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

April 13, 2022

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name : Hao-Hsuan Teng

Signature : Hao-Hsuan Teng

Date : 12/4/2021

Abstract

With the inventions of supercomputer with powerful processing power, a common way to utilize it is to run complex computational models that can be used to analysis the behavior or outcome of a certain event. These modules often involve large number of inputs and outputs and can be extremely time consuming to test. A tool name Causcumber that can be uses to execute the test has already been in development, but it still requires some functions to make the system user-friendly. The goal of this project is to allow users who don't have extend knowledge in software testing to be able to use the Causcumber system. This tool needs to be easy to understand, and accessible for people. A user interface with the ability to display results and create new test sets have been created, this streamed line the testing process, minimize the need for users to interact with the coding part of the testing.

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 superior computational power compared to normal computers. 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 behavior or 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 large number of inputs and outputs, and it takes a lot of times to process them, this means in order to test them, it also requires a lot of time to test all inputs and outputs' accuracy with traditional method.

There is a new testing method in development name Causcumber, it determines 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, then with these result, testers can decide whether or not this system meets their expectations. By using this method tester can 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 [11]. The problem with this method is that there's currently lack of a user-friendly way for people to access Causcumber, it requires a level of knowledge in software testing to understand how to operate this tool. The solution provided by this project is to implement a user interface, this interface minimizes the need for user to interact with the programming part of the testing process. Instead of require user to interact with the code directly, it visualizes the code into a way that is easy to read for user, highlight the parts that require edit and prompt what to edit for user. With this interface, testing with Causcumber become more accessible for people who don't have extensive knowledge in software testing.

1.1 Aims and Objectives

This project will be primary based on Causcumber, a tool for testing 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.) [7]. 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.

Since Causcumber is a new system that is still in development, there isn't a convenient way for user to interact with the system. So, the primary goal of this project is to provide a user interface that can streamline the testing process, minimize the need to interact with programming code directly, allowed users to get a grasp of how to use this system more easily.

Another goal of this system is that it needs to be easy to read, and to achieve this, we use 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 explaining what scientific software is, and what are their purpose. Follow up with why they are hard to test and why they aren't usually tested in a proper method. Then provide some of the current testing methods and why those methods are insufficient. Then this will be followed up with introductions to Cucumber and Behave, two of the main factors of the Causcumber system. After that will be explain what Causal testing is and how computational model testing is could benefit from it. Finally, explaining the current flaw of this method and what can be done to improve it.

In the third chapter, Requirements and Analysis, will be mainly talking about the requirements and objectives of the system. And what solution is possible based on these requirements

Part four, Design, will be discussing the design philosophy of the system, the design process

and discuss what the system should do.

Part five, Implementation and testing, will be demonstrate the final system by introduce different parts of it and what functions do those parts have.

Part six, Evaluation, will provide three(MIGHT CHANGE IN THE FUTURE) walkthrough of testing process with different models using the implemented user interface, each walkthrough will include the full testing process, the results, and a discuss about the result. Part seven, Results and discussion, will discuss what does the results mean and how does the user interface help simplify the testing process, and what can be improve with the user interface.

Part eight, Conclusions, will summarize what has been achieved in this project, provide a overview on what the final product performs.

Chapter 2

Literature Survey

This section will explain why there's a need for this system by exploring background literature related 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 explaining why most of them are not tested correctly. Then we will be discussing some of the current method in testing and why are they insufficient. Next we will be discussing the two main components 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 a proper interactive system for non software engineering users.

2.1 Computational model

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

2.1.1 Aspects of computational model

What defines a scientific computational model, it's by using computers to simulate and study real life events by using mathematics, statistics, physics, and computer science to study the mechanism and behavior of complex systems by computer simulation [1].

There are two main ways to process model's data, empirical and mechanistic:

For empirical, researcher don't need to know exactly how the system works, they just need to observe the outcome of certain scenario and predict what's going to happen in the future[Empirical vs. Mechanistic Models REFERENCE]. For example, by observing how tides are going to change in a year, people can predict how the tides will change in

the upcoming years. They don't need to know the exact mechanism behind why and how the tides change, only require knowing the circumstance and the outcome. By using this method, it simplifies the model since there's less complex calculation required. And it's better to comprehend compared to mechanistic. The problem for empirical is it require a lot more input data to have good accuracy in the model, there won't be much problem if the model is developed with big data and machine learning tools, but if the model don't have enough data at the start, it will affect the model's accuracy.

For mechanistic, it requires fewer input data to achieve the same accuracy as the empirical, but it is also more computationally intensive. Compared to empirical, it is easier to extrapolation, researchers can make predictions based on the previous input data.

Model also split into two types, agent based and equation-based model.

For agent-based model, is a system is modeled as a collection of autonomous decision-making entities called agents. Agents will act based on the situation its in and the abilities it has(AGENT BASED MODEL REFERENCE). Agents with react appropriate to the rule defined for them, for example, in an agent-based model for honeybee, if a bee detect a flower near by with resource in it, the bee will move toward the flower and collect the resource. In this case, the bee agent is in a situation where there is flower with resource nearby, according to the rules define for the bee agent, it reacts by moving towards the flower and collect from it.

For equation-based model, instead of have agents with rules defined, it have a set of differential equations, the model uses equations to represent relationships between observables(EQUATION BASED MODEL REFERENCE), the situation of observables will change depend on the equation. For example, modeler can define an equation on how fast a disease will spread, by define a start infected population, the equation can then calculate how many people are infected in each given timestep.

2.1.2 Purpose of computational model

By using models that contain a number of input variables, and algorithms that will define the system. Researchers use these models to simulate real-life situations and 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 [10]. 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 [4], 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 grouped into a social network depending on their attributes. After that the model will start looping, in each step, the model will apply various operations on the individual, then collect the result and apply analysis.

2.2 Testing computational model

Computational models are often very complicated, and require special knowledge related to the field, so is testing computational model. This often results in some issue in testing, some methods have been developed to help with the testing process, but most are either to complicate or insufficient. There's currently a more efficient way to test these models which is by using behavior-driven development (or BDD) and Cucumber. Combined with casual model testing, a way to draw relationships between variables, it will make testing these complex models a lot easier.

2.2.1 Why computational models are hard to test

Computational models are often developed by scientists themselves, but most scientists aren't software developers and may not have knowledge in some common software engineering practices, this may cost the quality of the scientific software [13]. 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 researchers 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 [13].

Second is that scientific software is mostly developed with scientists being the lead role instead of a software engineer, and the value of the software system is often underestimated [13]. And most scientists never went through the training in software design like software engineer [12], this means limited understanding of the testing process and not applying known testing methods. This will make the core design of the software unfriendly for effective testing.

2.2.2 Current testing method

After modeler complete the modeling process, they are required to verify the model. Despite the rarely carried out practice, verification is an important step in modeling, it helps making sure the results produced by the model is accurate enough to represent the reality or the theory of the model. Verification is done by comparing the predication of the model with actual data. The type of compared data provide will determine whether the models is validated at the pattern, point, distribution, or value level, or some combination of these (CURRENT TESTING METHOD REFERENCCE).

This verification step can have some issue when it comes to models with multiple agents, modelers may be required to test the agent as an individual or in as a group, this is often determined by the requirement of the model. If the model's main goal is focus on the broader environment, then its agents need to be verified as a group, for example in a model for researching bee colony's behavior, the bee agents will be assessed as an entire hive instead of as individual bees. On the contrast, if the model's goal is focus on individual agent, it will need to be verify as an individual. The modelers will be required to collect data to compare it with the results of the model. This can make the verification process become very tedious and time consuming.

2.2.3 Behavior-driven development with Behave

Behave is a python API based on behavior-driven development, where the goal is for people who don't really understand programming code to be able to participate in the testing process. Behavior-driven development is a development technique that encourages collaboration between participants in a software project [2]. Behavior-driven development or BDD, aims to have a clear understanding of the desired software behavior between stakeholders and software developers, and communicate by writing test cases in a natural language that both sides can understand. Behave is a Python API created for this, it consists of tests written in Cucumber, another API written in natural language style.

When a test case is written, it requires a tool to read its specifications and see if the system works as the test-case specified. Cucumber is the tool that was developed for this

purpose, it reads specifications written in plain text with a certain format and validates if the software does what those specifications say [8] and creates a report.

For the non-developers who want to participate in testing, they are required to create a feature file. The feature need 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 [9]. A proper Gherkin grammar starts by giving a context, then describing an event, and after that what should happen after the event. With these grammars, testers 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 2.1: A feature file example.

For developers, they are required to create a step decorator the matches the steps, for example in the above feature file example, there is a “given” type, therefore in step decorator will need to match it as below:

```
@given(u"a simulation with parameters")
def step_impl(context):
    for row in context.table:
        var = Input(row["parameter"], locate(row["type"]))
        context.types[row["parameter"]] = locate(row["type"])
        var.distribution = eval(row["distribution"])
        context.scenario.modelling_scenario.variables[row["parameter"]] = var
```

Figure 2.2: A feature file example.

With the step file implemented, tester can execute the .feature file in terminal and Behave can read what needs to be tested and return a report. In this way, tester who don’t have sufficient knowledge can involve in testing phase, therefore making collaboration between developers and non-developer in a project become a lot easier.

2.2.4 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 mistakes. 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 testers monitor causal relationships in the data [6]. 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 [3]. Tester can purposefully change a certain input to observe the change in behaviour, when change in input cause change in output, tester can observe causal relationship between the variables. When there’s a difference between in executions of the system, whether its in input or the execution process, no matter how small it is, these differences can help testers identify any unusual behaviour in the system.(CAUSAL TESTING REFERENCE)

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

2.2.5 Testing computational model with causal testing

Causal testing can be a really useful testing method for software engineers [5], however computational models are often developed by scientists or researchers who have limited knowledge in software testing [2]. 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 means people who may not have that much knowledge in software engineering, for example researchers in other fields, might not be able to conduct this method of testing in their software.

By incorporate Behave and Cucumber into Causal testing, testing with Causal testing become more accessible for people who don't have specific knowledge in software testing. Without Behave, testers will need to either design inputs them self or using auto test value generation, but either option will require testers to have sufficient training in software testing, which are not common among scientists or researchers who developed computational model.

With Behave, user can specify the testing cases in a more natural language, making creating testing case more accessible. After creating test case for computational model, user can compare different test cases to identify the behaviour of the system and use the behaviour to identify if the subject system is behaving correctly.

Despite the existing implementation of combining Behave and causal testing, Causcumber, it still require more assist in order to make this system more easy to approach. In the current version of Cucumber, it requires users to edit the test case file directly, which still require user to use a certain syntax to make sure the test cases work with the decorator. There's still a need to further streamline this process, in order to achieve this, we can implement a GUI to guild and assist tester to in the testing process. With this GUI, tester can use it to create test case without the need to interact with the coding part of the system. This approach can make the testing process even more accessible.

2.3 Summary

Testing a computational model isn't a easy task, not only because there can be lots of unknowns in the developing of computational model, but also because among the people who develop those software, most have limit experience in software testing. The current testing method is by gather lots of data and compare those data with the result produced by the model, this method may be useful, but can become tedious with more complex model. To make this process more convenient, several tools can be used to help simplify the process. With Behave, an API based on the behavior-driven development method, tester can specify the test case in a natural language, making test process a more approachable, encouraging people who do not specialize in software engineering to take part in testing. With causal model, testers can effectively test large and complex computational models. By using different test cases, tester can compare the inputs, outputs and the execution path of the test cases

to identify the behaviour of the system. And with the behaviour, tester can verify if the software is working as intended. But the problem is that when using causal model testing, it is inaccessible for most people, due to its requirement for user to have knowledge in software testing. To make it more user friendly, Behave is integrated to make it where users can create test case with a relatively natural language. But it still require a certain syntax to work, so a way to further simplify the testing process is needed, which leads to the development of a GUI to help streamline this process.

Chapter 3

Analysis

Computational models are often too large and complex for traditional methods to test, this often causes developers of those models unwilling to test their models thoroughly. Furthermore, computational model developers are often scientists or researchers that don't have extensive knowledge in software engineering or software testing. Due to the different training received by scientists compared to software developers, 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 models with causal model testing easier for people. By integrating Cucumber testers 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.

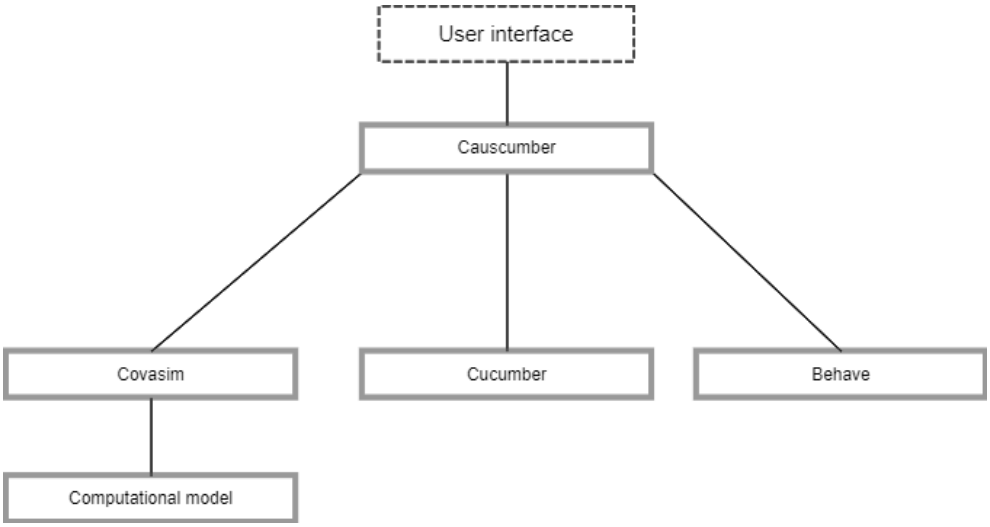

Figure 3.1: A diagram analysis for the system.

3.1 Project Requirements

The main goal of this project is to develop an easy-to-use system where people can use it to execute the feature file they have written, the system will examine the feature file and go through the computational model to check if the model performs as the test specifies. 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 has been in development for a while, what this project aims to accomplish is adding more to this tool. Currently the state of Causcumber is capable of executing some feature files that are used to test a computational model named Covasim. And it is capable of returning a 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 for a user interface for people to operate the system. Last is that currently the only way to create a feature file is by hand typing 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 functions that are 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 produced 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 options automatically based on different situations.

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 focused 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 integrating the causal model testing can be difficult and confusing, requirements 3, 5, 6 are focused on simplifying 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 models 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 is open-source and free to use, the tools used to develop the user interface are also open-source and free to use, so this means there won't be any legal problems either.

Bibliography

[1] Computational modeling. *National Institute of Biomedical Imaging and Bioengineering* (May 2020), 1.

[2] Benno Rice, R. J., and Engel, J. Behavior driven development.

[3] Brittany Johnson, Yuriy Brun, A. M. Causal testing: Finding defects' root causes. 123 – 146.

[4] Cliff C. Kerr, Robyn M. Stuart, e. Covasim: An agent-based model of covid-19 dynamics and interventions. *PLOS Computational Biology* (July 2018).

[5] Gaskin, P. B. L. J. Partial least squares (pls) structural equation modeling (sem) for building and testing behavioral causal theory: When to choose it and how to use it.

[6] Hitchcock, C. Causal models.

[7] Kerr CC, Stuart RM, M. D. e. Covasim.

[8] Marit van Dijk, Aslak Hellesoy, e. Cucumber.

[9] Marit van Dijk, Aslak Hellesoy, e. Gherkin reference.

[10] Muffy Calder, Claire Craig, e. Computational modelling for decision-making: where, why, what, who and how. *Royal Society Open Science* (June 2018).

[11] Robert C Wilson, A. G. C. Ten simple rules for the computational modeling of behavioral data. *eLife* (November 2019), 1.

[12] Segal, J. Scientists and software engineers: A tale of two cultures. *Open research Online*, 1 (2008), 2.

[13] Upulee Kanewala, J. M. Testing scientific software: A systematic literature review. *ScienceDirect* (May 2014).