ISBN 978-981-14-1684-2
Proceedings of 2019 the 9th International Workshop on Computer Science and Engineering
(WCSE 2019 SUMMER)
Hong Kong, 15-17 June, 2019, pp. 475-481
doi: 10.18178/wcse.2019.06.071

A Case Study of Applying Rigorous Testing in Practice

Yufeng Xue [1], Lan Lin [1 +], John C. Tucker [2], Becky Hammons [2] and Michael Wolfe [2]

[1] Department of Computer Science, Ball State University, Muncie, Indiana, USA
[2] Ontario Systems, LLC, Muncie, Indiana, USA

Abstract. This paper reports on experiences and preliminary results we obtained through a case study, in which we applied a rigorous testing method, namely statistical testing based on a Markov chain usage model, to a real-world testing problem in an industrial setting. Although model-based statistical testing has been around for more than two decades with well-established theory and engineering practices, it remains problem- and application-specific to develop a workable testing solution and framework that enables automatic test case generation-execution-evaluation, to provide meaningful and quantifiable statistics/data to make informed management decisions, and to support software certification. We describe the challenge involved in testing an Interactive Voice Response (IVR) module, and demonstrate our approach to tackle the problem following statistical testing practices, from usage modeling all the way to the test case analysis.

Keywords: Software testing, rigorous testing, statistical testing, model-based testing, Markov chain usage models, software quality

1. Introduction and Related Work

As a rigorous testing method developed by the University of Tennessee Software Quality Research Laboratory (UTK SQRL) in the 90's, *statistical testing* based on a *Markov chain usage model* [5, 6, 8, 10, 11] has been around for more than two decades, with well-established theory and engineering practices [7], and well-developed tool support [1, 9]. It frames the testing problem in science, with comprehensive application of statistical science to all the phases of software testing, bringing the direct benefit of quantification of the projected system reliability solely based on the testing data.

This form of statistical testing has been successfully applied to a variety of application development, ranging from medical devices, automotive components, to scientific instrumentation, to name a few. Although proven to be practical and effective, it remains problem- and application-specific to develop a statistical testing solution and framework that enables automatic test case generation-execution-evaluation, to run a large sample of test cases for valid statistical inference. We report in this paper our experiences in applying it to a real-world testing problem, the challenge involved, our approach and some preliminary results.

2. The Case Study: IVR Module Testing

The real-world problem that interested us is a rigorous testing of an Interactive Voice Response (IVR) module, a part of a voice-based billing system that allows users to dial in, verify accounts, make payments, update contact information, or talk to a customer service representative. We obtained functional requirements of the IVR module in a ten-page Word document.

After the user dials in, the system first locates the user's account through user-entered account number, or an Automatic Number Identification (ANI) search based on the caller ID or user-entered primary phone

[+] Corresponding author. Tel.: + 1 (765) 285-8641; fax: + 1 (765) 285-2614.
E-mail address: llin4@bsu.edu.

number, and performs needed account verification. Then the system communicates with the account balance, and enters the main menu. From there the user can make a payment, advise the system of an address or phone number change, or have other inquiries. The system supports different payment methods, e.g., by credit card, by debit card, by Electronic Fund Transfer (EFT), by electronic-check (e-check), or by check or money order sent in the mail, and different payment arrangements, e.g., paying in full, making a one-time partial payment, or making a payment arrangement. If payment is being made, the system takes and processes payment information, and confirms the payment. In a usage scenario the system guides the user through voice messages, and the user interacts with key presses or voice responses.

3. Statistical Testing of the IVR Module

Effective testing of the IVR module involves frequent judgement calls on what important usage scenarios to include in the test suite. In this section we describe how we approached this testing problem with statistical testing.

3.1. Usage modeling

The first step in statistical testing is to develop a usage model that characterizes all and expected uses of the System Under Test (SUT), in the form of a discrete-parameter, finite-state, time-homogeneous, and recurrent Markov chain [5, 6]. Usage modeling of the IVR module takes four iterations with feedback from our collaborator. Figure 1 shows a very small fraction (expansion from the beginning state, or the model source) of our developed usage model. The nodes represent usage states (e.g., "Account number known", "Account located"). The directed arcs represent transitions between states of use triggered by usage events/inputs (e.g., "ANI success" triggers the transition from "Valid account typed" to "Account located"), with expected software's outputs (e.g., "palm" is a shorthand for "the system playing account located message"). Each arc is associated with a probability of taking this transition from the "from" state (e.g., the next event being "ANI success" from "Valid account typed" has a probability of 0.05). While the model structure defines all possible uses of the SUT, the arc probabilities define all expected uses as represented by a usage profile.

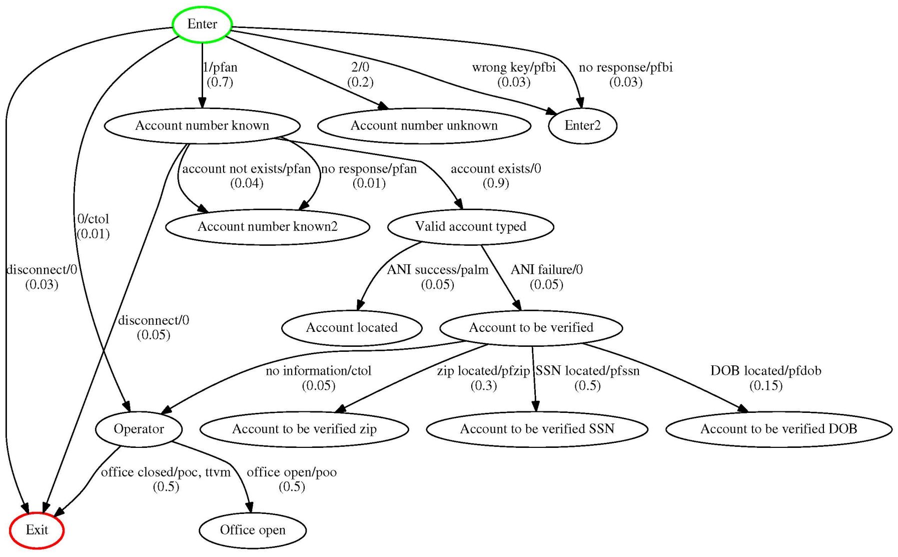


Fig. 1: A very small fraction of an IVR usage model, with "Enter" being the model source and "Exit" being the model sink. Arcs are labelled with "inputs/outputs (probabilities)".

Figure 2 shows the beginning few lines of our developed usage model in The Modeling Language (TML)[1] [2], with states in angle brackets, inputs/outputs in quotation marks, and probabilities in a pair of "$" signs within parentheses. Outgoing arcs from the same "from" state are grouped together.

The complete IVR usage model contains 271 states and 1,473 arcs, making it impossible to show clearly on one page all the model details (hence omitted here). We hope to mention that our collaborator helped us acquire usage data based on which we assigned all the arc probabilities.

[1] The Modeling Language (TML) was developed by UTK SQRL in the late 90's.

```
// A usage model for the IVR module
// A use of the IVR starts with hearing the welcome message
// and ends with hearing the welcome message again
// or disconnect or being transfered to the operator.


model IVR

[Enter]
            ($0.01$)"0/ctol"           [Operator]
            ($0.7$)"1/pfan"            [Account number known]
            ($0.2$)"2/0"               [Account number unknown]
            ($0.03$)"wrong key/pfbi"   [Enter2]
            ($0.03$)"no response/pfbi" [Enter2]
            ($0.03$)"disconnect/0"     [Exit]

[Enter2]
            ($0.01$)"0/ctol"           [Operator]
            ($0.7$)"1/pfan"            [Account number known]
            ($0.2$)"2/0"               [Account number unknown]
            ($0.03$)"wrong key/pfbi"   [Enter3]
            ($0.03$)"no response/pfbi" [Enter3]
            ($0.03$)"disconnect/0"     [Exit]

[Enter3]
            ($0.01$)"0/ctol"           [Operator]
            ($0.7$)"1/pfan"            [Account number known]
            ($0.2$)"2/0"               [Account number unknown]
            ($0.03$)"wrong key/ctol"   [Operator]
            ($0.03$)"no response/ctol" [Operator]
            ($0.03$)"disconnect/0"     [Exit]

...          ...                        ...
```

Fig. 2: An excerpt of an IVR usage model in TML.

3.2. Model analysis

We performed a model analysis using our statistical testing tool, the JUMBL[2] [1, 9]. Important statistics of the model were computed solely based on the Markov chain, and were used to validate the model (in four iterations) against what was known or believed about the application domain and the environment of use.

Table 1 shows model statistics such as the number of nodes, the number of arcs, the number of stimuli (usage events), and the average test case length (with variance). Other important model statistics (omitted due to the page limit) include:

- **Occupancy.** The amount of time in the long run that one will spend testing a node/arc/stimulus.
- **Probability of Occurrence.** The probability of a node/arc/stimulus appearing in a random test case.
- **Mean Occurrence.** The average number of times a node/arc/stimulus will appear in a random test case.
- **Mean First Passage.** The number of random test cases one will need to run on average before testing a node/arc/stimulus for the first time.

Table 1: Model statistics

Node Count	271 nodes
Arc Count	1473 arcs
Stimulus Count	72 stimuli
Expected Test Case Length	8.322 events
Test Case Length Variance	4.905 events
Transition Matrix Density (Nonzeros)	14.7056821E-3 (1,080 nonzeros)
Undirected Graph Cyclomatic Number	810

3.3. Model annotation

Since our collaborator used Cucumber [3] for automated testing, next we needed to annotate our IVR usage model with testing scripts (in this case, Gherkin [4] statements, as the Cucumber-executable test cases use the Gherkin syntax to define test steps) to integrate with their existing automated testing framework (and to facilitate automated test case execution with Cucumber).

We wrote Python code for this purpose. Figure 3 shows an excerpt of the annotated IVR usage model.

[2] The J Usage Model Builder Library (JUMBL) was developed by UTK SQRL in the early 2000s.

```
model IVR


[Enter]

    ($0.01$)"0/ctol"
        a:|$    When I send DTMF 0
          |$    Then the transfer event is raised
    [Operator]

    ($0.7$)"1/pfan"
        a:|$    When I send DTMF 1
          |$    Then I wait 30 seconds to listen to the account number prompt
    [Account number known]

    ($0.2$)"2/0"
        a:|$    When I send DTMF 2
          |$    Then nothing happens
    [Account number unknown]

    ($0.03$)"wrong key/pfbi"
        a:|$    When I send DTMF 9
          |$    Then I wait 30 seconds to listen to the welcome message
    [Enter2]

    ($0.03$)"no response/pfbi"
        a:|$    When I wait 30 seconds
          |$    Then I wait 30 seconds to listen to the welcome message
    [Enter2]

    ($0.03$)"disconnect/0"
        a:|$    When I hang up
          |$    Then nothing happens
    [Exit]
...
```

Fig. 3: An excerpt of the annotated IVR usage model in TML. Arcs are annotated with input/output Gherkin statements.

3.4. Test case generation

With the JUMBL we are able to automatically generate test cases from a Markov chain usage model based on different sampling options, in almost no time. Our test plan is as follows. First, we generated 789 minimum coverage test cases that cover every arc (and hence every node) of the IVR usage model with the minimum total number of steps (due to the model size and complexity). Next, we generated 200 weighted test cases, representing the 200 most probable paths. This was followed by random sampling. We generated 800 random test cases based on the probabilities on the arcs. In total, our test plan contains 1,789 test cases. Figure 4 shows an example of a weighted test case generated and exported by the JUMBL. Each test step maps to Gherkin statements that issue a test input (the “When” clause) and check/verify a test output (the “Then” clause). Test oracle is included in the usage model and the generated test cases through the specified outputs and the “Then” clauses.

```
#Test case name: IVR_wt_142
# ========================================
# Trajectory: 0
# Model:      IVR
# Key:
# Method:     weight
#
# Events:     8
# Including failure information.
# ========================================
# Step: 1, Trajectory: 0
    When I send DTMF 1
    Then I wait 30 seconds to listen to the account number prompt
# Step: 2, Trajectory: 0
    When I send DTMF as my account number
    Then nothing happens
# Step: 3, Trajectory: 0
    When ANI search is a success
    Then the system plays account located message
# Step: 4, Trajectory: 0
    When IVR stop logic is false
    Then I wait 60 seconds to listen to the mini-Miranda, my account balance, and the main menu
# Step: 5, Trajectory: 0
    When I send DTMF 1
    Then I wait 30 seconds to listen to the payment menu options
# Step: 6, Trajectory: 0
    When I send DTMF 1
    Then I wait 20 seconds to listen to the pay by credit card prompt
# Step: 7, Trajectory: 0
    When I send DTMF 1
    Then I wait 30 seconds to listen to the payment options prompt
# Step: 8, Trajectory: 0
    When I hang up
    Then nothing happens
# TESTING STOPPED AT 8
```

Fig. 4: An example of a weighted test case generated from the IVR usage model by the JUMBL. Test steps are mapped to Gherkin statements.

3.5. Test case execution

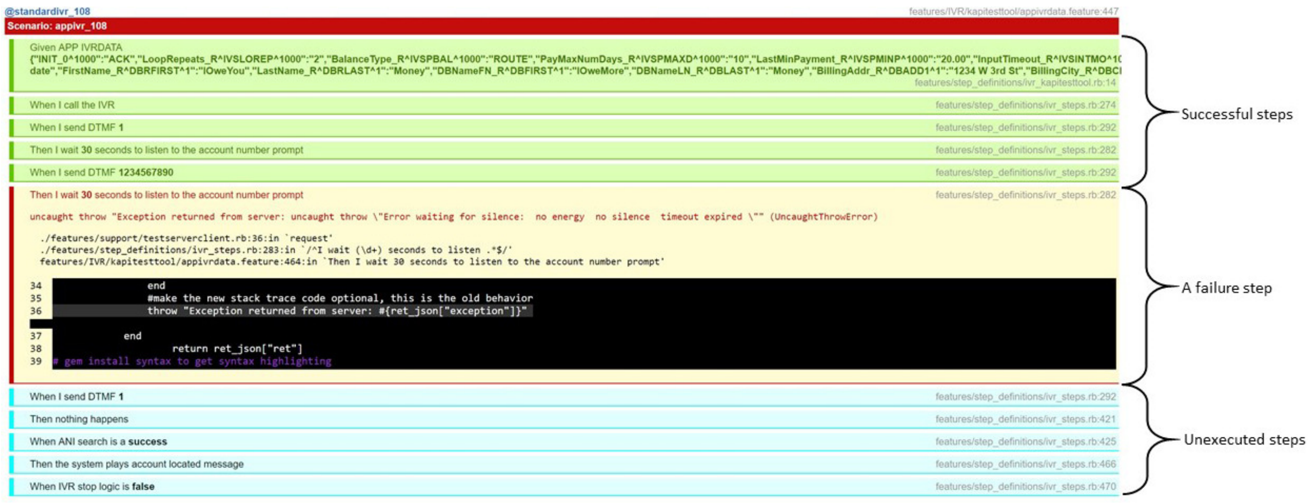
Our next task was to integrate the generated test suite with an existing Cucumber-based automated testing framework (our collaborator had developed), to achieve automatic test case execution. Since the IVR module is only a part of a larger sub-system for which the framework was developed, we needed to (1) analyse and collect all the preconditions required for the execution of a particular path (as reflected by the sequence of events in a generated test case), to simulate the effect of isolating the IVR module while running tests in the larger context, and (2) read and analyse test results from the Cucumber-generated HTML result file, and collect more details needed for statistical analysis, such as whether each test case is successful; for each failed test, what are the failure steps; and whether each failure step is a continue failure (meaning the following step in the test can still run to completion), or a stop failure (meaning the test aborts after this failure step).

We wrote Ruby code to first construct a big “Given” statement, to put in front of the sequence of events in each exported test case, that spells out all the collected preconditions required for running this test case, and then convert each big “Given” statement to a JSON statement that our collaborator needed to automatically retrieve information to determine which user account would be used to run each test.

We also wrote Ruby code to extract the needed test execution details from the generated HTML test report (see Figure 5). Together with another shell script we wrote, we were able to record the test results in the JUMBL for statistical analysis.

Figure 6 illustrates the sequence of processing we did to facilitate automatic test case generation-execution-evaluation.

At our first attempt, it took 3,122 minutes and 26 seconds (about 52 hours) to run all the 1,789 test cases, of which 395 passed and 1,394 failed. After examining some failed test cases we found that there are still certain adjustments that need to be made on the generated test cases to automatically execute them using the existing test framework. Work is under way to make all the needed adjustments, each could affect many test cases, for the next round of test case execution and evaluation.



```
@standardivr_108                                                                       features/IVR/kapitesttool/appivrdata.feature:447
Scenario: appivr_108
  Given APP IVRDATA
  {"INIT_0^1000":"ACK","LoopRepeats_R^IVSLOREP^1000":"2","BalanceType_R^IVSPBAL^1000":"ROUTE","PayMaxNumDays_R^IVSPMAXD^1000":"10","LastMinPayment_R^IVSPMINP^1000":"20.00","InputTimeout_R^IVSINTMO^1
  date","FirstName_R^DBRFIRST^1":"IOweYou","LastName_R^DBRLAST^1":"Money","DBNameFN_R^DBFIRST^1":"IOweMore","DBNameLN_R^DBLAST^1":"Money","BillingAddr_R^DBADD1^1":"1234 W 3rd St","BillingCity_R^DBCI
                                                                                       features/step_definitions/ivr_kapitesttool.rb:14
  When I call the IVR                                                                  features/step_definitions/ivr_steps.rb:274
  When I send DTMF 1                                                                   features/step_definitions/ivr_steps.rb:292
  Then I wait 30 seconds to listen to the account number prompt                        features/step_definitions/ivr_steps.rb:282
  When I send DTMF 1234567890                                                          features/step_definitions/ivr_steps.rb:292
  Then I wait 30 seconds to listen to the account number prompt                        features/step_definitions/ivr_steps.rb:282
  uncaught throw "Exception returned from server: uncaught throw \"Error waiting for silence:  no energy  no silence  timeout expired \"" (UncaughtThrowError)

    ./features/support/testserverclient.rb:36:in `request'
    ./features/step_definitions/ivr_steps.rb:283:in `/^I wait (\d+) seconds to listen .*$/'
    features/IVR/kapitesttool/appivrdata.feature:464:in `Then I wait 30 seconds to listen to the account number prompt'

                  end
                  #make the new stack trace code optional, this is the old behavior
                  throw "Exception returned from server: #{ret_json["exception"]}"

              end
                      return ret_json["ret"]
  # gem install syntax to get syntax highlighting
  When I send DTMF 1                                                                   features/step_definitions/ivr_steps.rb:292
  Then nothing happens                                                                 features/step_definitions/ivr_steps.rb:421
  When ANI search is a success                                                         features/step_definitions/ivr_steps.rb:425
  Then the system plays account located message                                        features/step_definitions/ivr_steps.rb:466
  When IVR stop logic is false                                                         features/step_definitions/ivr_steps.rb:470
```

Fig. 5: An example of a recorded test case in a Cucumber-generated test report, with different color codes for successful, failed, and unexecuted test steps.

3.6. Test case analysis

We report here a preliminary test case analysis based on our first attempt at running the test suite, using the JUMBL. Table 2 shows some important statistics to help with management decisions:

- **Single Event Reliability.** The probability of a randomly selected event (arc/step) being successful.
- **Single Use Reliability.** The probability of a randomly selected use (path/demand) being successful, given a specification of correct behavior.
- **Relative Kullback Discrimination.** An information-theoretic measure of how close the performed testing matches the software use as described by the usage model.
- **Optimum Reliability.** The estimated reliability if all *generated* test cases were executed successfully.

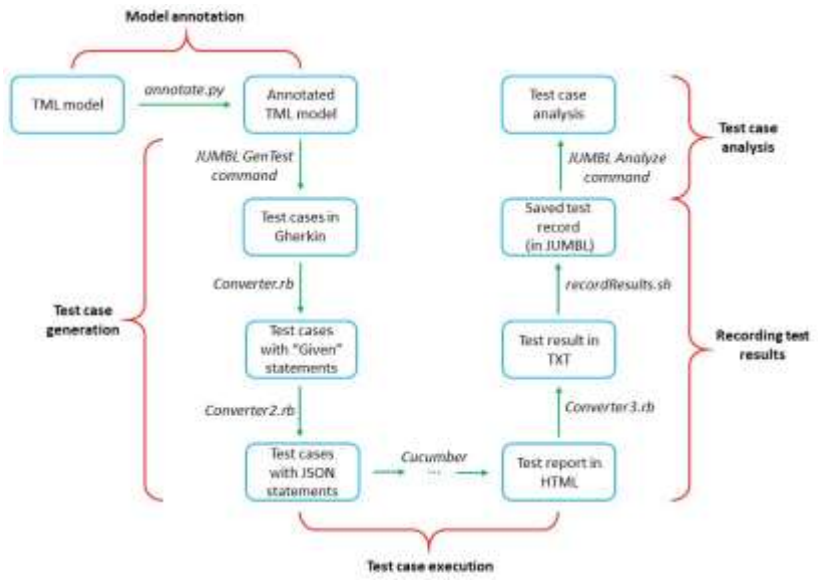


Fig. 6: The sequence of processing to facilitate automated statistical testing of the IVR module.

We hope to point out that once we are done with all the needed adjustments in the test framework and re-run the test suite, we will be able to repeat the process and update with more accurate analysis. What we have obtained provides a basis for refinement of this step and the result appears promising.

Table 2: Excerpts of the test case analysis: Reliabilities

Single Event Reliability	0.765847157
Single Event Variance	18.3936833E-6
Single Event Optimum Reliability	0.967487301
Single Event Optimum Variance	1.55073316E-6
Single Use Reliability	0.285513535
Single Use Variance	0.127557329
Single Use Optimum Reliability	0.805339308
Single Use Optimum Variance	60.3787671E-3
Arc Source Entropy	1.302 bits
Kullback Discrimination	4.834 bits
Relative Kullback Discrimination	371.214%
Optimum Kullback Discrimination	0.219023429 bits
Optimum Relative Kullback Discrimination	16.82%

4. Conclusion and Future Work

This paper reports our experiences and preliminary results obtained through a case study of conducting model-based statistical testing to the real-world testing of an IVR module in a voice-based billing system. We developed a test plan and testing strategies following statistical testing practices, from usage modeling all the way to test case analysis, and demonstrated a viable and rigorous testing solution to quantify the quality of the system under test for (part of) a commercial product.

Future work is along the line of making more accurate reliability estimates through (1) adjustments in the generated test suite for more accurate test case execution using the existing test framework, and (2) state verification for better classification of stop/continue failures and location of the first failure step. The results we have obtained appear promising.

5. Acknowledgements

This work was generously funded by Ontario Systems through the NSF Security and Software Engineering Research Center (S^{2}ERC).

6. References

[1] J Usage Model Builder Library (JUMBL). Software Quality Research Laboratory, The University of Tennessee. http://jumbl.sourceforge.net/jumblTop.html. 2019.

[2] The Modeling Language (TML). Software Quality Research Laboratory, The University of Tennessee. http://jumbl.sourceforge.net/tml/index.html. 2019.

[3] Cucumber. https://cucumber.io. 2019.

[4] Gherkin Syntax. https://docs.cucumber.io/gherkin. 2019.

[5] Jesse H. Poore. Theory-practice-tools for automated statistical testing. *DoD Software Tech News: Model-Driven Development* 2010, 12(4): 20-24.

[6] Jesse H. Poore, Lan Lin, Robert Eschbach, and Thomas Bauer. Automated statistical testing for embedded systems. In: J. Zander, I. Schieferdecker, and P. J. Mosterman (Eds.). *Model-Based Testing for Embedded Systems in the Series on Computational Analysis and Synthesis, and Design of Dynamic Systems*, CRC Press-Taylor & Francis. 2011, pp. 111-146.

[7] Jesse H. Poore and C. J. Trammell. Engineering practices for statistical testing. *Crosstalk (DoD Software Engineering Journal-Newsletter)* (April 1998): 24-28.

[8] Jesse H. Poore and Carmen J. Trammell. Application of statistical science to testing and evaluating software intensive systems. In: Michael L. Cohen, Duane L. Steffey, and John E. Rolph (Eds.). *Statistics, Testing, and Defense Acquisition: Background Papers*, National Academies Press. 1999, pp. 124-170.

[9] Stacy J. Prowell. JUMBL: A tool for model-based statistical testing. In: *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*. Big Island, HI, 2003, 337c.

[10] James A. Whittaker and Jesse H. Poore. Markov analysis of software specifications. *ACM Transactions on Software Engineering and Methodology* 1993, 2(1): 93-106.

[11] James A. Whittaker and Michael G. Thomason. A Markov chain model for statistical software testing. *IEEE Transactions on Software Engineering* 1994, 30(10): 812-824.