University of Sheffield

A tool to specify testable causal models of software behaviour

Hao-Hsuan Teng

Supervisor: Dr Neil Walkinshaw

COM3610

A report submitted in fulfilment of the requirements
for the degree of MSc in Advanced Computer Science

in the

Department of Computer Science

December 4, 2021

Declaration

Name:Hao-Hsuan Teng

Signature:Hao-Hsuan Teng

Date:

Abstract

With the inventions of supercomputer with powerful processing power, a common way to use it is to run complex computational modules that can be used to determine the different outcome of a certain event. These modules often involve large number of inputs and outputs and can be extremely time consuming to test. The goal of this project is to create a tool that can be used to test these complex computational modules, this tool needs to be easy to understand, and accessible non-software engineering users. Parts of the tool for this project has already been in development, but it still requires a way for non-software engineer users to access it. A simple user interface with the ability to display results and create new test sets has been implanted, the results shows that this system capable of assist non-software engineer users access this system without too much trouble.

Contents

List of Figures

Chapter 1

Introduction

Throughout the years, companies have developed CPU with more and more powerful processing power, and these more powerful CPU comes the invention of supercomputer, these supercomputers have stronger computational power compared to normal computer. With this computational power, people start to develop large and complex computational models. Computational model is a way to try and use computer system to simulate real life situation, these models use large amounts of parameters to determine the outcomes of a certain event, researchers can utilize these models to simulate experiments and real-life event [1] and aid their research with the results.

But these models aren't without problems, one problem is these models has a large number of inputs and outputs, and it takes a lot of times to process them, this means in order to test them, it requires a lot of time to test all inputs and outputs' accuracy with traditional method.

The solution provided by this project is to use the causal models, by determine the relationship between different input and output parameters and use this information to create graphs that focus on certain behaviours. Then just simply compare the difference between the test sets and the output of the tested models, people can know how accurate a certain interaction is between input and output parameters, and by doing so saved a lot of time compared to traditional method where the best ways to test it is to run the model repeatedly with new data [2]. Then with these result, researchers or people who tests the model can decide whether or not this system meets their expectation.

1.1 Aims and Objectives

Despite the existence of the theory, there isn't a software where the user can store these test sets and used these sets to test other tools. So, primary goal of this project is to provide a system that can be used to store the test sets, and when needed, people can specify the scenarios they want and use the stored result to test other computational models.

This system will be primary based on a tool called Causcumber, it is a tool to test computational model and is mainly based on a model called Covasim, Covasim is a computational model focus on COVID-19 analyses, it can determine the infection rate and how will the rate change based on different interventions (such as quarantine, social distance, etc.) [3]. Causcumber is a in developing tool and is aimed to provide a testing method for computational model like Covasim, it uses Cucumber specification, another tool reads executable specification in plain text and validates if the software does what those specifications say, to produce casual model graph.

The main problem with Causcumber is that it lacks an accessible user interface, so for users who haven't really studied in software, it is pretty much impossible to understand. So, the goal is to eventually provide a user interface that can be accessible for all level of users.

Another goal of this system is that it needs to be easy to read, and to achieve this, we use the gherkin reference to help with it, by using a set of special keywords with a certain structure, it can give meaning to executable specifications. For the users who are not specialize in software engineering, this is a way to make the system easy to understand and can avoid a lot of confusion.

1.2 Overview of the Report

In the next chapter, Literature Survey, a list of literature will be providing information and reasons why there's a need for this type of testing tool to exist. It will start with explain what scientific software is, what do they do and why they aren't usually tested in a proper method. Then this will fellow up with introductions to Cucumber and behave, two of the main factors of the Causcumber system.

In the third chapter, Requirements and Analysis, will be mainly talking about the requirements and objectives of the system. And the design choice made based of these requirements.

Part four, Progress, will be discussing the current state of the system, how it is received and what feedback does it get.

In the fifth chapter, conclusions and project plan, will summarize what have been achieve till this date, and the overall plan for the upcoming months.

Chapter 2

Literature Survey

This section will be explained why there's a need for this system by explore background literature relate to this subject. Mainly the need of a proper system to test scientific software, start with explaining what a scientific software is, following up by explain why most of are not tested correctly. Then we will be discussing the two main component of this system, cucumber and behave. Then finally, explaining what casual testing is, the main method used by this system to test other software, and exhibit why it lacks proper interactive system for none software engineering users.

2.1 Scientific software

Using computational model for scientific research is common practice with today's technology, scientist use these models to predict the result of an experiment or the outcome of event. But whether these models can accurately predict the result, is still a problem since most of them lacks some form of proper testing, below will be discussing what is a computational model and why are they poorly tested.

2.1.1 Computational model

What defines a scientific computational modeling, it is by using computers to simulate and study real life event by using mathematics, statistics, physics, and computer science to study the mechanism and behavior of complex systems by computer simulation [4].

By using models that contains a number of input variables, and algorithm that will define the system. These models will simulate real-life situation and researchers can adjust these models by changing their variable and algorithm according to the results to make the model more conform to reality. Then researchers can use these models to see how one or more variables can affect the outcome of a certain event. Computational models provided some level of prediction of being able to calculate an anticipated result from a given set of variables [5]. This forecasting system can be used to predict complex systems, such as weather or disease spread, and help researchers or decision makers to decide their next move.

An example of this kind of model is Covasim [6], Covasim uses agents to simulate individual people, the model mainly focused on one type of calculation, what is the probability of an individual in a given time step will change from not infected to infected, or badly ill to death.

The simulation starts from loaded the parameters, then it will start creating individual with different age, sex, and comorbidities based on the selected location (i.e., Different country). Then will be group into a social network depend on their attribute. After that the model will start looping, in each step, the model will apply various operations on the induvial, then collect the result and apply analysis.

2.1.2 Why computational models are hard to test

Computational models are often very complicate, and require special knowledge related to the field. These models are often developed by scientist themselves, but most scientist aren't software developer and may not have knowledge in some common software engineering practices, this may cost the quality of the scientific software [7]. Software testing is one of the aspects that is impacted, since the lack of knowledge in software development, lack of understanding in systematic testing is expected. Mistake can be made without notice and may affect the output of the system, causing the result to become inaccurate.

Testing a scientific software itself can be a difficult task to do, this may be the result of two types of challenges:

First is due to the software's main purpose is to predict something unknown or simulate areas even researcher have little knowledge in, but to test a software, it is crucial to know how the software should behave, what kind of output should it show. Without these knowledge, it is hard to tell if the software is working the right way.

Furthermore, these computation models are often consisting of hundreds of parameters and complex algorithm, this means it would require lots of test case in order to test every aspect of the system, and this leads high execution time making the testing process become very time consuming [8].

Second is that scientific software is mostly develop with scientist being the lead role instead of a software engineer, and the value of the software system is often underestimate [9]. And most scientist never gone through the training in software design like software engineer [10], this means limited understanding of testing process and not applying known testing methods. This will make core design of the software being unfriendly for effective testing.

2.2 Current testing method

Despite the difficulty in testing these models, there are still some common software testing methods that can be used to test them, such as behavior-driven development (or BDD) and cucumber. Combine with casual model testing, a way to draw relationship between variables, it will make testing these complex model a lot easier.

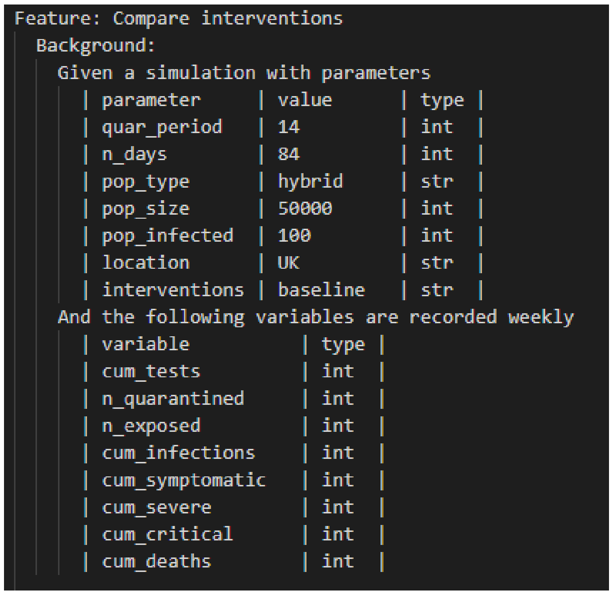
2.2.1 Behavior-driven development with Behave

Behave is a python API based on behavior-driven development, behavior-driven development is a development technique that encourages collaboration between participants in a software project [11]. Behavior-driven development or BDD, aims to having a clear understand of the desired software behavior between stakeholders and software developers, communicate by writing test cases in a nature language that both sides can understand. Behave is a Python API created for this, it consists of tests written in natural language style, or in other words Cucumber.

When test case is written, it requires a tool to read its specifications and see if the system works as the test-case specify. Cucumber is the tool that developed for this purpose, it read specifications written in plain text with a certain format and validate if the software does what those specifications say [12] and create a report.

For Cucumber to understand the specifications, it must be written in Gherkin language, Gherkin uses a set of grammar and special keywords to give plain text structure so that Cucumber could understand it, one of the pros for Gherkin is that keyword can be translated to other languages, making it usable for people who don't speak English [13].

A proper Gherkin grammar starts by giving a context, then describe an event, and after that what should happen after the event. With these grammars, tester can specify a situation in the system, describe a certain parameter, and what the results should be with those parameters. By writing a test using Cucumber and Gherkin grammar, tester can combine them into a .Feature file as display below:



```
Feature: Compare interventions
  Background:
    Given a simulation with parameters
      | parameter     | value     | type |
      | quar_period   | 14        | int  |
      | n_days        | 84        | int  |
      | pop_type      | hybrid    | str  |
      | pop_size      | 50000     | int  |
      | pop_infected  | 100       | int  |
      | location      | UK        | str  |
      | interventions | baseline  | str  |
    And the following variables are recorded weekly
      | variable          | type |
      | cum_tests         | int  |
      | n_quarantined     | int  |
      | n_exposed         | int  |
      | cum_infections    | int  |
      | cum_symptomatic   | int  |
      | cum_severe        | int  |
      | cum_critical      | int  |
      | cum_deaths        | int  |
```

Figure 1. A feature file example

With this .Feature file, tester can execute the file in terminal and Behave can read what needs to be tested and return a report.

2.2.2 Causal testing

With the help of behave and cucumber, we have the ability to test computational models, it is possible to test the entire system by writing an enormous feature file, but the execution time will become really long, and since computational models are usually in big scale it is also easy to make mistake. This is when causal model can help simplify the testing process.

Causal model is a way to represent causal relationship within a system through mathematical model. It helps tester monitor causal relationship in the data [14]. A causal module can predict the behavior of a system, by comparing different inputs and the outputs a system produces, it can explore the cause and influence of a certain relationship [15]. By understand these relationships, tester can test parts of the system by see if the inputs and outputs follow the same behaviour.

With this method tester can reduce the time required to test a complicate system by only test parts of it one at a time, then combine the results, and testers can have a full picture of how the system perform. This is unlike the tradition method, where it requires to test the entire system all at once, and may require lots of time if it is a complicate system.

2.2.3 Testing computational model with causal testing

Causal testing can be a really useful testing method for software engineer [16], however computational models are often develop by scientist or researchers who have limited knowledge in software testing [8]. Tester who wants to use causal model as a way to test system needs to have prior knowledge in how variables work in a software, this mean people who may not have that much knowledge in software engineering, for example researchers in other field, might not be able to conduct this method of testing in their software.

2.3 Summary

Behave is a powerful testing tool for software tester's, based on the behavior-driven development method, encourage people who are not specialize in software engineering to take part in testing. Assist by cucumber, a tool where user can describe testing steps in an easy-to-understand manner, and cucumber compile and produce the results. With causal model, tester can effectively test large and complex computational models, saving time by only test parts of the system one at a time and combine the result together. But the problem is that when using causal model testing, its is inaccessible for most people, due to its lack of user-friendly function. In order to make it more user friendly, we can integrate cucumber and behave with it to make it where user can use causal model testing by using behave and cucumber specify the testing process.

Chapter 3

Analysis

Computational models are often too large and complex for traditional method to test, this cause developers of those models unwilling to test their models thorough. Furthermore, computational model developer are often scientist or researchers that don't have extent knowledge in software engineering or software testing. Due to the different training receive by scientist compared to software developer, the importance of software testing is often overlooked.

Causal model testing can be a powerful tool to test computational model, but causal model testing requires software testers to integrate the causal model with software model that is being tested, consider the complexity of the computational model being tested can be, it is hard to integrate the test for people who don't have software engineering background.

It is possible to integrate behave and cucumber to make test computational model with causal model testing easier for people, by integrate Cucumber tester can use easy to understand language to specify the testing process and behave can read and execute these processes. By combining all these researchers who want to test their computational model can have an easy to access tool to test their software.

A tool with this function has already been in development for a while called Causcumber, a modify version of cucumber for testing computational model, it has the ability to use cucumber specification and behave combine with causal module testing to test computational model, but currently it still lacks an easy to use method and can only be execute in terminal with specific command.

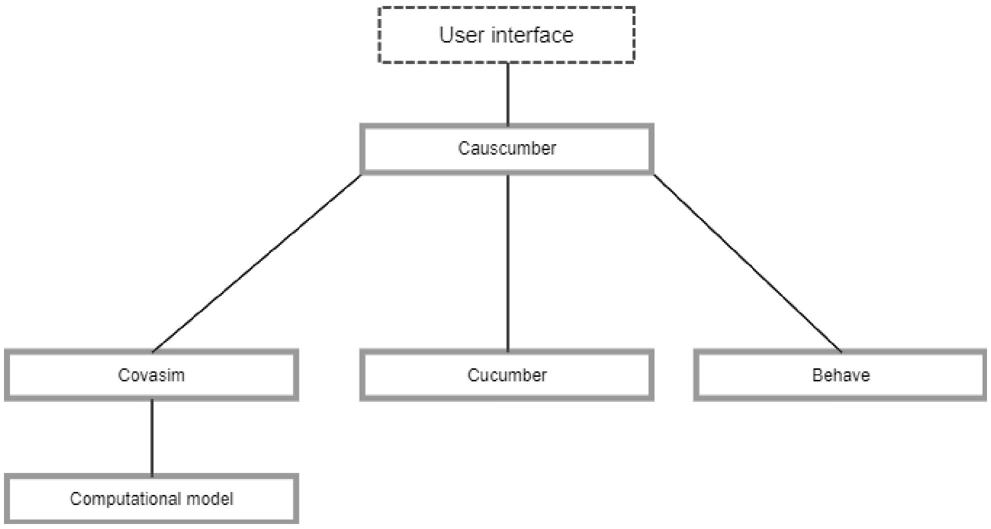


Figure 2. A diagram analysis for the system

3.1 Project Requirements

The main goal of this project is to develop an easy-to-use system where people can use it the execute the feature file they have written, the system will exam the feature file and go through the computational model to check if the model perform as the test specify. As a result, the system should produce a coherent result detailing the accuracy of the tested model. Thus, the systems need to accomplish the following:

R1. As a user, I want this system to be easy to understand, when I use the system, I want to immediate know what I need to do to get the result I want.

R2. As a user, I want the system to go through the feature file I provide and test the computational model with it.

R3. As a user, I want the system to produce a result where I can know the accuracy of the computational model.

Since Causcumber is already in developing for a while, what this project aims to accomplish is adding more to this tool. Currently the state of Causcumber is capable of execute some feature files that use to test a computational model named Covasim. And it is capable of returning lot of useful information, but the information still requires some organization to make the result easier to read. And currently there isn't a way to execute the testing system without using the terminal to execute the command, so there's a need of a user interface for people to operate the system. Last is that currently the only way to create a feature file is by hand type the files, and this might cause some error or confusion. There's also some other requirements according to the feedback of the Causcumber team, these feedback consist of multiple function that is potent to the system and will be included during the development. Therefore, by the end of the project Causcumber should be able to accomplish the following:

1. As a user, I want the result produce by the system to be clean and easy to understand, focusing on the important part.

2. As a user, I want to execute and interact with the system through a user interface.

3. As a user, I want to have a more convenient way to create a feature file, to avoid any mistake during the creation of the feature file.

4. As a user, I want to have the ability to toggle different types of result display for different levels of the users.

5. As a user, I want to have the ability to have different input optional automatically based on different situation.

6. As a user, I want to have an option to save the feature file I created.

3.2 Analysis

One of the main themes for this project is ease of use, and therefore that's what most of the requirements are focus on. For requirement R 1 – 3, the goal is to make the testing process as simple as possible, with a simplified testing process, people will be more willing to test their computational model.

Since integrate the causal model testing can be difficult and confusing, requirements 3, 5, 6 are focus on simplify this process, by making this part reduce to only require testers to input the parameters and their value, this should help mitigate a lot of errors.

Finally, this project's main focus is on enhancing the user experience for Causcumber, this is what requirement 1, 2, 4 are for, these requirements make using Causcumber to test computational model a more convenient experience.

3.3 Ethical, Professional and Legal Issues

The execution of this project won't require any real human or animal involvement, every part of this project is simulated, so there won't be any ethical problem. All the source code used in this project are open-source and free to use, the tools used to develop the user interface are also open-source and free to use, so this mean there won't be any legal problems either.

Chapter 4

Planning

4.1 Risk Analysis

The main purpose of this system will be providing a tool for people to test computational model, researchers can use this system to test if the model is working as intended and company or others who want to use the model can use it to see the model's accuracy. The following are the stake holder of this project:

1. Developers of the Causcumber project
2. Researchers who will be using Causcumber to test their model
3. Organization that uses this tool to check their model's accuracy

One of the main goals of this model is to return a clear and correct result for the user, if the returned result is incorrect or misleading. This might affect the development of the computational models that use this tool for testing, the result might end up being inaccurate and people who uses these results for scientific research or other purpose may affect by this mistake.

4.2 Project Plan

For this project, I plan to start with understand the how Causcumber work, what kind of result does it produce. Then, following by understand how behave and cucumber function in the system, by understand how these tool works, it would benefit a lot for future plan.
The next phase will be designing the user interface for Causcumber, this phase will be focus on design the user interface and making sure it is easy to understand, and easy to operate.
After the design the is done, I plan start implement the system, combine the system with the interface, in this phase will be working in Scrum method, where the project will focus on rapidly adding or change function based on feedback, this is the current phase where the project is in right now, a list of features is either being implanting or will be implement in

the near future.
The last phase will be polishing the system, focusing on debugging, and making sure the system runs smooth.

4.3 Current progress

This project already has some progress, I already have grasp on how Causcumber work and have move into the scrum phase where a user interface being develop and adding or adjust functions according to feedback, below will be some figure showing the current progress of the system:

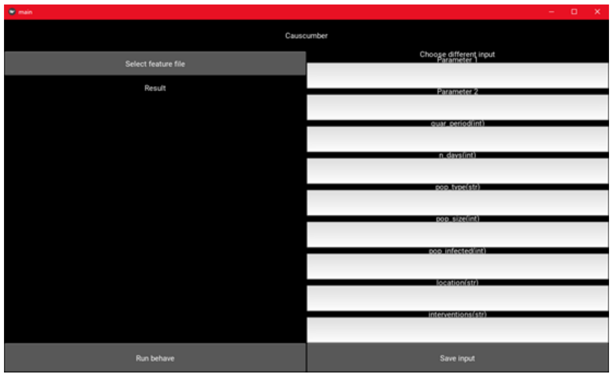
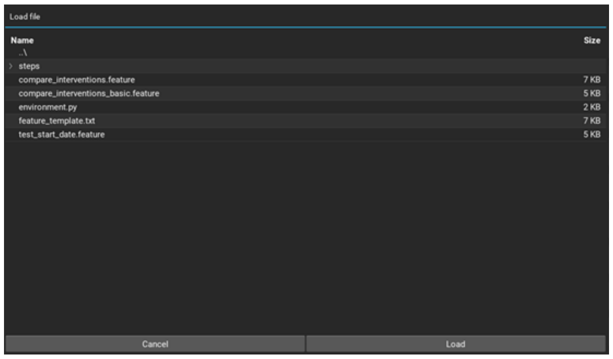


Figure 3. An overview of the user interface

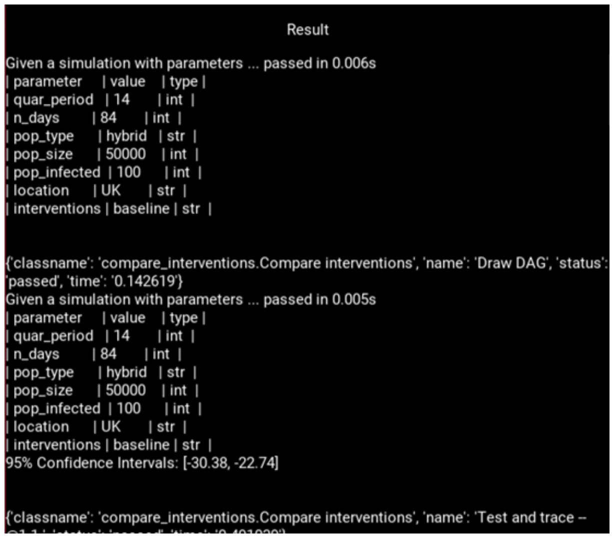
This is an overview of the user interface, the left part will display the result, and the right part is where the users can custom their feature file.



Name	Size
..\	
steps	
compare_interventions.feature	7 KB
compare_interventions_basic.feature	5 KB
environment.py	2 KB
feature_template.txt	7 KB
test_start_date.feature	5 KB

Figure 4. Select feature file screen

User can select which feature file they want to execute, then Causcumber will load and produce the result into a xml file.



```
Result
Given a simulation with parameters ... passed in 0.006s
| parameter    | value   | type |
| quar_period  | 14      | int  |
| n_days       | 84      | int  |
| pop_type     | hybrid  | str  |
| pop_size     | 50000   | int  |
| pop_infected | 100     | int  |
| location     | UK      | str  |
| interventions | baseline | str |


{'classname': 'compare_interventions.Compare interventions', 'name': 'Draw DAG', 'status':
'passed', 'time': '0.142619'}
Given a simulation with parameters ... passed in 0.005s
| parameter    | value   | type |
| quar_period  | 14      | int  |
| n_days       | 84      | int  |
| pop_type     | hybrid  | str  |
| pop_size     | 50000   | int  |
| pop_infected | 100     | int  |
| location     | UK      | str  |
| interventions | baseline | str |
95% Confidence Intervals: [-30.38, -22.74]


{'classname': 'compare_interventions.Compare interventions', 'name': 'Test and trace --
```

Figure 5. Result produced by Causcumber and display in the result section

After Causcumber load the feature file and produce to the xml file, the user interface can read from the xml file, retrieve the useful information and remove the unwanted ones, users can view the input parameters and how accurate a parameter in the computational models are.

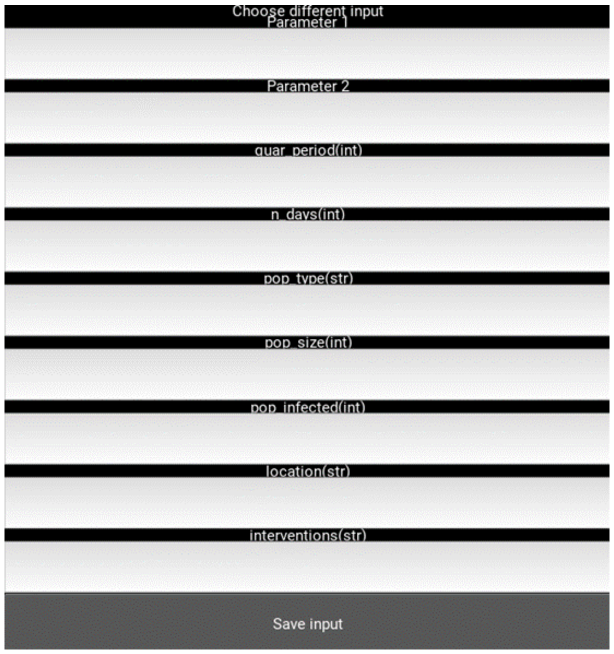


Figure 6. Choose different input value, and save it into a feature file

In the right side of the interface, user can create their own feature file with their own parameter to test the system, the system will compile the input value and save as a new feature file.

Chapter 5

Conclusions

A basic User interface has already been built, with the ability to retrieve useful information and allow users to create feature files with specific value. Users can choose which feature file they want to use to test the computational model, then the system will produce the result and the users can view it. User can also customize the feature file by choosing their own value and parameter and test the model with their own feature file. In the future, it is planned to add more ways to display the result, allow user to pick and choose what type of result display they want. Another plan is to add a more flexible feature file create system, allow people to create feature file with more variety. Below is a detail the plan of work:

Phase1: Understand how Causcumber work

Phase2: Have a good idea on how behave and cucumber function in Causcumber

Phase3: Design and implement user interface and focus on scrum to gather feedback and adjust the system according to it

Phase4: Polish the system and debug or add new function according to feedback if needed

*This schedule is subject to change due to the uncertainty in various phase of the project

Bibliography

[1] U.S. Department of Health and Human Services Computational modeling Section.1 *Nation Institute of Biomedical Imaging and Bioengineering*, 12 (May 2020), 1.

[2] Robert C Wilson, Anne GE Collins Ten simple rules for the computational modeling of behavioral data *eLife 2019;8:e49547*, 26 (November 2019), 1.

[3] Kerr CC, Stuart RM, Mistry D, Abeysuriya RG, Rosenfeld R, Hart G, Núñez RC, Cohen JA, Selvaraj P, Hagedorn B, George L, Jastrzebski M, Izzo A, Fowler G, Palmer A, Delport D, Scott N, Kelly S, Bennette C, Wagner B, Chang S, Oron AP, Wenger E, Panovska-Griffiths J, Famulare M, Klein DJ Covasim: an agent-based model of COVID-19 dynamics and interventions. *PLOS Computational Biology 17 (7): e1009149.*,(2021)

[4] U.S. Department of Health and Human Services Computational modeling Section.3 *Nation Institute of Biomedical Imaging and Bioengineering* , 12 (May 2020), 1.

[5] Muffy Calder, Claire Craig, Dave Culley, Richard de Cani, Christl A. Donnelly, Rowan Douglas, Bruce Edmonds, Jonathon Gascoigne, Nigel Gilbert, Caroline Hargrove, Derwen Hinds, David C. Lane, Dervilla Mitchell, Giles Pavey, David Robertson, Bridget Rosewell, Spencer Sherwin, Mark Walport and Alan Wilson Computational modelling for decision-making: where, why, what, who and how Chapter.3 *Royal Society Open Science*, 20 (June 2018), 3.

[6] Cliff C. Kerr ,Robyn M. Stuart ,Dina Mistry ,Romesh G. Abeysuriya,Katherine Rosenfeld,Gregory R. Hart,Rafael C. Núñez,Jamie A. Cohen,Prashanth Selvaraj,Brittany Hagedorn,Lauren George,Michał Jastrzebski,Amanda S. Izzo,Greer Fowler,Anna Palmer,Dominic Delport,Nick Scott,Sherrie L. Kelly,Caroline S. Bennette,Bradley G. Wagner,Stewart T. Chang,Assaf P. Oron,Edward A. Wenger,Jasmina Panovska-Griffiths,Michael Famulare,Daniel J. Klein Covasim: An agent-based model of COVID-19 dynamics and interventions *PLOS Computational Biology*, 26 (July 2021), 1.

[7] Upulee Kanewala, James M.Bieman Testing scientific software: A systematic literature review *ScienceDirect*, 23 (May 2014), 3.2.

[8] Upulee Kanewala, James M.Bieman Testing scientific software: A systematic literature review *ScienceDirect*, 23 (May 2014), 3.2.2.

[9] Upulee Kanewala, James M.Bieman Testing scientific software: A systematic literature review *ScienceDirect*, 23 (May 2014), 1.

[10] Judith Segal Scientists and software engineers: A tale of two cultures *Open Research Online*,(2008), 2.1

[11] Benno Rice, Richard Jones and Jens Engel Behavior Driven Development *behave*,(2017), 1.

[12] Marit van Dijk, Aslak Hellesoy, etc. Cucumber *Cucumber*,(2019), 1.

[13] Marit van Dijk, Aslak Hellesoy, etc. Gherkin Reference *Gherkin Reference*,(2019), 1.

[14] Christopher Hitchcock Causal Models *Stanford Encyclopedia of philosophy*, 7 (Augest 2018), 1.

[15] Brittany Johnson, Yuriy Brun, Alexandra Meliou Causal Testing: Finding Defects' Root Causes *Cornell Universtiy*, 19 (February 2020), 8.

[16] Paul Benjamin Lowry; James Gaskin Partial Least Squares (PLS) Structural Equation Modeling (SEM) for Building and Testing Behavioral Causal Theory: When to Choose It and How to Use It *IEEE*, 22 (April 2014), 1.