

“Generic Kubernetes based load testing framework ”



Trinity
College
Dublin

The University of Dublin

Project Development Report

Client | Client: Rapid7

Group 5 | Developers:

Second Years:

Holly Daly - 20331814
Declan Quinn - 20334565
Miguel Arrieta - 20332427
Abushark Abdelaziz - 20332134

Third Years:

Aaron Byrne - 19334098
Tomasz Bogun - 19335070
Vitali Borsak - 19335086

Table of Contents

1. Introduction	3
1.1. Background & Problem Statement	3
1.2. Technical Approach	4
2. Requirements	4
2.1. Functional Requirements	4
2.2. Non-functional Requirements	4
2.3. User Interaction Scenarios	5
3. Design	5
3.1. Architecture Design	5
3.2. UML Class and Sequence Diagrams	6
4. Implementation	7
4.1. Tools, Libraries, Platforms	7
4.2. User Interfaces	7
4.3. Algorithms	9
5. Conclusions	9
5.1. Design and Implementation	9
5.2. Project Objectives	10
5.3. The Team	10
5.4. Algorithms	10
Appendix 1	11
Appendix 2	20

1. Introduction

1.1. Background & Problem Statement

Background : Our assignment involved designing file format with testing configuration and designing the testing framework as well as creating a mock for the project under test and deploying it to MiniKube

Problem Statement: Our objective was to have a configurable performance testing framework, which can be used with Docker Containers and Kubernetes to run requests against HTTP APIs and obtain results. Our goal was to do so by creating a mock search engine that will search through vast amounts of data for the information that the user needs. The system will run performance testing on this piece of software and report the results. The test runner and project under test will be packaged up to a Docker Container and deployed to a Kubernetes cluster. The user interface will monitor and graph the performance of the system.

1.2. Technical Approach

We approached the problem by assigning ourselves up into three teams. The project under test team, the Test Runner team and the Docker and Kubernetes Team. Each team had assigned a third year to lead and one or more second years. During our weekly meetings the third years who have done this project last year were able to convey and understand what was required to be completed by their team and for when. Also during our meetings it was discussed how various parts of the project would interact with each other and how each team should design their files to be intractable with other aspects of the project. GitHub was used for version control as teams worked on their own branches and merged back to main once completed. For the most part we settled on languages that group members were familiar with during design where applicable. Finally we had scheduled weekly meetings with our client when available to run through our progress with the project and to get feedback to address any inconsistencies early.

2. Requirements

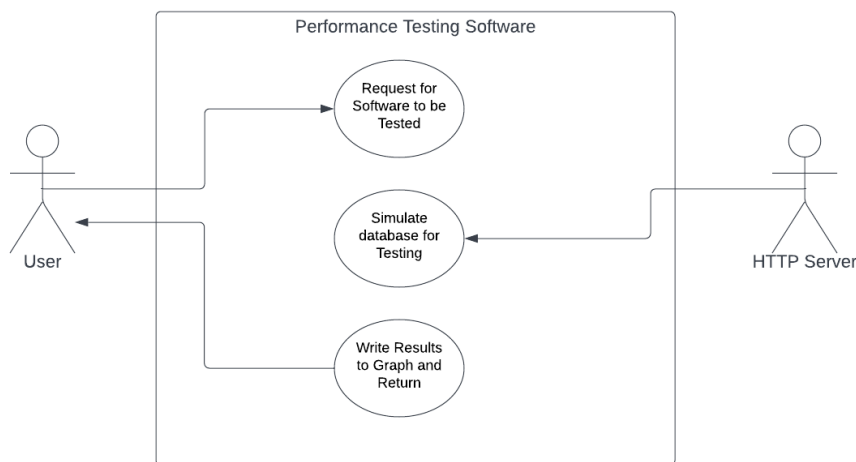
2.1. Functional Requirements

- Performance testing application accessed through a command line interface.
- Include testing of a HTTP request's speed.
- Ability to keep track of changes in performance, for example in different versions of the software.
- Needs to be reproducible for every software release.
- Provides tests in isolation, through the use of a separate Kubernetes cluster.

2.2. Non-functional Requirements

- Ease of use. The project should make it easy for the client to test the project under test within a Kubernetes cluster and obtain and view the results.
- The project should be easily scalable to fit the needs of the client's actual project they want to test. Include the possibility of automating the expansion process to add in more docker containers to improve performance of the test runner.
- The project under test has to perform well as it needs to handle files with millions of lines of text and search through them.
- Our client's project is a long term project and requires this test runner to be easily maintainable and portable (Hence docker and Kubernetes).

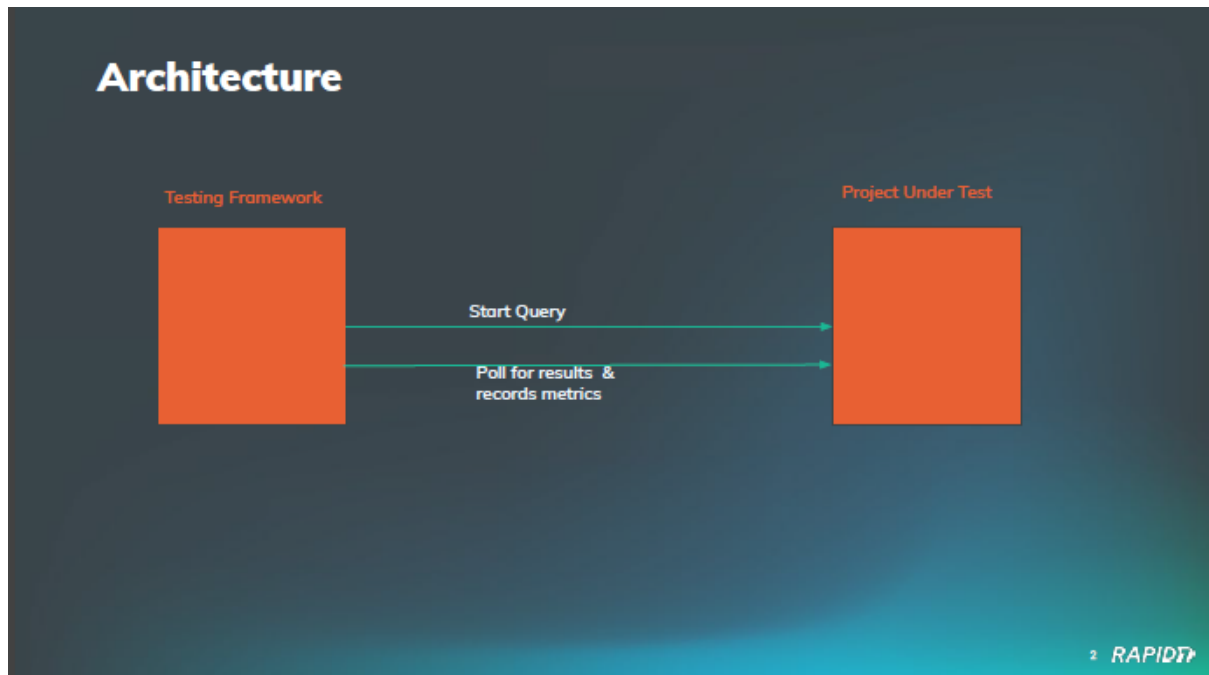
2.3. User Interaction Scenarios



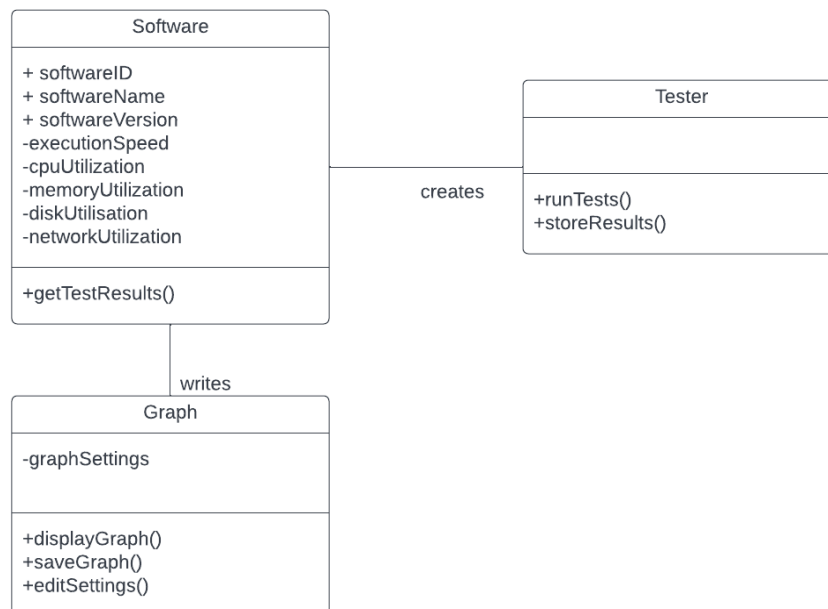
As outlined in the use case diagram above the user typically will send a request for one or more different pieces of software to be tested. The HTTP server in our case is just a server where we are simulating a database as we were required to make one ourselves. Once the user has entered the desired number of requests the system will return an interactive graph of the various runtimes of these requests.

3. Design

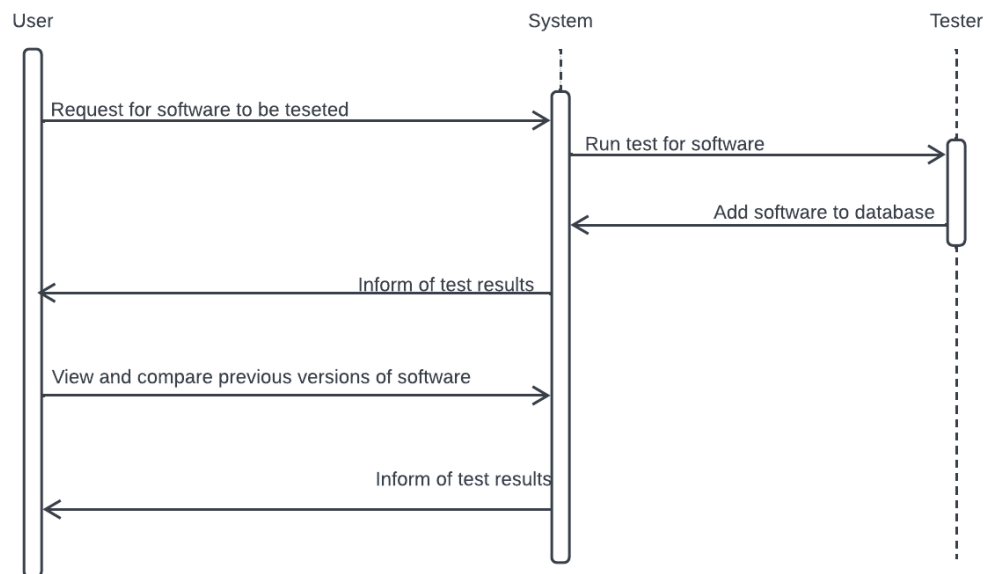
3.1. Architecture Design



3.2. UML Class and Sequence Diagrams



The object model diagram above consists of a software object that in turn creates a test object that runs performance tests on the various pieces of software created.



Above shows a sequence diagram where a user requests to test a piece of software and compare it with previous versions.

- The user sends a request to the system to test a piece of software
- The system runs this test on our test runner
- The test runner returns the software and stores in a database
- The user may request a graph of the test result
- The user can then add previous versions of the data to the system for comparison
- The user can then get this comparison of results back from the system.

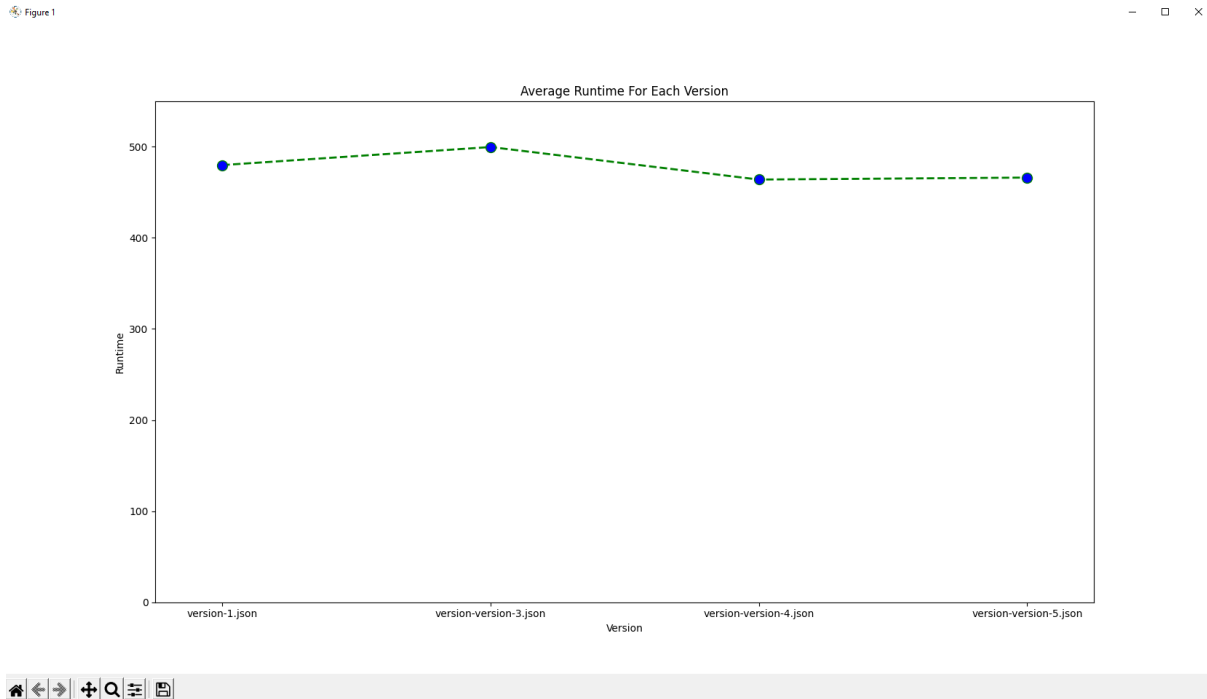
4. Implementation

4.1. Tools, Libraries, Platforms

- **Tools and Libraries**
 - The project under test (mock server) was built using python and Flask.
 - The Test Runner was built in Java, making use of the Jackson API to read and create JSON files efficiently and Apache libraries for HTTP requests and statistics.
 - Docker was used for containerisation and Kubernetes to deploy these containers.
- **Platforms**
 - GitHub was used for version control, along with Git.
 - The software is containerised with Docker, which can then be deployed through a Kubernetes cluster.

4.2. User Interfaces

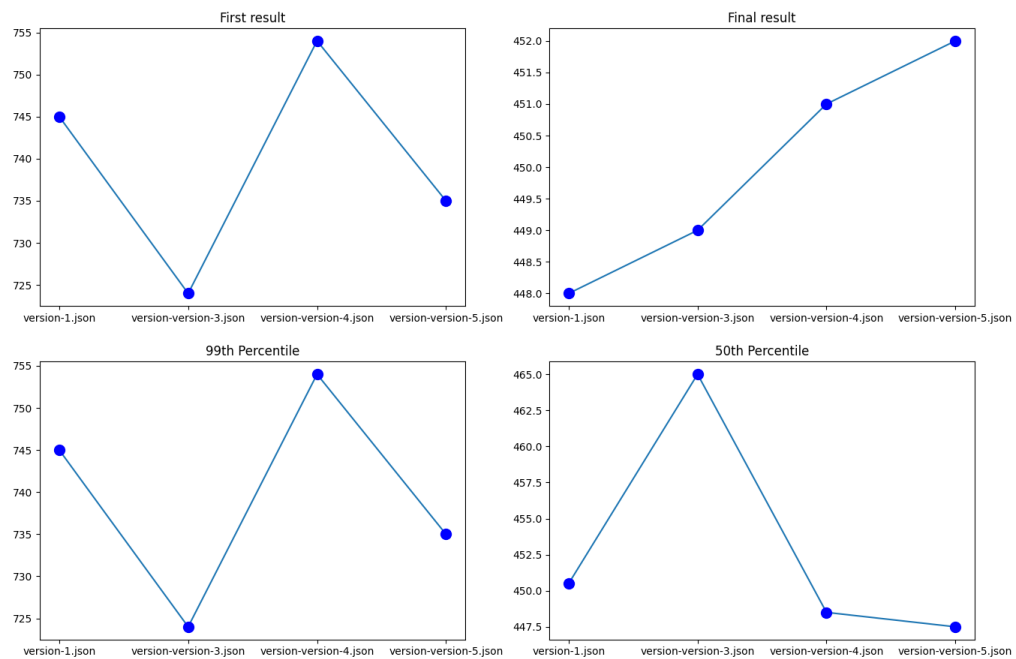
- Due to the nature of our project, the user mainly interacts with a terminal and enters command line parameters to use our software. We implemented a graphing user interface using matplotlib to show the results of our tests.



- The program would generate a screen where a representation of the average runtimes of each version of a file was plotted, where version was plotted along the x-axis and the runtime was plotted along the y-axis.



- Various buttons at the bottom left of the screen allowed the user to perform actions with the graph. The home icon would restore the original graph and undo any edits performed by the user. Both arrow icons <- and -> acted as undo and redo buttons to help the user restore a previous view of the screen. The cross button allowed the user to use their mouse to drag the data on the screen to change the position of the data (Ex to view the data exclusively from above a runtime of 200). The magnifying glass acted as a zoom where the user could draw a box and zoom the graph in on that selected box. After the magnifying glass was the configurations button where users could configure the display of the subplots on the screen. The save icon at the end allowed the user to save the graph for future use.



- The next screen that would open once the average runtime was closed was a screen of subplots displaying the runtimes for various versions for a specific piece of data. Likewise the user could also interact similarly with these as in the previous screen.



Subplot configuration tool

- □ ×

Click on slider to adjust subplot param



Reset

- The subplot configuration tool (set to default settings) allowed the user to drag various sliders that would change the positions of each subplot on the screen. The reset button returned the sliders to their default positions.

4.3. Algorithms

Algorithm	Method
This algorithm performs the requests provided through the configuration file and times each one, storing each time to an efficient data structure for calculating the necessary statistics	TestRunner::performTests
This algorithm is to be called after TestRunner::performTests and it first calculates the necessary statistics for the results file, then creates the file or overwrites an existing one and finally writes the calculated values to the file in a structured format.	TestRunner::writeResultsFile

5. Conclusions

5.1. Design and Implementation

- We believe we have designed and implemented an efficient solution to the clients problem addressed to us. One of the biggest challenges we faced when designing our project was that many members of the team had to adjust themselves to using new technologies. We found that our third years were more than happy to help us with our coding as they had more experience than us, and proved very helpful when engaging with difficult tasks. Our client also provided us with advice on how we should implement various tasks and gave us good starting points to work from when we faced challenges.
- We had to be realistic with our implementation of our design. We had hoped to implement our UI by using react.js. We chose this as it seemed to have many useful features for creating user interfaces and was recommended by one of our third years as he had experience with it. However it was realised due to time constraints, two second years assigned to the UI could not pick up react.js in the timeframe we had left to finish the project for the presentation so we chose to use python instead, which also proved to be very effective.
- Our one downfall as a group was that we were unable to get the program working as part of a kubernetes cluster. Although our program performs the task at hand we could not implement this on a lightweight kubernetes cluster due to a combination of inexperience with the software and the difficulty of the task at hand. We still managed however to produce a quality product that fit the description of what the client wanted the software to do. Our client, with more experience in the use of kubernetes should be able to implement a kubernetes cluster easily with the product we have given should they choose to use it for more purposes.

5.2. Project Objectives

We believe that we have delivered a very good project to the client and have addressed many of their needs and requirements they outlined to us.

The following tasks were performed by the team during the project:

1. We created a Test Runner, a configuration file driven test executor that would perform various tests as configured
2. We created a Project under test HTTP server to simulate a database
3. We acquired a sample datasheet (.txt files) that were used for performance testing
4. We Created a results UI using python to graph results from the Test Runner
5. We containerised the Project under test and the Test Runner in docker containers

5.3. The Team

- The general consensus among the group is that we worked very well together.
- Attendance to meetings in general was perfect throughout the entire semester and had a welcoming atmosphere with plenty of team spirit.
- Discord was used to communicate to other members and was active on a daily basis so that team members could easily request help from each other.
- Second years and third years integrated very well with each other in almost all aspects. There was a clear understanding of the level of each individual and that helped to build support among the members.
- All members completed their work before the deadline.
- During the meetings the ideas of each member were considered and spoken about.

5.4. Algorithms

- The performTests algorithm performs the HTTP requests to the HTTP server specified in the configuration file, and then times these requests, repeating it as many times as was specified by the configuration file. This is the main algorithm used for the project as this collects all the necessary data for creating the results file.
- The writeResultsFile algorithm is to be called after performTests and, as the name suggests, writes the results file. This is how we summarise the data collected in performTests in a structured format, that can easily be read by humans and machines.

Appendix 1

CS202122-CSU33013 Requirements Documentation

“Generic Kubernetes based load testing framework”



Trinity College Dublin

The University of Dublin

Requirements Documentation

This document contains, through both diagrams and descriptions a listing of what functionalities we wish to implement in our testing framework.

Client | Client: Rapid7

Group 5 | Developers:

Second Years:

Holly Daly - 20331814

Declan Quinn - 20334565

Miguel Arrieta - 20332427

Abushark Abdelaziz - 20332134

Third Years:

Aaron Byrne - 19334098

Tomasz Bogun - 19335070

Vitali Borsak - 19335086

Table of Contents

Appendix 1	11
Introduction	13
1.1. Overview - Purpose of system	14
1.2. Scope	14
1.3. Objectives and success criteria	14

1.4. Definitions, abbreviations	15
2. Current system	15
3. Proposed System	15
3.1. Overview	15
3.2. Functional Requirements	16
3.3. Non-functional requirements	16
3.4. System prototype (models)	16
3.4.1. User interface mock-ups	16
3.4.2. Use cases (including text narratives)	17
3.4.3. Object model	18
3.4.4. Dynamic model	19
4. Proof of client sign off:	20

1. Introduction

1.1. Overview - Purpose of system

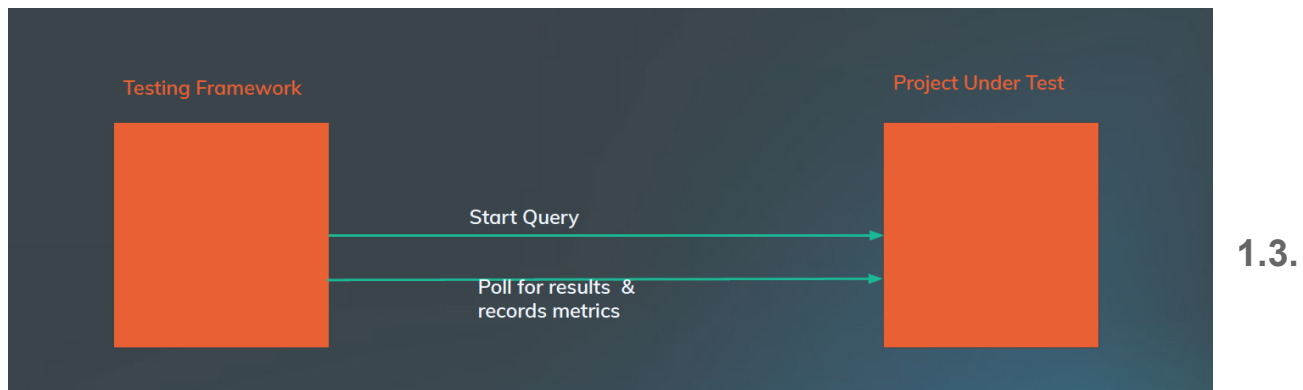
This software is a search engine that searches vast amounts of data for specific information they need. The data that is being searched can reach terabytes in size, so it is crucial for the search engine to be as efficient as possible to speed up workflow. The proposed system will run performance tests on the piece of software and report back the results. In order to eliminate any external factors that can affect the performance, this testing will be run on an isolated Kubernetes Cluster. This system will be able to analyse the changes in performance of the different software release versions and be able to see if a new release has a negative or positive impact on the performance. The user will be able to query and poll the software under test using custom file formats and testing configurations.

1.2. Scope

The scope for this project is a configurable testing framework which can be used with Kubernetes and Docker Containers to run requests against HTTP API'S, time their responses and obtain results. The main goals for this project: ability to track changes in performance, reproducible for every software release and performance testing of a search

engine. This project will include different components such as Test Runner, Sample Dataset, Kubernetes, etc.

Template of the whole process:



Objectives and success criteria

Our objectives are to split the work up into three sections and have groups working on each. One group will be working on a project under test using a sample database, another will work on a test runner and test runner configuration and the last on Dockers and Kubernetes. Our project will be deemed as a success if we have a configurable performance testing framework, which can be used with Docker Containers and Kubernetes to run requests against HTTP APIs, time their responses, and obtain results.

1.4. Definitions, abbreviations

- HTTP - Hypertext Transfer Protocol
- API - application programming interface
- CPU - central processing unit
- Docker - A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.
- Kubernetes - Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

2. Current system

Currently there is no system in place for performance testing that works with Docker Containers and Kubernetes that runs requests against HTTP APIs in an isolated test environment. Due to this the system needs to be created from scratch.

3. Proposed System

3.1. Overview

The main objective of the proposed system is to analyse the performance of a piece of software. In our client's case, this software is a search engine that searches vast amounts of data for specific information they need. The data that is being searched can reach terabytes in size, so it is crucial for the search engine to be as efficient as possible to speed up workflow. The proposed system will run performance tests on the piece of software and report back the results. In order to eliminate any external factors that can affect the performance, this testing will be run on an isolated Kubernetes Cluster. This system will be able to analyse the changes in performance of the different software release versions and be able to see if a new release has a negative or positive impact on the performance. The user will be able to query and poll the software under test using custom file formats and testing configurations. Our client has asked us to deploy the project under test to MiniKube which is a local Kubernetes cluster. The system will start off as a command line program, but if time permits, may also have a graphical user interface. After discussion with our team, we decided that it will be best to use either python or java for this project as our team is familiar with these two languages.

3.2. Functional Requirements

- Performance testing application accessed through a command line interface.
- Should include time testing and if constraints permit, testing of:
 - CPU utilisation
 - Memory utilisation
 - Disk utilisation
 - Network utilisation
- Ability to keep track of changes in performance, for example in different versions of the software.
- Needs to be reproducible for every software release.
- Provides tests in isolation, through the use of a separate Kubernetes cluster.

3.3. Non-functional requirements

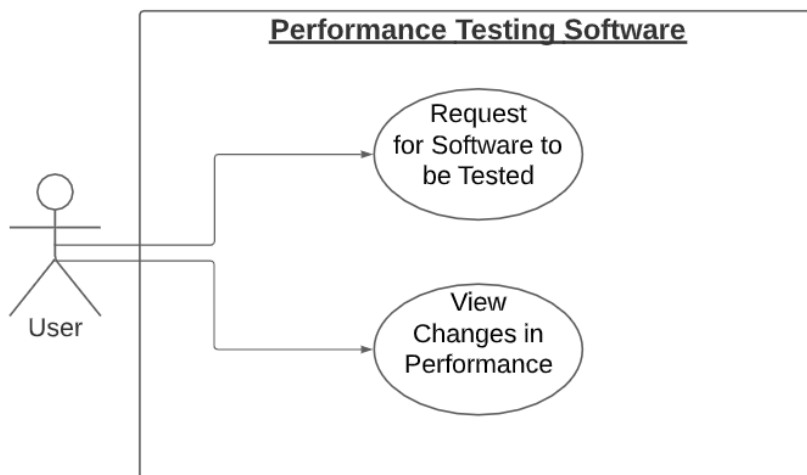
- Ease of use. The project should make it easy for the client to test the project under test within a Kubernetes cluster and obtain and view the results.
- The project should be easily scalable to fit the needs of the client's actual project they want to test. Include the possibility of automating the expansion process to add in more docker containers to improve performance of the test runner.
- The project under test has to perform well as it needs to handle files with millions of lines of text and search through them.
- Our client's project is a long term project and requires this test runner to be easily maintainable and portable (Hence docker and Kubernetes).

3.4. System prototype (models)

3.4.1. User interface mock-ups

A user interface is not required, as a command-line interface will suffice.

3.4.2. Use cases (including text narratives)

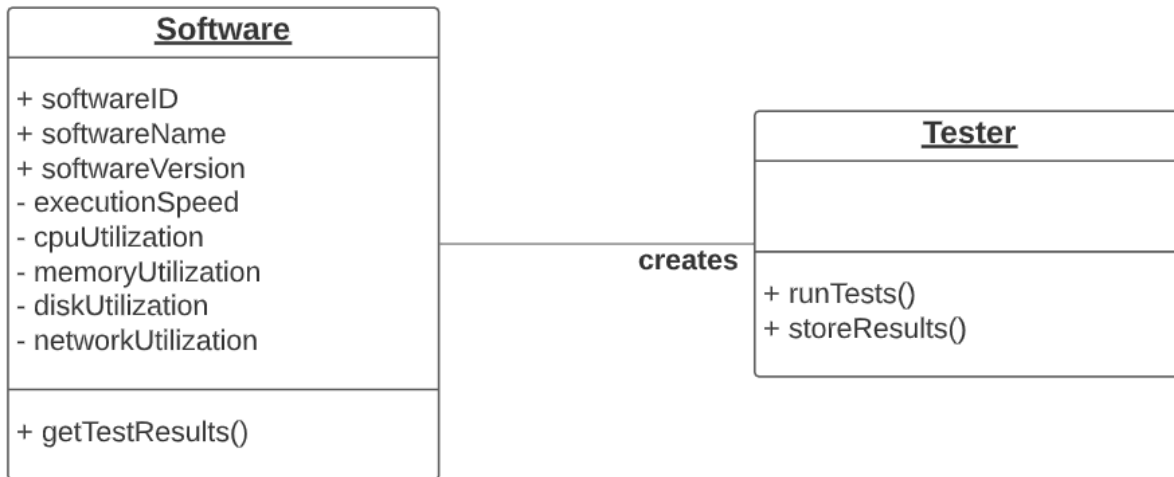


Request Software to be Tested	User	System
1.1	Request for a software to be tested for its efficiency	The system runs test on the software with various data.
1.2		The system returns the test results to the user and stores them.

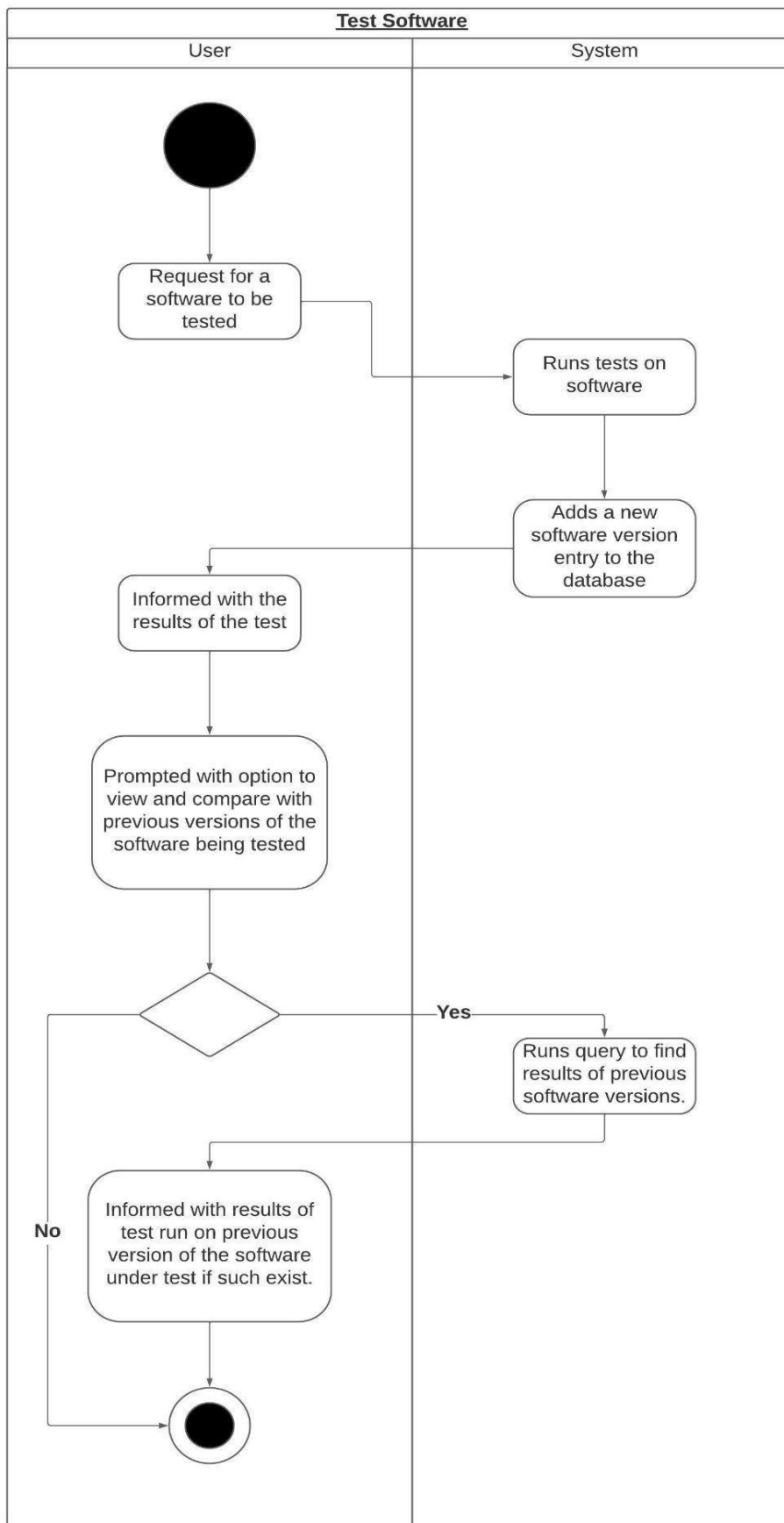
Changes in Performance	User	System
1.1	Request to view and compare the test results of previous versions of the software under test.	The system checks if there is any test data on previous versions of the current software under test.
1.2 (a)		The system returns the old test results and compares them with the most recent test results for the software under test.
1.2 (b)		The system returns null as there

		is no test results for the software under test.
--	--	---

3.4.3. Object model




3.4.4. Dynamic model



4. Proof of client sign off:

Client signed off on the document and we have organised weekly meetings to review progress.




Aaron Byrne

Brilliant thanks very much Ilya!

11:58 (1 hour ago)

☆



Ilya Biryukov

to me ▾

The requirement looks good.
I guess the detailed requirements will be worked through each time and we can review them separately?

12:53 (21 minutes ago)

☆

↩

⋮

Ilya

...

...

...

“Generic Kubernetes based load testing framework”



Trinity
College
Dublin

The University of Dublin

Software Design Specification Document

Client | Client: Rapid7

Group 5 | Developers:

Second Years:

Holly Daly - 20331814
Declan Quinn - 20334565
Miguel Arrieta - 20332427
Abushark Abdelaziz - 20332134

Third Years:

Aaron Byrne - 19334098
Tomasz Bogun - 19335070
Vitali Borsak - 19335086

Table of Contents

1. Introduction	22
1.1. Overview - Purpose of the System	22
1.2. Scope	22
1.3. Definitions, abbreviations	23
2. System Design	23
2.1. Design Overview	23
2.1.1 High-level overview of how the system is implemented, what tools, frameworks and languages are used etc.	24
2.2 System Design Models	25
2.2.1 System Context	25
2.2.2 Use Cases	26
2.2.3 System Architecture	26
2.2.4 Class Diagrams	27
2.2.5 Sequence Diagrams	27
2.2.6 State Diagrams	28

1. Introduction

1.1. Overview - Purpose of the System

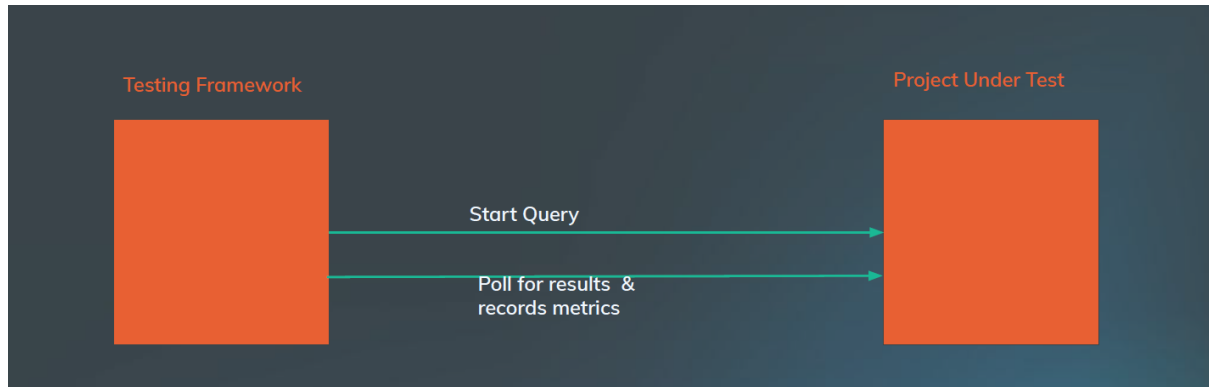
This software is a search engine that searches vast amounts of data for specific information they need. The data that is being searched can reach terabytes in size, so it is crucial for the search engine to be as efficient as possible to speed up workflow. The proposed system will run performance tests on the piece of software and report back the results. In order to eliminate any external factors that can affect the performance, this testing will be run on an isolated Kubernetes Cluster. This system will be able to analyse the changes in the performance of the different software release versions and be able to see if a new release has a negative or positive impact on the performance. The user will be able to query and poll the software under test using custom file formats and testing configurations.

1.2. Scope

The scope for this project is a configurable testing framework that can be used with Kubernetes and Docker Containers to run requests against HTTP APIs, time their responses, and obtain results. The main goals for this project: the ability to track changes in performance, be reproducible for every software release, and performance testing of a

search engine. This project will include different components such as Test Runner, Sample Dataset, Kubernetes, etc.

Template of the whole process:



1.3. Definitions, abbreviations

- HTTP - Hypertext Transfer Protocol
- API - application programming interface
- CPU - central processing unit
- Docker - A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings.
- Kubernetes - Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

2. System Design

2.1. Design Overview

The system has multiple components that can be worked on together in multiple separate teams. There are three sub-teams in our group with 2-3 members each.

The first component is the project under test. This is the piece of software whose performance will be measured. The project under test can be any piece of software, but our client has requested that we make a specific software for testing. It will be an HTTP server that will search through a large text file and search for a given string and report back the results.

The HTTP server will be able to accept a POST and a GET request. When a POST request is received, the server will begin to search for a string provided in the body of the request, in

a file whose name is also provided in the body of the request. Then, the server will begin the search and send back a unique ID. This ID can be then used to view the results of this search using a GET request. When this happens, the server will report the number of matches found, along with the status of the search. The client will be testing internal software that resembles the functionality of the project under test that we will develop.

The second component is the test runner and test runner configuration. This is the software that will perform the tests on the project under test. The test runner will be a script that sends commands to the project under test, waits for responses, and records the results and performance. The test runner will be able to take in a configuration file as input called the test runner configuration. This file will contain all the commands that will be run by the test runner. This means that the test runner will be able to run different tests based on the configuration file that is provided.

The third component will be packaging up the above components into docker and Kubernetes containers so that they can be run in isolation. A dockerfile will be required for this that initialises the containers. The containerization will also allow for the performance monitoring of containers using built-in functionality. This component will also involve an auto-scaling feature for the project under test that will allow it to use all of the resources it has access to efficiently.

2.1.1 High-level overview of how the system is implemented, what tools,

frameworks and languages are used etc.

The main objective of the proposed system is to analyze the performance of a piece of software. In our client's case, this software is a search engine that searches vast amounts of data for specific information they need. The data that is being searched can reach terabytes in size, so it is crucial for the search engine to be as efficient as possible to speed up workflow. The proposed system will run performance tests on the piece of software and report back the results. In order to eliminate any external factors that can affect the performance, this testing will be run on an isolated Kubernetes Cluster.

→ Programming Languages and Framework

◆ The project will be built using:

- Python
- Java
- [Dropwizard](#)
- [Flask](#)
- Docker
- Kubernetes

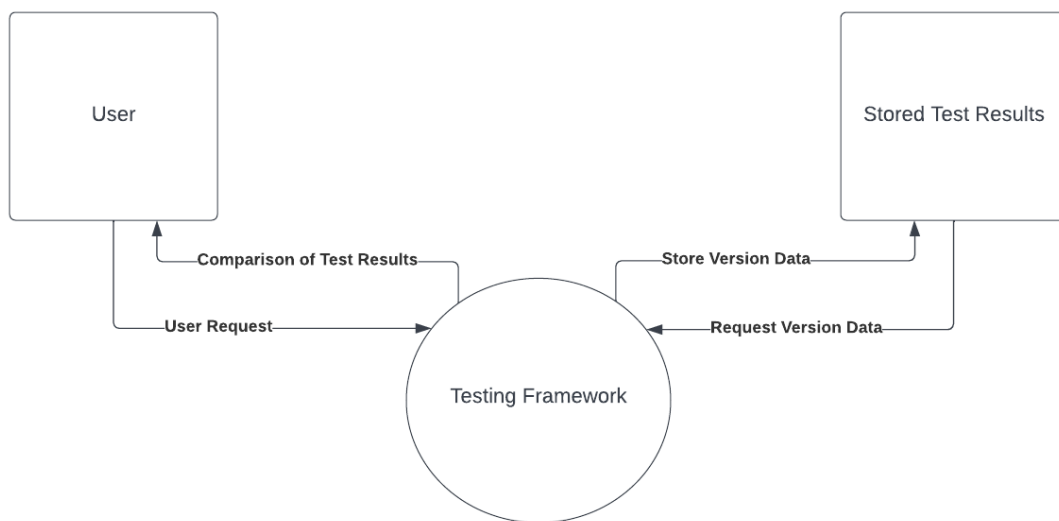
→ Version Control

- ◆ Git and GitHub are used for version control.
- ◆ Jira will be used for workflow management
- ◆ Jenkins will be used as our CI/CD platform

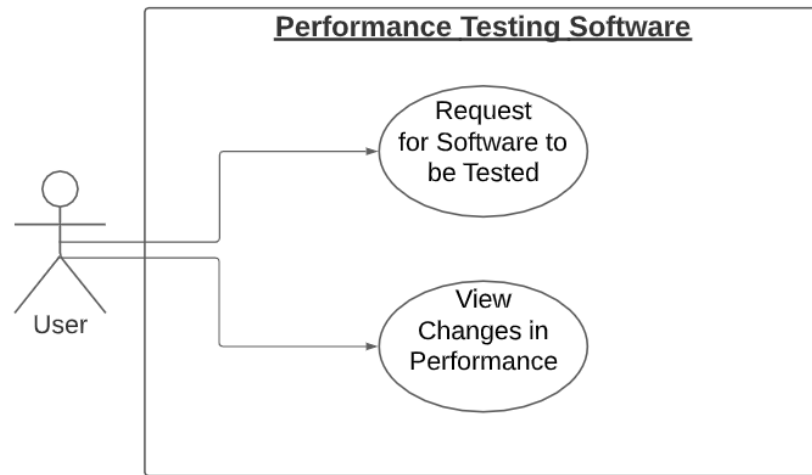
2.2 System Design Models

2.2.1 System Context

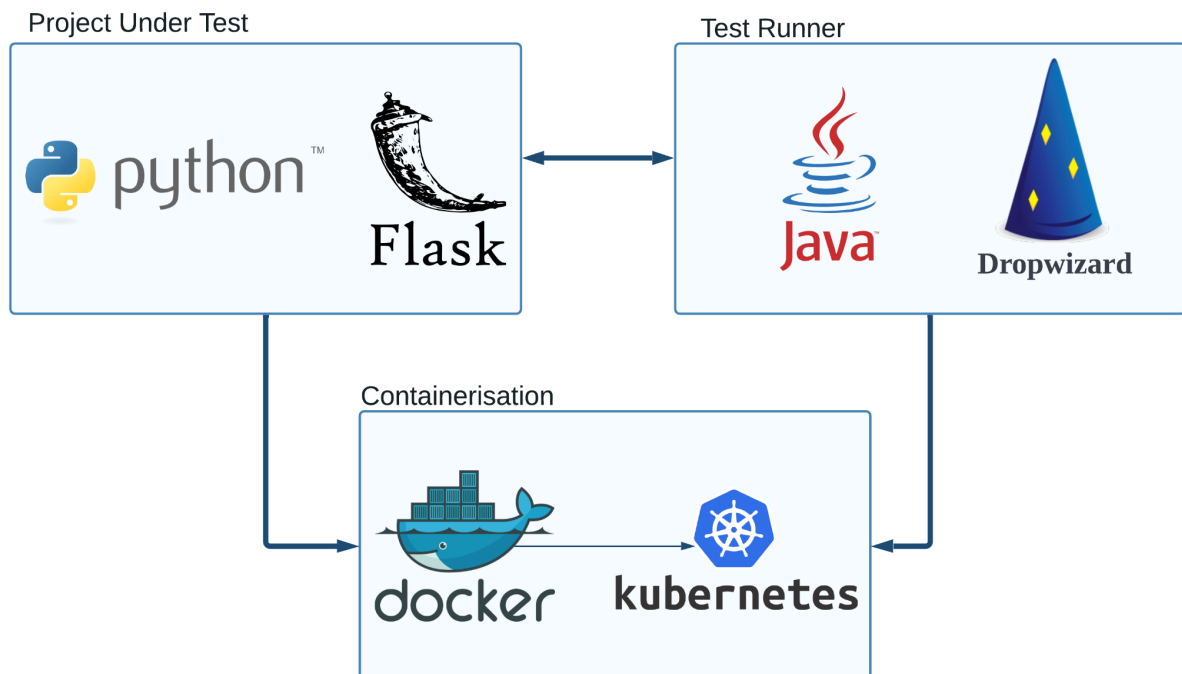
Our system allows the user to send a request to test a piece of software. This is tested and the results of the current version are saved. The user is then returned the test data along with information on how previous versions of the software performed under testing if requested.



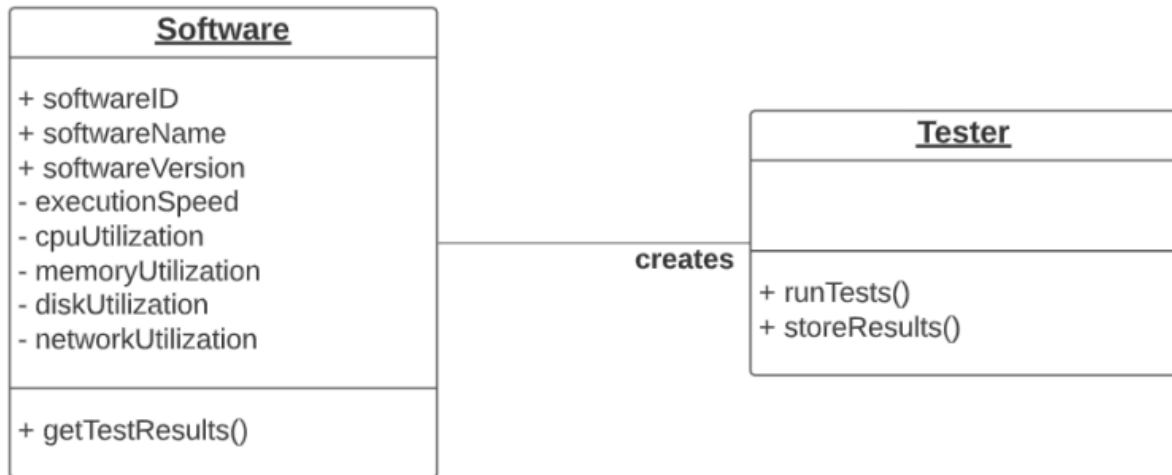
2.2.2 Use Cases



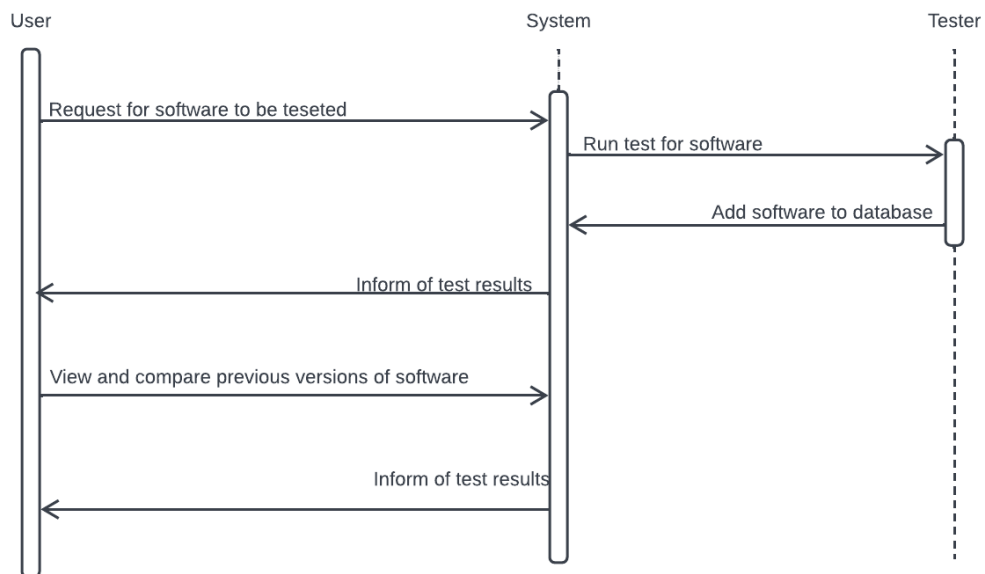
2.2.3 System Architecture



2.2.4 Class Diagrams



2.2.5 Sequence Diagrams



2.2.6 State Diagrams

