On Effective Testing of Health Care Simulation Software

Christian Murphy[1], M. S. Raunak[2], Andrew King[1], Sanjian Chen[1], Christopher Imbriano[1], Gail Kaiser[3], Insup Lee[1], Oleg Sokolsky[1], Lori Clarke[4], Leon Osterweil[4]
[1]Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia PA 19104
{cdmurphy, kingand, sanjian, imbriano, lee, sokolsky}@cis.upenn.edu
[2]Dept. of Computer Science, Loyola University Maryland, Baltimore MD 21210
raunak@loyola.edu
[3]Dept. of Computer Science, Columbia University, New York NY 10027
kaiser@cs.columbia.edu
[4]Dept. of Computer Science, University of Massachusetts Amherst, Amherst MA 01003
{clarke,ljo}@cs.umass.edu

ABSTRACT

Health care professionals rely on software to simulate anatomical and physiological elements of the human body for purposes of training, prototyping, and decision making. Software can also be used to simulate medical processes and protocols to measure cost effectiveness and resource utilization. Whereas much of the software engineering research into simulation software focuses on validation (determining that the simulation accurately models real-world activity), to date there has been little investigation into the testing of simulation software itself, that is, the ability to effectively search for errors in the implementation. This is particularly challenging because often there is no test oracle to indicate whether the results of the simulation are correct. In this paper, we present an approach to systematically testing simulation software in the absence of test oracles, and evaluate the effectiveness of the technique.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Reliability, Verification

Keywords

Software Testing, Oracle Problem, Metamorphic Testing

1. INTRODUCTION

Simulation software is used in the health care field to model anatomical and physiological elements for predicting the effects of medical procedures [33] and for purposes of training and education [3, 13]. Such software may simulate only a specific organ such as the heart [20], or may incorporate an entire physiological system [19]. Simulation software is also used to evaluate medical processes, such as the impact of resource allocation on patient waiting time in a hospital Emergency Department (ED) [15, 27], or the response time of Emergency Medical Service technicians [30].

From a software engineering perspective, research into simulators typically is concerned with validating how well the simulation matches the real-world events it is trying to model [5], and not necessarily whether the implementation is free of defects. Researchers who investigate the verification of simulation software often suggest well-known best practices from software engineering [29], such as test-driven development and formal code reviews, but to date there has been little assessment of how effective these techniques are at discovering errors in the implementation.

Testing simulation software is particularly challenging because such software falls into a category that Davis and Weyuker describe as *"Programs which were written in order to determine the answer in the first place. There would be no need to write such programs, if the correct answer were known"* [14]. Without a "test oracle" [32] to indicate that the output is incorrect, it becomes difficult to find subtle calculation errors that could adversely affect the correctness of the simulation.

In this paper, we seek to systematically test simulation software, specifically focusing on the domain of health care, with the aim of discovering defects in the implementation. Our approach is based on the observation that many applications without test oracles, including simulation software, exhibit the following property: although we cannot in advance know the relation between an input and its corresponding output (since there is no oracle), it may be possible to then modify the input in a certain way, such that we can predict what the change to the output should be. If the application appears to exhibit such a property and the new output is as expected, that does not necessarily mean that the implementation is working correctly. However, if the property is violated, and the output is *not* as expected, then there is either a potential error in the implementation, or the property is not sound and needs to be modified for the particular domain or software. In either case, our approach leads to

SEHC'11, May 22-23, 2011, Waikiki, Honolulu, HI, USA

better understanding of the software and its inner working.

This approach, introduced by Chen et al. [9], is known as metamorphic testing, and has previously been shown to be applicable to testing applications without oracles [11]. To our knowledge, we are the first to use this technique to test software in the field of simulation in general, and healthcare-oriented simulation software in particular. In this paper, rather than focusing on validation, i.e., determining whether the *model* used in the simulation is correct, we focus on the *implementation*, and seek to test the implementation systematically to discover any error that may exist in the code. In addition to describing our observations about the practical application of the technique, we also present evidence that demonstrates that metamorphic testing is an effective approach for testing simulation software.

2. BACKGROUND

In this section, we further discuss the metamorphic testing approach, and introduce the simulation software to which we have applied it.

2.1 Metamorphic Testing

An early approach to testing applications without test oracles, including simulation software, was to use a "pseudo-oracle" [14]. This approach is based upon the notion of "N-version programming" [8], in which independent programming teams develop an application (perhaps using different technologies or programming languages) from the same specification; then, identical sets of input data are processed and the results are compared. If the results are the same, that does not necessarily mean that they are correct (since the implementations may all have the same defect, for instance), but if the results are *not* the same, then a defect has likely been revealed in at least one of the implementations.

Among the many limitations related to the applicability of the N-version programming approach [22], one obvious issue is that multiple versions of a program may simply not exist. In the absence of multiple implementations, however, metamorphic testing [9] can be used to produce a similar effect. Metamorphic testing is designed as a general technique for creating follow-up test cases based on existing ones, particularly those that have not revealed any failure.

In metamorphic testing, if input x produces an output $f(x)$, the function's so-called "metamorphic properties" can then be used to guide the creation of a transformation function t, which can then be applied to the input to produce $t(x)$; this transformation then allows the expected value of the output $f(t(x))$ to be predicted based on the (already known) value of $f(x)$. If the metamorphic property is clearly applicable to the function (and is sound), but the new output is not as expected, then a defect must exist.

Of course, like any testing approach, this can only show the existence of defects and cannot demonstrate their absence, since the correct output cannot be known in advance (and even if the outputs are as expected, both could be incorrect). This approach, however, provides a mechanism of revealing potential defects in such programs that do not have a test oracle [11].

2.2 Health Care Simulation Software

In this paper, we apply metamorphic testing to two different simulators that are used in the field of health care.

2.2.1 JSim

JSim [34] is a rigorous process based Discrete Event Simulation (DEVS) engine that has been effectively used to model and simulate, among other human-centric processes, the flow of patients through a hospital emergency room [27, 26]. Developed in Java by researchers at the University of Massachusetts Amherst, JSim's input consists of a complete *process* definition, represented graphically as a tree of steps and constituent substeps; a set of special resources called *agents* who perform the steps, such as a nurse in the emergency department; a set of non-agent *resources*, which include entities that *agents* may need to carry out the steps, such as beds, x-ray machine, medicine, etc.; specification of artifacts that carry information from one step to another, such as a patient chart; a resource manager that manages the availability and assignments of agent and *non-agent* resources; and an *oracle*, which dictates when process events should occur and how long they should take (not to be confused with a test oracle, of course). The oracle in the case of JSim is primarily a specification of agent behavior in terms of when an agent starts working on a task(step), when an agent completes a task, etc.

The JSim simulation engine maintains a timeline module into which events are placed. When the simulation is run, each agent is assigned items on its *agenda*, or a list of step instances to perform. The simulator goes through the process, and the JSim Agent Behavior Specification (the "oracle") indicates whether it is time for an agent to perform the next step on its agenda. If it is, and sufficient resources are available, then the agent performs the step and moves on to the next item on its agenda; if the required agent or the non-agent resources are not available, then depending on the configuration, the simulation engine either blocks (waits for the required resource) or throws an exception, in which case the process may be terminated. This continues until all agents have completed all items on their agendas and there are no more events in the timeline. The output of the simulation is the sequence of events related to each step's lifecycle, including the times at which they were started and completed, as well as the agents involved.

Testing JSim effectively has been challenging, as it is very complex and consists of many interacting components [28]. Adding to the challenge is the fact that JSim can be non-deterministic depending on the system configuration and resource constraints. For instance, the amount of time each step in the simulation takes to complete may be random over a range, either using a *uniform* distribution or using a *triangular* distribution with an inflection point at a specified mode. Thus the time it takes for the entire process to complete can often be non-deterministic in practice.

2.2.2 GCS: Glycemic Control Simulator

The Glycemic Control Simulator, or GCS, is a MATLAB program being developed by researchers at the University of Pennsylvania to simulate the behavior of different closed-loop insulin titration algorithms on a virtual patient. GCS can be integrated into larger simulations in order to support the model driven development of closed loop medical systems, similar to the general approach described in [2].

The GCS has two different simulation components: (1) the patient simulation, which is an implementation of the type I diabetic patient model from [19]; and (2) a closed loop glycemic control algorithm that computes insulin in-

fusion rates from the patient's blood glucose readings. In the current version, the control algorithm was derived from the glycemic control guidelines in use at the University of Pennsylvania Hospital. The simulator is run by specifying the initial conditions (such as patient weight, in addition to other metabolic parameters) and the amount of time to simulate. The simulation then produces the trajectory of the virtual patient's blood glucose, insulin infusion rate, and nutrition rate versus time.

Although the patient model and control algorithm could conceivably be separated and tested in isolation, we choose to consider them as a whole, to demonstrate that metamorphic testing can be applied to such closed-loop simulation software, as well as discrete event simulators such as JSim.

3. APPROACH

To demonstrate that metamorphic testing is an effective technique for testing health care simulation software, we started by identifying the metamorphic properties that we would expect most simulators (including our target applications) to exhibit, and then applied those properties to see if we could detect any defects or inconsistencies. Then, to determine the effectiveness of metamorphic testing, we systematically inserted defects into the software and measured how many of the defects were detected, as described in Section 4.

3.1 Identifying Metamorphic Properties

In [24], we enumerated six different classes of metamorphic properties, all of which are typically suited to applications that are principally numerical, such as machine learning or scientific computing. Here, we present some general guidelines for identifying metamorphic properties, using a running example of simulation software such as JSim that is used to model the flow of patients through a hospital's emergency department (ED).

First, we consider the metamorphic properties shared by **all applications in the given domain**. In the case of discrete event simulators, regardless of the particular algorithm, there are generally "resources" that are modeled in the simulation. These resources may be doctors and nurses in a hospital, developers and testers in a software company, or postal workers who deliver mail. No matter what algorithm is used, and no matter what is being simulated, all of these share some common metamorphic properties. For instance, if resources are being fully utilized, increasing the number of all resources would be expected to lower each resource's average utilization rate, assuming the amount of work to be done remains constant. On the other hand, if certain resources are underutilized due to a bottleneck resource, increasing that bottle-necked resource should result in increased utilization of other under-utilized resources. As another example, if the timing of all events in the simulation is multiplied by a constant factor, then the resource utilization should not change, since the ratio of the time spent working to the total time of the simulation should not be affected (because each are scaled up by the same factor).

We can also consider the properties specific to **the implementation of the algorithm** used to solve the problem. A given application that uses the chosen algorithm may have particular metamorphic properties based on features of its implementation, the programming language it uses, how it processes input, how it displays its output, etc. For instance, in simulating the operation of a hospital ED, the process definition language Little-JIL [6] and its corresponding simulator tool (in this case, JSim) may be used to specify the steps that an incoming patient goes through after arriving. In this implementation, the unique identifiers and the descriptions for the different resources (doctors, nurses, etc.) are specified in a relational database. Thus this implementation exhibits the metamorphic property that permuting the order of the resources in the database table should not affect the simulated process.

Last, we consider properties that are applicable only to **the given input** that is being used as part of the test case. Often it is the case that some metamorphic properties of an application will only hold for certain inputs. Consider an input to the hospital ED simulation in which the number of resources is sufficiently large so that no patient ever needs to wait. For this particular input, increasing the number of resources should not affect the simulation, since those resources would go unused.

Although the examples provided here are specific to the domain of discrete event simulations in general, and simulations of a hospital ED in particular, this approach can be applied for other types of medical (as well as non-medical) simulation software, such as modeling of a diabetic patient's response to insulin [19], as discussed in section 3.1.2.

3.1.1 JSim Metamorphic Properties

To identify the metamorphic properties specific to Jsim, we have considered the effect that changes in event timings would have on the utilization rates for the different resources in the simulation. Specifically, if the event timings were increased by a positive constant, then the utilization rate (i.e., the time the resource is used, divided by the overall process time) of the most utilized resource would be expected to decrease, since the total increase in the overall process time would outweigh the small increase in the resource's time spent working.

As a simple example, consider a process that has steps that take times a, b, and c to complete, and a resource that is used only in the third of these steps. Its utilization would thus be $c/(a + b + c)$. If the time for each step were increased by one, then the utilization rate would change to $(c + 1)/(a + b + c + 3)$. If this resource has the highest utilization, i.e. $c > a > 1$ and $c > b > 1$, then the new utilization rate will be lower, because the change to the numerator is proportionally smaller than the change to the denominator.

In the cases in which JSim uses non-deterministic event timing, we applied statistical metamorphic testing (SMT), which has been proposed as a technique for testing non-deterministic applications that do not have test oracles [16]. SMT can be applied to programs for which the output is numeric, such as the overall event timing in JSim, and is based on the statistical properties of multiple invocations of the program. That is, rather than considering the output from a single execution, the program is run numerous times so that the statistical mean and variance of the values can be computed. Then, a metamorphic property is applied to the input, the program is again run numerous times, and the new statistical mean and variance are again calculated. If they are not as expected, then a potential defect has been revealed.

For instance, we could specify a non-deterministic event

timing over a range [α, β] and run the simulation 100 times to find the statistical mean μ and variance σ of the overall event timing in the process (i.e., the time to complete all the steps). We could then configure the simulator to use a range [10α, 10β], run 100 more simulations, and expect that the mean would be 10μ and the variance would be 10σ. Of course, the results would not *exactly* meet those expectations, so we could use a Student T-test to see if any difference was statistically significant.

3.1.2 GCS Metamorphic Properties

As described in Section 2.2.2, the GCS contains two main components: the patient simulator and the control algorithm. A software defect in either component could have a large impact on the simulation result. We investigated two domain-specific metamorphic properties and one algorithmic property of the GCS. Each property, whether domain- or algorithm-based, relates to how the patient simulation models insulin sensitivity in the virtual patient. Insulin sensitivity describes the tendency for insulin to affect a change in the body's blood glucose level. A patient with high insulin sensitivity will require less insulin than a low sensitivity patient to achieve normal glycemic levels.

The first domain property specifies that patients who weigh more will be less sensitive to insulin and require more insulin to achieve normo-glycemia: given a simulation of a patient with weight w_1, if the total insulin delivered during the therapy is I_1 and the simulation is then executed with weight $w_2 > w_1$ then the resulting total of insulin delivered I_2 should be greater than I_1.

The second property is similar. Instead of weight, the endogenous glucose production EGP parameter is varied: given a simulation of a patient with endogenous glucose production EGP_1, if the total insulin delivered during the therapy is I_1 and the simulation is then executed with $EGP_2 > EGP_1$, the resulting total amount of insulin delivered I_2 should be greater than I_1.

We also identified a metamorphic property which should hold in the specific simulation algorithm we implemented. The insulin kinetics described in [19] is a collection of differential equations that relate insulin absorption, blood glucose levels and nutritional intake. One of these equations describes the rate of insulin absorption in the human body:

$$\frac{dI(t)}{dt} = \frac{k_a S_2(t)}{V_I} - k_e I(t)$$

Among all the equations that are collectively used to model the patient dynamics, this is the only equation with the term V_I, which represents the insulin distribution volume. Note that the insulin absorption rate varies inversely with V_I. If V_I is increased, the absorption rate will decrease, and more insulin should be required to maintain normoglycemia. Thus we can construct a metamorphic property with the parameter V_I: given a simulation of a patient with V_I, if the total insulin delivered during the therapy is I and the simulation is then executed with $V_I' > V_I$, the resulting total of insulin delivered I' should be greater than I.

Note that these properties might not hold for all control algorithms. For example, if the simulator executes a control algorithm with poor stability (and exhibits lots of oscillatory behavior) then more insulin might be delivered than necessary. In our testing, we chose a metamorphic interval for each of these properties that we believed would exhibit the above properties if the control algorithm were implemented correctly; these are shown in Table 1.

Parameter	**Variable**	**Value 1**	**Value 2**
Patient weight	w	60kg	160kg
Endogenous glucose production	EGP	4×10^{-3}	25×10^{-3}
Insulin distribution volume	V_I	10^{-2}	15^{-2}

Table 1: Values used in metamorphic testing of GCS.

3.2 Applying Metamorphic Testing

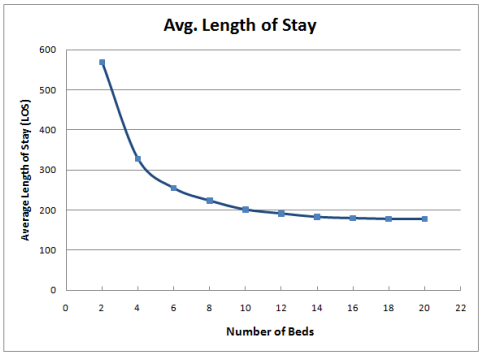
After identifying metamorphic properties, we applied them to the simulation software, and present our most interesting results here.

For JSim, we applied the metamorphic property related to expected trends in resource utilization that result from changing resource availability. As a necessary part of performing this test, we kept variability at a minimum by keeping all the step times fixed. We ran simulations using a simplified model of the emergency department (ED), which we shall refer to as the 'VerySimpleED' process. Space limitation allows us to only provide a very brief description of the process; the detailed description is available in [28].

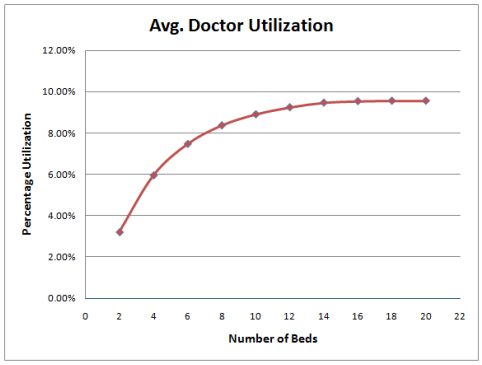
In the 'VerySimpleED' process, when a patient arrives at the ED, she first gets seen by a triage-nurse (TriagePatient step) and consequently gets a triage acuity level assigned. The patient then goes to the registration clerk for registration (RegisterPatient step). The registration clerk collects information from the patient including insurance information and puts it in the patient's record. The registration clerk also generates and places an ID-band on the patient. The patient then goes inside the treatment area of the ED (often referred to as main-ED) if a bed is available. If all beds inside the main-ED are occupied, the patient waits in the waiting room until a bed becomes available. This is modeled by a *blocking* acquisition request for a bed resource instance in PatientInsideED step.

Once a bed is successfully acquired, the patient is placed in a bed (PlacedInBed step) inside the main-ED. Here the patient is first seen by a nurse in the RNAssessment step, followed by an assessment by the attending doctor (MDAssessment step). The doctor assessment may result in some tests. These test related activities have been represented as a single abstract step named Tests. There are also some bedside procedures that may be performed on the patient by the nurse (RNProcedure step) and by the doctor (MDProcedure step). Once all the tests and procedures are done, the attending doctor makes a final assessment of the patient and decides whether to admit the patient or to discharge her (MDDischarge step), which is followed by paperwork that is performed by the nurse (RNDischarge step).

Both the above description and actual observed experiences suggest that beds are potentially bottleneck resources in EDs. We thus ran a set of simulations where the number of beds was varied, keeping other resources constant, starting with a single bed and adding more beds to the resource mix. We computed and plotted the average length-of-stay (LOS) against the increased number of beds. The output of



(a) Avg. Length-of-Stay



(b) Avg. doctor utilization

Figure 1: **Validating simulation results by increasing the number of beds.**

this experiment, shown in Figure 1a, demonstrates that as more and more beds were added into the resource mix, the LOS metric improved (i.e., was reduced). However, the improvement diminished with the increase of this one resource only and there was no impact of adding that resource after a certain point. This graph demonstrates a simple case of the "law of diminishing returns".

To continue with other metamorphic properties of this domain, it is reasonable to expect that if more beds are added to the resource mix, the utilization levels of other underutilized resource instances should increase. This is because having more beds should result in more patients simultaneously getting treatment inside the ED, thus requiring services from other resource instances such as doctors, nurses, etc. Consequently, that suggests that these other resources would be utilized more heavily with increases in the number of beds in the simulation. Further, we can expect that the amount of improvement would decrease as we continue to increase only the bed resource instances. To test this property, we collected utilization levels of all resource instances (not just the bed resource instances) used in the previously described set of simulations. This set of simulations used a resource mix that contained two triage nurses, two registration clerks, four doctors and four nurses. We took the utilization levels of each of the doctors as determined by the JSim simulation runs, and computed average doctor utilization for each. Figure 1b shows the graph of these results. As we expected, the average doctor utilization improved as more and more patients were allowed inside the ED simultaneously as a result of adding more beds. However, the improvement in the utilization was diminishing and gradually flattened out as the number of beds continued to increase. This also points to another metamorphic property: if there is no more waiting taking place due to resource contention, adding more resources should not improve the LOS.

As we applied the metamorphic property to JSim with a variety of different configurations, we did encounter one particularly interesting case in which the metamorphic property was violated. In this test, we simulated five patients arriving one after another every six minutes, with a resource mix of three beds, one doctor, one nurse, one clerk and one triage nurse. The simulation resulted in an average LOS of 217.4. When we increased the number of nurses to two, we saw the average LOS slightly increase to 220.2. This unexpected increase in average LOS led us to a deeper investigation of

1 nurse

Patients	Arrivals	Departures	Durations
1	2	159	157
2	8	185	177
3	14	197	183
4	20	295	275
5	26	321	295
Avg LOS:	217,4		

2 nurses

Patients	Arrivals	Departures	Durations
1	2	159	157
2	8	185	177
3	14	194	180
4	20	312	292
5	26	321	295
Avg LOS:	220,2		

Figure 2: **Length-of-Stay computation per patient**

what was happening. As shown in Figure 2, we discovered that with the addition of one nurse, the LOS for Patient 3 decreased from 183 to 180, but the LOS for Patient 4 increased from 275 to 292.

Further investigation by the JSim developers revealed that this was happening due to the particular scheduling algorithm in place. In the case of one nurse, when Patient 4 was ready for step MDDischarge, Patient 5 was not yet ready for its next step, MDProcedure. Consequently, the doctor agent first picked up the job of completing step MDDischarge for Patient 4 and then performed step MDProcedure for Patient 5. When one more nurse was added to the resource mix, Patient 5 became ready for MDProcedure early, but Patient 4 was still not ready for MDDischarge. JSim follows a greedy scheduling mechanism: it assigns the task of MDProcedure for Patient 5 first, and consequently the doctor performs the step MDDischarge for Patient 4 later, hence the additional time for patient 4. If the simulation software somehow could always ensure an optimum schedule in terms of the smallest average LOS, the scheduler would come to the conclusion that for the doctor, instead of starting to work on step MDProcedure for Patient 5 as soon as the step was ready, it is better to just wait and then start working on step MDDischarge for Patient 4.

This discovery led us to revise our metamorphic property for discrete event simulations. We realized that the property that "adding one resource should not increase average service time" is not applicable for all situations, especially where interrelated resource contentions are possible. Nevertheless, starting out with this metamorphic property led the developers to understand the impact of JSim's scheduling choices.

4. EVALUATION

This section describes the results of experiments in which we demonstrate the effectiveness of metamorphic testing in detecting injected defects in the implementations of the applications of interest.

4.1 Methodology

In this experiment, we used mutation testing to systematically insert defects (e.g., changing "+" to "-", ">" to "<", etc.) into the source code and then determined whether or not the defects could be detected using metamorphic testing. Mutation testing has been shown to be suitable for evaluation of effectiveness, as experiments comparing mutants to real faults have suggested that mutants are a good proxy for comparisons of testing techniques [1].

To determine which mutation variants were suitable for testing, the output of each was compared to the output of the application with no mutants, which was considered the "gold standard". If the outputs of the gold standard and the variant were the same, the mutation would be considered unsuitable for testing, since the mutation may not have been on the execution path, or may have been an "equivalent mutant" that did not affect the overall output. Additionally, if the mutation yielded a fatal error (crash), an infinite loop, or an output that was clearly wrong (for instance, being nonsensical to the application expert, or simply being blank), that variant was also discarded since any reasonable testing approach would detect such defects.

For JSim, of the 104 mutants we generated, only 25 mutants could be used in the experiment. The two principle reasons why we were only able to create a small number of mutants were that there simply are not many mathematical calculations in JSim, and that many of the mutants we generated led to obvious errors, such as crashing. We investigated the use of a mutant generator tool such as μJava [23], which would also create other types of mutations specific to Java (such as modifying inheritance hierarchies, variable scope, etc.) but the current implementation of μJava (version 3) as of this writing does not support Java generics [1], which are used throughout the JSim implementation.

Most of the JSim mutants were related to the event timing in the LinearRangeDuration and TriangleRangeDuration classes, so that when the configuration specified a timing range from A to B for a given event, the actual range would be [A+1, B] or [A, B-1] or [A+1, B-1]. We could not use defects that had ranges starting at A-1 or ending at B+1 because of checks that already existed within the code that would notice such out-of-range errors and raise an exception, further reducing the number of mutations that we could use in the experiment.

As described above, we used the statistical metamorphic testing approach for these cases that involved non-determinism. We validated the properties with the gold standard implementation (i.e., in which we had not inserted any defects), and found that the distributions that resulted from applying the properties were not significantly different, with $p < 0.05$. In the metamorphic testing of JSim, we used the same data set used in the emergency room simulation work presented in [27].

For the GCS, 950 mutants were generated using MATmute [2]. Of those mutants, 226 either did not produce output different than the unmutated program, or were not on the execution path, and were excluded from the experiment. Of the remaining mutants, 487 were mutations of the patient model implementation and 237 were mutations of the control algorithm. All three metamorphic properties of the GCS were applied to detect defects sequentially; if no defect was detected with the weight-based property, then the EGP property was applied, and so on.

4.2 Results

For both JSim and GCS, we applied the metamorphic properties discussed in Section 3.1 to each mutated version of the code, and measured how many mutants were killed (i.e., how many injected defects were revealed) using this approach.

The metamorphic testing approaches were able to kill all of the mutants in JSim, and 60% of the mutants in the GCS. Metamorphic testing was more effective killing mutants in the patient simulation component of the GCS (68% of defects detected) versus the control algorithm section of code (25%), as shown in Table 2.

	Control Alg.	**Patient Simulation**
Det. by Weight	39	254
Det. by EGP	9	41
Det. by V_i	10	38
Not detected	179	154
Det. rate	25%	68%

Table 2: Applying metamorphic testing to GCS.

4.3 Analysis

Here we discuss our results and explore the effectiveness of the different metamorphic properties.

4.3.1 Analysis of JSim Results

The most interesting result here is that in previous experiments to demonstrate the effectiveness of metamorphic testing when applied to machine learning applications [25], the metamorphic property based on multiplication was typically ineffective at killing off-by-one mutants; in this case, though, it was very effective. Upon further investigation, we see that this is due to the nature of the particular way in which the off-by-one mutant is manifested.

Consider a simple function $f(a, b) = a + b$, and a mutated version with an off-by-one error $f'(a, b) = a + b - 1$. We would expect f to exhibit the metamorphic property that $f(10a, 10b) = 10f(a, b)$; obviously, f' does not exhibit this property, since $f'(10a, 10b) = 10a + 10b - 1 = f'(a, b) - 1$. In this case, the property based on multiplication reveals the defect. The code we mutated in JSim included such simple functions, thus the property was very effective.

Although we did not compare metamorphic testing to other approaches in this study, we note that techniques such as assertion checking would be unlikely to detect any of the JSim defects related to event timing because they only consider a single execution of the program, or of the function that produces the random number in the specified range.

[1] `http://cs.gmu.edu/~offutt/mujava/`

[2] `http://matmute.sourceforge.net`

Consider a function that is meant to return a number in the range [*A*, *B*], but has a defect so that the range is actually [*A*, *B*-1]. No single execution will violate the invariant that "the return value is between *A* and *B*", so assertion checking would not reveal this defect.

However, statistical metamorphic testing *will* detect this defect because over the course of 100 executions of the program (as in our experiment), the mean and variance show a statistically significant difference compared to what is expected; other testing approaches may only run the program once, and would not consider the trend of the program over a number of independent executions.

4.3.2 Analysis of GCS Results

While our metamorphic properties of the GCS were able to detect and kill many of the patient simulation mutants, they were less successful detecting defects injected in the control algorithm implementation. The control algorithm used in this version of the GCS is composed of 10 conditionals of the form "*if patient blood sugar is x then adjust infusion rate by y*". The defect introduced by any single operator mutation will affect only one of the conditionals (either changing the guard, or linearly altering the control action the algorithm would take in that instance.) In many simulations, the condition related to the patient's blood sugar may only rarely be true under a given guard and any mutation protected by the guard would have little impact on the overall amount of insulin delivered to the patient. For this kind of control algorithm, less coarse metamorphic properties would be desired to achieve a high defect detection rate.

4.4 Summary

This experiment shows the feasibility of metamorphic testing as a technique for detecting defects in simulation software. By following the guidelines from Section 3.1 above, we were able to identify metamorphic properties that detected 100% of the defects in JSim, and 60% of the defects in GCS.

5. RELATED WORK

Researchers concerned with the validation and verification of simulation software often acknowledge the need for software testing, but typically do not present techniques beyond using some sort of formal specification [31], or approaches that are common for all types of software, such as test-driven development or code reviews [4, 21, 29]. Others have suggested similar techniques for testing software in the field of scientific computing [17, 18], as well, but we are not aware of any work that has attempted to reveal coding defects in simulators by using a systematic approach as we do here.

Applying metamorphic testing to situations in which there is no test oracle has previously been studied by Chen et al. [11]. In some cases, these works have looked at situations in which there cannot be an oracle for a particular application [12], as in the case of non-testable programs; in others, the work has considered the case in which the oracle is simply absent or difficult to implement [7]. Much of the work on applying metamorphic testing to applications without test oracles has focused on specific domains, such as machine learning [24], computational biology [10], or graphics [16]. To our knowledge, we are the first to focus on applying metamorphic testing to the domain of simulation software in general, and health care related simulation software in particular.

6. CONCLUSION

In this paper, we have demonstrated that metamorphic testing is an effective technique for testing simulation software that is used in the field of health care. We provided guidelines for deriving the properties on which the technique relies, and discussed how we used metamorphic testing to systematically test real-world simulation systems for effective discovery of potential defects. We also presented the results of a study that measured the effectiveness of metamorphic testing, revealing it to be very useful at finding injected defects in medical simulation software.

Future work could expand our study to include more implementations of simulation software, and to compare metamorphic testing to other techniques. Additionally, a possible direction for this research could be to consider using metamorphic testing for validation purposes, i.e., to discover whether the implementation of simulation software sufficiently matches the phenomenon it is attempting to model. If a metamorphic property based on domain knowledge and expectation of real-world behavior is violated, it may be the case that the error exists in the understanding of how to simulate the activity, and not in the implementation itself.

As practitioners rely more and more on simulation software to model the results of medical procedures and related processes, it is clear that the ability to reliably test these applications is critical. We believe that our work is an important first step in improving the quality of simulation software used in the health care field.

7. ACKNOWLEDGMENTS

We would like to thank T.Y. Chen for his guidance with metamorphic testing, and Guillaume Viguier-Just and Sandy Wise for their assistance with the JSim software. We also thank Roman Hovorka and Gosia Wilinska for their help with understanding their diabetic patient models including the one we implemented for the experiments presented in this paper.

Murphy and Kaiser are members of the Programming Systems Lab, funded in part by NSF CNS-0717544, CNS-0627473 and CNS-0426623, and NIH 2 U54 CA121852-06. King, Chen, Lee, and Sokolsky are members of the PRECISE Center, funded in part by NSF CNS-0834524, CNS-0930647, CNS-1035715, and NSF CNS-0720518. Raunak, Osterweil, and Clarke are members of the Laboratory for Advanced Software Engineering Research, funded in part by CCR-0427071, CCR-0204321 and CCR-0205575.

8. REFERENCES

[1] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *Proc. of the 27th International Conference on Software Engineering (ICSE)*, pages 402–411, 2005.

[2] D. Arney, M. Pajic, J. M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky. Toward patient safety in closed-loop medical device systems. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '10, pages 139–148, New York, NY, USA, 2010. ACM.

[3] N. Arshad, A. Akhtar, S. Khan, and D. Sabih. Software engineering for simulation systems in medical training - some initial experiences. In *Proc. of 2nd Wkshp on Software Engineering in Health Care*, 2010.

[4] O. Balci. Verification, validation and accreditation of simulation models. In *Proc. of the 29th conference on winter simulation*, pages 135–141, 1997.

[5] J. Banks, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation, 5th ed.* Pearson/Prentice Hall, 2010.

[6] A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, S. M. Sutton Jr., and A. Wise. Little-JIL/Juliette: A process definition language and interpreter. In *Proc. of the 22nd International Conference on Software Engineering (ICSE)*, pages 754–757, 2000.

[7] W. K. Chan, S. C. Cheung, and K. R. P. H. Leung. A metamorphic testing approach for online testing of service-oriented software applications. *International Journal of Web Services Research*, 4(1):60–80, April-June 2007.

[8] L. Chen and A. Avizienis. N-version programming: A fault-tolerance approach to reliability of software operation. In *Proc. of the 8th Symposium on Fault-Tolerance Computing*, pages 3–9, 1978.

[9] T. Y. Chen, S. C. Cheung, and S. Yiu. Metamorphic testing: a new approach for generating next test cases. Technical Report HKUST-CS98-01, Dept. of Computer Science, Hong Kong Univ. of Science and Technology, 1998.

[10] T. Y. Chen, J. W. K. Ho, H. Liu, and X. Xie. An innovative approach for testing bioinformatics programs using metamorphic testing. *BMC Bioinformatics*, 10(24), 2009.

[11] T. Y. Chen, T. H. Tse, and Z. Q. Zhou. Fault-based testing without the need of oracles. *Information and Software Technology*, 44(15):923–931, 2002.

[12] T. Y. Chen, T. H. Tse, and Z. Q. Zhou. Semi-proving: an integrated method based on global symbolic evaluation and metamorphic testing. In *Proc. of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 191–195, 2002.

[13] D. Chodos et al. Healthcare education with virtual-world simulations. In *Proc. of 2nd Workshop on Software Engineering in Health Care*, 2010.

[14] M. D. Davis and E. J. Weyuker. Pseudo-oracles for non-testable programs. In *Proc. of the ACM '81 Conference*, pages 254–257, 1981.

[15] G. W. Evans, T. B. Gor, and E. Unger. A simulation model for evaluating personnel schedules in a hospital emergency department. In *Proc. of 28th Conf. on Winter Simulation*, pages 1205–1209, 1996.

[16] R. Guderlei and J. Mayer. Statistical metamorphic testing - testing programs with random output by means of statistical hypothesis tests and metamorphic testing. In *Proc. of the Seventh International Conference on Quality Software*, pages 404–409, 2007.

[17] M. A. Heroux and J. M. Willenbring. Barely sufficient software engineering: 10 practices to improve your CSE software. In *Proc. of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 15–21, 2009.

[18] D. Hook and D. Kelly. Testing for trustworthiness in scientific software. In *Proc. of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 59–64, 2009.

[19] R. Hovorka et al. Nonlinear model predictive control of glucose concentration in subjects with type 1 diabetes. *Physiological Measurement*, 25(4), 2004.

[20] Z. Jiang, M. Pajic, A. T. Connolly, S. Dixit, and R. Mangharam. Real-time heart model for implantable cardiac device validation and verification. In *Proc. of 22nd Euromicro Conference on Real-Time Systems*, 2010.

[21] J. P. C. Kleijnen. Verification and validation of simulation models. *European Journal of Operational Research*, 82(1):145–162, April 1995.

[22] J. Knight and N. Leveson. An experimental evaluation of the assumption of independence in multi-version programming. *IEEE Transactions on Software Engineering*, 12(1):96–109, 1986.

[23] Y.-S. Ma, J. Offutt, and Y. R. Kwon. MuJava: An automated class mutation system. *Journal of Software Testing, Verification and Reliability*, 15(2):97–133, June 2005.

[24] C. Murphy, G. Kaiser, L. Hu, and L. Wu. Properties of machine learning applications for use in metamorphic testing. In *Proc. of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 867–872, 2008.

[25] C. Murphy, K. Shen, and G. Kaiser. Automated metamorphic system testing. In *Proc. of the 2009 ACM International Conference on Software Testing and Analysis (ISSTA)*, pages 189–199, 2009.

[26] M. Raunak, L. Osterweil, and A. Wise. Developing discrete event simulations from rigorous process definitions. In *Proc. of the 2011 conference on Theory of Modeling and Simulation/Discrete Events Simulation (TMS/DEVS11)*, 2011.

[27] M. Raunak, L. Osterweil, A. Wise, L. Clarke, and P. Henneman. Simulating patient flow through an emergency department using process-driven discrete event simulation. In *Proc. of the 2009 ICSE Workshop on Software Engineering in Health Care*, pages 73–83, 2009.

[28] M. S. Raunak. *Resource Management in Complex, Dynamic Environments*. PhD thesis, University of Massachusetts Amherst, 2009.

[29] R. G. Sargent. Verification and validation of simulation models. In *Proc. of the 37th conference on winter simulation*, pages 130–143, 2005.

[30] S. Su and C. L. Shih. Modeling an emergency medical services system using computer simulation. *International Journal of Medical Informatics*, 72(1-3):57–72, Dec 2003.

[31] W. T. Tsai, X. Liu, Y. Chen, and R. Paul. Simulation verification and validation by dynamic policy enforcement. In *Proc. of the 38th annual Symposium on Simulation*, pages 91–98, 2005.

[32] E. J. Weyuker. On testing non-testable programs. *Computer Journal*, 25(4):465–470, November 1982.

[33] N. Wilson, K. Wang, and R. W. Dutton. A software framework for creating patient specific geometric models from medical imaging data for simulation based medical planning of vascular surgery. *LNCS*, 2208/2001:449–456, 2001.

[34] A. Wise. JSim agent behavior specification language. http://laser.cs.umass.edu/documentation/jsim.