# GENERIC KUBERNETES BASED LOAD TESTING FRAMEWORK

Client: Ilya Biryukov – Rapid7

Group 5

**Second Years:**

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

**Third Years:**

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

## Contents

## 1. Project Goals and Objectives

### 1.1. Background

Our client is Rapid7. They are a cyber-security company that provides corporations with security data and analytics solutions that help enable them to keep an active approach to cyber security which is becoming crucial in every industry. The company's IT security solutions deliver visibility and insight that help to make informed decisions, create credible action plans, and monitor progress. The project we have been given will load test their products.

### 1.2. Objectives

Rapid7 wants us to develop a load testing framework that runs on Kubernetes. Their goal is to use this framework to test their software that searches documents and files with terabytes of data. The testing framework should be able to time their responses and record CPU and memory utilization. When we gather this data, we aim to graph this data in an informative and easy-to-read manner.

This project will help the client quickly develop code and find the flaws immediately during the testing phase. Before it can be deployed the software will be tested rigorously to make sure it is as efficient as it can be.

### 1.3. Goals

This project will be deemed a success if we meet all the following goals:

- Build a load testing framework that can take any executable.
- Take a configuration file with a list of HTTP API calls.
- Package them up to docker images and run them in an isolated Kubernetes server.
- Time and record performance metrics and graph the results in an easy-to-read manner.

## 2. Project Scope

### 2.1. Project Deliverables

- Requirements Document: covers a variety of aspects of the project such as an introduction to the topic, the current system, and the proposed system with the purpose of better understanding the project and the client's need.

- Software Design Specification: similar to the requirements document but more design-oriented, as it will explain the system design model using diagrams and also how it is implemented, what tools and languages are used, etc.

- Project Plan: accompanies the requirement document as a planning document for the project; however, the project plan is more focused on team management as it presents the project organization and controls, risk analysis, communication, and team roles.

- Development Plan: details the entire development process including implementation and design and evaluates the process as a whole with suggestions for improvements.

- Management Report: a final evaluation of the project planning process, including management, conclusion, and improvements.

- In terms of code our deliverables are the load testing framework, and the CI/CD infrastructure that will automatically package the test runner and project under test and run it in a Kubernetes server.

## 2.2. Project Boundaries

In scope:

- Allow the developer to load test their software with a list of HTTP API requests.
- Package the executable and the test runner up to docker containers and test it in a Kubernetes server.
- Record the results in a file. Results vary from performance times and computer utilization.
- Graph the results to compare the other software versions.

Out of scope:

- Autoscaling the Kubernetes server.
- A UI so the user can easily input configuration files themselves.

## 2.3. Product Backlog

### Requirements:

- Allow the developer to load test their software with a list of HTTP API requests.
- Package the executable and the test runner up to docker containers and test it in a Kubernetes server.
- Record the results in a file. Results vary from performance times and computer utilization.

### New features:

- Graph the results to compare the other software versions.
- Autoscaling Kubernetes. Create more instances of the project under test.
- CI/CD implemented to automatically package and build the code in a dockers image and run in a Kubernetes server.

### Changes to existing features:

- Using Jenkins on AWS to automatically build our code every time it is pushed to the GitHub repository.
- Changes to our mock project under test. Now has a more efficient search algorithm.

### Bug Fixes:

- Jenkins was not set up correctly. Changes to that so GitHub was connected, and the pipeline was working in its intended way. Been resolved
- We found a bug in our search algorithm and would not count the number of searches correctly every time. Been resolved.

### Infrastructure Changes:

- Jenkins (CI/CD) was set up locally. Now has been set up in an EC2 with AWS Code Deploy.
- MiniKube will be used for Kubernetes local testing, but we will migrate to AWS Elastic Kubernetes Service.

## 3. Project Approach

When we were assigned this project, we knew we had to take a very open-minded approach when developing this project. All concepts and technologies recommended were not familiar to any of us whatsoever. For the first few weeks, we had to use our time in a different way than we thought. We thought we had enough experience and a strong collection of knowledge of technologies to hit the ground running on any project given to us. However, we had to spend the first few weeks learning about new concepts of programming and industry ways of development. We knew this would help us in the long run. None of us had a strong knowledge of continuous integration and continuous development (Jenkins). The same with technologies such as Docker and Kubernetes. These 3 technologies are pivotal components of our project.

When the development stages began, we decided to split up into 3 teams.

1. Project Under Test
2. Test Runner and Configuration
3. Docker and Kubernetes

As 3 smaller teams, we had to keep in constant communication with each other when developing the 3 main components. We had to be certain that what we were developing would work the way it should when it all comes together. The test runner must have a format that can take a list of HTTP requests and easily run them against the project under test. It should be developed in a certain way that it will do it for different types of software with their own custom test configuration file.

The Docker and Kubernetes setup must be configured for different software. The build scripts must be focused on the software that is being tested. So, this will have to be changed. Using Kubernetes observability, we can capture the data in a file and graph it using React to compare performance results with previous software versions.

### 3.1. Scrum Sprints

**Sprint 1** – 31/1/22 - 11/2/22:

During the first sprint, we scheduled meetings with our client to get a better understanding of the project. After numerous of our questions were answered we began to split the project into the 3 teams mentioned above. We also had discussions on the best and appropriate technologies to use for this project. After splitting the team up accordingly, we took time to learn about the technologies we were going to use.

**Sprint 2** – 14/2/22 - 25/2/21:

Our objective of this sprint was to have the skeleton of the project planned and developed. Every week, we have numerous meetings to catch up on progress and retro for people to recommend things we should add or do differently if we have time. During this sprint, we just had a small amount of learning backlog which was quickly handled with.

**Sprint 3** - 28/2/22 - 11/3/22:

Our objective for this sprint was to have the 3 components working in their own separate ways. But developed in a way that integrating the 3 parts would cause little to no hassle. We ran into a few issues with the CI/CD integration due to lack of experience with the Jenkins platform but after talks with the team, we were easily able to handle it. During retro, we thought ahead about the technologies we could use to integrate the graphical interface.

**Sprint 4** - 14/3/22 - 25/3/22

Our plan for this sprint is to get all the main components up and running and get the project working the way it should be. This is without the graphical interface. As of the time of writing, we have not dealt with the backlog and retro has not happened.

**Sprint 5** - 28/3/22 – 8/4/22:

Our plan for this final is sprint is to get all documentation finished and complete the graphical interface so our client can view the performance data. As of the time of writing, we have not dealt with the backlog and retro has not happened.

## 4. Project Organisation

### 4.1. Staff

Provide a list of the team members and their prior experience in projects and prior technical skills

| | Prior Experience |
|---|---|
| Aaron Byrne | Previous experience in NodeJS technologies such as ExpressJS, ReactJS and NextJS from college projects and individual projects. Also, have experience in Java and C++ from college work. Some knowledge in MySQL from SWENG Project last year. |
| Tomasz Bogun | Experience with Python, Swift, Dart, and Java while working on individual projects. While working on college projects, Tomasz has used React, Typescript, and Java. |
| Vitali Borsak | Experience with Java on individual projects and has used ASP.NET, C#, JavaFX, and Python as part of college projects. |
| Miguel Arrieta | Experience with Scala, Python, and Java while working on both college projects and individual projects. |
| Holly Daly | Experience in web development using HTML, CSS, and Python. Worked on projects using Java and python in College. |

| | |
|---|---|
| Declan Quinn | Experience with HTML, and Java while working on college projects. While working on individual projects, Declan has used Python and has developed C applications. |
| Abdelaziz Abushark | Aziz has used Java and processing to develop college projects and has used Python to develop his own personal applications and programs. |

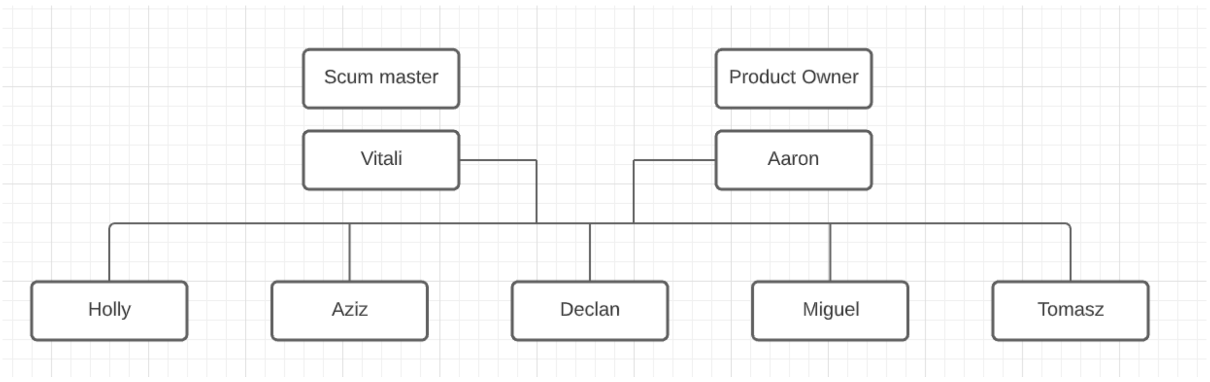
### 4.2. Staff Chart

Provide a scrum staff charts outlining the following people for each scrum sprint in the project: Product owner; Scrum master; team members.

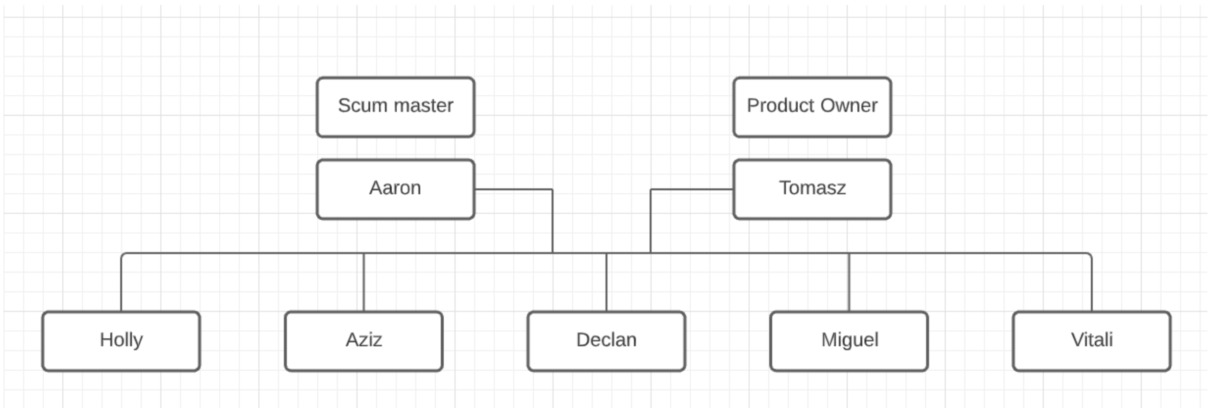
For each sprint staff chart outline what each member of the team will be working on.

For each scrum sprint the roles of Product Owner, Scrum master, and team members were alternated as follows:

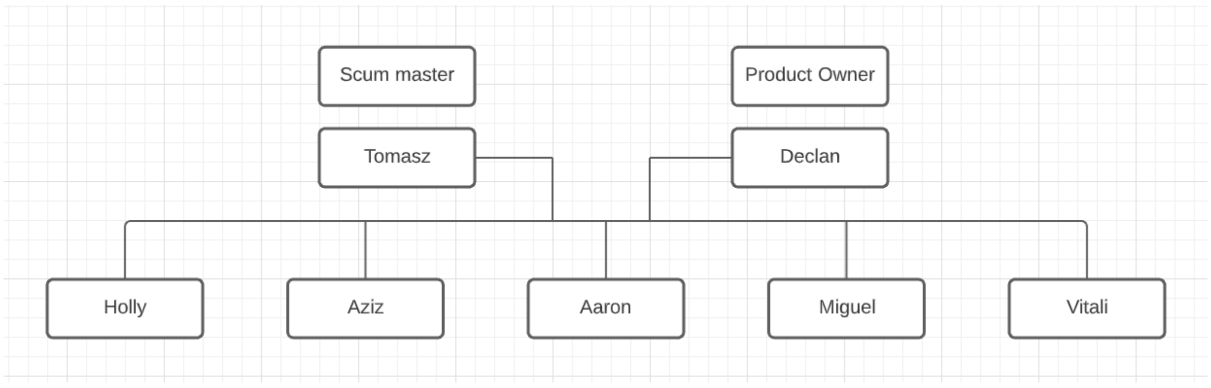
**Sprint 1:**



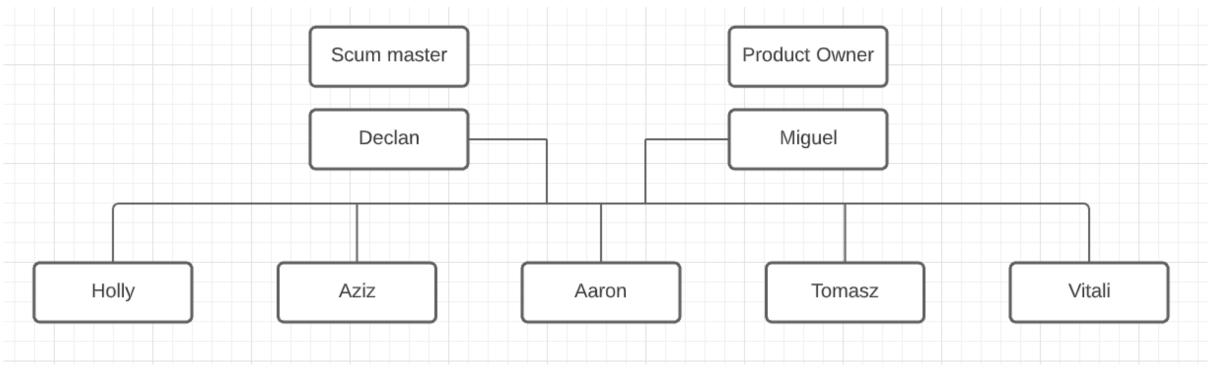
## Sprint 2:



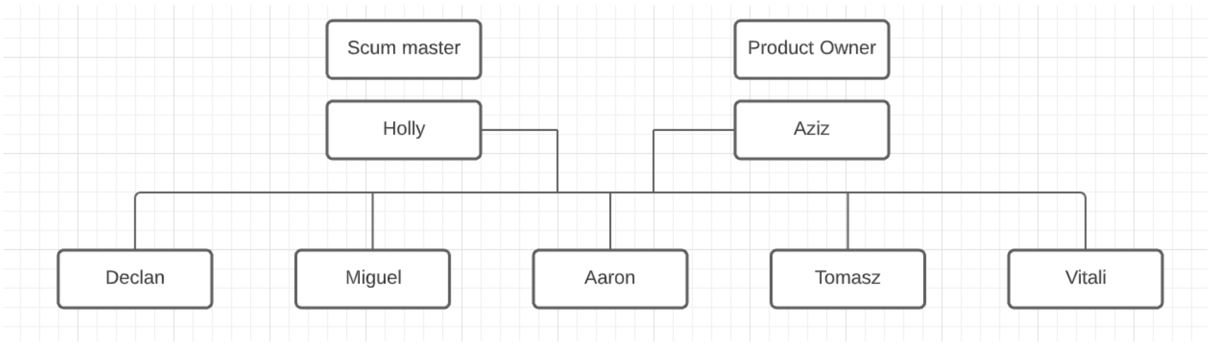
## Sprint 3:



## Sprint 4:



## Sprint 5:



The following table shows what we worked on through the sprints and what we will work on.

| | **Sprint 1** | **Sprint 2** | **Sprint 3** | **Sprint 4** | **Sprint 5** |
|---|---|---|---|---|---|
| **Aaron** | (Product Owner) Started learning about Docker and Kubernetes. Worked on Requirements Documentation. | (Scrum Master) Researched CI/CD technologies to automate the code management and testing cycle. | Setup the CI/CD with AWS and Jenkins. Build and tests the code when it's pushed. | Create YAML files to automate the packaging to docker process and run-on Kubernetes process | Work on Kubernetes autoscaling and fix any outstanding bugs. |
| **Tomasz** | Began learning about technologies for the project under test. Worked on Requirements Documentation. | (Product Owner) Developed the project under test. | (Scrum Master) Optimize the project under test and work on the project plan. | Begin to work on the graphical UI interface to display the data. | Finish the graphical UI interface and fix any final bugs. |
| **Vitali** | (Scrum master) Began looking at suitable technologies for the test runner. E.g Dropwizard and Springboot. Worked on Requirements Documentation. | Began the development cycle of the test runner. | Further development on the test runner and configuration and worked on the project plan. | Work on final documentation and outstanding bugs. | Final review of documentation and code bundle. |

| | | | | | |
|---|---|---|---|---|---|
| **Miguel** | Began looking at suitable technologies for the test runner. E.g Dropwizard and Springboot. Worked on Requirements Documentation. | Began the development cycle of the test runner. | Further development on the test runner and configuration and worked on the specification document. | (Product Owner) Create tests to automate the test process for the test runner. | Fix any final bugs and help with graphical UI and documentation. |
| **Aziz** | Started learning about Docker and Kubernetes. Worked on Requirements Documentation | Worked on testing other sample applications within Docker and Kubernetes. | Setup the CI/CD with AWS and Jenkins. Build and tests the code when it's pushed. | Create YAML files to automate the packaging to docker process and running on Kubernetes. | (Product Owner) Work on Kubernetes autoscaling and fix any outstanding bugs. |
| **Declan** | Began learning about technologies for the project under test. Worked on Requirements Documentation. | Developed the project under test. | (Product Owner) Optimize the project under test and work on the specification document. | (Scrum Master) Begin to work on the graphical UI interface to display the data. | Finish the graphical UI interface and fix any final bugs. |
| **Holly** | Began looking at suitable technologies for the test runner. E.g Dropwizard and Springboot. Worked on Requirements Documentation. | Looked into formats we can use for the configuration file. | Further development on the test runner and configuration and worked on the specification document. | Create tests to automate the test process for the test runner. | (Scrum Master) Fix any final bugs and help with graphical UI and documentation. |

## 5. Risk Analysis

### 5.1. Risk Analysis

Identify the risks to the successful conclusion of the project and calculate their Risk Factor

| Risk Element | Impact (1 to 5) | Likelihood (1 to 5) | Risk Factor (I*L) |
|---|---|---|---|
| The test-runner reports false test results. | 5 | 1 | 5 |
| The project under test runs into an error. | 5 | 2 | 10 |
| The test-runner records test results incorrectly. | 5 | 1 | 5 |
| The test-runner runs into an error. | 5 | 1 | 5 |
| Jenkins (CI/CD) runs into an error | 5 | 1 | 5 |

### 5.2. Risk Mitigation

Present the risks (ordered by their risk factor) and identify actions to reduce them

| Risk | Measures to Reduce Risk |
|---|---|
| The project under test runs into an error. | This can be reduced primarily by the client themselves. The client can make sure that their software works without any errors before using it on the test-runner. We can also have a failsafe where if the project under test runs into an issue, we terminate the program and report the issue to the client. |
| The test-runner reports false test results. | We can give the option to the client to re-run the test or we can re-run the test ourselves multiple times to make sure that the results match up, otherwise take the worst-case/ average case of the results. |
| The test-runner records test results incorrectly. | We can output the test results to the client, record the results, and then check if they match up with the results outputted to the client, if they don't then re-write the results to the ones originally outputted to the client. |
| The test-runner runs into an error. | If this happens, we can try to re-run the tests before reporting an issue back to the client. |
| Jenkins (CI/CD) runs into an error. | The automated build that Jenkins performs should previously test the software before letting it proceed through the pipeline. Managing the infrastructure will stop it from failing. |

## 6. Project Controls

**Project execution -** We control our project execution using various IDEs (Integrated Development Environment) and tools. We tested the execution of the project under test which is a http server by running it in a python IDE called Pycharm and then performing GET and POST requests using software called Postman. We then observed the values that the server returned and made sure that they match what is expected from the specification document. Whenever unexpected behaviour occurred, it was an indication that there was a bug in the project under test.

**Progress and quality -** We control the progress and quality of our project by reviewing each other's work and giving feedback and suggesting recommendations. This way, bugs can be eliminated, and improvements can be made that would not normally be spotted by a single author. We also reach out to other team members whenever we have difficulty with a piece of work. This is very helpful because other team members might have more experience in a specific topic, and it reduces the overall completion time.

**Deliverables -** We controlled the deliveries using delegation and proactiveness. Whenever we had a delivery to complete, we would try to meet up as soon as possible to discuss it and plan it out. This is because if a difficulty occurs in completing the deliverable, there is enough time to sort everything out. During those meetings we split up the work of the deliverable into many smaller tasks that can be completed independently. Then each task was assigned to the person who felt most comfortable completing it. We found this quite effective because each task was completed by the person that could do it best, and therefore we didn't have much trouble finishing each of our tasks.

**Deadlines - We** controlled the deadlines primarily by planning our work and using schedules. Each of us had individual timetables where we made sure to allocate enough time for completing each deliverable for their assigned deadlines. In addition to this, we also sometimes made intermediary guidance deadlines for large pieces of work, for each big chunk of a single deliverable. It was also important to start the work as early as possible in order to have as much time as possible to complete the work before the deadline.

**Communication** - We control our communication using discord. We have a channel set up where we can either type or go on voice and video call whenever it is convenient. We also use it to share documents, feedback, and ideas. We all stay up to date on all of the communication because whenever there is a new message, we immediately get a notification about it to our devices. We control the communication with our clients using zoom and email. Whenever we have a quick question, we send a short email and use zoom for longer communication. We schedule weekly meetings on email and then all meet on zoom. There we discuss our progress, give demonstrations, and ask questions.

## 7. Communication

### 7.1. Client Communication

We meet with the client every Friday at 12am. We use zoom for this because our client has it and it is easy to use. We chose Friday for the meetings because it suits everyone's different timetables, and because the end of the week is a suitable time to discuss the progress made in the week and the plans for next week.

### 7.2. Project Team Meetings

We usually meet on a Friday around 1 pm. We sometimes meet on a Thursday when someone is unavailable for a meeting on Friday. We use our discord channel for this. We do this straight after the client meeting because it is more convenient to schedule the two meetings together rather than scheduling them separately.

## 8. Appendixes

Requirements and specification documents.

# “Generic Kubernetes based load testing framework”

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

# 1. Introduction

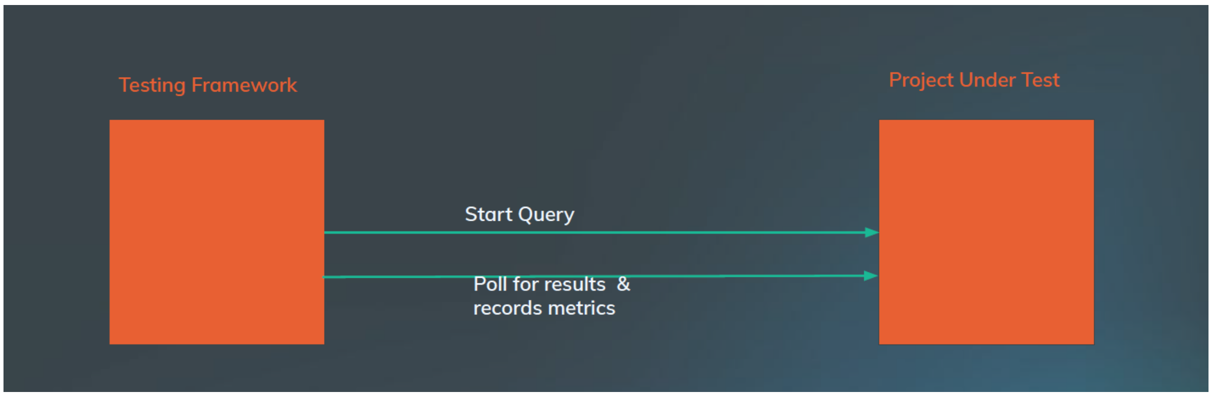
## 1.1. Overview - Purpose of the System

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 analyze 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 initializes 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.

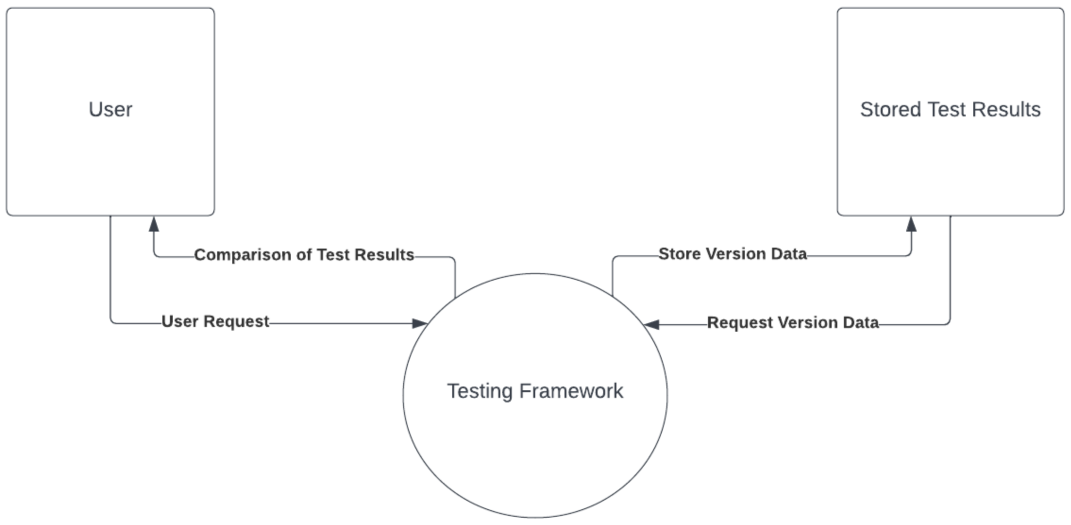
The main objective of the proposed system is to analyze the performance of a piece of software. In our client's case, this software is a search engine that searches vast amounts of data for specific information they need. The data that is being searched can reach terabytes in size, so it is crucial for 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

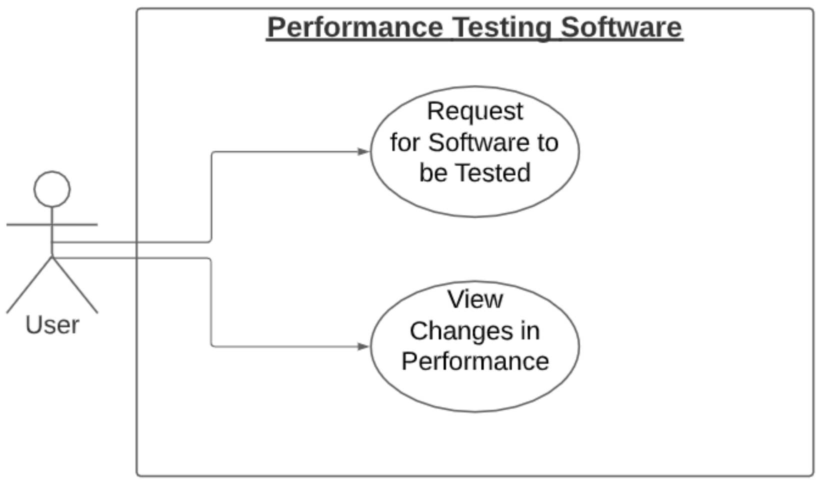
## 2.2 System Design Models

### 2.2.1 System Context

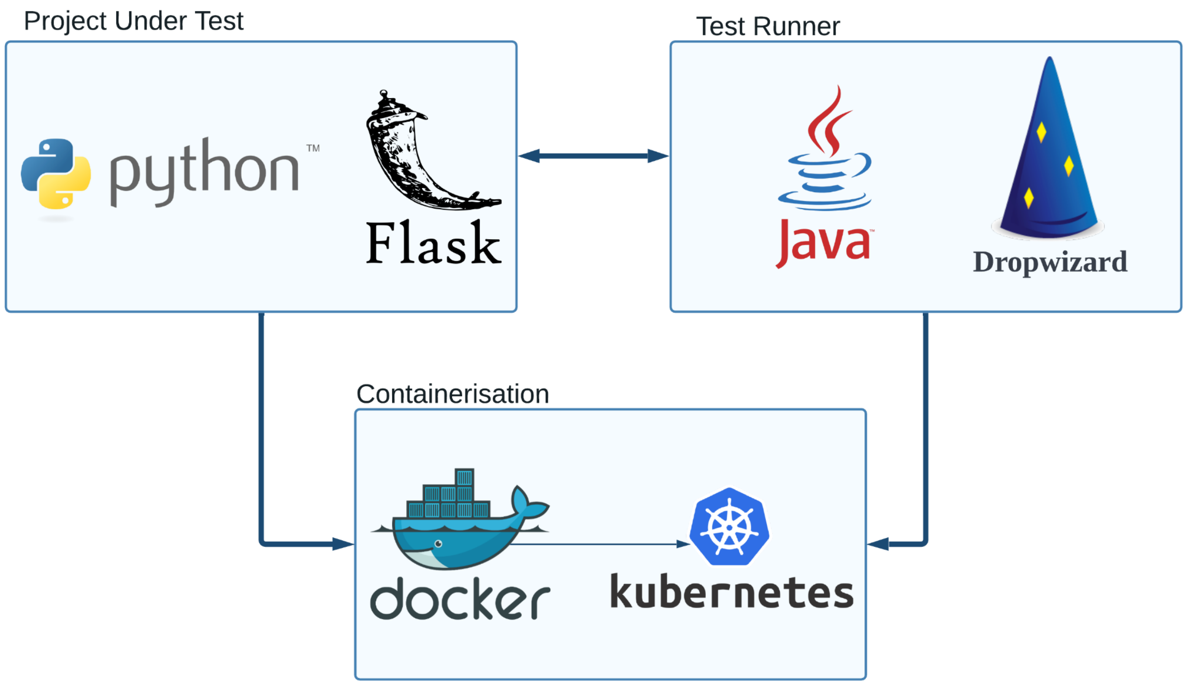
Our system allows the user to send a request to test a piece of software. This is tested and the results of the current version are saved. The user is then returned the test data along with information on how previous versions of the software performed under testing if requested.



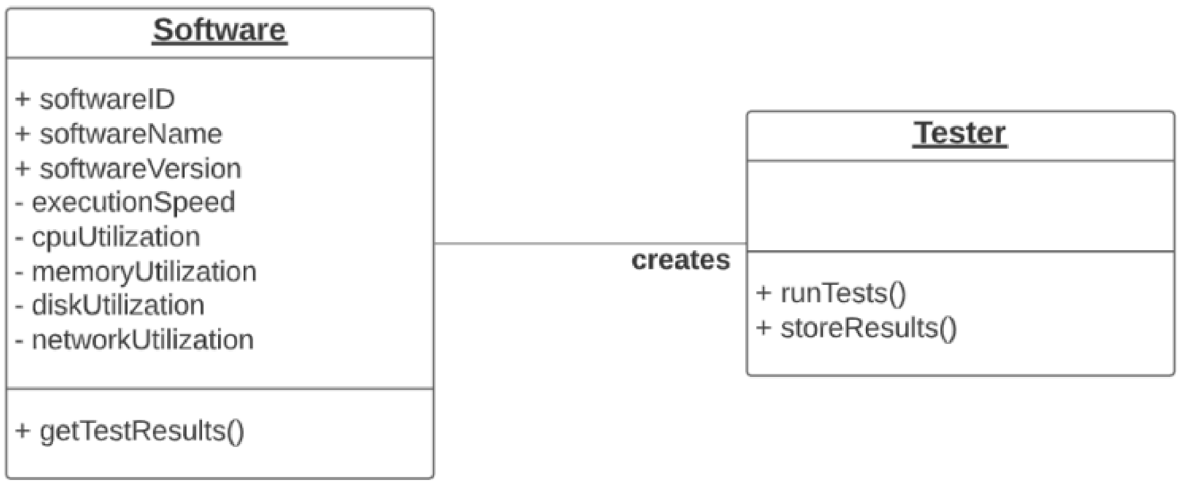
### 2.2.2 Use Cases


Performance Testing Software
Request for Software to be Tested
View Changes in Performance
User

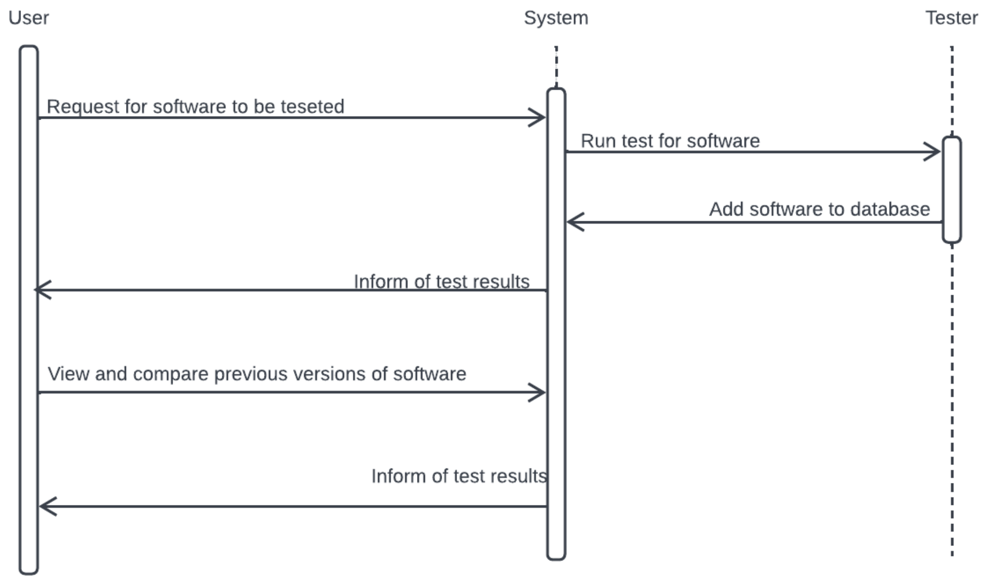
### 2.2.3 System Architecture



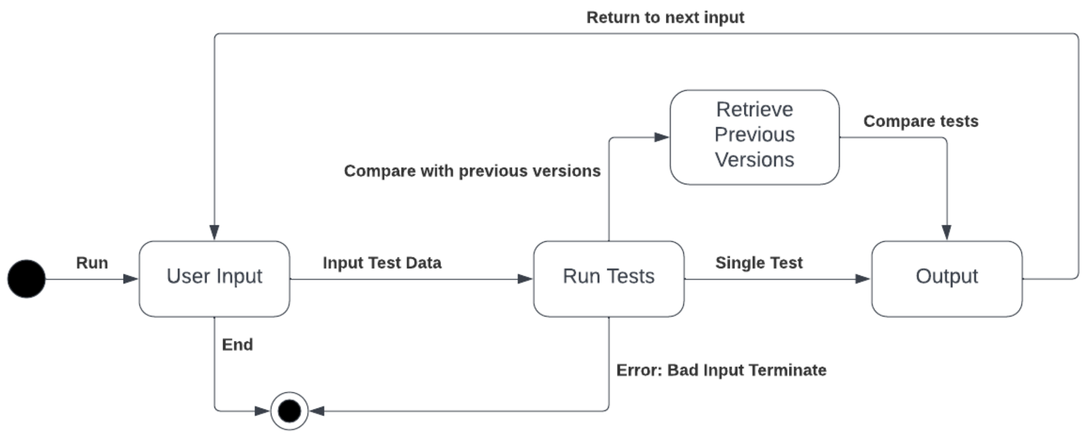
### 2.2.4 Class Diagrams



### 2.2.5 Sequence Diagrams



### 2.2.6 State Diagrams



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

# 1. Introduction

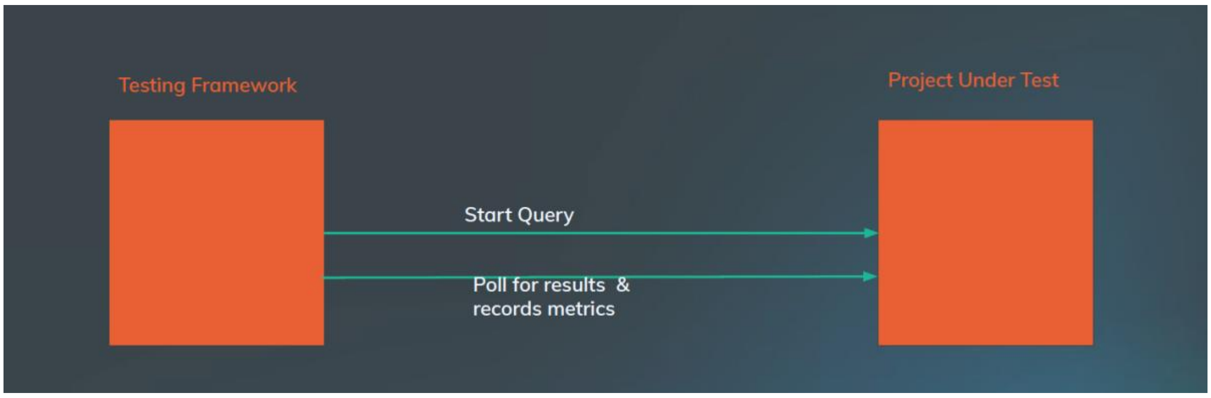
## 1.1. Overview - Purpose of system

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:**



## 1.3. 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

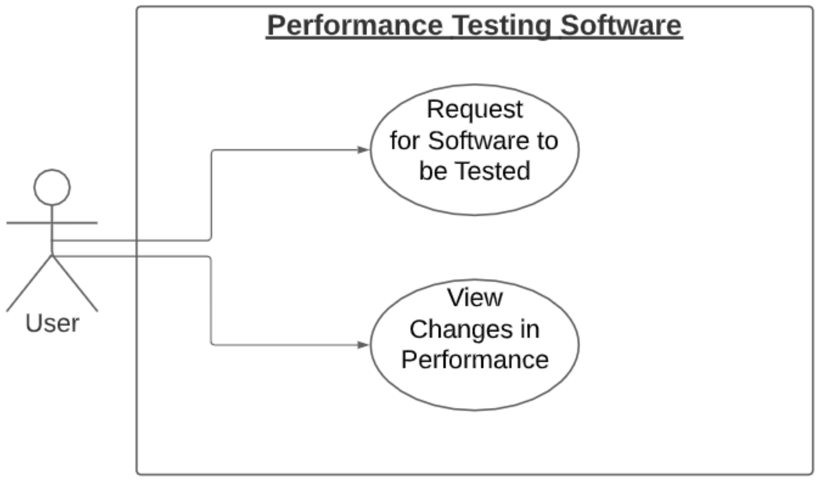
- Ease of use. The project should make it easy for the client to test the project under test within a Kubernetes cluster and obtain and view the results.
- The project should be easily scalable to fit the needs of the client's actual project they want to test.Include the possibility of automating the 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.

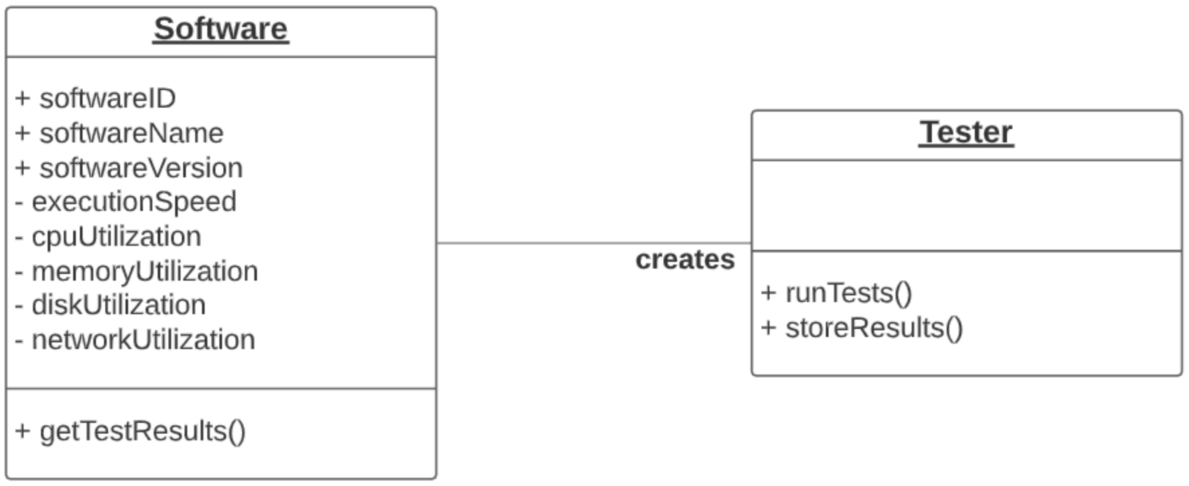
### 3.4.2. Use cases (including text narratives)


Performance Testing Software
Request for Software to be Tested
View Changes in Performance
User

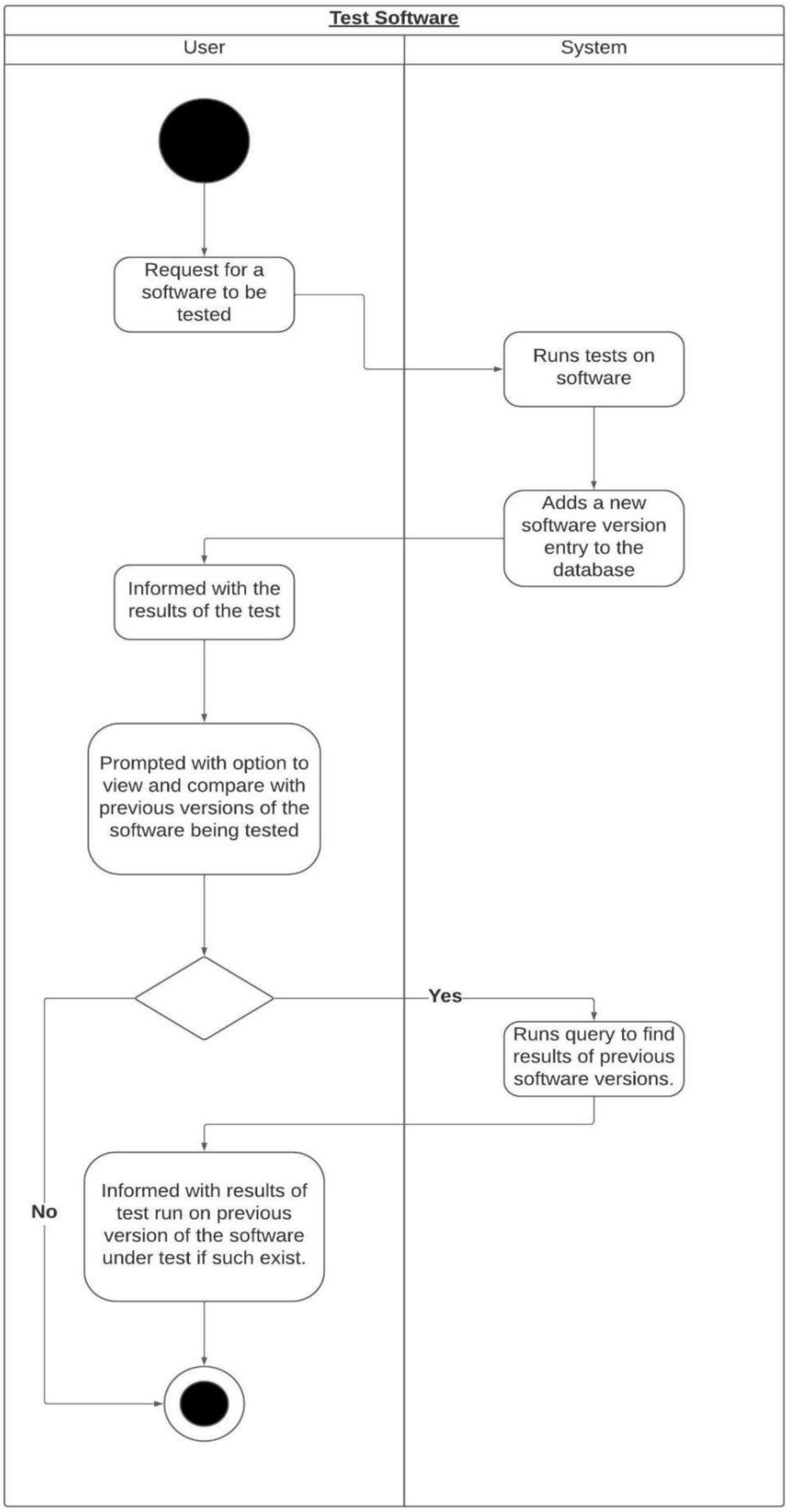
| Request Software to be Tested | User | System |
|---|---|---|
| 1.1 | Request for a software to be tested for its efficiency | The system runs test on the software with various data. |
| 1.2 | | The system returns the test results to the user and stores them. |

| 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

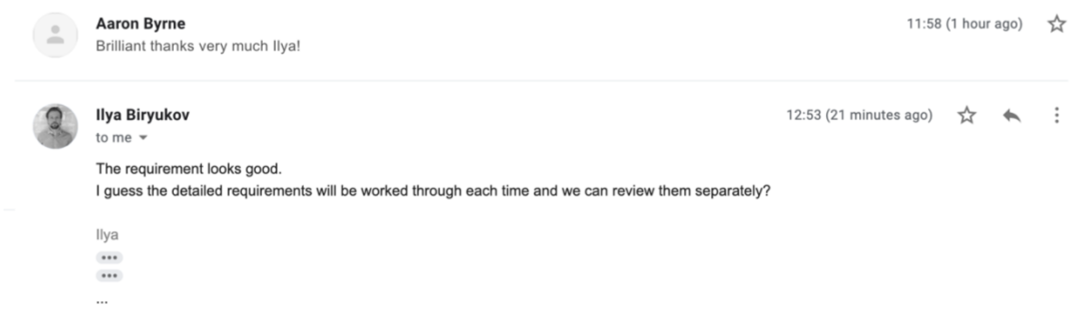


### 3.4.4. Dynamic model


Test Software
User
System
Request for a software to be tested
Runs tests on software
Adds a new software version entry to the database
Informed with the results of the test
Prompted with option to view and compare with previous versions of the software being tested
Yes
Runs query to find results of previous software versions.
No
Informed with results of test run on previous version of the software under test if such exist.

## 4. Proof of client sign off:

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



**Aaron Byrne** 11:58 (1 hour ago)
Brilliant thanks very much Ilya!

**Ilya Biryukov** 12:53 (21 minutes ago)
to me

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

Ilya

...