Understanding Unexpected Behaviors in Exploratory Simulations

Ross Gore
April 9, 2010

School of Engineering and Applied Science
University of Virginia
151 Engineer’s Way, P.O. Box 400740
Charlottesville, VA 22904-4740

A Dissertation Proposal
Presented in Partial Fulfillment
of the Requirement for the Doctor of Philosophy Degree
at the University of Virginia
School of Engineering and Applied Science
Department of Computer Science

ABSTRACT

Simulations and computational models have become the common tool of subject matter experts (SMEs) in a variety of disciplines to explore systems with inherent uncertainty. Predictions from these models and simulations have entered the mainstream of critical public policy and research decision-making practices. Unfortunately, methods for gaining insight into unexpected simulation outcomes have not kept pace. SMEs need to understand and explain unexpected behaviors in exploratory simulations to determine if the behaviors reflect an error or if they represent new knowledge in the discipline. Common practice is to apply classic debugging techniques to identify the program statements and interactions that lead to the unexpected behaviors. This practice is largely manual, it can consume years of effort, and it will not scale as models increase in complexity. Automation of at least a portion of the process has become essential. The automated process proposed here, Bayesian Program Slicing (BPS), will combine program slicing and Bayesian networks in a novel manner to identify program statements that are relevant to understanding unexpected behaviors. BPS will facilitate focusing SME attention on understanding and explaining the interactions of program statements whose execution results in variable state changes that are most relevant to the unexpected behaviors. Issues that make the proposed approach research challenging include: identifying prior knowledge in exploratory simulation that can be employed by Bayesian networks, efficiently sampling variable states and dynamic program slices and identifying an approach to cluster similar dynamic program slices. Evaluation of BPS will employ established methods for evaluating emerging software tools and quantitative metrics. The effectiveness of BPS will be compared to that of established, leading tools. The thesis of the proposed work is that by these measures BPS will be deemed more effective for facilitating SME explanation and understanding of unexpected behaviors than existing tools and will be a useful contribution by bringing automation to a challenging task.
1. INTRODUCTION

The research proposed here addresses improved methods and tools for explaining unexpected outcomes in simulations used for exploratory analysis. The combination of existing software analysis methods such as program slicing, and data analysis methods such as Bayesian networks, presents an opportunity for a beneficial outcome. Among the expected benefits are: incorporation of a higher degree of automation in a process that is currently largely labor intensive, development of a more effective methodology for the process of understanding unexpected outcomes, and development of a more effective tool for retrieving program statements relevant to unexpected outcomes.

This proposed research is needed. Simulation and computational modeling have become mainstays for exploration in a broad landscape of disciplines. For example, subject matter experts (SMEs) in biology, chemistry, physics and astronomy employ models and simulations of systems (Fortnow 2009), where a system is defined as the domain or process of interest to the SME. The National Science Foundation expects that the use of models and simulations in more disciplines will increase over time (NSF 2007). Uncertainty about system structure is inherent in the modeling and simulation of many systems. This uncertainty results in unexpected simulation outcomes. As the use of models and simulations increases, the need for an automated methodology for facilitating SME understanding and explanation of the unexpected outcomes is increasing. Further compelling arguments regarding the need and importance of this proposed research follow the presentation of relevant, currently accepted definitions.

The combination of models, simulations, uncertainty and exploration possesses an established terminology and structure that provides a framework for the proposed research. SMEs develop models by specifying logical and mathematical relationships to describe how a system works (Law and Kelton 2000). The computational evaluation of the model is a simulation of the system the model represents, and the simulation is the artifact that results from modeling systems (Law and Kelton 2000). SMEs construct two different types of models for systems: consolidative models and exploratory models (Bankes 1993). A consolidative model is the description of the logical and mathematical relationships within the system, where the relationships are accepted by SMEs in the field. The computational evaluation of a consolidative model is a consolidative simulation. The simulation consolidates the information about the system into a particularly useful form and can be used as a surrogate for the system itself (Bankes 1993). However, for other systems there exists uncertainty about some of the logical and mathematical relationships. The uncertainty about these relationships exists because experiments cannot be realized due to constraints imposed by the nature of these systems (Whipple 1996; Arthur 1999; Hooke 2000; Cha 2005; Elderd 2006; NSF 2006). As a result consolidative models cannot be constructed for these systems, instead SMEs form exploratory models by making estimations about the uncertain mathematical and logical relationships within the system (Bankes 1993). Constructing an exploratory model is a theory construction task where the model is the expression of the theory, and the simulation is the research artifact that allows the theory to be evaluated. Other simulations of the same system, any data from experiments, or other SME opinions in the field are used to test the theory. This gives exploratory simulations their experimental nature (Trenouth 1991).

Model requirements exist for both consolidative and exploratory models to identify and document, in native language, required model output values for model input values. These requirements are generally accepted in the field and established through experiments and observation of the system (Law and Kelton 2000; Ghezzi et al. 2003). Simulation outputs that are not defined by the model requirements and do not match simulations of the same system, data from experiments, or SME opinions are unexpected behaviors. Unexpected behaviors in consolidative simulations are rare because of the completeness of consolidative model requirements and the accepted SME knowledge in the field. However, due to the incomplete nature of exploratory model requirements and uncertainties regarding relationships within the system, unexpected behaviors are common in exploratory simulations. SMEs require the capability to understand and explain unexpected behaviors in exploratory simulations to determine if the behavior is due to an error or if it represents new knowledge about the system. If the behavior does represent new knowledge the model requirements are modified to reflect this established knowledge for the system. The process of understanding and explaining unexpected behaviors is further complicated in exploratory simulations because of the use of
continuous and discrete random variables to estimate the uncertain relationships in the system. The use of random variables in an exploratory simulation results in outputs that are random (Law and Kelton 2000). As a result exploratory simulations often exhibit unexpected behaviors that are not immediately repeatable.

An effective methodology for facilitating SME understanding and explanation of unexpected behaviors in exploratory simulations can affect millions of people and billions of dollars. Exploratory simulations are commissioned and employed by decision makers for policy analysis to answer questions where experimentation and observation is infeasible. For example, the inability of researchers to explain the results of Episims, a simulation of the worldwide spread of the smallpox virus under various vaccination strategies, has led to public policy debate (Eubank et al. 2004). The Institute of Medicine of the National Academies has published a collection of critical opinions of the predictions from Episims. The chief complaint is that the Episims simulation developers cannot provide a clear explanation for the difference between their predictions under the vaccination strategies and previously established estimates (Baciu et al. 2005). Methodology to facilitate the understanding and explanation of Episims’ behaviors would help to resolve this debate.

To address the need for explaining unexpected behaviors in exploratory simulations, I propose Bayesian Program Slicing (BPS), which combines program slicing and Bayesian networks to provide an effective tool for SMEs to explore unexpected behaviors in exploratory simulations. Program slicing is used in a variety of software analysis tools to extract statements that are relevant to a particular computation within the program. Bayesian networks are used to identify the structure of measurable variables in artificial intelligence, medical, economic and social science studies. The combination of Bayesian networks with program slicing is attractive because the analysis can be focused on identifying how variables interact to cause a phenomenon. This emphasis on discovery does not require an unexpected behavior in an exploratory simulation to be classified as correct or an error before Bayesian networks can be applied. Furthermore, because Bayesian networks are based in probability theory they are applicable to exploratory simulations which use random variables. These tools motivate my thesis statement: The purposeful combination of program slicing and Bayesian networks will result in a more effective methodology than existing tools and practices for facilitating SME understanding and explanation of unexpected behaviors in exploratory simulations.

BPS will demonstrate this thesis statement. BPS will employ program slicing and Bayesian networks in combination to identify how the program statements of an exploratory simulation change the states of variables within the simulation creating an unexpected behavior. Furthermore, BPS will focus SME attention on understanding and explaining the interactions of program statements whose execution results in variable state changes that are most relevant to the unexpected behavior. The goal of BPS is to enable users to gather insight into unexpected behaviors of exploratory simulation by understanding and explaining the interactions of the program statements that are most relevant to the unexpected behavior.

Two groups of SMEs employ exploratory simulations. The first is source code level SMEs who implement exploratory simulations in source code written in a high-level programming language. Accordingly, these SMEs are comfortable receiving insight into unexpected behaviors of exploratory simulations in terms of source code written in a high-level programming language. The second group of SMEs is graphical tool or library SMEs who implement exploratory simulations using graphical tools and libraries such as Simulink (Mathworks 1999), Java Modeling Tools (Bertoli et. al 2007), and VisSim (Planung Transport Verke 2005). These SMEs need insight into unexpected behaviors in exploratory simulations expressed in the same graphical tool or library in which the simulation is designed and implemented. BPS will be designed, implemented and evaluated for source code level SMEs. However, any decisions that could preclude BPS from being adapted for use by graphical tools or library SMEs will be weighed carefully. The effectiveness of BPS will be evaluated for the source code level SMEs through quantitative metrics and an observational study, which are discussed further in Section 4.

2. BACKGROUND AND RELATED WORK

BPS draws on the areas of program slicing, Bayesian networks and cluster analysis and it is related to work in the program understanding and fault localization communities. A review of this work follows.
2.1 PROGRAM SLICING

Program slicing is a decomposition technique that extracts statements relevant to a particular computation within the program (Weiser 1981). A program slice provides the answer to the question, “What program statements affect the computation of variable \( v \) at statement \( s \)?” (Binkley and Gallagher 1996) An important distinction is that between static and dynamic slices. Figure 1 (a) shows an example program that reads an integer input \( n \), and computes the sum and the average of the first \( n \) positive numbers. If the sum of the first \( n \) integers is evenly divisible by \( n \) the program assigns \(-1\) to \( x \). Otherwise the program assigns \( \text{sum} \) to \( x \). The criterion for a static slice is a 2-tuple consisting of \{line number of statement \( s \), the name of variable \( v \} \), where \( v \) is the variable of interest and \( s \) is the statement of interest. Figure 1 (b) shows a static slice of this program using criterion \{13, \( x \}\}. Slices are computed by identifying consecutive sets of transitively relevant statements, according to data and control flow dependences (Tip 1995). Only statically available information is used for computing slices; hence, this type of slice is referred to as a static slice.

```
read(n);
1
i := 1;
2
x := 0;
3
sum := 0;
4
average := 0;
5
while i <= n
6     sum := sum + i;
7     i := i + 1;
8     end
9
if (sum mod n == 0)
10     x := -1;
11
else
12     x := sum;
13
print(x);
14
average := sum/n;
15
print(average);
```

(a)

```
read(n);
1
i := 1;
2
x := 0;
3
sum := 0;
4
average := 0;
5
while i <= n
6     sum := sum + i;
7     i := i + 1;
8     end
9
if (sum mod n == 0)
10     x := -1;
11
else
12     x := sum;
13
print(x);
14
average := sum/n;
15
print(average);
```

(b)

```
read(n);
1
i := 1;
2
x := 0;
3
sum := 0;
4
average := 0;
5
while i <= n
6     sum := sum + i;
7     i := i + 1;
8     end
9
n = 4, 13, x
10
x := -1;
11
else
12     x := sum;
13
print(x);
14
average := sum/n;
15
print(average);
```

(c)

Figure 1: (a) An example program. (b) A static slice of the program using criterion \{13, \( x \}\}. (c) A dynamic slice of the program using criterion \{n = 4, 13, \( x \}\}.

In the case of dynamic program slicing, only the dependences that occur in a specific execution of the program are taken into account. A dynamic slicing criterion specifies the input; it consists of \{input, line number of statement \( s \), name of variable \( v \}\}. The difference between static and dynamic slicing is that dynamic slicing assumes fixed input for a program, whereas static slicing does not make assumptions regarding the input. Figure 1(c) shows a dynamic slice of the program in Figure 1(a) using the criterion \{n = 4, 13, \( x \}\}. Note that for input \( n = 4 \), the assignment \( x := \text{sum} \) is executed, and the assignment \( x := -1 \) is not executed. The “if (sum mod n == 0)” branch of statement 9, and statement 10 in Figure 1(a) is omitted from the dynamic slice because the assignment of \( x := -1 \) is not executed.

BPS uses static program slicing to identify those program statements which may affect an unexpected behavior in an exploratory simulation. This limits, sometimes significantly, the number of program statements and variable states that need to be considered in the BPS analysis (Atkinson and Griswold 2001). BPS also collects dynamic program slices to quantify and analyze the different executions that are possible for an exploratory simulation given a set of input configurations.

2.2 BAYESIAN NETWORKS

A Bayesian network is a graphical model that compactly represents the joint probability distribution of a set of variables in a directed acyclic graph (DAG). Formally, a Bayesian network is a pair \((G, \Theta)\), where: \( G \) is a DAG over the set of variables \( Z \), and \( \Theta \) is a set of Conditional Probability Tables (CPTs), such that a CPT
exists for each variable $z \in Z$. A Bayesian network is formed by the user supplying any prior knowledge about the independence relationships among the variables in $Z$. The user also supplies any temporal information that s/he possesses about the relationship among variables. Temporal information about two variables $p$ and $q$ means that a change in variable $p$ occurs in time before a change in variable $q$. The user supplied temporal information is used to orient edges in the DAG, $G$. The prior knowledge is then augmented with a CPT for each variable in $Z$. Given a variable $z \in Z$, the CPT for $z$ contains an entry for each value of $z$, for the values of the other variables $x \in Z$. This allows the Bayesian network to provide the probability of $x$ conditional on $z$, $P(x|z)$, for all values of $z$ for every instantiation of $x$ (Pearl 1989; Darwiche 2009).

Using the prior knowledge supplied by the user and the CPTs for each variable, existing inference algorithms can be employed to construct and orient the edges between vertices forming the DAG, $G$, of the Bayesian network. In the DAG, each vertex represents a variable $z \in Z$ and each edge represents a lack of conditional independence between the variables. Within the network the following condition, called the Markov Condition must hold, “each variable is conditionally independent of its non-descendants given its parent variables.” The following notation is introduced to formally describe the Markov Condition. Given a variable, $z$ in a DAG, $G$:

- Parents $(z)$ is the set of variables $N$ with an edge from $N$ to $z$.
- Descendants$(z)$ is the set of variables $M$ with a directed path from $z$ to $M$.
- Non-Descendants$(z)$ are all variables in DAG, $G$ other than $z$, Parents$(z)$ and Descendants$(z)$.
- $a \perp\!\!\!\!\perp B | C$ denotes that the variable $a$ is independent of the set of variables $B$ conditional on the set of variables $C$.

Given this notation the DAG, $G$ of the Bayesian network compactly represents the following independence statements: $z \perp\!\!\!\!\perp$ Non-descendants$(z)$ | Parents$(z)$ for all $z \in Z$ (Pearl 1989; Darwiche 2009).

The goal of BPS is to employ Bayesian networks to represent the interactions of variable states creating an unexpected behavior in an exploratory simulation. The Bayesian networks will be used by SMEs to understand the interactions within an exploratory simulation and explain why an unexpected behavior arises.

2.3 PROGRAM UNDERSTANDING

Understanding how a program feature is implemented is a major research area of program understanding, especially when maintaining or modifying an existing program. Gallagher and Lyle (Gallagher 1989; Gallagher and Lyle 1991) use static slicing for the decomposition of a program into a set of components capturing part of the original program’s behavior. Several other researchers have used dynamic slices to identify code in legacy systems that relate to a specified feature (Kang and Bierman 1996; Lakhotia and Deprez 1998; Wong et al. 2000; Wilde et al. 2001). Eisenbarth (Eisenbarth et al. 2001; Eisenbarth et al. 2003) has incorporated static and dynamic slices along with concept analysis to further automate this process.

Ernst's program understanding work, Daikon, is an approach that is closely related to BPS. The approach offers an alternative to requiring programmers to annotate code with invariants. Daikon executes a program on a collection of inputs and extracts variable values and then infers the invariants (Ernst et al. 2006). The invariants assist users in understanding and maintaining software. Recently, Brun and Ernst have extended the Daikon approach to finding latent code errors via machine learning over program executions (Brun and Ernst 2004). The technique generates machine learning models of program properties known to result from errors, and applies these models to program properties of user-written code to classify and rank properties that may lead to errors. Given a set of properties produced by the program analysis, the technique selects a subset of properties that are most likely to reveal an error.

2.4 FAULT LOCALIZATION

Work on automatic fault location closely matches BPS’ proposed goal of understanding and explaining unexpected behaviors in exploratory simulations. The goal of fault localization is to identify a fault or bug created by a programmer in source code. The fault creates an infection in the program state during execution, causing an externally observable error. Once the error is observed fault localization tools are applied (Zeller 2002). An initial solution to fault localization records the dynamic program slice from an execution that passes a test case and the dynamic program slice from an execution that fails the same test case. Program
statements within the dynamic program slice failing the test case but not in the slice passing the test case are presented to the user (Agrawal et al. 1993).

Jones et al. significantly improved fault localization solutions by offering a statistic to determine the likelihood that a program statement in a slice that fails a test case contains a fault (Jones et al. 2002). Reiniers offered a nearest neighbor algorithm to analyze the distance between statements in passing and failing executions to attempt to compute the same statistic more accurately (Reiniers and Reiss 2003). Delta Debugging is a different approach that isolates the causes of failing test cases by assessing outcomes of altered executions of the program to determine changes in the program state that create the difference in test case outcomes (Zeller 2002; Cleve and Zeller 2005). However, fault localization tools require users to distinguish between valid and invalid behavior and are only applicable to software that does not employ random variables. These restrictions are not suitable for exploratory simulations and unexpected behaviors.

2.5 CLUSTER ANALYSIS

The goal of clustering is to separate a finite unlabeled data set into a finite and discrete set of clusters (Xu and Wunsch 2005). The data within clusters is grouped according to whether a piece of data is similar to the other pieces of data in the cluster. Clustering is ubiquitous and a wealth of clustering algorithms and definitions of similarity have been developed to solve different problems in specific fields. However, there is no clustering algorithm or definition of similarity that can be universally employed to solve all problems (Xu and Wunsch 2005). Despite the lack of a universal clustering algorithm, several research efforts have successfully applied clustering algorithms to group program executions (Liu et al. 2007; Leon et al. 2007). BPS will use clustering analysis to group similar executions of exploratory simulations together as discussed further in Section 3.1.4.

3. PROPOSED RESEARCH

Four research objectives in support of realizing BPS are presented. These will demonstrate my thesis and constitute novel work in computer science:

1. The demonstration that static analysis tools can be used to identify prior knowledge about independence and temporal relationships among variable states within exploratory simulations.

2. The demonstration that the variable states and executable dynamic program slices within exploratory simulations can be efficiently sampled.

3. The demonstration that Bayesian networks revealing the structure of program statements creating an unexpected behavior can be generated for clusters of exploratory simulation executions using the data provided in research objectives (1) and (2).

4. The demonstration that the clusters of Bayesian networks provided in research objective (3) are more effective in facilitating user understanding and explanation of the unexpected simulation behaviors in exploratory simulations than existing tools and practices.

In the remainder of this section the proposed shape of BPS, a research plan to complete the development of BPS and the expected contributions of BPS are presented.

3.1 THE SHAPE OF BPS

The goal of BPS is to employ Bayesian networks to represent the interactions of variable states creating an unexpected behavior in an exploratory simulation. BPS begins with the SME identifying the unexpected behavior. Once the unexpected behavior has been identified, the BPS process will start with the application of existing static analysis tools to the exploratory simulation exhibiting unexpected behavior to provide prior knowledge about the independence and temporal relationships among variable states. Next, the exploratory simulation will be instrumented to collect the values of each variable state which can affect the unexpected behavior. Then, the exploratory simulations will be run for a set of SME supplied inputs to collect the data required to generate the CPT (conditional probability table) associated with each variable state. The executions resulting from the set of exploratory simulation runs will be grouped based on their similarity.
using cluster analysis. Finally, existing algorithms will be employed to generate a Bayesian network for each cluster of executions. The resulting Bayesian networks will identify, for each cluster of executions, how the interactions of variable states create the unexpected behavior. These Bayesian networks will be used by the SME to understand and explain the unexpected behavior in the exploratory simulation.

![Figure 2: A variation of the program in Figure 1(a).](image1.png)

![Figure 3: A static slice of Figure 2 with criterion \{13 ,x\}.](image2.png)

The program shown in Figure 2 will be used as an example throughout this section to motivate and describe each component of BPS. The program is a variation of the program shown in Figure 1(a) that uses random variables. The program takes an integer input n and computes the sum of n samples taken from a uniform random number distribution between 0 and n, where each sample is an integer. If the sum of the n random samples is evenly divisible by n then -1 is assigned to x. If the sum of the n random samples is not evenly divisible by n then sum is assigned to x. The value of x is printed. Finally, the average of the n random samples is computed and printed.

3.1.1 APPLYING STATIC ANALYSIS TO PROVIDE PRIOR KNOWLEDGE FOR BPS ANALYSIS

BPS will begin with the user identification of the unexpected behavior. The program statement in the source code of the exploratory simulation at which this state can be observed is identified by its line number, s. The variable storing the value of interest related to the unexpected behavior is identified by the variable, v. In the example in Figure 2, the unexpected behavior is the printed value of x in statement 13. This behavior is identified with \{13, x\}.

Next, static program slicing will be applied using the static slicing criterion \{s, v\}. The static program slice will identify all statements in the simulation’s source code containing variables that may influence the unexpected behavior. Figure 3 shows the static program slice of the program in Figure 2 given the static slicing criterion representing the unexpected behavior, \{13, x\}. Notice statements 5, 14 and 15 which compute the average of the n randomly drawn integers and are not relevant to the computation of x in statement 13 and thus have been removed. In preliminary work, static program slicing has been used successfully within BPS to conservatively focus the analysis on the variable states and program statements that affect the unexpected behavior within the exploratory simulation (Gore and Reynolds 2009a; Gore and Reynolds 2009b).

Next, static data dependence and flow analyses will be applied to the statements in the static program slice. The analyses will be used to provide prior knowledge about the independence and temporal relationships among variable states in the slice. This application of theses analyses is proposed work. The expectation is that static data dependence analysis will identify independence relationships among variable states and static data flow analysis will identify the temporal relationships among variables. The challenges associated with this work and directions for solution are discussed further in Section 3.2.1.
By employing these static analysis techniques the scope of the BPS analysis will be conservatively bounded affording more efficient, yet still complete analysis. Furthermore, the prior knowledge about the independence and temporal relationships of variable states will be supplied to BPS to help generate the Bayesian network of the interactions creating the unexpected behavior without any effort from the SME.

3.1.2 PREPROCESSING EXPLORATORY SIMULATION PROGRAM STATEMENTS

Following the application of static data dependence and flow analyses the statements in the static program slice will be preprocessed by a BPS preprocessor. The preprocessor will instrument the source code in the exploratory simulation to collect the value and address of each variable state in the static program slice when the exploratory simulation is executed. These variable state value samples will be used to form the CPTs required for the Bayesian network. Currently, the BPS preprocessor does not capture the values of all array component variable states and all variable states within loops (Gore and Reynolds 2009a). It also does not capture the addresses of the program statements in which these variable states appear. Instead, the preprocessor aggregates all components of an array in a single variable state sample and all iterations of a loop into a single variable state sample. Refining the BPS preprocessor in these capacities is future work. The challenges associated with this future work and directions for solution are discussed further in Section 3.2.

3.1.3 SAMPLING THE VARIABLE STATES AND DYNAMIC PROGRAM SLICES

Once the preprocessing step is complete, the SME will identify the set of input parameter configurations for which s/he is interested in gaining insight into the unexpected behavior. The exploratory simulation will then be executed for each configuration. The source code inserted by the BPS preprocessor will collect the samples for the value of each variable state and the address of each executed program statement which is included in the static program slice. The collection of addresses for each executed program statement will form a dynamic program slice. Each dynamic program slice will be associated with the set of samples that it generated.

Recall, for exploratory simulations which use random variables, there exist different possible outputs (or behaviors) for a fixed input (Law and Kelton 2000). Most software analysis tools for sequential programs assume that for each input there exists only one possible output and that for each input there exists only one possible dynamic program slice to be executed. The first assumption is never true for exploratory simulations that employ random variables and often the second assumption is not true either.

Figures 4(a) and 4(b) help elucidate the issues that can arise in analysis when these assumptions are made. Figures 4(a) and 4(b) show the two possible dynamic program slices using criterion \{n = 5, 13, x\} for the program in Figure 2. Figure 4(a) shows the dynamic program slice, when the sum of n random numbers is not evenly divisible by n. Figure 4(b) shows the slice when the sum is evenly divisible by n.

![Figure 4](image-url)

Figure 4: (a) One possible dynamic program slice of Figure 2 using criterion \{n=5, 13, x\}. (b) Another possible dynamic slice of Figure 2 using criterion \{n = 5, 13, x\}.
Analysis that only employs one dynamic program slice for the program in Figure 2 will not capture all the possible behaviors. Furthermore, even if only one of these dynamic program slices is possible different values for a single variable state are possible. In the program in Figure 2, given input n=5, the value of the variable state of x in statement 12 can be 0, 5, 10, 15, 20, or 25 if the dynamic program slice in Figure 4(b) is executed. Currently, BPS addresses these issues by performing Monte-Carlo sampling for each input configuration multiple times when the exploratory simulation exhibiting unexpected behavior uses random variables (Gore and Reynolds 2009b; Gore and Reynolds 2009c). However, this solution to sampling is inefficient, in terms of time and space. Developing a more efficient approach to sampling variable states and dynamic program slices of exploratory simulations is future work. This challenge along with several directions for solutions is described in Section 3.2.3.

3.1.4 APPLYING CLUSTER ANALYSIS TO GROUP THE SAMPLES

Following the BPS collection of a representative sample of dynamic program slices and variable states for the SME supplied set of input configurations for the exploratory simulation, cluster analysis will be applied to group similar dynamic program slices together. The application of cluster analysis addresses the following problem: many different dynamic program slices can be possible for a set of SME supplied input configurations. Each configuration can result in at least one different dynamic program slice. When the exploratory simulation uses random variables many dynamic program slices may be possible for each configuration. For some exploratory simulations over 10,000 different dynamic program slices are possible for a single SME supplied input configuration (Gore and Reynolds 2009c).

The current solution employed by BPS to address this problem is to generate a single Bayesian network for all the dynamic program slices collected for the set of SME supplied input configurations (Gore and Reynolds 2009a; Gore and Reynolds 2009b). The example program in Figure 2 and a SME supplied configuration set of only n=5 elucidate the shortcoming of this solution. Recall, for this example, the unexpected behavior is the value of x in statement 13. The single Bayesian network generated for this example is shown in Figure 5(a). The Bayesian network in Figure 5(a) aggregates the dynamic program slice and the associated variable state values shown in Figure 4(a) with the dynamic program slice and the associated variable state values shown in Figure 4(b). This is the shortcoming of the single Bayesian network solution; the SME cannot determine the interactions of the program statements which create the unexpected behavior because the different dynamic program slices are aggregated.

In this example the dynamic program slices should be grouped into two clusters: one cluster that corresponds to the dynamic program slice in Figure 4(a) and a second cluster that corresponds to the dynamic program slice in Figure 4(b). The Bayesian networks generated for each of these clusters are shown in 5(b) and 5(c) respectively. Here, the interactions of program statements creating the unexpected behavior are separated and evident to the SME. Identifying an existing clustering algorithm and developing a definition of similarity for exploratory simulation dynamic program slices that will enable clustering to be effectively applied within BPS is future work. This challenge and direction for solution is described in Section 3.2.4.

3.1.5 GENERATING BAYESIAN NETWORKS OF THE PROGRAM STATEMENTS

Once the cluster analysis within BPS has grouped the different dynamic program slices, a Bayesian network will be generated for each cluster. Each network will be generated from the prior knowledge identified by the static analysis tools and the samples of the variable states associated with the dynamic program slices in the cluster. CPTs will be formed for each unique variable state address from the samples associated with the dynamic program slices forming the cluster. The variable states will serve as the variables in the network.

Next, established algorithms will be employed to construct Bayesian networks describing the interactions among the variable states which cause the unexpected behavior. The result will be a chain of variable states describing how each variable state influences the other variable states, and the unexpected behavior. The proposed combination of Bayesian networks with program slicing enables the influence variable states have on one another and the unexpected behavior to be quantified.

The strength of the influence is measured as the absolute value of the correlation coefficient (between [0, 1] inclusive) between two variable states; given that the two variable states cannot be made conditionally
independent by conditioning on any other variable or set of variables within the Bayesian network. This condition is reflected through an edge between the two variables in the network (Spirtes et al. 2000).

Each variable state in the Bayesian network with an influence that is over a SME specified threshold will be mapped back to the program statement that changed the variable’s state value. Finally, a graph of the chain of program statements that have an influence on the unexpected behavior will be displayed to the SME. The graph is annotated with the influence each program statement has on the unexpected behavior or another program statement over threshold. The graph focuses SME attention on understanding those statements in the program’s source code with the strongest influence on the unexpected behavior. The idea that the interactions of some variable states are more important to the explanation of a behavior than others, is established within the fault localization community (Weiser 1981; Zeller 2002; Cleve and Zeller 2005). BPS has successfully generated Bayesian networks for dynamic program slices from exploratory simulations in a variety of fields (Gore and Reynolds 2008; Gore and Reynolds 2009a; Gore and Reynolds 2009b). However, exploring other quantitative definitions of influence to improve BPS’ effectiveness in facilitating SME understanding and explanation of unexpected behaviors is future work. This exploration is described in Section 3.2.5.

BPS will also specify to the SME how frequently each of the different Bayesian networks associated with the clusters is executed for the set of SME supplied input configurations. This allows the SME to understand how often a given Bayesian network reflects the interactions of the program statements creating the unexpected behavior. The two Bayesian networks associated with the two clusters formed from applying BPS to the example program in Figure 2 are shown in Figure 5(b) and 5(c). The frequency of each cluster of dynamic program slices is also shown. It is important to note that in such a small example different clusters are needed and the Bayesian network of each cluster has a very different structure and execution frequency.

<table>
<thead>
<tr>
<th>Executed 80% of the time</th>
<th>Executed 20% of the time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 read(n);</td>
<td>1 read(n);</td>
</tr>
<tr>
<td></td>
<td>1.000</td>
</tr>
<tr>
<td>6 while k&lt;n</td>
<td>6 while k&lt;n</td>
</tr>
<tr>
<td>7 sum := sum + rand(0,n);</td>
<td>7 sum := sum + rand(0,n);</td>
</tr>
<tr>
<td>9 end</td>
<td>9 end</td>
</tr>
<tr>
<td>10 if (sum mod n == 0)</td>
<td>10 if (sum mod n == 0)</td>
</tr>
<tr>
<td>11 x := -1;</td>
<td>11 x := -1;</td>
</tr>
<tr>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td>12 x := sum</td>
<td>12 x := sum</td>
</tr>
<tr>
<td>13 print(x)</td>
<td>13 print(x)</td>
</tr>
</tbody>
</table>

(a) (b) (c)

Figure 5: (a) A single Bayesian network for all the dynamic program slices in Figure 2. (b) The Bayesian network for the dynamic program slices in Figure 4(a). (c) The Bayesian network for the dynamic program slices in Figure 4(b). All three Bayesian networks (a-c) have a SME specified influence threshold of .5.

3.2 RESEARCH PLAN

The research plan for this proposed work consists of the sequence of tasks that must be completed in order to meet the research objectives presented at the beginning of this section.
1. **Develop an approach to applying static analysis tools to identify prior knowledge about temporal and independence relationships among variable states within exploratory simulations.** This prior knowledge must be in a form that can be used by Bayesian networks. **Challenges:** Two types of prior knowledge need to be supplied to the Bayesian network: information regarding the independence relationships among variable states and the temporal relationships among variables states.

   Static data dependence analysis is expected to provide prior knowledge about the independence relationships among variables states. The analysis identifies any variable state that may reach a candidate variable state (Ranganath 2006). The expectation is that for each candidate variable state in each statement remaining from the application of static program slicing, BPS will apply static data dependence analysis tools to determine the variable states in the remaining statements that cannot be reached and thus are independent of the candidate variable state.

   Static data flow analysis is expected to provide prior knowledge about the temporal relationships among variable states. Static data-flow analysis is a technique for gathering information about the possible set of values calculated within various variable states for a given program variable (Ranganath 2006; Chang and Jo 2002). Data-flow analyses often employ a control flow graph (CFG) for the program which is used to determine the variable states where a particular value assigned to a variable might propagate. The expectation is that BPS will apply static data flow analysis tools to identify the temporal order of the variable states which propagate values to the unexpected behavior.

2. **Refine the BPS preprocessor to capture the values of array component variable states and variable states within loops.** Also capture the addresses of the program statements in which these variable states appear. **Challenges:** Due to the current implementation of BPS, which only generates a single Bayesian network for the set of SME supplied input configurations, the BPS preprocessor does not collect these variable states and addresses. The future application of cluster analysis in BPS is expected to alleviate this restriction. A refined BPS preprocessor will use a program tracing utility, such as ptrace (Padala 2002a; Padala 2002b) or the Java Debugging Interface (JDI 2010), to capture all variable states values and the addresses of all executed program statements that affect the unexpected behavior.

3. **Develop an approach to efficiently sample variable states and dynamic program slices within an exploratory simulation; efficiency is measured in terms of space and time.** **Challenges:** BPS requires a representative set of samples of variable states and dynamic program slices to generate Bayesian networks of the interactions of variables states within an exploratory simulation. For exploratory simulations which employ random variables each input configuration needs to be sampled multiple times. Applying a straightforward approach of repeatedly executing the simulation and recording the variable states and dynamic program slice for each input configuration will require an overwhelming amount of space and time. Exploring symbolic execution algorithms used within the model checking community to improve the time efficiency of generating these samples is promising (Clarke et al. 2000). Also compression algorithms will be explored to improve the spatial efficiency of storing the dynamic program slice executions that result from an exploratory simulation (Wang and Roychoudhury 2004).

4. **Develop a definition of similarity and identify an existing clustering algorithm to effectively group exploratory simulation dynamic program slices.** **Challenges:** Actual exploratory simulation dynamic program slices will not be as simple to cluster as the dynamic program slices of the program in Figure 2. Identifying the characteristics of dynamic program slices that will be included in the definition of similarity required by clustering algorithms will be challenging. Furthermore, there is not a unified clustering framework that works for all problems. The expectation underlying the application of cluster analysis is that many of the dynamic program slices share similar characteristics, and those characteristics can be identified and categorized in a small number of groups. This expectation has been shown to be true of dynamic program slices within software employed in a variety of domains (Liu et al. 2007; Leon et al. 2007). The exploration for an effective clustering algorithm will begin with algorithms used in these research efforts.

5. **Explore alternative quantitative definitions of influence to improve BPS’ effectiveness in facilitating SME understanding and explanation of unexpected behaviors in exploratory simulations.** **Challenges:** While the correlation of the values of one variable state on the values of another variable state is currently used in BPS to measure influence, it is not necessarily the definition of influence that will result in the most
effective BPS analysis for SMEs tasked with understanding and explaining unexpected behaviors in exploratory simulations. Several different quantitative metrics have been developed in the fault localization community to identify program statements whose executions are most likely to cause a program run to fail a test case (Zeller 2002; Cleve and Zeller 2005; Renieris and Reiss 2003; Jones et al. 2002). Each of these established definitions will be explored to identify the definition of influence that offers the most effective BPS analysis for facilitating SME understanding and explanation of unexpected behaviors.

6. Conduct the evaluation of BPS. Challenges: The challenges of the evaluation are discussed in Section 4.

3.3 Expected Contributions

1. The quantification of the probability distribution of dynamic program slices in exploratory simulations that employ random variables. Currently, there is a lack of software analysis methodology for programs that use random variables. This is due, in part, to the uncertainty associated with the dynamic program slice that is executed for a given input, for this class of software. BPS will address this issue enabling research in software analysis for programs using random variables.

2. The development of a more effective tool for facilitating SME understanding and explanation of unexpected behaviors in exploratory simulations than currently exists. The effectiveness of BPS for this contribution will be evaluated through the observational study component of the evaluation in Section 4.1.

3. The development of a more effective tool for extracting statements that enable an explanation of an unexpected behavior in an exploratory simulation than currently exists. The effectiveness of BPS for this contribution will be evaluated through the quantitative metrics component of the evaluation in Section 4.2.

4. Evaluation

The success of BPS in addressing the thesis and objectives of the proposed research will be evaluated. The two components of the evaluation are: (1) the observational study and (2) quantitative metrics. The observational study is described in Section 4.1 and the quantitative metrics are described in Section 4.2. The goal of the evaluation is to demonstrate that BPS warrants further exploration as a tool for facilitating source code level SME understanding of unexpected behaviors in exploratory simulations.

4.1 Observational Study

BPS will be evaluated through an observational study crafted after observational studies in (Murphy et al. 1999, Walker et al. 1999; Williams et al. 2000; Murphy et al. 2001; Hughes 2002; Sfetsos et al. 2006; Falke et al. 2008; Salinger and Prechelt 2008; Cleary et al. 2009). The objectives of the proposed study are to: (1) determine how effectively BPS facilitates source code level SMEs to understand and explain unexpected behaviors in exploratory simulations, as compared to performing the same task using an established method, in this case the Eclipse Debugger and (2) identify challenges facing the effectiveness of BPS.

Planning the Study: The observational study will be constrained by four factors: (1) the pool of potential participants available will be small, (2) the amount of time each participant can devote to the study will be short – especially in comparison to the time spent understanding and explaining unexpected behaviors in an exploratory simulation, (3) the cost in man hours of running the study and analyzing the results will be high and (4) because the evaluation of participant understanding of an unexpected behavior in an exploratory simulation is complex, some precision of measurement will be forfeited (McGrath 1995). Due to these constraints this study will not be a controlled statistically valid experiment. Instead it is an observational study modeled after the “semi-controlled experiments” in (Murphy et al. 1999; Walker et al. 1999; Murphy et al. 2001). The study will take advantage of the lessons learned in (Murphy et al. 1999) and other guidelines in empirical software engineering (Guindon 1987; Kitchenham 1995; Kitchenham 2002; Kitchenham 2008) to achieve sufficient validity to be accepted by the simulation and software engineering communities.

Participants in the observational study will be 12-16 senior level undergraduate students, graduate students and faculty researchers working with exploratory simulations at The University of Virginia. At least two participants will be faculty researchers. Previous observational studies in empirical software engineering studies have successfully recruited this size and demographic of participants for 90-180 minute studies (Murphy et al. 1999; Walker et al. 1999). The inclusion of food and gift certificates ($25-$50 value) will
encourage participation. The objects of the study are the Eclipse Debugger tool and BPS tool. Each tool will be integrated into the Eclipse Integrated Development Environment (IDE) (Eclipse 2010). The observational study will consist of a series of sessions in which the participants use either the Eclipse Debugger or BPS to facilitate understanding and explanation of an unexpected behavior in an exploratory simulation. Each session is modeled after a session within the “semi-controlled debugging experiment” in (Murphy et al. 1999; Walker et al. 1999). An administrator (me) will be present during each session and available to answer questions about the Eclipse IDE and usability issues with both tools. This role of the administrator is consistent with that applied by (Murphy et al. 1999; Walker et al. 1999; Weick 1968; Guindon et al. 1987).

Each session will be audiotaped, videotaped and the mouse activity, keystrokes and other activity within the Eclipse IDE will be captured via the Morae tool (Morae 2010). Participants will be asked to think aloud while attempting to understand and explain an unexpected behavior in the simulation. The purpose of the think aloud protocol is to make explicit the implicit cognitive process of explaining and understanding the behavior. The think aloud transcript of each participant session will be transcribed and classified using the categories defined in the published coding scheme, A Flexible Expandable Coding Scheme (AFECS) (von Mayrhouser and Lang 1999). AFECS is designed to be flexible, expandable, reduce the effort required to perform protocol analysis and enable comparable results. These properties allow the scheme to be tailored to a variety of observational software studies, while maintaining a degree of consistency between studies. Each code within AFECS explicitly identifies each participant action (von Mayrhouser and Lang 1999). Many of the measures in the observational study will be classified actions from the participant think aloud transcripts.

At the end of the session the administrator will perform a semi-structured interview with the participants. The semi-structured interview will also be recorded. The interview will include questions about how and why the unexpected behavior occurred and the effectiveness of the tool used by the participant. Closing the session with the semi-structured interview to receive participant feedback is recommended (Adams and Cox 2008).

Many pilot sessions will be performed prior to any actual sessions in the study. The goal of the pilot sessions is to refine any usability or data collection issues that are presented. These sessions will ensure fewer usability issues occur during the actual sessions where data is collected and analyzed in the study. The pilot sessions will also ensure the administrator provides a consistent and appropriate level of assistance during the actual sessions. The use of pilot sessions is recommended (Adams and Cox 2008; Murphy et al. 1999).

Prior to the sessions the administrator will form a forced choice questionnaire and identify correct answers regarding how and why the unexpected behavior occurs in the simulation. Questions in the questionnaire will directly reference source code within the simulation and will be formed from publications accompanying the simulations. During the session as the participant feels s/he understands and can adequately explain parts of the unexpected behavior in a simulation s/he will answer the forced choice questions. Once the participant completes the questionnaire the administrator will mark the end time of the session. The questionnaire will be scored by another administrator (not me) for correctness.

Validity: The validity of the observational study will be evaluated according to the established four tests: construct validity, internal validity, external validity and reliability validity (Yin 1994). The primary concerns of the observational study are construct and internal validity rather than with external validity and reliability validity. The focus on construct and internal validity is meant to ensure the results will be meaningful to the overall question of interest: does BPS show promise of easing the task of facilitating source code level SME explanation and understanding of unexpected behavior. If the analysis reveals that participants using BPS take longer to understand and explain the unexpected behavior, provide a correct explanation less often, or experience other difficulties during the studies, their difficulty should not be blamed on the programming environment, usability issues or the administrator; it should be blamed on BPS. Each of these levels of validity for the study will meet at least the same level as (Murphy et al. 1999; Walker et al. 1999; Murphy et al. 2001) and match the desired level for exploratory studies in (Kitchenham 1995; Kitchenham 2002).

The Hypothesis and Evaluation Criteria: The hypothesis of the observational study is: BPS will better facilitate participant understanding and explanation of an unexpected behavior in an exploratory simulation than the Eclipse Debugger. The measures collected during the sessions in the observational study will be analyzed by the following evaluation criteria as evidence to support or refute the hypothesis. For each
evaluation criterion the min, max, median, mean and standard deviation for all the participants assigned a
given tool will be calculated. Analysis of the AF ECS encoded think aloud transcripts will yield criteria 1-4.
These criteria are similar to evaluation criteria used in empirical software engineering, program
comprehension and human computer interaction studies (von Mayrhauser and Vans 1996; von Mayrhauser
Analysis of the forced choice questionnaire will yield criteria 5 and 6. These criteria evaluate the efficiency
and correctness of the participant’s explanation of the unexpected behavior. The criteria are similar to the
number of faults corrected by a participant and the time required to correct each fault used in Walker et al.’s
evaluation of the utility of aspect oriented programming (Murphy et al. 1999; Walker et al. 1999).

1. **Hypothesis Resolution Ratio** (HRR) - The number of hypotheses confirmed or determined to fail by a
participan divided by the number of hypotheses abandoned by a participant in a given session will be
calculated. The tool with the higher HRR will better facilitate participant understanding of unexpected
behavior. The rationale behind this criterion is that the tool with a higher HRR enables the participant to
come to a fulfilling resolution of a hypothesis more frequently.

2. **Hypothesis Type Ratio** (HTR) – The number of hypotheses that are of the type *why* divided by the number
of hypotheses of type *what* will be calculated. The tool with a higher HTR will better facilitate participant
understanding of unexpected behavior. The rationale behind this criterion is that the tool with a higher
HTR enables users to ask more questions about *why* certain effects in the simulation are manifested and
fewer questions about *what* effects in the simulation can be manifested.

3. **Information Manipulation Time** (IMT) - The time spent by a participant manipulating different
information sources (searching, reading, reviewing and editing code and reading comments) will be
calculated. The tool with a lower IMT will better facilitate participant understanding of unexpected
behavior. The rationale behind this criterion is that the tool with a lower IMT enables the participant more
time to set goals and test hypotheses by limiting the time required to manipulate information.

4. **Goal To Change of Direction Ratio** (GCDR) – The number of goals generated by the participant during
the session divided by the number of times the participant changed direction will be calculated. The tool
with the higher GCDR will better facilitate participant understanding of unexpected behavior. The
rationale behind this criterion is that the tool with the higher GCDR enables the participant to set
achievable goals more frequently.

5. **Forced Choice Questionnaire Score** (FCQS) – The number of questions unanswered or answered
incorrectly by the participant will be calculated. A score of zero means each forced choice question was
answered correctly. The tool with the lower FCQS will better facilitate participant understanding of unexpected
behavior. The rationale behind this criterion is that the tool with a lower FCQS has enabled the
participants to answer more questions regarding the unexpected behavior correctly.

6. **Forced Choice Questionnaire Time** (FCQT) – The time required by the participant to complete the forced
choice questionnaire will be calculated. The tool with the lower FCQT will better facilitate participant
understanding of unexpected behavior. The rationale behind this criterion is that the tool with a lower
FCQT has enabled participants to answer questions regarding the unexpected behavior faster.

7. **Excerpts** from the semi-structured interviews at the end of the session will be collected in the analysis to
corroborate or contradict the calculated evaluation criteria described in 1-6. This approach to triangulating
evaluation criteria to strengthen results is recommended (Walker et al. 2003; Adams and Cox 2008).

It is expected that measures and evaluation criteria that are currently undefined, ultimately will be
collected and analyzed in the observational study. This expectation is held “because it is not at all evident how
to determine an analysis strategy especially when conducting exploratory studies in which some analysis is
required to determine appropriate observations that drive further analysis (Murphy et al. 1999).” The record of
participant sessions and the use of an established flexible coding scheme designed for a variety of
observational software studies will enable measures to be collected that currently do not seem to be of interest
but ultimately may need to be analyzed.

The hypothesis of the observational study will be supported with sufficient evidence to consider BPS
successful if: the median score of BPS participants for each evaluation criterion better facilitates participant
understanding than the median score of the Eclipse Debugger participants for the respective criterion. There is the possibility that BPS will not better facilitate participant understanding for each evaluation criterion. If this occurs the results of the observational study will be taken back to the committee and discussed. The think aloud transcripts, semi structured interviews and questionnaires will be explored to determine: (1) how and why this occurred, (2) the impact on the perception of BPS’ utility and (3) how to address these issues moving forward with BPS. If the committee decides that this exploration shows the issues facing the utility of BPS can be addressed in a straightforward manner, such as reinterpreting the collected data, then the observational study will be evidence of a positive technical result and BPS will be considered successful once the issues are addressed. This position is consistent with the exploratory observational studies the evaluation is modeled after. In these studies the intent in evaluation is not to categorically determine whether the new approach can or can not meet all of its claims, “but rather to explore whether the approach is useful, and which parts of the approach might help or hinder various parts” of the process being studied (Murphy et al. 1999). However, if the committee decides the issues facing the utility of BPS cannot be addressed in a straightforward manner then additional work and evaluation will need to be performed before sufficient evidence of a positive technical result exists.

**Candidate Simulations and Unexpected Behaviors:** One simulation and unexpected behavior will be used in the study. The following example candidates capture the essence of the kind of simulation and unexpected behavior being considered for use in the observational study.

The first candidate simulation is the SEIR Dunham simulation that predicts disease spread by modeling interactions on a two dimensional torus (Dunham 2005). The predictions of the SEIR Dunham simulation match the results of an established SEIR differential equation simulation for a variety of parameterizations (Li and Muldowney 1995). However, the predictions of the SEIR Dunham simulation do not match the predictions of the established SEIR differential equation simulation for a population of size 1000. The variance between the predictions from the two simulations for each of the four SEIR disease stages (Susceptible, Exposed, Infected, Removed) with population of size 1000 is the unexpected behavior participants will be asked to explain in the observational study. The candidate has the following attributes: (1) it is a new exploratory simulation that produces predictions that match the predictions of an established exploratory simulation for some inputs, (2) for other inputs the simulation predictions do not match the predictions of the established simulation and (3) the simulation is instrumented with an output(s) that displays the disparity or agreement of the predictions from itself and the established simulation. Since the administrators of the study will have previously explored and understand how and why the prediction from the new simulation differs from the established simulation, extra attention will be given to the preparation of the materials to ensure that they do not bias the participants’ exploration of the unexpected behavior. Also, the pilot sessions and use of published guidelines will ensure the administrator does not bias the participant. The ability to compile, execute and view the source code of the established simulation will also be available.

The second candidate simulation is the self-driven particle simulation (Phet 2010). Here, particles move on a two dimensional torus at a constant speed. At each time step in the simulation, the orientation of each particle is set to be the average orientation of all particles within a radius plus a random term. Under some parameterizations particles form clusters as they follow the given set of rules. This is a complex behavior given the set of simple rules and thus unexpected. The simulation will be instrumented to capture and output the median number of particles in a cluster. Participants in the study will be asked to explain how and why the particles cluster in the simulation. The candidate has the following attributes: (1) it is an exploratory simulation that for an input(s) produces a complex behavior based on a simple set of rules and (2) it is instrumented to output the computation(s) reflecting the complex behavior.

**4.2 QUANITATIVE METRICS**

The goal of the quantitative metrics is to measure and compare, through established criteria, the quantitative features of the set of program statements returned by automated tools that facilitate user understanding of an unexpected behavior in a simulation. Fault localization tools and BPS return a set of program statements that facilitates user understanding of an unexpected behavior. The experiment will manipulate one independent
variable: the tool that returns the set of statements to the user. BPS will be a tool; the fault localization tools that will be included are described in (Jones and Harrold 2005). These tools are a representative sample of the state of the art in fault localization. Each tool will be applied to several published simulations, for an input exhibiting the unexpected behavior, including a discrete-event, equation-based and agent-based simulation.

**Measures of Effectiveness:** Each set of program statements returned by a tool will be evaluated through four measures of effectiveness: efficiency, succinctness, cumulative relevance and expected relevance.

1. **Efficiency** will be measured by the wall clock time required by the tool to return the set of statements.

2. **Succinctness** will be measured by: $\frac{\text{# of returned statements}}{\text{# of statements in simulation}}$. This percentage represents the portion of statements in the program that needs to be examined by the user.

3. **Cumulative Relevance** - In a program $x$, a program statement $s$ is relevant to the variable $r$ in source code line number $l$, if $s$ lies on the path, according to data and control flow dependences, to the computation of variable $r$ in line number $l$. The computation of variable $r$ in line number $l$ is referred to as variable state $r_l$. For a program $x$, that employs random variables, and a specified input $i$, it is possible that for some executions $x(i)$, $s$ will be relevant to $r_l$ and for other executions $x(i)$, $s$ will not be relevant to $r_l$. Based on this observation a continuous measure of the relevance of a program statement $s$ to variable state $r_l$ can be proposed. Each execution, $x(i)$, can be considered a Bernoulli trial $Y$ such that: if $s$ is relevant to $r_l$ in the execution $x(i)$, $Y=1$ and if $s$ is not relevant to $r_l$, $Y=0$. Thus for each execution $x(i)$, $s$ is relevant to $r_l$ with probability $p$. It is important to note that probability $p$ is influenced by the input $i$. In other words, the choice of $i$ influences the probability $p$ that $s$ is relevant to $r_l$ for a given execution, $x(i)$. For a set of $N$ executions an unbiased estimator, $\hat{p}$, of the probability that statement $s$ is relevant to variable state $r_l$ for program $x$ with input $i$ is $\frac{\# \text{ of } x(i) \text{ where } s \text{ is relevant to } r_l}{N}$. Let $S$ be the set of statements returned by the tool, $s_j$ be a statement in $S$ and $\hat{p}_j$ be the estimate of the probability that $s_j$ is relevant to the variable state representing the unexpected behavior. The cumulative relevance of $S$ is the sum of the $\hat{p}_j$ of all $s_j \in S$; or symbolically: $\sum_{j=1}^{|S|} \hat{p}_j$.

4. **Expected Relevance** - The expected relevance of $S$ is the sum of the $\hat{p}_j$ of all $s_j \in S$ divided by the cardinality of $S$; or symbolically: $\sum_{j=1}^{|S|} \hat{p}_j / |S|$.

**Measures of Success:** BPS will be successful if it is included on the Pareto-frontier for each of the simulations as or more frequently than each of the other tools (Chavas 2004).

**5. CONCLUSION**

Every day public policy debates surrounding exploratory simulations such as Episims (Cha 2005) hinge on the ability to understand and explain unexpected behaviors. How can policy makers make informed decisions involving billions of dollars and millions of people in confidence when poorly understood behaviors in exploratory simulations are pervasive? Methodology to improve SME understanding and explanation of unexpected behaviors in exploratory simulations is needed; this motivates BPS.

BPS will automate the process of understanding unexpected behaviors with a methodology to construct Bayesian networks of clusters of exploratory simulation executions. The Bayesian networks will gather the insight that enables a SME explanation of an unexpected behavior. The evaluation plan will measure the effectiveness of this insight to SMEs and BPS’ ability to retrieve relevant program statements. All together the proposed research will provide SMEs with an improved methodology to facilitate the understanding and explanation of unexpected behaviors in exploratory simulations.
6. REFERENCES


