COVID-19 Model Testing
Getting Important Work Done in a Hurry

Christian Wiswell

cwiswell@idmod.org

Abstract

Hitting milestone dates can be challenging even with all of the right planning and scheduling.

The COVID-19 work disruptions are unprecedented. Policy decisions about how to respond are informed by sophisticated computational models, which in turn need to undergo scrutiny.

Testing computational models is challenging because the desired behavior is "accurately simulates the real world," but the real world behavior is at least partially unknown or theoretical.

This paper will tell the story of how the Institute for Disease Modeling (IDM) tested its Covasim model, and how that work was divided between Subject Matter Experts and professional software testers. This story includes discussion of collaboration, test methodology, and balancing risk with time constraints.

Biography

Christian Wiswell's career in software testing began as a manual link-clicker in 1999, and proceeded through jobs at web startups, Microsoft, and Google. He started work at the Institute for Disease Modeling in 2013, becoming a test manager in 2015.

1 Introduction

The purpose of this paper is threefold. Firstly, it is to equip people to test specialized software without becoming a subject matter expert. Secondly, it is to discuss testing of computational models of infectious disease transmission. Lastly, it is to tell the story of how a particular piece of software was efficiently tested so that others can learn from the experience.

To these ends, the paper will discuss the Institute for Disease Modeling and its Covasim model, which simulates outbreaks of COVID19. It will also discuss some issues with testing stochastic and scientific software and tell how some tests were developed in the Covasim case. Hopefully, the narrative parts of the paper will also be helpful to people trying to understand some of the soft skills of working with subject matter experts on the testing of software.

2 Background: Institute for Disease Modeling (IDM)

The Institute for Disease Modeling (IDM) is a group working in Washington State on applying the power of computational modeling to the study of infectious disease transmission. The Institute consists of a mix of academic researchers with expertise in computational modeling, and software engineers with experience in designing, building, and testing software[1]. The Institute has existed with different names and organizational structures for about ten years, and has published dozens or papers in academic journals, and worked with many other institutions (including NGOs and government institutions) to help people understand disease dynamics.

Most of the work of IDM in the past has been on the diseases Malaria, Polio, and HIV, and in more recent years has expanded to include Typhoid, Measles, and Tuberculosis / HIV coinfection.

2.1 The EMOD Model

A large part of IDM's research on these diseases uses IDM's Epidemiological MODeling software (EMOD). The following is a quick overview of EMOD software to give context for the discussion of the Covasim model. For a more thorough discussion of the EMOD software, please see the linked paper in references.

EMOD is a stochastic, agent-based engine for simulating epidemics. Agent-based means that every time an EMOD simulation is run, a separate object in memory is created to represent each person in the simulation who may get infected. Agent-based models are powerful tools for answering modeling questions where each individual is potentially unique. For example, a malaria simulation where some individuals have no bednets, others have new bednets, and others have bednets that are aging and less effective. Stochastic means that some events in the simulation are dependent on a random number draw. This means that before the simulation executes, it is not possible to know the exact state of the simulation at the end. While stochastic, agent-based models can be computationally expensive, EMOD was designed around the concept of transmission pools, which makes some kinds of simulations faster.

2.1.1 Transmission pools in EMOD

EMOD's transmission pool concept is how the model implements agent-based transmission. Consider the following example. In this simple model, we have five agents. Two of the agents are infected, which is represented by lightly shaded people. Three of the agents are susceptible to a new infection, which is represented by being dark. However, the two infected agents are shedding different levels of infection. This could be for any number of reasons. For example, perhaps one of the agents is washing their hands more frequently, which reduces that person's shedding. It is also possible that one of the agents is at a more infectious part of their infection than the other one (in a disease where infectiousness goes up and down over time).

[1] More information about IDM is available at https://idmod.org/.

Let's consider how EMOD models transmission.

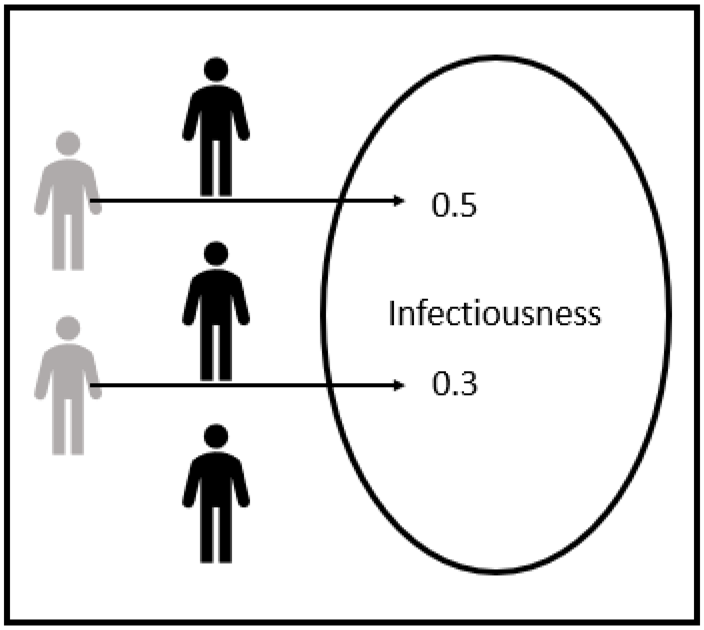


Figure 1: Two people contribute to the infectiousness pool through shedding.

In Figure 1, we see the two infected agents contributing to the transmission pool. This is called shedding. In the case of a respiratory disease like Measles or Tuberculosis, can be caused by coughs or sneezes.

After shedding is complete, susceptible agents are exposed to the total infectiousness, and some of them are selected to be newly infected. In the example, one agent becomes infected during exposure (Figure 2). During the next shedding phase, this individual may contribute to shedding.

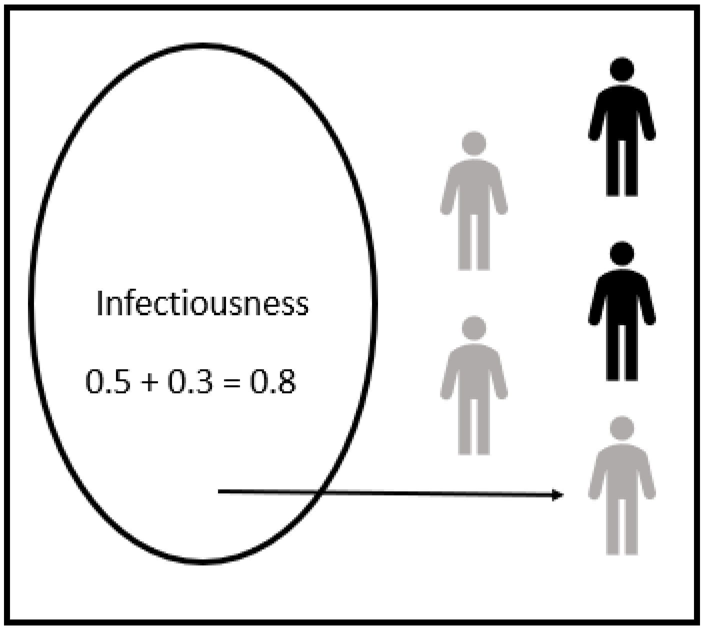


Figure 2: With a total infectiousness of 0.8, one of the three susceptible agents becomes infected.

2.1.2 EMOD inputs and outputs

EMOD requires a configuration file defined in JSON format, where each key is the name of a model parameter and the corresponding value tells the model what to use. For example

```
 "Simulation_Duration": 50
```

tells the model to run for 50 time steps before finishing and writing reports.

Most of the model outputs are channelized JSON, where each key is the name of a reporting channel and the corresponding value is a list of results per time step. For example

```
"New_Infections": [0,1,0,3,5,12]
```

Indicates that there was 1 new infection on the second time step, 3 on the fourth time step, and so forth.

2.2 Testing of Modeling Software

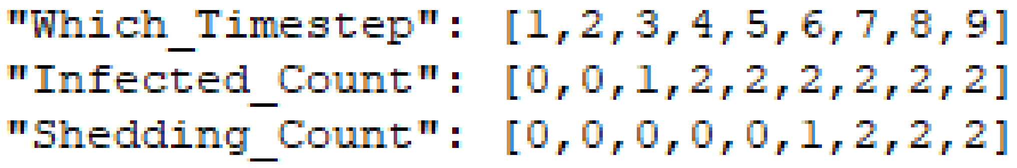
Validation of this kind of software is very challenging. Consider the following question:

Does this model correctly determine how long after becoming infected a person starts shedding virus?

Answering this question by looking at the outputs of a normal simulation is impossible.

- Each individual who becomes infected will draw a pre-infectious period from a probability distribution (for example, a bell curve distribution). To see if the distribution is correct, you would need to consider several people at the same time.

- Models that only output total counts at each time step make knowing the state of each person impossible. Consider a simulation with two people. One of them is infected at time 3, the other at time 4. One of them begins shedding at time 6, and one at time 7. Output files similar to EMOD's would contain data to represent these totals like so

  ```
  "Which_Timestep": [1,2,3,4,5,6,7,8,9]
  "Infected_Count": [0,0,1,2,2,2,2,2,2]
  "Shedding_Count": [0,0,0,0,0,1,2,2,2]
  ```

  But this data doesn't tell you *which* of the two infected people began shedding on time 6. When the simulations contain hundreds or thousands of individuals, any chance of getting data about individuals from these reports disappears.

- Lastly and perhaps most importantly, the word *correctly* here is impossible to answer, because the scientists using these models are making estimates based on the best data they can find.

Fortunately, there are techniques for dealing with the first two problems. For testing to see if draws from a probability distribution are correct, there are powerful statistical techniques one can use to see if lists of numbers are likely to have come from a given distribution. In the case of EMOD, we've found that the Kolmogorov-Smirnov[2] and Anderson-Darling[3] tests have proven useful but required a great deal of time and effort to implement them properly. Interpretation of statistical test results is also non-trivial. A discussion of how to use these tests for software validation is out of the scope of this paper.

[2] scipy.stats.kstest, from scipy.org https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.kstest.html (accessed September 4, 2020)
[3] scipy.stats.anderson from scipy.org https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.anderson.html (accessed September 4, 2020)

Resolving the problem with getting the proper outputs can be resolved through additional logging or careful crafting of test simulations. In EMOD, our development teams have enabled specialized logging outputs that can tell us the state of every individual at each simulation time. Creating this required a lot of effort on the part of the development staff and must be disabled in production environments to prevent non-test simulations from overwhelming file storage (these debug logs can be very large).

In some cases, it may be possible to configure a simulation in a way that highlights a specific feature of the model. For example, if we want to find out how long infections last without confounding our results with new infections, we can configure the simulation to force every individual to be infected at the first time, and just measure the total count of infected people at each time to make those durations visible at the whole simulation level.

3 The Challenge

In late 2019, a novel coronavirus was first identified in the Wuhan region of China that produced a very serious respiratory condition, and the disease began spreading rapidly. By early 2020 it had reached the United States and was beginning to cause infections and deaths in Washington State, where IDM is located.

Among the many important areas of COVID-19 modeling were questions around what interventions could be taken to limit the spread of the disease. Among the proposed interventions were the following items.

- **Testing** Including testing some percentage of all people and testing some percentage of symptomatic people
- **Contact tracing** After a person is diagnosed, trying to find the people they have been in contact with and testing them for infection
- **Quarantining families** Keeping the families of infected people away from other people in case they have infections that haven't been detected
- **Isolating infected people** Keeping people known to be infected away from other people
- **Social distancing** Closing workplaces, schools, or both, and encouraging policies that would limit disease transmission among people in contact (masks and handwashing)

The EMOD model, while very flexible and powerful, had some limitations at the time that would have made modeling these interventions difficult[4]. EMOD's implementation of transmission pools (section 2.1.1) didn't track human-to-human transmission. This meant that the model was unaware from which person a newly infected person got their infection, so contact tracing in EMOD wasn't possible. Also, EMOD did not track families, workplaces, or schools, which made complicated social distancing scenarios (Shut down middle schools and large workplaces, but not elementary schools and small workplaces) not feasible.

4 Our Solution: Covasim

Fortunately, IDM had a new modeler with strong computer science skills who brought a different computational model to help answer these questions. This is the origin of the Covasim model. Covasim was developed specifically for answering questions around COVID-19 and the interventions mentioned above, so it had many features built in to make this easy.

[4] Many of the limitations of EMOD for Covasim modeling have since been overcome, but this code is not yet publicly released.

The Covasim model is like EMOD in that it is also a stochastic, agent-based model. A critical difference between the two is in how disease transmission works. Instead of using transmission pools, each person in Covasim can have from one to four contact networks. A contact network for an individual is a list of people in their household, work, school, or community. Instead of shedding into the transmission pool, individuals in Covasim expose some number of their personal contacts. A contact tracing intervention considers the contacts that an individual has and either tests those contacts for COVID-19, puts those contacts in quarantine, or performs some other operation.

Because the model was built with these questions and functionalities in mind, the modeling work began quickly, and researchers were able to begin work on answering questions about COVID-19.

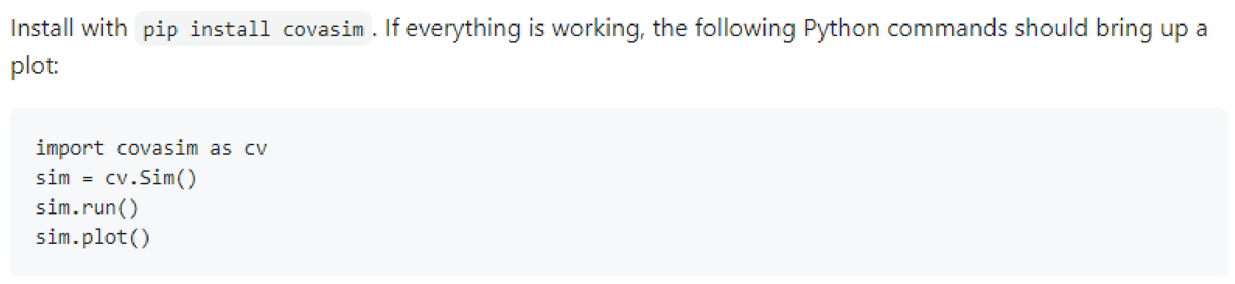
4.1 Testing Covasim

From a software testing point of view, the model was completely new. IDM's software engineering teams had years of experience in developing and testing new EMOD features and disease models but had never seen the Covasim software before. At the same time, IDM's research teams were using Covasim to inform policy decisions. This led to a real sense of urgency in testing the model quickly to ensure that the model produced correct results and provided good policy guidance.

To further complicate matters, IDM had issued a work from home order in March, so in-person collaboration with the research team was going to be difficult if not impossible. My initial attempts at testing the model were not very productive. Based on my experience with EMOD, I expected to find configuration files that I could begin editing, JSON reports produced as default outputs, and the idea of implementing statistical tests for every property of the model was quite intimidating.

The modeling team had created tests of the model that consisted of running it in different ways, enabling different features and then viewing the resulting graphs that were produced. When I reached out to see if I could be of help, they asked if it would be possible for me to generate code coverage numbers for their existing tests.

To make sure that some professionally designed test coverage could be implemented, I decided to lean on humility by simply following the directions in the README for running simulations. The instructions were straightforward. The software was in pure Python, and could be installed through the pip utility, and the simplest simulation could be run in Python through four lines of code.


Install with `pip install covasim`. If everything is working, the following Python commands should bring up a plot:

```
import covasim as cv
sim = cv.Sim()
sim.run()
sim.plot()
```

Figure 3: Covasim's quick start guide

Once I got simulations running, I began outlining a number of automated tests, using Python's unittest framework. I created skeletal outlines of a number of test areas, creating several test methods as "Not Yet Implemented" (NYI) and building entirely empty classes for each area of the model where I thought some test verification could be added.

This served two purposes. It showed the modeler the areas I intended to investigate. Also, it allowed me to have a sense of scope and progress toward completion. Each time I implemented another one of these tests and got it enabled, I could run the unit tests and watch the number of passing tests increase, and the number of skipped tests (all of the NYI tests were skipped) decline.

Ait this point, I reached out to the Covasim modeler again. Given that we were both working from home, we ended up having an extended conversation on a Slack chat about how to add software test coverage with verification to the model. When I showed the modeler the skeletal tests I wanted to implement, the modeler kindly volunteered to add documentation as comments into the parameters.py file, including suggestions for appropriate tests for each parameter. This allowed me to receive instruction about the observable behavior of each of the parameters, as well as get some excellent suggestions about tests that could be easily implemented.

Perhaps most importantly, agreeing upon the parameters file as a shared responsibility began to build a sense of mutual trust with a new colleague. With this information as a clear point of agreement between us, I was able to write the support classes for my tests using parameter names that were familiar to me. At the time, Covasim's parameters dictionary had a key named "n" and another key named "n_infected". My support layer created a property called "number_agents" that I could use to refer to "n", and "initial_infected_count" that I could use to refer to "n_infected". Using the parameters.py file from the model code and my unittest_support_classes.py file in my test code, I could respond quickly to changing parameter names by only changing my code in the support layer, rather than having to change each of the individual tests every time a parameter was renamed.

4.2 Tests produced

This section contains some examples of tests that were produced for testing the Covasim model.

4.2.1 Testing reproducibility

One of the first tests produced was test_random_seed.

Like a lot of stochastic software, Covasim allows the user to set a random seed. This test is meant to show that the random seed is honored. To do so, it simply runs three simulations. The first two simulations use the same random seed, and the third uses a different one. All other parameters are the same. From a scientific sense, this test is worthless, as it shows nothing about a scientific property of the model.

From an engineering point of view this test is interesting. Consider an outside modeler who wants to reproduce results from a Covasim publication. This modeler may begin by running the exact same simulation that the original modeler did. If they get a different result than the published one, this would make reproducing the results difficult. This test gives reason to believe that reproduction of results is possible, because that modeler has the ability to use the exact same series of random number draws that were used in the original publication.

During model development, this test would fail when something added to the model made draws against the random number table before the seed was set. In other words, this test found defects in regression.

4.2.2 Testing of variance

A good example of testing variance in the model is test_exposure_to_infectiousness_delay_deviation_scaling. While the title of this test is really long, the thing it is testing is very small. The feature under test is identified in the test case title. This test is examining the delay between when an individual becomes exposed to when they become infected. In other words, "how long after a person is exposed to their infection do they begin shedding virus?" Within that delay is a random draw from a probability distribution, and this test is meant to see that the standard deviation of that distribution is honored. Consider several standard deviations here:

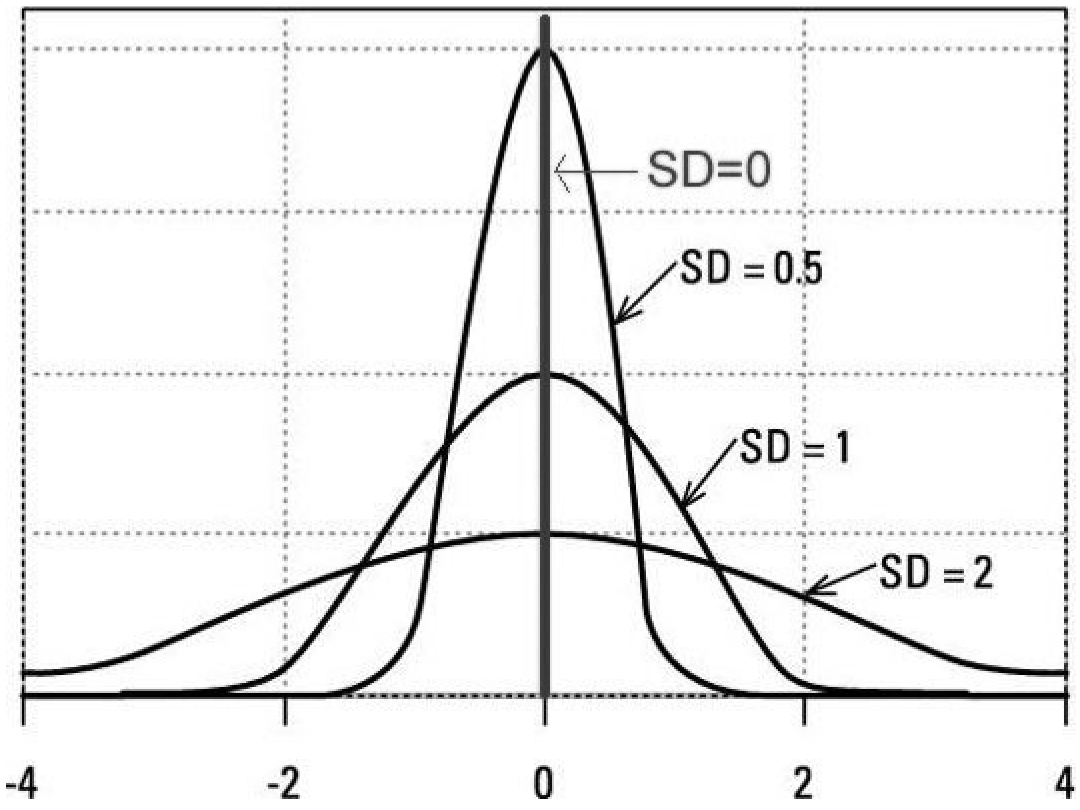


Figure 4: The effect of changing standard deviations on a normal distribution

As the standard deviation increases, the lowest and highest values in the distribution get farther from the mean, and the probability of the mean value decreases. So long as the mean of the distribution and the number of events (people who become infectious) remains the same, all of the cumulative values remain the same: mean period times the number of individuals.

In a realistic simulation, testing this feature is impossible, as discussed earlier (section 2.2). Basically, the information about any individual duration is lost in the noise of the other hundreds of infections reporting as cumulative numbers. This means that the configuration of this test will be very different from a realistic simulation.

A software tester can exercise their craft by configuring the simulation in a way that will illuminate the feature in question. In this case, the initial infected count (how many people are infected at the start of the simulation) is set to the number of agents (the total count of people in the simulation). This does several useful things. Firstly, it prevents any new infections from muddying the data, as there are no susceptible people. Secondly, it gives every single infection the very same start time, so every time a new person becomes infectious, we can tell how much time that took by subtracting the start time.

Consider a single individual, where we can express this as an equation:

Days for person X to become infectious = Day person X is infectious – Start day of simulation

With a simulation configured this way, what the test does is run a series of simulations with increasing standard deviations and stores three pieces of data:

- The first day any person became infectious
- The last day any person became infectious

- The highest number of people who became infectious on a single day

Consider the graph from Figure 4 again. As the standard deviation increases, the first day a person becomes infectious should be earlier. The last day a person becomes infectious should be later. And the number of people becoming infectious on the peak day should decrease, because the curve is flattening out (as the extremes spread farther, there are less people to change on the peak day).

Like the random number seed test, this test doesn't tell us anything really interesting about the science of COVID-19, it merely tells us that the parameter that controls the standard deviation seems to be working.

4.2.3 Testing of duration in the model

Building upon the previous test of variance, we have test_exposure_to_infectiousness_delay_scaling.

Look at the graph from Figure 4 once more. At the center of the graph, it shows that with a standard deviation of zero, the bell curve becomes a straight line, and every event should happen at the mean.

Having already shown that the variance is honored (with the previous test), we run a series of simulations where the standard deviation is zero, and we change the mean to several different days. Since the distribution tells us that all of the events must occur on the same day, the verification is simple. We look at the number of people who are infectious for each time in the simulation. If the current time is less than the mean, that number of people should be 0. If the current time is equal or greater than that time, the number of people should be equal to the whole population.

This is a strong prediction, and when the test confirms this prediction, it gives us a very strong result.

5 Results and Conclusions

5.1 Testing modeling software in a hurry

Earlier in this paper, I considered this question someone might ask about an infectious disease model:

Does this model correctly determine how long after becoming infected a person starts shedding virus?

Without the engineering effort to create voluminous logging, and the test automation effort to use powerful statistical tests and log parsing, we had to find a different way than we did for EMOD. Instead, for Covasim, we can now answer this question this way:

- The parameter that defines the standard deviation of the pre-shedding period is honored. We use relative validation (the results for standard deviation 3 are earlier, later, and broader than for 0, 1, or 2) for this.

- The parameter that defines the mean of the pre-shedding period is honored. We use absolute validation (with standard deviation of 0, and mean time of 13, every human in the simulation becomes infectious on day 13) for this.

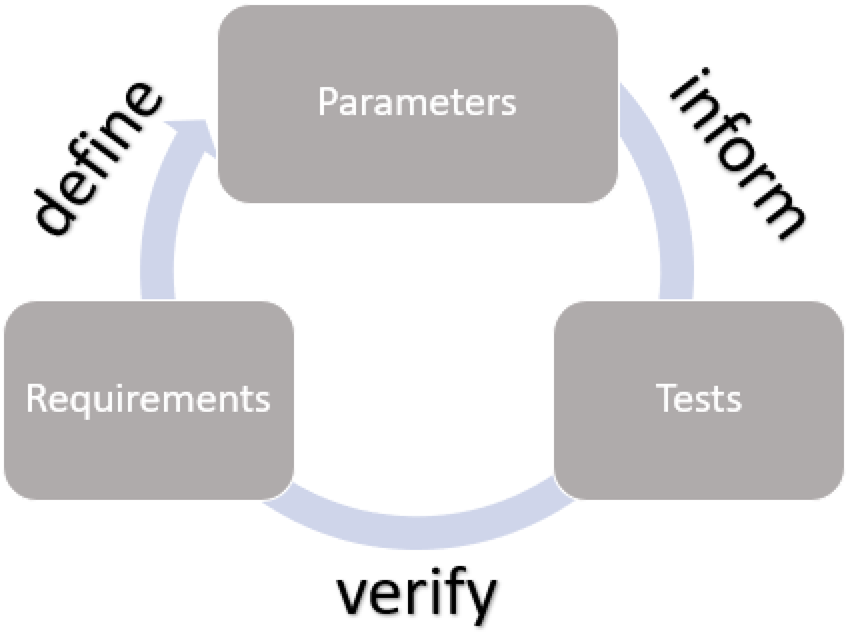
Therefore: Whatever distribution of pre-shedding period the modeler specifies is what the modeler gets.

This reduces the concern of "Does this software accurately simulate the real world," because the question changes to "Does this software simulate the world that is specified by the user?" This becomes a question that can be answered through software testing.

5.2 Conclusions

A great deal of work was put into testing the Covasim model, but only part of that came from software engineering teams. One of the lessons learned here was one of humility. When testing specialized software, lean on your specialists to test things you cannot. Some of their methods will be unlike your own, and you need to allow them the space to apply those methods. As someone who writes automated tests, it is hard to wrap my head around "plot a graph and look at it" as a test methodology, but when the final deliverable is a graph that answers a question, having an expert looking at that graph and pondering it may be the most powerful tool in our toolbox.

The second lesson here is of confidence. As Jenny Bramble said, "anyone who has made their career in testing software is an expert in testing software," and that includes you. So even when working on highly specialized, highly sophisticated software solutions, there's always ground to cover from an engineering point of view. Can you install the software, can you uninstall it, and can you run existing examples?



The last lesson is one of collaboration. In working on highly complex, highly specialized software, find some place where the Subject Matter Expert (SME) can tell you about where the knobs in the software are, how you can turn them, and how you should observe them working. For a mailto: link on a web page, you probably don't need any help, but for modeling contact tracing among school age children you probably do. In my case, that was the parameters dictionary that the SME was able to document in about an hour. This took very little effort on their part but was immensely useful for the work I did based on it. The researcher wrote requirements, which described the parameters. I consumed the parameters in tests, and then validated against the requirements.

If you can find this common ground with your SMEin and apply your expertise to the software parts of the problem, you can effectively test specialized software.

References

Anna Bershteyn, Jaline Gerardin, Daniel Bridenbecker, Christopher W Lorton, Jonathan Bloedow, Robert S Baker, Guillaume Chabot-Couture, Ye Chen, Thomas Fischle, Kurt Frey, Jillian S Gauld, Hao Hu, Amanda S Izzo, Daniel J Klein, Dejan Lukacevic, Kevin A McCarthy, Joel C Miller, Andre Lin Ouedraogo, T Alex Perkins, Jeffrey Steinkraus, Quirine A ten Bosch, Hung-Fu Ting, Svetlana Titova, Bradley G Wagner, Philip A Welkhoff, Edward A Wenger, Christian N Wiswell, for the Institute for Disease Modeling, "Implementation and applications of EMOD, an individual-based multi-disease modeling platform" *Pathogens and Disease*, Volume 76, Issue 5, July 2018, fty059, https://doi.org/10.1093/femspd/fty059

Fok, Luis "What is the shape of normal distribution when standard deviation is least?" https://www.quora.com/What-is-the-shape-of-normal-distribution-when-standard-deviation-is-least (accessed August 20, 2020).

Bramble, Jenny "Building Automation Engineers from Scratch" http://uploads.pnsqc.org/2019/papers/Bramble-Building-Automation-Engineers-From-Scratch.pdf (accessed August 20, 2020)

The more powerful statistical tests mentioned in section 2 are available in the SciPy library for python and are listed here https://docs.scipy.org/doc/scipy/reference/stats.html#statistical-tests (accessed August 24, 2020)

- Komolgorov-Smirnov test https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.kstest.html (accessed September 04, 2020)
- Anderson-Darling test https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.anderson.html (accessed September 04, 2020)

For more information about the Institute for Disease Modeling, see https://idmod.org/

For more on the Scipy library, see the following paper

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) **SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python**. *Nature Methods*, in press.