Calibrating Probabilistic GUI Testing Models
Based on Experiments and Survival Analysis

Cristiano Bertolini∗†, Alexandre Mota∗ and Eduardo Aranha‡

∗Informatics Center - Federal University of Pernambuco
P.O. Box 7851, 50732-970 - Recife-PE, Brazil
Email: {cbertolini,acm}@cin.ufpe.br

†International Institute for Software Technology - United Nations University
P.O. Box 3058, Macao SAR, China
Email: cbertolini@iist.unu.edu

‡Informatics and Applied Mathematics Department - Federal University of Rio Grande do Norte
P.O. Box 1524, 59072-970 Natal-RN, Brasil
Email: eduardouranha@dimap.ufrn.br

Abstract—Models abstract reality. Although abstract, such models can capture the essence of real world phenomena as long as they are sufficiently accurate. The development of new techniques or its usage in a different environment cannot always be satisfactory and can be expensive. In this paper we propose a novel strategy to create GUI probabilistic testing models and calibrating them with the aid of real experiments based on survival analysis. Survival analysis is used to transform exact responses of the real experiment into probabilistic predictions, comparable to the responses obtained from our testing models. Thus calibration means searching for model parameter instances that yield model predictions almost equal to survival analysis predictions. Using our strategy, we improved the accuracy of our models showing that the models has a result closely with a real experiment.

Keywords—Testing; Survival Analysis; Simulation; GUI Testing.

I. INTRODUCTION

Performing experiments on concrete systems is the best choice to obtain high accurate results [1]. However, experiments on real systems can sometimes take a (almost) prohibitive time to be accomplished completely. For instance, when humans are involved in exploratory testing or to compare testing techniques. In these circumstances, working with models instead of real systems is desirable [2]. The major drawback of working with models instead of real systems is related to how good the results are provided by the model with respect to the real system. That is, how accurate is the model. An additional difficulty occurs when the model needs certain knowledge about the real system. In this way, humans provide information based on (limited) observable (visible or external) information about the system behavior [2].

The process of adjusting a model to obtain as close results as possible with respect to a real system is known as calibration [3]. The purpose of calibration is to determine the values of model parameters (configuration) according to observations from experiments performed with the real system. We have exact responses of the real experiment and probabilistic predictions from the model. Thus, we use survival analysis to transform the exact responses (from the real experiment) into probabilistic predictions to be able to calibrate the model.

In previous work [4] we propose a strategy that consists in modeling two GUI testing techniques as well as a GUI-based system using a Markov-based language (PRISM) [5], [6] (Section II). These techniques, named DH and BxT, come from a partnership between Motorola Inc. and the Informatics Center (called CIn-BTC). The GUI-based system is a cellular phone. Motorola created and uses DH and we created BxT, as a possible improvement of DH. Our goal is to determine if it is possible to replace DH with BxT (if we can show that BxT is more efficient and effective than DH) and when (if we can replace in the current testing stage of the development or in another stage). Moreover, to overcome some drawbacks pointed by [1], concerning the use of models instead of real systems, we modeled the GUI-based system following an also proposed framework which is compositional and parameterized. The compositional aspect is employed to ease adapting the framework to different contexts (GUI-based systems beyond cellular phones) [7], [8]. The parameters are used to instantiate the model to specific characteristics based on test hypothesis obtained from the application domain. However, the work reported in [4] instantiates some parameters based only on user intuition (limited information).

In this paper we address this problem by showing how to calibrate the parameters of these models using a proposed calibration strategy (Section III). We start by performing a design of experiments [9] to determine the influence of the several parameters considered and collect data about the system. As the data collected (exact responses of the real experiment) do not correspond to the probabilistic predic-
tions obtained by the PRISM model, we use survival analysis to make them compatible. Calibration aims at searching for the best model parameter instances that produce model predictions almost equal to the survival analysis predictions (obtained from the instances that produce the least possible standard error and satisfy a criteria stated in terms of the ANOVA [9]). We show evidences of our strategy (Section IV) using the same material of [4] and show some threats to validity in Section IV-E. We also discuss about the limitations, applicability and costs associated with our strategy (Section V). Thus, we compare our work to others in Section VI and present our conclusions and future work in Section VII.

The main contributions of this paper are:

- A strategy to calibrate probabilistic models;
- To show how to combine experimental software engineering and formal models;
- The evaluation of some GUI testing techniques in order to increase the confidence of our strategy.

II. FORMAL MODELS

In this section we consider a top-down presentation of our GUI-based model. In the PRISM language [5], we represent a system as a special and unique module system ...endsystem, which allows parallel combinations of defined modules. For instance, the following PRISM system module captures the interaction between a GUI-based system (which is composed of a HANDLER and an ORACLE) and the testing technique BxT. We define the system module as follows:

```prism
module GUI
state : [0..1];
sControl : [1..pressKey] init 1;
[getEvents] state<=1 -> 1:(state '=0);
[pressEvent] state=0 ->(1-event_fire):(state '=0)+
event_fire:(state '=1);
[failure] state=1 & sControl<pressKey =>
(1-event_fire):(state '=0)&
(sControl'=sControl+1)+
event_fire:(state '=0)&
(sControl'=sControl+1);
endmodule
```

Figure 1. GUI Model.

Figure 2 shows the HANDLER module. It is responsible to determine, for each possible event (pressEvent), the probability to exercise the handler and then put the system in a crash state. The variable handlerFail means the reliability of the handler and we are not sure about its precise value. Determining its best value is the main motivation of the material of Section IV.

```prism
module HANDLER
h : [0..1] init 0;
[pressEvent] h=0 -> (1-handlerFail):(h'=0) +
(handlerFail):(h'=1);
endmodule
```

Figure 2. HANDLER Module.

Figure 3 shows the ORACLE module, which is used to determine if a crash has occurred in the system due to a state change. The ORACLE, presented in Figure 3, represents a Boolean function where 0 means no crash found and 1 means crash found. The synchronized event failure is responsible to synchronize the modules GUI and HANDLER with ORACLE.
Our two GUI testing techniques consist in generating (a sequence of keys) and executing (pressing a key on a cellular phone application) test sequences. This kind of technique can be easily generalized to other kinds of applications based on GUIs. They use the following inputs: a predetermined sequence of screens (created manually), a random seed to use in the event generator, a global timeout, and a local timeout (numRept—the number of events to be pressed from a chosen screen). The detailed algorithms can be found in [11].

Both techniques, BxT and DH, work by choosing some valid screen randomly, pressing a certain number of keys and repeating this process on another screen until a total number of keys is pressed. Thus, we define two control variables: local_key that keeps the number of pressed keys from each initial screen and global_key that holds the total number of pressed keys the technique has pressed (the sum of all local_key). The variable local_reset determines when a test sequence is completed and a new one must be started from a new screen (that is, when local_key equals the maximum number of pressed keys (TechPressKey)). Finally, this process repeats until global_key equals presskey (the maximum number of total pressed keys exercised by the testing technique).

Figure 4 shows the DH and BxT modules in PRISM. Basically, DH can press any key (enabled or not). The main difference from DH is the ability of BxT to press only those keys that are enabled in a given screen, maximizing its change of changing the system state. It uses a special event called getEvents that returns all the enabled events in the current screen.

III. MODEL CALIBRATION STRATEGY

In a modeling process it is important to observe that the response variable (the output of the model or experiment) is dependent on the values used in its various input parameters. In many cases, these input parameters assume values based on intuition of experts. In these circumstances, such models are calibrated in an ad hoc way. Following Heuberger [12] the calibration process is one of the most critical phases of the whole modeling process. The calibration process tries to overcome two main problems: (1) parameter uncertainty and (2) measurement error. In our proposed calibration strategy we also intend to solve both problems.

To introduce our strategy, suppose a real experiment $E_{\text{real}}(in, out, err)$ where the parameter $in$ means a set of inputs, $out$ is a response variable and $err$ is the error resulting from noise in the experiments. The real experiment can be simulated by a model-based experiment $E_{\text{prob}}(in', out', err')$ where the parameter $in'$ corresponds to a set of input parameters, such that $in' \subseteq in$ (where $\subseteq_A$ is the usual subset relation except that it relates abstract (in') with concrete values (in)), $out'$ is a probability-based response variable which quantifies the possible occurrences of $out$ and $err'$ is the error inherited from numerical methods used to solve stochastic models (for instance, Jacobi and Gauss-Seidel methods).

Figure 5 presents an overview of our strategy. It consists in four steps (phases), described as follows:

- **Step 1 (Collect data):** it consists in performing some experiments using the real system. We observe how the response variable (out) is influenced by the various input parameters (in). In this step, controlled experiments are suggested to be applied [13]. In this case we obtain a set of (real) data that is used, for example, to study the correlation among parameters, compute mean, mean and standard deviation, and also to infer observations about the system. It is important to observe that the parameter $err$ is associated with the level of control of the experiment and the number or samples used. The result of this step is the real experiment $E_{\text{real}}(in, out, err)$ itself.

- **Step 2 (Create a Markov-based model):** it consists in creating a Markov-based model and experiment $E_{\text{prob}}(in', out', err')$ able to represent the real experiment $E_{\text{real}}(in, out, err)$. The key idea is to create several modules to represent each part of

```plaintext
module ORACLE
  crash : [0..1] init 0;
  [failure] crash=0 ->
  (l=failureRateFunction):(crash'=0) +
  (l=FailureRateFunction):(crash'=1);
  [] crash=1 -> 1:(crash'=1);
endmodule
Figure 3. ORACLE Module.
```

```plaintext
module DH
  local_key : [0..TechPressKey] init 0;
  global_key : [0..pressKey] init 0;
  local_reset : bool init true;
  [pressEvent] local_key<TechPressKey &
  global_key<pressKey ->
  1:(local_key=local_key+1)&
  (global_key'=global_key+1);
  [] local_key=TechPressKey &
  global_key <pressKey-TechPressKey) ->
  1:(local_key'=0) & (local_reset= trái true);
endmodule
Figure 4. DH and BxT Technique Modules.
```
$E_{real}(in, out, err)$, similarly to what was shown for the GUI-based model in Section II. In this model we use known values for (a subset of) $in'$ and approximate values for those elements in $in'$ whose values are uncertain, according to expert opinions.

- **Step 3 (Perform a survival analysis):** in many cases the response variable of the real experiment is not directly compatible with the one provided by the formal model. Because in general the response variable of the real experiment is an exact value (for example, a time instance) and the response variable of the formal model is an estimate of this exact value (a probability). When $out$ is a temporal variable (it assumes time values) and $out'$ is a probability variable (it assumes probabilities), we can use a statistical analysis named survival analysis [14], which can be used to map elements of $out$ (response variable of $E_{real}(in, out, err)$) into probabilities, creating a new experiment $E_{real}(in, out_{prob}, err)$. After this transformation, the response variables (of the real experiment and the model) become comparable. It is worth noting, however, that survival analysis gives a probability about survivability, which is in general a complement of the probability obtained with models. Thus to compare then the results of survival analysis must be previously complemented, creating a new set of responses such that $out_{comp} = 100\% - out_{prob}$. To apply survival analysis, we first need to set the censored data (some values of $in$) and then perform the analysis (for instance, using a statistical tool like R [15]).

- **Step 4 (Calibration):** it consists in searching for the best values of the various input parameters ($in'$) of the model, whose exact value is uncertain and we only have expert opinions about them. It is worth noting that at this point, the response variables $out$ (or $out_{comp}$ if it is the case) and $out'$ must be comparable. As this search uses the ANOVA method as a terminating condition, we can provide an algorithm for this step. Furthermore, the ANOVA method is always used to accept or discard some statistical hypothesis. In our case, this statistical hypothesis is always the same and it is as follows:
  - Null Hypothesis ($H_0$): the responses (probabilities) are significantly different.

That is, the goal of the calibration (reject the null hypothesis) is to obtain an almost 45° straight line (the best possible match) between the responses provided by the real experiment against those provided by the model.

Algorithm 1 shows the calibration process. To simplify the presentation of the algorithm we will consider the search of a single parameter. Its goal is to find the best value for variable intMean, according to the ANOVA method applied to our model using certain instances of the parameter being adjusted. The algorithm takes as input the model-based experiment $M_{prob}(in', out', err')$, the response-compatible real-based experiment $E_{real}(in, out_{comp}, err)$ (in the algorithm we assume that out is the original set of responses or, if necessary, the one based on survival analysis—out$_{comp}$), and the search interval, represented by min (minimum starting value of the parameter) and max (maximum starting value of the parameter), for the parameter. Our algorithm divides the current interval in two equal parts (intMean) and sets its min/max to this intMean according to the new best interval of searching (this is determined by the least error found). Note that each ANOVA is calculated using the model-based experiment with the parameter handler instantiated by (initially) min, max and (later on by) intMean. This algorithm always terminates and this occurs when the difference between min and max is less than or equal to 1. The min and max are computed according to the standard error represented by errorMax and errorMin.

**Algorithm 1: Calibration process**

**Input:** $M_{prob}(in', out', err')$, $E_{real}(in, out, err)$, min, max  
**Output:** intMean, errorMean

1. begin
2. errorMin = 
3. errorMax = 
4. repeat
5. intMean = $\frac{min + max}{2}$
6. errorMean = 
7. if (errorMean > errorMax) then
8. min = intMean
9. else
10. max = intMean
11. endif
12. until (max − min ≤ 1)
13. return (intMean, errorMean)
14. end
IV. APPLYING THE PROPOSED STRATEGY

In this section we illustrate how we use the proposed strategy in our previous work reported in [4]. Thus our goal here is to calibrate the models of [4] with respect to our real experiments.

A. Step 1: Collect Data

As we previously said, to collect data, a controlled experiment was performed in [13]. Thus we first present the elements of our controlled experiment.

We characterize each phone configuration used in the experiment by:

- the phone model (that is, a list of external and internal phone features to identify a set of similar phone functions);
- the hardware version;
- the software version (that is, some build containing the current phone applications);
- the flex bit (FB) version (the flex bit configuration allows the user to dynamically configure the phone prior to normal usage. An example of such configurations includes enabling the phone to send and receive Bluetooth signals, and setting the phone to debug mode).

We configure our techniques with some parameters to determine how the exploration of the systems may occur. These parameters are used in the statistical assessment and we call them treatments. In what follows, we provide a brief explanation about the treatments we consider:

- **Re-initialization (Driven):** each technique consists in a repetitive behavior of setting a given initial state (screen) and then pressing keys on the phone during some time or until a crash is found. By re-initialization we mean the way the initial state is chosen: (i) always the same initial state; (ii) randomly chosen from a set of possible initial states.

- **Probabilities (KeyProb):** each key to be pressed on the phone can have its specific probability (frequency of occurrence). It is a reasonable assumption that some keys a more likely to occur than others. For example, a key like “Cancel” may have a smaller probability than “Ok” because the probability of “Ok” changing the system state is higher. And we assume that state change is directly related to finding bugs faster.

- **Test Case Size (SizeTC):** a test case is a sequence of keys to be pressed. This parameter concerns how much keys should be considered in a test case after some initial state was chosen.

The treatment structure used in this experiment is a $2^4$ full factorial design [9] (2 levels with 4 factors) resulting in 16 treatments (possible combinations). These treatments are applied in a randomized complete block design (RCBD) [9]. We use such experimental design to control the effect of the different phone configurations (blocks) and to investigate the effect of each factor and their interactions. The design is randomized in 4 blocks, where each block represents one phone configuration that we will perform 16 treatments. Therefore, the total number of executions is 64 runs (4 blocks $\times$ 16 treatments) or at most $64 \times 40h = 2560h$.

Table I shows a summary of the results (based on the previous experimentation [13]). As we can observe, we used 4 phone configurations (Config B, C, D and H) where two phones have the same phone model and hardware. We only vary the software version and the flex bits. The column **Config** presents the configurations used in this study and it has the same identifier used in previous experiments [13] to ease in a historical comparison. The columns B, C, D, and H correspond to the time to find a crash (our response variable) up to 40h observed in several treatments. In our previous work [13], we consider all values of our response variable, but we stop some executions (when 40h is reached) due time and resources limitations. However, as we will use survival analysis (next step), we can deal with all of them by setting such situations as censored. In practice, it means that we do not know the exact time when crashes occur, but we do not consider times greater than 40h. These observations are said to be censored.

B. Step 2: Create a Markov-based Model

Recall from Section II that we already have the required models. Thus we simply reuse them. However, if no model is available we need to determine a Markov-based model corresponding to the real experiment in order to use our strategy.

The creation of good abstract models is the most difficult part of the strategy. To simplify the construction we define different modules and compose them. Our goal is to ease the application of the strategy [4]. As a consequence, the models in our strategy are created using a component view. It means that the systems we are interested in are, in general, composed by a GUI component that represents the interaction between the user and the business rules (different components). In our models we observe some aspects like the interaction with the handler (responsible to accept or refuse each input event in the GUI) and the change of states (state changing has some effect on how the system works over time).

The modeling process used in this paper is done manually, but abstractions also can be mechanized and even automated. But this is transparent to the previous and next steps.

C. Step 3: Perform a Survival Analysis

Our models fit with the prerequisites of using the survival analysis. That is, our real experiment exhibits exact instances about where crashes are found and our PRISM model yields the probabilities of finding those crashes. Thus we apply survival analysis to transform the exact instances into probabilities.
Survival analysis deals with time-dependent data. We study the occurrence of crash and the time to crash. This time, which represents the lifetime until the occurrence of the crash, is called survival time or failure time [14]. Considering the data in Table I, we perform survival analysis in order to obtain the probability of crash over time (out).

1) Survival Definitions: Survival analysis is a collection of statistical procedures for data analysis where the outcome variable of interest is the time (failure time or survival time or lifetime) until an event \(X\) occurs [14]. The survival time is defined as a random variable \(T\) with censored data represented by a variable \(\delta\) where \(T \geq 0\) and \(\delta = (0, 1)\), where each treatment has one associated \(\delta\) that assumes two values: 0 in \(\delta\) means a censored data and 1 is the failure that occurs. In our case, the failure is censored when the study ends without finding a crash. We consider time in hours and an event means a crash occurrence. Also, this kind of analysis considers the occurrence of only one event at a time.

There are two quantitative terms to use in survival analysis. Equation 1 represents the survival function:

\[
S(t) = P(T > t) \tag{1}
\]

where \(P(T > t)\) is the probability of a technique to survive longer than a specified time \(t\) \((T > 40)\). This function obtains survival probabilities for different values of \(t\) providing a summary information from survival data. It has the following characteristics: (i) when \(t\) increases the probabilities decrease; (ii) at time \(t = 0\), we have \(S(t) = S(0) = 1\). That is, at time zero the survival probability is 1; and (iii) at time \(t = \infty\), we have \(S(t) = S(\infty) = 0\). That is, theoretically if the time increases without limit, eventually the probability becomes 0.

Equation 2 represents the hazard function \((h(t))\), also called failure function, and Equation 3 is the lifetime distribution function \((F(t))\) which is defined as the complement of the survival function:

\[
h(t) = 1 - F(t) \tag{2}
\]

\[
F(t) = P(T \leq t) = 1 - S(t) \tag{3}
\]

where \(P(T \leq t)\) is the probability of a technique to find a crash before the timeout occurs \((T \leq 40)\). Note that, the hazard function focuses on failure, that is, it is the dual of the survival function.

2) Evaluation: We estimate and draw the survival curves using the Kaplan-Meier (KM) method. KM is used to estimate the mean time to failure with censored data, which follows a nonparametric model because there is no needed (and we don’t know) to specify the distribution for the variable time until the failure. The estimated survival probabilities are computed using a product limit formula that estimates the survival function from lifetime data. The KM estimator of \(S(t)\) and it is given by:

\[
\hat{S}(t) = \prod_{t_i \leq t} \frac{n_i - d_i}{n_i} \tag{4}
\]

where
- \(\hat{S}(t)\) is the KM survival probability of a failure to occur at time \(t\);
- \(\prod_{t_i \leq t}\) is the product of time \(t\);
- \(n_i - d_i\) is the number of survivors \((n_i)\) less the number of censored cases \((d_i)\).

<table>
<thead>
<tr>
<th>Pattern</th>
<th>KeyProb</th>
<th>Driven</th>
<th>SizeTC</th>
<th>Technique</th>
<th>Config B</th>
<th>Config C</th>
<th>Config D</th>
<th>Config H</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>with</td>
<td>initial</td>
<td>50</td>
<td>BxT</td>
<td>censored</td>
<td>censored</td>
<td>1.4</td>
<td>1.2</td>
<td>1.3</td>
</tr>
<tr>
<td>2</td>
<td>without</td>
<td>initial</td>
<td>50</td>
<td>BxT</td>
<td>36.9</td>
<td>censored</td>
<td>1.3</td>
<td>2.8</td>
<td>13.7</td>
</tr>
<tr>
<td>3</td>
<td>with</td>
<td>initial</td>
<td>500</td>
<td>BxT</td>
<td>censored</td>
<td>33.4</td>
<td>1.1</td>
<td>9.7</td>
<td>14.5</td>
</tr>
<tr>
<td>4</td>
<td>with</td>
<td>random</td>
<td>50</td>
<td>BxT</td>
<td>20.1</td>
<td>16.6</td>
<td>6.8</td>
<td>27.5</td>
<td>17.8</td>
</tr>
<tr>
<td>5</td>
<td>without</td>
<td>random</td>
<td>500</td>
<td>BxT</td>
<td>12.8</td>
<td>32.0</td>
<td>10.2</td>
<td>29.9</td>
<td>21.2</td>
</tr>
<tr>
<td>6</td>
<td>without</td>
<td>random</td>
<td>50</td>
<td>BxT</td>
<td>15.1</td>
<td>21.9</td>
<td>9.7</td>
<td>15.1</td>
<td>15.3</td>
</tr>
<tr>
<td>7</td>
<td>with</td>
<td>random</td>
<td>500</td>
<td>BxT</td>
<td>23.5</td>
<td>28.7</td>
<td>23.2</td>
<td>38.9</td>
<td>28.6</td>
</tr>
<tr>
<td>8</td>
<td>without</td>
<td>initial</td>
<td>500</td>
<td>BxT</td>
<td>27.1</td>
<td>38.9</td>
<td>1.9</td>
<td>1.8</td>
<td>17.4</td>
</tr>
</tbody>
</table>

Mean: 22.6, 27.9, 6.9, 15.8 -

<table>
<thead>
<tr>
<th>Pattern</th>
<th>KeyProb</th>
<th>Driven</th>
<th>SizeTC</th>
<th>Technique</th>
<th>Config B</th>
<th>Config C</th>
<th>Config D</th>
<th>Config H</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>with</td>
<td>random</td>
<td>50</td>
<td>DH</td>
<td>38.9</td>
<td>22.3</td>
<td>3.2</td>
<td>21.8</td>
<td>23.0</td>
</tr>
<tr>
<td>10</td>
<td>with</td>
<td>initial</td>
<td>500</td>
<td>DH</td>
<td>censored</td>
<td>censored</td>
<td>8.5</td>
<td>censored</td>
<td>8.5</td>
</tr>
<tr>
<td>11</td>
<td>with</td>
<td>initial</td>
<td>50</td>
<td>DH</td>
<td>censored</td>
<td>censored</td>
<td>5.6</td>
<td>28.2</td>
<td>16.9</td>
</tr>
<tr>
<td>12</td>
<td>without</td>
<td>initial</td>
<td>50</td>
<td>DH</td>
<td>censored</td>
<td>censored</td>
<td>1.1</td>
<td>censored</td>
<td>1.1</td>
</tr>
<tr>
<td>13</td>
<td>with</td>
<td>random</td>
<td>500</td>
<td>DH</td>
<td>censored</td>
<td>34.5</td>
<td>14.1</td>
<td>33.7</td>
<td>27.4</td>
</tr>
<tr>
<td>14</td>
<td>without</td>
<td>random</td>
<td>500</td>
<td>DH</td>
<td>censored</td>
<td>censored</td>
<td>10.9</td>
<td>censored</td>
<td>10.9</td>
</tr>
<tr>
<td>15</td>
<td>without</td>
<td>initial</td>
<td>500</td>
<td>DH</td>
<td>censored</td>
<td>censored</td>
<td>2.1</td>
<td>20.2</td>
<td>11.2</td>
</tr>
<tr>
<td>16</td>
<td>without</td>
<td>random</td>
<td>50</td>
<td>DH</td>
<td>30.7</td>
<td>25.4</td>
<td>7.1</td>
<td>10.9</td>
<td>18.5</td>
</tr>
</tbody>
</table>

Mean: 34.3, 29.3, 7.4, 23.0

Table 1
FULL FACTORIAL DESIGN AND RESULTS.
Figure 6 shows the complement of the survivability patterns over time (that is, \(1 - \text{survival rate}\)) corresponding to our two testing techniques (BxT and DH) using all phone configurations. We consider all configurations and parameters of the techniques because we want to determine what is the survivability of each GUI testing technique. The failure rate starts at 0.0 (because at time 0 the probability of failure is 0%), indicating the start of the study (0% of bugs are uncovered), and the lines indicate the probability to failure over time. Figure 6 was generated using the KM estimator (Equation 4) provided by the software JMP [16].

Another thing we can observe is that the survival rates are closer to both techniques up to 15 hours (survivability around 70%) and far away after 15 hours. Thus, survival analysis indicates that BxT has a higher probability of finding crashes up to 40h than DH, that is, BxT has the highest failure rate over time.

![Figure 6. Failure Plot.](image)

**D. Step 4: Calibration**

Following our strategy, now we need to adjust some parameters of our PRISM model in such a way that the model predictions are closer enough to the responses of the real experiment. Concerning our models, we have a set of variables that we know their exact values (like configuration of our techniques) and as we previously said such variables are not a problem to the model accuracy (calibration). However, the value corresponding to the reliability of the handler is unknown and thus we only have an intuition from the test engineers. The test engineers consider the handler’s reliable but this is a qualitative measure and we need a numerical value. In [13] we consider an ad hoc solution and set the reliability of the handler as 90%. Thus here we follow our proposed strategy to adjust this parameter.

The adjustment is relatively simple. It consists in searching for possible values of the handler’s reliability that better approximates the values provided by the PRISM model with respect to the complement of those provided by Figure 6.

According to our strategy, this means testing the hypothesis about the linearity between the response variables of the real against the model-based experiments. Thus we use the ANOVA method to try to discard the following statistical null hypothesis (note that this hypothesis is used twice, where it is instantiated with the testing technique DH as well as BxT):

- Null Hypothesis (\(H_0\)): the responses (probabilities) are significantly different.

To determine the best value for the handler’s reliability and simultaneously avoiding any tendency inherited from the test engineer opinions, we consider a broad range of values: we set min initially to 1% and max to 99%. By running our algorithm we obtain the following search intervals: (50%,99%), (75%,99%), (87%,99%), (93%,99%), (96%,99%), (98%,99%). This search ends with the result 98% as the best approximation for the handler’s reliability and it is shown graphically in Figure 7. The x (\(dh\_sa\) and \(dh\_prism\)) and y (\(bxt\_sa\) and \(bxt\_prism\)) axis show the probability to find a crash obtained by the survival analysis and our simulation using PRISM, respectively.

To have a better idea of the calibration we provide the results of the ANOVA method as well (Table II shows the ANOVA table for DH and BxT). It is important to observe the standard error. The smaller is the value of the standard error; the best is the value for the variable being investigated. We observe that the best value for the handler’s reliability is 98% for both techniques, which has the standard error of 0.0006889 for DH and 0.006599 for BxT. As the p-value of this table (the probability under the column Prob > F) is less than 0.05 (we assume \(\alpha = 0.05\)), then we reject the null hypothesis.

<table>
<thead>
<tr>
<th>Reliability</th>
<th>Estimate</th>
<th>StdErr</th>
<th>t value</th>
<th>Prob &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>99%</td>
<td>0.033394</td>
<td>0.007532</td>
<td>-4.434</td>
<td>0.000364</td>
</tr>
<tr>
<td>98%</td>
<td>0.035608</td>
<td>0.006889</td>
<td>-5.169</td>
<td>0.000077</td>
</tr>
<tr>
<td>75%</td>
<td>0.04445</td>
<td>0.019142</td>
<td>-7.596</td>
<td>0.000000734</td>
</tr>
<tr>
<td>50%</td>
<td>0.07063</td>
<td>0.09182</td>
<td>-8.393</td>
<td>0.000000189</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reliability</th>
<th>Estimate</th>
<th>StdErr</th>
<th>t value</th>
<th>Prob &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>99%</td>
<td>0.050579</td>
<td>0.008407</td>
<td>-6.016</td>
<td>0.00000327</td>
</tr>
<tr>
<td>98%</td>
<td>0.04046</td>
<td>0.006599</td>
<td>-6.131</td>
<td>0.00000247</td>
</tr>
<tr>
<td>75%</td>
<td>0.05454</td>
<td>0.01682</td>
<td>-7.87</td>
<td>4.21E-08</td>
</tr>
<tr>
<td>50%</td>
<td>0.06659</td>
<td>0.07527</td>
<td>-8.856</td>
<td>0.00000005</td>
</tr>
</tbody>
</table>

**E. Threats to validity**

This section describes threats to internal and external validity of our experiments. Internal validity determines whether the techniques have a cause-and-effect relationship
in the experimental observations. External validity determines whether or not one can generalize the experimental observations to other scenarios.

One threat to internal validity is internal randomness. It is possible to simulate situations where the real system does not execute due to the random-based nature of the techniques. Thus, there is a probability to real experiments differ from our simulations.

One threat to external validity is the portability of the strategy to other contexts, including other probabilistic models. We need a better understanding on the kind of contexts our strategy can be applied. However, we can assure its use in a slightly different context by simply using different testing techniques. Another external validity is whether our calibrated model is still accurate for different situations (changes in the system and the techniques) considering our GUI testing technique and system. In principle, there is no reason to believe that they are not applicable to other situations. Anyway, the model can be recalibrated to consider different scenarios.

V. DISCUSSION

The main advantage of this kind of simulation is to predict new situations to support the decisions of the software engineers. We can also check different scenarios (GUI-system based) of our case study, for example, using a web application as a GUI-system.

Another reason to simulate, for example, a new technique, is to give to software engineers the opportunity to evaluate such a technique in its intended environment without the effort of implementing it up-front. Techniques, mainly in the embedded systems domain, are expensive to implement, test and debug. Also, if one has a new testing technique to insert in the production environment, it is desirable to know its effectiveness and efficacy as soon as possible.

VI. RELATED WORK

Bayarri et al. [17] present a six step to calibrate models based on comparison of computer model runs with field data of the process being modeled. They use wavelet coefficients in order to compare computer model output with field output. This work differs from ours mainly in the use of the field data in the models and how to determine whether the model
is calibrated or not. In our work we use survival analysis and simulate (with models) until the models fit with experimental data.

Janssen and Heuberger [12] present a detailed study of the calibration process. They discuss its limitations, the various involved choices and indicates techniques for performing the calibration. We do not discuss the alternative approach and limitations of the calibration process as a whole, but we propose a novel approach and we focus the discussion to our approach.

Survival analysis is applied in many science fields like human, biology and medicine. However, in software engineering and computer systems a few insights are found in the literature. The simple approach is to obtain the field data and apply survival analysis. For example, to analyze of software reliability and network security [18], [19]. Wedel et al. [20] propose the use of survival analysis in order to verify the correlation of code repositories with bug databases over time.

Clarke et al. [21] present the algorithm for performing statistical Model Checking using Bayesian Sequential Hypothesis Testing. Basically, they are using biological models that are often too large for classical model checking. They apply an algorithm that requires fewer system simulations to check a property. This allows faster verification. In our work we use Statistics to calibrate our models by checking if our model has an approximation with the real system.

The validation problem is reported in the literature and the most relevant solutions are based on Bayesian methodology [22]. Rebba et al. [23] propose a Bayesian methodology for assessing the confidence in model prediction by comparing the model output with experimental data when both are stochastic. They use a Bayes factor as a metric in order to perform model assessment. Recently, Leye et al. [24] report an overview of the model validation field. They show the importance of this area in simulation and indicate as a future direction the need of tools for guidance of experimental model validation. In our work, we propose an algorithm to calibrate probabilistic models, which can be easily implemented in a tool to help the model validation process. Also, we already compute a survival analysis due to the characteristics of our data (some executions are censored).

Simulating real systems is faster and in general cheaper than real experiments. However, we have some basic (foundation) problems with our mathematical models like the state explosion problem. Therefore, how to compute probabilities of large systems. Fortunately, a new research topic is attracting the attention of the model checking community: Statistical Model Checking (SMC). Sen et al. [25] present a statistical approach to analyze stochastic systems against specifications written in the CSL language. Also they implement it in a tool called VESTA [26]. Clarke et al. [27] are using ideas and proposing new algorithms in the SMC area. They are introducing Bayesian analysis and checking properties based on SMC where classical model checkers cannot handle due to scalability problems.

VII. Conclusions and Future Work

As far as we know, we present a novel strategy to calibrate probabilistic models and show how the approach can be used. This strategy deals with parameter uncertainty in Markov-based models. We are able to confirm by statistical confidence ($\alpha = 0.05$) that the probabilistic model of our case study approximates the real experiment. This results in the best value for the handler’s reliability as 98% for both techniques, with standard error of 0.006889 for DH and 0.006599 for BxT. Thus, the main advantage of our strategy is to calibrate models with strong statistical confidence.

Our model can be used to simulate new experiments that could not be executed due their costs, like evaluating new GUI testing techniques or considering new aspects of the system. For example, we can simulate a new GUI testing techniques that combines DH and BxT by pressing controlled keys (a fixed number of keys that changes the state and a fixed number of keys that does not change the system state). We believe that this kind of testing technique can be useful in more general applications, but a preliminary experimentation with the model can give us insights about its real execution.

In a future investigation, we plan to verify our strategy in other contexts and experiment other models. Another future work is to check whether our PRISM models conform with linear changes applied to the real experiment. As an extension to our current work, we intend to consider real time systems and different modeling choices, for example, to model the behavior of the system using CTMC instead of DTMC.

Acknowledgment

We would like to thank Cristiano Ferraz for comments about this work. This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES\(^1\)), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08. Also, the first author was partially supported by the CNPq fellowship 142905/2006-2.

References


\(^1\)http://www.ines.org.br