So, instead, the boost.dll library was used as a robust and portable way of dynamically loading shared objects into the program. The caveat with this rested with boost not being totally compliant with C++1x, especially relating to shared and unique pointers, having to use the boost variants as opposed to the C++ ones[[5]](#footnote-6).

However, using one of the constructors of the shared\_ptr we move the boost::shared\_ptr into a std::shared\_ptr and vice-versa. While this may add an extra step when switching between libraries, it will certainly help with keeping things more uniform and, more importantly, standard within the software.

Boost came with its own complications. Especially regarding C++ standards. Due to the genuine requirement to use the latest and greatest parts of the C++ standard, the packaged boost libraries needed to be rebuilt to make use of this new standard and to actually build correctly. This was, eventually, successful by passing “cxxstd=17” to the “./b2” command upon building[[6]](#footnote-7).

This caused many of the boost libraries already present to break and cause dependency issues on the PC the project is being developed on. It was eventually resolved only by removing the offending, newly built, packages. Re-installing boost through the package manager. Then rebuilding the special version of boost needed to build the project and then retargeting the CMake module to pick up the correct version of the libraries for building against the project.

Once a stage was reached where it made sense to perform a build, several template issues were thrown up. Issues arose around what is returned by the import alias function. The sharedMap\_t<> was changed several times to reflect changes along with the import\_alias function. However, after countless configurations, it was discovered that the template of the import alias should contain only what the factory method itself returns and not other types (such as the boost/std::function in the import template). Areas that contained too many types in templates were also discovered.

Eventually a build was performed and was successful, however, it required rethinking many approaches to the templating in the within that area of code.

### Parsing Complications

After many false starts on attempting to creating a lexer/parser It was realised that that was beyond the scope of the project and not beneficial. After some research, the header-only library rapidxml was discovered[[7]](#footnote-8). It is fast and simple and does everything required of it without too much bloat.

After time in a separate spike understanding rapidXML and how it works, it was decided that it would be best to abstract the calls to rapidXML out to another class. While this is required for the purposes of unit testing and mocking, it also provides a platform that allows the parser to be swapped out at a later date without having to change the code in the gubbins of the project.

During development and feature verification, it was also discovered that rapidxml’s implementation of std::exception (rapidxml::parse\_error) provides a *where()* method for detailing where an error has been found. It returns a pointer to the location that can then be expanded by the user upon a failed analysis to detail where the parse error occurred. Thus, requirement 5.2.1 is immediately ticked off and making the implementation for it that much easier.

### Running a test

One of the final libraries that need to be created was that of the test runner itself. It consists of two components. The TestRunner “handler” class and “TestThread” objects. The TestRunner will setup the environment, create the test threads and finally run them.

Test threads will actually send the data using the specified communication using the created protocol and run for a certain period or until they’re cancelled/killed. Each test thread handles its *own* lifetime duration by a ratio of the:

*(th**e* *rate per second)/(nu**mber of threads)*

Currently, the software only has the “scaffolding” for more of the advanced features that need to be implemented due to time constraints and the ever so important need for the code to be easily extendable for future features, developers and maintainers to tinker with it.

### First, run

As soon as the testRunner was completed, a minimal test was done to make sure that what was currently implemented works to some degree. A simple executable was mocked up.

Immediately the software threw an exception. The plugin library claimed that the shared object was in an incorrect ELF format. This was because there was no file extension check. Having added that check another run was performed which instead caused a segfault as soon as any method within the plugin is called. This error is still being investigated.

### API

The API was one of the harder parts of the program as it was using technologies more foreign concepts than previously dealt with. It also required a lot of third-party libraries to make work; both in the API itself and in the communication using Google’s protobuf technology.

The first step was in figuring out the best way to package the required libraries or potentially to just distribute them with the software itself (to make building easier later on).

gRPC can be included as a part of the source in the “otherLibraries” directory which, with some tinkering, can handle its protobuf dependency too.

The next issue was using protobuf to generate classes based on the protobuf format that will be used to request information through gRPC and then integrating those source files into the projects own source tree.

## Testing

## Unit Tests

> All unit test output will be parsed and displayed here in a more readable format. I just need to figure out the best way of doing that. It’ll probably follow a similar layout as given below. <

## Acceptance Tests

Below is a table of acceptance tests. A successful test will contain a reference to proof of the success (if possible/applicable) in the “Action Note” field. The “Notes” field will contain any special notes. The “Action Note” field will contain notes relating to the success of the test; if a test fails it will contain what actions have been taken to make the software pass the test. Finally, the “Actioned” field will mark whether that was done. For tests that repeatedly fail, a log will be kept and placed into the Appendix of the report and referenced accordingly in the table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Notes | Success | Action Note | Actioned |
| PD\_1 |  |  |  |  |
| PD\_2 |  |  |  |  |
| PD\_3 |  |  |  |  |
| PD\_4 |  |  |  |  |
| PD\_5 |  |  |  |  |
| PD\_6 |  |  |  |  |
| PD\_7 |  |  |  |  |
| PD\_8 |  |  |  |  |
| PD\_9 |  |  |  |  |
| PD\_10 |  |  |  |  |
| PD\_11 |  |  |  |  |
| PD\_12 |  |  |  |  |
| PD\_13 |  |  |  |  |
| PD\_14 |  |  |  |  |
| PD\_15 |  |  |  |  |
| PD\_16 |  |  |  |  |
| PD\_17 |  |  |  |  |
| PD\_18 |  |  |  |  |
| PD\_19 |  |  |  |  |
| PD\_20 |  |  |  |  |
| PD\_21 |  |  |  |  |
| PD\_22 |  |  |  |  |
| PD\_23 |  |  |  |  |
| PD\_24 |  |  |  |  |
| PD\_25 |  |  |  |  |
| PD\_26 |  |  |  |  |
| PD\_27 |  |  |  |  |
| PD\_28 |  |  |  |  |
| PD\_29 |  |  |  |  |
| PD\_30 |  |  |  |  |
| PD\_31 |  |  |  |  |
| PD\_32 |  |  |  |  |
| PD\_33 |  |  |  |  |
| PD\_34 |  |  |  |  |
| PD\_35 |  |  |  |  |
| PD\_36 |  |  |  |  |
| PD\_37 |  |  |  |  |

# Evaluation

## What has been shown

It has been shown, to some extent, that a highly configurable and extensible load testing software can work and can exist. While every requirement and every nicety may not exist within the software, those are exactly as stated, niceties.

It can definitely be stated that what was set out to discover is definitely possible

## What is still outstanding

No idea

# Conclusion

## Future enhancements

In improving the software based on leftover requirements (and to keep it more in-line with current software trends) implementing the "XaaS” requirements would be a great boon to the software; allowing it to compete in a very real way with current enterprise solutions.

As a way of improving the *types* of protocols tested and increasing the number of different test modes/test starts, different runners can be written that behave slightly differently. Perhaps first waiting for a signal, or even needing to work with two separate protocols.

Non-functionally, it would be very advantageous to include some template classes for implementing plugins with examples on how to write more complicated protocol plugins that allow more complex interpretation of the parsed testcase.

## Into the future

The project has a great deal of room to grow. Feature-wise, efficiency wise, Ease-Of-Use wise. In its current state it does leave a lot to be desired, however, it has been designed and created in such a way that adding new parts would be a near pleasure to do.

It would even be possible to try to create a greater focus on *hardware* load testing in the communication aspect of the program. Investigations should definitely be carried out regarding expanding the industries that would benefit from the use of this software.

# References

Ahmad, M.O., Dennehy, D., Conboy, K. & Oivo, M. 2018, “Kanban in software engineering: A systematic mapping study”, *The Journal of Systems & Software,* vol. 137, pp. 96-113.

Akhshani, A, Akhavan, A., Mobaraki, A., Lim, S. and Hassan, Z. (2014) Pseudo-Random Number Generator Based on Quantum Chaotic Map. Communications in Nonlinear Science and Numerical Simulation. 19 (1), pp. 101-111.

Baltazar, H. 1998, “Iometer is best for testing servers but shouldn't be used when buying them”, *PC Week*, vol. 15, no. 50, pp. 104.

Bayan, M.S. and Cangussu, J.W. (2006) Automatic Stress and Load Testing For Embedded Systems. Computer Software and Applications Conference (Compsac'06).

Bayan, M and Cangussu, J.O.A.O. (2008) Automatic Feedback, Control-based, Stress and Load Testing. Symposium on Applied Computing., pp. 661-666.

Bhatia, S., Schmidt, D., Mohay, G. and Tickle, A. (2014) A Framework For Generating Realistic Traffic For Distributed Denial-of-service Attacks and Flash Events. Computers & Security. 40, pp. 95-107.

Casado, R., Tuya, J. & Younas, M. 2015, “Evaluating the effectiveness of the abstract transaction model in testing Web services transactions”, *Concurrency and Computation: Practice and Experience*, vol. 27, no. 4, pp. 765-781.

Chapuis, B. & Garbinato, B. 2017, “Scaling and Load Testing Location-Based Publish and Subscribe”, *IEEE*, pp. 2543.

Cico, O. & Dika, Z. 2014, “Performance and load testing of cloud vs. classic server platforms (Case study: Social network application)”, *IEEE*, pp. 301.

Draheim, D. and Weber, G. (2005) Modelling Form-based Interfaces with Bipartite State Machines. Interacting with Computers. 17 (2), pp. 207-228.

Draheim, D., Grundy, J., Hosking, J., Lutteroth, C. and Weber, G. (2006) Realistic Load Testing of Web Applications. Conference on Software Maintenance and Reengineering (Csmr'06).

Ee Mae Ang, K.W.Y.H.P.K.K.P. (2015) A performance analysis on packet scheduling schemes based on an exponential rule for real-time traffic in LTE. *EURASIP Journal on Wireless Communications and Networking*, (2015), p.201.

Garg, M. & Lavhate, N. 2017, “Web Service Testing Automation using SoapUI Tool”, *International Journal of Computer Applications*, vol. 167, no. 2, pp. 23-28.

Grehan, R. 2005, SOAPtest 4.0 targets Web services — New security tests, load testing make for squeaky-clean Web services, *InfoWorld Media Group, Inc.*

Hasenleithner, E. and Ziegler, T. (2003) Comparison of Simulation and Measurement Using State-of-the-Art Web Traffic Models. Ieee Symposium on Computers and Communications.

Hari Balakrishnan, V.N.P.R.H.K. (1999) The effects of asymmetry on TCP performance. *Mobile Networks and Applications*, (4), pp.219-41.

Hwang, G., Chang, S. & Chu, H. 2004, “Technology for testing nondeterministic client/server database applications”, *IEEE Transactions on Software Engineering*, vol. 30, no. 1, pp. 59-77.

Jiang, Z.M. & Hassan, A.E. 2015, “A Survey on Load Testing of Large-Scale Software Systems”, *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1091-1118.

Jiang, Z.M.J. 2015, “Load Testing Large-Scale Software Systems”, *IEEE*, pp. 955.

Jumar, S. (2017) Simulating DDoS Attacks on the U.S. Fiber-optics Internet Infrastructure. 2017 Winter Simulation Conference (WSC).

Koh, N., Li, Y., Li, Y., Xia, L., Beringer, L., Honoré, W., Mansky, W., Pierce, B.C. & Zdancewic, S. 2018, “From C to Interaction Trees: Specifying, Verifying, and Testing a Networked Server”, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*.

Malik, H. (2010) A Methodology to Support Load Test Analysis. Acm/IEEE International Conference on Software Engineering. 2, pp. 421-424.

Na, Q. & Huaichang, D. 2015, “Extension based on robot framework and application on Linux server”, *IEEE*, pp. 293.

Pagotto, T., Fabri, J.A., Lerario, A. & Goncalves, J.A. 2016, “Scrum solo: Software process for individual development”, AISTI, pp. 1.

Menasce, D.A. (2002) Load Testing of Web Sites. Ieee Internet Computing. 6 (4), pp. 70-74.

AlertSite Launches Outsourced Load Testing Service; Web Monitoring Company Offering Full-Service Web Performance Management With Industry Leader Mentora 2004, *PR Newswire Association LLC*.

Rubin, K.S. 2013, *Essential Scrum: a practical guide to the most popular Agile process,* Addison-Wesley, London;Upper Saddle River, N.J;.

Sajal Bhatia, D.S.G.M.A.T. (2014) A framework for generating realistic traffic for Distributed Denial-of-Service attacks and Flash Events. *Elsevier*, (40), pp.95-107.

Shimomura, T. 2012, “Automated Server-Side Regression Testing for Web Applications”, *International Journal of Computers and Applications*, vol. 34, no. 2, pp. 119-126.

Shojaee, A., Agheli, N. & Hosseini, B. 2015, “Cloud-based load testing method for web services with VMs management”, *IEEE*, pp. 170.

Sumeet Kumar, K.M.C. (2017) SIMULATING DDOS ATTACKS ON THE US FIBER-OPTICS INTERNET INFRASTRUCTURE. In *Winter Simulation Conference*. Las Vegas, NV, 2017. IEEE.

Sun, J. & Mannisto, T. 2012, “Usefulness Evaluation of Simulation in Server System Testing”, *IEEE*, pp. 158.

Yan, M., Sun, H., Liu, X., Deng, T. and Wang, X. (2014) Delivering Web Service Load Testing as a Service with a Global Cloud. Concurrency and Computation. 27 (3), pp. 526-545.

Yates, D. (2006) Editorial: Uncertainty and chaos. *SOFTWARE TESTING, VERIFICATION AND RELIABILITY*, (16), pp.69-70.

Zhang, P., Elbaum, S. & Dwyer, M. 2011, “Automatic generation of load tests”, *IEEE Computer Society*, pp. 43

APPENDIX

1. <https://softwareengineering.stackexchange.com/a/363706> [↑](#footnote-ref-2)
2. <https://www.raywenderlich.com/585-scrum-of-one-how-to-bring-scrum-into-your-one-person-operation> [↑](#footnote-ref-3)
3. <https://rix0r.nl/blog/2015/08/13/cmake-guide/> [↑](#footnote-ref-4)
4. <https://stackoverflow.com/questions/5877779/c-logger-class-without-globals-or-singletons-or-passing-it-to-every-method> [↑](#footnote-ref-5)
5. <https://stackoverflow.com/questions/12314967/cohabitation-of-boostshared-ptr-and-stdshared-ptr#comment28669000_12315035> [↑](#footnote-ref-6)
6. <https://gist.github.com/dennycd/5890475> [↑](#footnote-ref-7)
7. <https://stackoverflow.com/questions/9387610/what-xml-parser-should-i-use-in-c> [↑](#footnote-ref-8)