Known Unknowns: Testing in the Presence of Uncertainty

Sebastian Elbaum
Dept. of Computer Science and Engineering
University of Nebraska - Lincoln
Lincoln, NE, USA
elbaum@cse.unl.edu

David S. Rosenblum
Dept. of Computer Science
National University of Singapore
Singapore
david@comp.nus.edu.sg

ABSTRACT

Uncertainty is becoming more prevalent in the software systems we build, introducing challenges in the way we develop software, especially in software testing. In this work we explore how uncertainty affects software testing, how it is managed currently, and how it could be treated more effectively.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*reliability, statistical methods*; D.2.5 [**Software Engineering**]: Testing and Debugging; G.3 [**Mathematics of Computing**]: Probability and Statistics—*Markov processes, nonparametric statistics, stochastic processes*

General Terms

Experimentation, Measurement, Reliability, Theory

Keywords

Uncertainty Quantification, Software Testing, Fault Masking, Hidden Markov Models

1. INTRODUCTION

It is a cliché to observe that software systems are becoming ever more sophisticated, and that providing this sophistication makes their initial and ongoing development significantly complex. Much of the complexity arises from the need to support deep interactions between interconnected software systems and their users and execution environments. A little-explored consequence of this complexity is that software increasingly embodies a great deal of *uncertainty* with respect to its behavior.

Uncertainty in software is prevalent not just in its development and understanding, but also in its design and implementation. Indeed, uncertainty is sometimes introduced *deliberately* into software designs and implementations, leading to occasional misbehavior or incorrect output that is deemed to be acceptable. Consider the following examples:

- The popular application *Google Maps* provides functionality to estimate the location of the user's mobile device, based on GPS signals, triangulation of signals from cellular communication towers, and crowdsourced location data for WiFi access points. It thus fuses data from multiple sources to optimize the estimate and tolerate the source variability. The outcome is a computed location of uncertain accuracy that depends to a great extent on the performance of those external resources at the time the data is needed, and the quality of the data fusion library. Developers and users are willing to accept reported locations that are tens of meters off.
- A *context-aware music recommender system* (CAMRS) selects music from a user's smartphone-based music library to complement their current activity, which is inferred via input from the smartphone's sensors [20]. A key component of a CAMRS is an activity classifier built using machine learning techniques. As is typical with machine learning, the resulting classifier has less than 100% precision, but the developer accepts this uncertainty on the assumption that users are willing to tolerate suboptimal recommendations.
- *Quadcopter stabilization* systems rely on sensors for attitude adjustment [9]. The sensor readings are used by a series of controllers to change thrust values continuously in order to keep the vehicle stable during flying. The outcome of this stabilization behavior depends heavily on the precision and accuracy of the sensors. Even if those sensors could operate perfectly, the variability of the physical components and the environment introduce uncertainty in the system behavior. That uncertainty leads to conservative controller configurations, and users willing to accept temporary oscillations.

These are just a few of the many possible examples we could present, but they illustrate the fact that uncertainty is present in most systems we build today, whether introduced by human decisions, machine learning algorithms, external libraries, or sensing variability. Garlan recognized this phenomenon and put forth a challenge to treat uncertainty as a first class citizen in software development [7]. In addition, some researchers have begun to deal with the uncertainty underlying system requirements, design and adaptation [4, 6, 8, 10, 18].

In the context of software testing, uncertainty increases the ambiguity of what constitutes the input space and what is deemed as acceptable behavior, which are two key attributes of a test. Existing approaches to deal with uncertainty in testing have been partial and of limited scope [3, 24], leaving the systematic treatment of uncertainty in testing still open. In the following sections, we further motivate this problem through an example and discuss these challenges in more detail, and then we delve into potential directions for dealing with uncertainty during testing.

FSE'14, November 16–21, 2014, Hong Kong, China
Copyright 2014 ACM 978-1-4503-3056-5/14/11...$15.00
http://dx.doi.org/10.1145/2635868.2666608

2. MOTIVATING EXAMPLE

Given that a software system can produce misbehaviors and incorrect outputs that are acceptable to some extent to its developers and users, how can we distinguish *acceptable misbehaviors* (caused by "known unknowns") from *unacceptable misbehaviors* (caused by "unknown unknowns")? How can we determine when there are (avoidable or fixable) faults present in the system that are being *masked* by the acceptable misbehaviors? There is no general answer to this question, since the answer depends very much on the nature of the underlying uncertainties of the system, and the extent to which those uncertainties can be quantified and analyzed.

Consider a simplified variant of the CAMRS system described earlier, in which the system merely senses the user's behavior via the smartphone's sensors and then periodically outputs its inference of what the user's current activity is. As we argue further in the next section, at some point a tester will want to perform system-level, black-box testing of the full application under realistic usage scenarios. This requires the tester to deploy the system on a smartphone and then perform a sequence of activities in order to determine how well the application infers each different activity. A test case for this application thus is a sequence of real user activities, and the test output is the associated sequence of inferred activities. And a test suite then is a set of such test cases, each one perhaps carried out in a different location, a different environment (such as indoors versus outdoors), with different activity sequencing, etc.

The tester knows not to expect perfect output results, by virtue of the system employing an imperfect machine learning-based classifier for the activity inference. The imprecision of the classifier is quantified by the *confusion matrix* produced by a machine learning tool as a normal by-product of its training of the classifier on a set of sensor data labeled with activity classes. The i, j-th entry of the confusion matrix is the probability that the classifier will classify a real instance of activity i as being an instance of activity j. Whenever $i = j$, the classification is correct; otherwise it is erroneous.

If the system is free of faults, and assuming that the classifier has been trained on a sufficiently broad range of sensor inputs, then over time the classifier should perform roughly as predicted by the confusion matrix. However, if the system contains a fault, then the *observed* performance of the classifier may deviate significantly from what the confusion matrix predicts. The problem for the tester of this system is to detect the presence of faults given only limited information about the acceptable uncertainty in the system's behavior (the confusion matrix) plus whatever the tester is able to observe by executing test cases on the system.

Clearly, a single activity is unlikely to reveal a fault in the system in this situation. Instead, test cases comprising large sequences of activities must be executed and observed in order to identify any statistically significant deviation from the uncertainty allowed by the confusion matrix. How can such a deviation be detected? And how meaningful is that deviation?

3. UNCERTAINTY IN TESTING TODAY

Two ways to deal with uncertainty are to xercise more control over the inputs provided to the unit under test, and to constrain the testing environment. For example, testing of the CAMRS application may utilize a smartphone emulator and a predefined music library. This approach is feasible and effective when the unit under test is small, especially with the availability of testing frameworks that provide extensive mocking and manipulation facilities that mimic the external resources or environment. As the unit under test becomes larger and more complex, there is a shift towards relaxing control of the execution, and thus towards validating the unit while operating more in the "wild". For the quadcopter stabilization, for example, there is no amount of simulation that can account for minuscule variations in the propellers' structure and placement; testing in the real physical environment is essential in order to assess the impact of these effects.

Clearly, attempting to control execution uncertainty is a laudable goal, but not a fully realizable one as the unit under test becomes more complex. Some uncertainties like those present in systems that deal with the physical world are *aleatoric*; that is, they cannot be known precisely because of their inherent variability and noise. *Epistemic uncertainties*—those that could be resolved with enough effort (e.g., over-engineering a sensor-based system to minimize inference errors about the environment)—are becoming more numerous and costly, as the systems we build increase in complexity. We need to learn to cope with the fact that, in general, the testing environment will not be able to control all sources of uncertainty.

A complementary way to deal with uncertainty at the other end of the testing process is to develop more sophisticated oracles. Determining whether a test passes or fails becomes much more difficult in the presence of uncertainty, as an output may be wrongly judged. There is an inherent risk for uncertain behavior to *mask* a fault if the oracles are relaxed too much to tolerate variations introduced by uncertainty, or to generate false positive results that waste a developer's time if the oracles are too narrow because they do not properly account for uncertainty.

Consider the Google Maps application. Given a test performed at a particular location, if the location reported by the application is off by one city block, then it may be that that is the best possible reported location given the input from the sources of location data, or it could be that the data fusion library is faulty, or that a hardware sensor is faulty, and/or that the program logic has a fault. Assuming that the test does not control all the sources of data, then the oracle must be able to differentiate between an acceptable albeit suboptimal difference between the true and reported location on the one hand, and a wrong location produced by some system fault on the other hand. Incorporating utility functions into oracles can provide better quantification of the strengths of a test result, but that requires having adequate knowledge of the input distribution and result likelihood. In general, more sophisticated oracles may help but the cost to enable such judgement may become prohibitive.

These difficulties are magnified by the lack of mechanisms to represent and quantify system uncertainty, and the inadequacy of existing testing measures (e.g., coverage criteria) to assess progress in exposing or controlling the sources of uncertainty.

4. FUTURE HANDLING OF UNCERTAINTY

The previous discussion leads us to identify a set of requirements for adequate handling of uncertainty in testing:

1. *richer testing frameworks* to specify input distributions instead of discrete inputs, and automated support to generate inputs from those distributions while favoring the discovery of uncertainty;
2. *probabilistic oracles* that can help distinguish between acceptable and unacceptable misbehaviors, and that enable the specification of likelihood of results; and
3. *richer models* to represent system and environment uncertainty, to connect uncertainty to test requirements and outcomes, and to enable automated uncertainty quantification.

Not all systems will impose these requirements, and not all techniques will satisfy them. In fact, we expect it will be necessary to develop a suite of techniques for dealing with uncertainty, selected according to system characteristics and the particular forms of uncertainty the system embodies, and based on the use of stochastic models and quantitative analyses.

As an illustrative example in this direction, one possibility we explore is to use *Hidden Markov Models* (HMMs) [12]. HMMs can be used to infer hidden properties of systems (such as the likelihood of latent faults) based on sequences of observations of the outputs produced by each state of the system. A HMM is a probabilistic state-machine model whose key elements are a set of states S and a set of possible observations O. As the machine transitions from one state of S to another, each successor state emits an observation from O. There are *transition probabilities* $A = [a_{i,j}]$ between all pairs of states i, j (which are assumed to be constant, and some of which may be zero), and there are *emission probabilities* $B = [b_{i,j}]$ for each possibility of making observation j in state i (also assumed to be constant). The sets S and O are known, but for many applications of HMMs, some of the probabilities in A and B are unknown and can be learned or estimated based on sequences of observations, allowing one to determine the probability of being in one of the (hidden) states.

Returning to the simplified CAMRS system, we can model it for testing so that each chosen activity within a test case is represented as a state in S, and the activity inferred by the system in response to the performed activity is represented as an observation in O . Thus, S and O both represent the same set of activities, and therefore $|S| = |O|$. In a testing scenario, the transition probabilities A can be controlled by the tester, and thus the states S are not hidden; for simplicity, we may assume that the tester chooses the next input in random fashion according to A. The confusion matrix gives us the emission probabilities B.

So how should one deal with the possibility of a fault? Suppose the system has been trained to recognize and classify four activities $\{S_1, \ldots, S_4\}$, and that the inferred activities are observed as outputs $\{O_1, \ldots, O_4\}$. Suppose further that the training of the classifier produced the following confusion matrix, which provides the emission probabilities for the states:

$$B = \begin{array}{c} \\ S_1 \\ S_2 \\ S_3 \\ S_4 \end{array} \begin{array}{c} \begin{array}{cccc} O_1 & O_2 & O_3 & O_4 \end{array} \\ \begin{bmatrix} 0.7 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.7 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.7 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.7 \end{bmatrix} \end{array}$$

For simplicity, we assume that the tester randomly selects, with the support of an automated tool and a pool of sample activities, each successive activity to perform with equal probability, and that any activity may follow any other activity. Therefore, we may assume that the HMM is always in equilibrium, with a stationary probability distribution of $\pi_{S_1 \ldots S_4} = (0.25, 0.25, 0.25, 0.25)$. Finally, to simplify the statistical reasoning, we assume that the test suite contains a single long test sequence of 4000 activities rather than several short test sequences.

Figure 1 depicts possible outcomes of testing four hypothetical implementations of the system, with each outcome represented in matrix form showing the number of times each activity S_i was observed as activity O_j. The progression from Figure 1a to Figure 1d shows increasing deviation from the frequencies predicted by the confusion matrix, with Figure 1a showing perfect correspondence and Figure 1d showing substantial deviation. Below each matrix is the p-value computed using Pearson's chi-squared test (with nine degrees of freedom) of the goodness-of-fit between the observed frequencies and the predicted frequencies. Thus, while the implementations tested in Figures 1a and 1b show a good fit with the expected frequencies, the outcomes of testing the implementations of Figures 1c and 1d are extremely unlikely, thus suggesting the existence of a fault in those implementations. Note that if behavior induced by faults in the system is indistinguishable from acceptable misbehavior via statistical testing, then arguably the faults are not sufficiently critical to impair the correct functioning of the system.

5. DISCUSSION

We are unaware of any previous research dealing with the problem of testing systems having acceptable misbehaviors.

Weyuker was the first to note the difficulty of creating oracles for "non-testable" programs, which in her case were mostly scientific programs (such as matrix multiplication routines) for which an oracle would need to perform the same computation as the system under test [21]. More recently, Raz and Shaw discussed the notion of "sufficient correctness" for systems for which one is willing to sacrifice some degree of reliability [16].

Several authors have applied statistical reasoning in testing, especially in software reliability testing, with its use of statistical sampling and reliability growth models [11, 15]. Arcuri and Briand discuss the lack of statistical rigor in experimental studies that use randomized algorithms such as random test generation, which produces a different test suite each time [3]. Some authors have studied Markov chains as a test generation mechanism [22, 25]. Other authors have employed HMMs or other Bayesian inference frameworks to model the debugging process and the rate of arrival of failures [5, 14, 23]. Xie et al. consider the problem of testing machine learning classifiers using metamorphic testing, but their focus is on testing the implementation of the machine learning algorithms rather than on testing systems that use the classifiers as components [24]. Note that all of these statistical approaches operate under the conventional assumption that outputs are either acceptably correct or unacceptably incorrect, without considering the third possibility of acceptably incorrect output.

One might try to relate uncertainty to nondeterminism, which has a long and honorable history in computer science. However, nondeterminism is essentially an abstraction mechanism that is used to simplify a program, model or computation, or to defer decisions about a model until later stages of its elaboration. Instead, given the likely need to obtain deep quantitative understanding of the impact of uncertainty in software testing, a more directly applicable framework would be the rich science of *uncertainty quantification* [19]. Our example using HMMs hopefully demonstrates the benefit of thinking along these lines.

HMMs provide just one of many possible approaches for dealing with uncertainty in software testing, and the example we presented does not even being to employ the full power of HMMs. A major feature of HMMs is the availability of algorithms for *learning* the parameters of an HMM (the state transition and observation emission probabilities) [12]. However, the most commonly used algorithms indiscriminately learn all parameters simultaneously, whereas for our problem it is likely that we need to develop new algorithms for learning a subset of the parameters. In addition, more specialized statistical techniques than our simple goodness-of-fit test are available for performing statistical inference on HMMs [1, 2, 13, 17], but they likely will need to be tailored to the characteristics of our problem. Perhaps the most challenging task is to determine how best to exploit the HMM formalism itself in order to model the existence of one or more error states.

In the CAMRS example, note that although the tester knows whether each observed classification matches the actual activity performed, this is insufficient as an oracle, because the tester also knows *a priori* to expect some incorrect classifications due to the imprecision of the classifier. In fact, there is no real oracle for this system, other than the confusion matrix. We therefore might extend the HMM model in various ways in hopes of increasing its fault localization power, such as extending S to include both a normal state and an error state for each activity, and/or extending O to allow for the explicit observation of failures (crashes, garbled output, hanging computation, etc.).

$$\begin{array}{ccccc} & O_1 & O_2 & O_3 & O_4 \\ S_1 & 700 & 100 & 100 & 100 \\ S_2 & 100 & 700 & 100 & 100 \\ S_3 & 100 & 100 & 700 & 100 \\ S_4 & 100 & 100 & 100 & 700 \end{array}$$

$p < 1.00$

(a) Perfect Correspondence

$$\begin{array}{ccccc} & O_1 & O_2 & O_3 & O_4 \\ S_1 & 720 & 90 & 95 & 95 \\ S_2 & 105 & 695 & 105 & 95 \\ S_3 & 98 & 98 & 705 & 98 \\ S_4 & 104 & 103 & 104 & 690 \end{array}$$

$p < 0.938$

(b) Close Correspondence

$$\begin{array}{ccccc} & O_1 & O_2 & O_3 & O_4 \\ S_1 & 770 & 60 & 85 & 85 \\ S_2 & 115 & 645 & 135 & 105 \\ S_3 & 88 & 88 & 755 & 68 \\ S_4 & 144 & 103 & 114 & 640 \end{array}$$

$p < 1.25 \times 10^{-15}$

(c) Notable Deviation

$$\begin{array}{ccccc} & O_1 & O_2 & O_3 & O_4 \\ S_1 & 800 & 100 & 50 & 50 \\ S_2 & 100 & 800 & 50 & 50 \\ S_3 & 150 & 150 & 550 & 150 \\ S_4 & 150 & 150 & 150 & 550 \end{array}$$

$p < 2.05 \times 10^{-68}$

(d) Significant Deviation

Figure 1: Testing of Four Possible Implementations of the Simplified CAMRS Application on Test Sequences of Length 4000

6. CONCLUSION

We have discussed the issue of uncertainty in testing, which we believe to be a fundamental challenge for development of future software systems. We have presented an extended example illustrating one particular form of uncertainty in software (the use of imprecise machine learning classifiers) and a statistical technique for determining the likelihood that the system contains a fault. Much work remains to be done in addressing this challenge. First we must characterize systematically the different ways uncertainty manifests itself in testing. Then we will be able to apply the full machinery of statistical reasoning and develop new statistical methods to improve the tester's ability to reveal faults masked by uncertainty.

7. ACKNOWLEDGEMENTS

This work was supported in part by Air Force Office of Scientific Research award FA9550-10-1-0406, National Science Foundation award NSF-1218265, and the National University of Singapore under grants R-252-000-478-133 and R-252-000-478-750.

8. REFERENCES

[1] R. M. Altman. Assessing the goodness-of-fit of hidden Markov models. *Biometrics*, 60:444–450, June 2004.

[2] T. Anderson and L. A. Goodman. Statistical inference about Markov chains. *Annals of Math. Stat.*, 28(1):89–110, 1957.

[3] A. Arcuri and L. C. Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proc. 33rd International Conf. on Software Engineering*, pages 1–10, May 2011.

[4] R. Calinescu, C. Ghezzi, M. Z. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Comm. ACM*, 55:69–77, May 2012.

[5] J.-B. Durand and O. Gaudoin. Software reliability modelling and prediction with hidden Markov chain. Technical Report INRIA/RR–4747–FR+ENG, INRIA, 2003.

[6] N. Esfahani, E. Kouroshfar, and S. Malek. Taming uncertainty in self-adaptive software. In *Proc. 19th ACM SIGSOFT Symposium and the 13th European Conf. on Foundations of Software Engineering*, pages 234–244, 2011.

[7] D. Garlan. Software engineering in an uncertain world. In *Proc. FSE/SDP Workshop on Future of Software Engineering Research*, pages 125–128, 2010.

[8] C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli. Managing non-functional uncertainty via model-driven adaptivity. In *Proc. 35th International Conf. on Software Engineering*, pages 33–42, May 2013.

[9] H. Jiang, S. G. Elbaum, and C. Detweiler. Reducing failure rates of robotic systems though inferred invariants monitoring. In *Proc. 2013 IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 1899–1906, 2013.

[10] E. Letier, D. Stefan, and E. T. Barr. Uncertainty, risk, and information value in software requirements and architecture. In *Proc. 36th International Conf. on Software Engineering*, pages 883–894, June 2014.

[11] M. R. Lyu. *Handbook of Software Reliability Engineering*. McGraw-Hill, 1996.

[12] S. Marsland. *Machine Learning: An Algorithmic Perspective*. CRC Press, 2009.

[13] L. A. Newberg. Error statistics of hidden Markov model and hidden Boltzmann model results. *BMC Bioinf.*, 10, 2009.

[14] A. Pievatolo, F. Ruggeri, and R. Soyer. A Bayesian hidden Markov model for imperfect debugging. *Rel. Eng. & Sys. Safety*, 103:11–21, July 2012.

[15] A. Podgurski, W. Masri, Y. McCleese, F. G. Wolff, and C. Yang. Estimation of software reliability by stratified sampling. *ACM Trans. Software Eng. and Methodology*, 8(3):263–283, July 1999.

[16] O. Raz and M. Shaw. An approach to preserving sufficient correctness in open resource coalitions. In *Proc. 10th International Workshop on Software Specification and Design*, pages 159–171, 2000.

[17] A. C. Titman and L. D. Sharples. A general goodness-of-fit test for Markov and hidden Markov models. *Statistics in Medicine*, 27:2177–2195, 2008.

[18] C. Trubiani, I. Meedeniya, V. Cortellessa, A. Aleti, and L. Grunske. Model-based performance analysis of software architectures under uncertainty. In *Proc. 9th International ACM SIGSOFT Conf. on Quality of Software Architectures*, pages 69–78, 2013.

[19] Uncertainty quantification. *Wikipedia*. Accessed from http://en.wikipedia.org/wiki/Uncertainty_quantification on June 26, 2014.

[20] X. Wang, D. Rosenblum, and Y. Wang. Context-aware mobile music recommendation for daily activities. In *Proc. 20th ACM Multimedia Conf.*, pages 99–108, Oct.–Nov. 2012.

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

[22] J. A. Whittaker and M. G. Thomason. A Markov chain model for statistical software testing. *IEEE Trans. Software Eng.*, 20(10):812–824, Oct. 1994.

[23] M. P. Wiper and M. T. Rodríguez. Bayesian inference for a software reliability model using metrics information. In *Proc. ADIS 2001 Workshop on Decision Support in Software Engineering*, Nov. 2001.

[24] X. Xie, J. W. K. Ho, C. Murphy, G. E. Kaiser, B. Xu, and T. Y. Chen. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software*, 84:544–558, Apr. 2011.

[25] B. Zhou, H. Okamura, and T. Dohi. Markov chain Monte Carlo random testing. In *2010 AST/UCMA/ISA/ACN Conferences on Advances in Computer Science and Information Technology*, pages 447–456, June 2010.