ELSEVIER

Information and Software Technology 42 (2000) 809–814

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.nl/locate/infsof

Improving a product with usage-based testing

A. Kouchakdjian[a,*], R. Fietkiewicz[b]

[a]*Teradyne SST, 44 Simon Street, M/S 33 Nashua, NH 03060, USA*
[b]*FAA Technical Center, Atlantic City, NJ 08405, USA*

Abstract

This paper presents the introduction and application of a usage-based testing approach for the acceptance of an aircraft traffic flow management system by the FAA. The FAA needed an approach to provide more effective testing than that typically achieved in their requirements-based acceptance testing. Testing in a manner more consistent with operational usage was desired. A testing technology that was new to the organization was introduced and applied successfully. The approach resulted in a greatly improved product being released to the field, a cost-effective approach for testing, and a commitment to apply the technology across the organization.

Keywords: Usage-based testing; Statistical usage testing; Black box testing; Models

1. Introduction

This paper reports on the application of usage-based testing technology to an aircraft traffic flow management system that the FAA was to evaluate for acceptance. The software was developed by contractors, but would be fielded and supported by the FAA. There were concerns about the reliability of the system. The FAA staff determined that their traditional requirements based acceptance test would not ensure high field reliability and that they needed a better acceptance testing method. The FAA staff decided upon Statistical Usage Testing (SUT) as the better method. SUT is a model based testing approach, where tests are generated from a model that is intended to represent all possible operational usage of the software system at a suitable level of abstraction. Using SUT, the FAA could test the product in a manner similar to the operational use of the product in the field. The techniques were applied successfully, resulting in a number of changes and corrections to the software. After the updates and additional testing, the product was released to Air Traffic Control Centers for use. The reliability of the product as assessed in SUT has been validated by the field experience.

A number of lessons were learned from application of this technology on the project, which are reported here. These will provide significant insight on application of SUT to large software systems. Additionally, the success in applying the methods to the aircraft traffic flow management system has resulted in the application of the methods to other systems.

The remainder of this paper will discuss the project and technology. Subsequently, it will delve into the schedule and project results. There will be a separate lessons learned section, which will provide insight on future applications of the technology. Finally, conclusions will be drawn.

2. Project overview

The project under consideration was an air traffic flow management system to be fielded in the Air Traffic Control centers. Contractors developed this product between 1993 and 1998. The FAA's role was to test and accept the system. After acceptance, FAA responsibility included fielding the product and post-delivery support. As a result, FAA needed to ensure as that the product was of acceptable reliability before accepting responsibility from the contractor. The fielding date for the system constrained the testing time available to the FAA (additional schedule details are provided in the Project Timeline section). The main computer–human interface portion of the system was more than 500,000 lines of C code running on workstations using Unix, with the overall system size in excess of 1.2 million lines of code. The size of the system, the limited schedule, and the complexity of the computer–human interface made it clear that a traditional acceptance testing (requirements testing) approach would be insufficient.

In 1997, FAA management focused on determining a test method to evaluate and accept the software. They made the

* Corresponding author.
E-mail address: ara@sst.teradyne.com (A. Kouchakdjian).

0950-5849/00/$ - see front matter
PII: S0950-5849(00)00126-9

Table 1

Test planning	Define test goals
	Stratify testing to meet goals
	Define boundary of system under test
	Define stimuli that cross boundary
	Define a 'use' of the system under test
	Allocate test budget to testing goals
	Determine approach to testing for each goal
Usage model development	Build model structure
	Assign transition probabilities
	Analyze model
	Validate and refine model as necessary
Generate tests	Generate test scripts
	Process scripts into test cases
	Study coverage of test cases
	Compute expected results
Test execution	Run tests on the system
	Determine success or failure of each test case
	Record pass/fail for each test
Certification	Compute quality measures
	Compute stoppage criteria
	Make release decisions

decision to adopt a usage-based approach, Statistical Usage Testing (SUT). The belief was that requirements-based testing could also be fulfilled with this approach. Additionally, by mimicking field usage (in all its breadth) as much as possible, the hope was that the failures a field user might have uncovered would be found and removed prior to fielding.

3. Technology overview

The intent of SUT is to exercise the system under test in the same manner as the product is used in the field. SUT is a black box testing approach [1], which provides the opportunity for objective measurement of product reliability. The high level SUT process is provided in Table 1 [3,4,5].

Since SUT is a black box testing approach, defining a boundary of the system under test is critical. Additionally, since it is a usage-based approach, the definition of a 'use' and its relationship to a field definition of use of the system is also important. These are sometimes difficult decisions, but once made correctly, they are leveraged throughout the remainder of the process.

The models developed represent the possible usage of the system under test. Models are developed as Markov Chains, which represent possible transitions (user events) that take a scenario from the beginning to the end of a use. A Markov Chain can be viewed as a class of state processes where decisions are dependent only upon the current state, not upon information available at other states. In a Markov Chain for software usage, a state represents a user situation, where a user has a certain set of possible inputs to apply to the software and a distribution representing the relative likelihood of inputs. Transitions from each state represent application of the inputs. Transitions have likelihoods and events, which are the actions that move the process from state to state. Likelihood's are probabilities (which sum to one across all transitions leaving a state). The models are created through analysis of product documentation and discussion with domain experts. The information gained is used to create the concrete model of how the product can be used. Typically, the models can generate all possible inputs, not just the 'expected' inputs.

Since the completed model is a mathematical object, it can be analyzed by mathematical methods. Analysis provides a variety of useful data such as expected test case length, expected number of visits for a particular state. Since Markov chains are memory-less, models that are at a low level of abstraction tend to be very large. Models that are at a higher level of abstraction tend to be smaller, but rely on post-processing of test scripts to provide the details necessary for the tests to be executed.

Test scripts are generated as a Monte Carlo simulation. Though any test (any path through the model) can be generated at any time, one would expect the more probable tests to be generated more often. In many cases, though not in this case study, significant work is necessary to convert test scripts into test cases. In this case study, the models are very granular, leaving little conversion necessary.

Tests are executed. Since the set of tests represent a random sample of expected field use, testing can be viewed as a statistical experiment. Tests that successfully execute provide positive product reliability information; failures provide negative information. The tests also provide information on product coverage, the sufficiency of testing. Reliability and stopping information provide the inputs necessary to make a product release decision. Note that in this case study, reliability measures were not used as release criteria; only coverage and failure density information was used to drive product release.

4. Project timeline

The project to test the aircraft traffic flow management system was conducted under the timeline shown in Table 2.

5. Project results

Consultants who had no prior knowledge of the application domain developed the models. As a result, some time spent in model development was actually spent in understanding the required behavior of the system. The boundary of the system was defined as the boundary between the human user and the software, essentially the keyboard, mouse and screen. A 'use' was defined as a set of commands

Table 2

Evaluating alternatives and determining the testing approach to take	Spring 1997	Done by FAA
Initial training in SUT	Summer 1997	Training given by consultants. Students were FAA staff and contractors
Initial modeling	Summer 1997	Done by consultants, with FAA support
Testing of system	Spring 1998–Summer 1999	Done by FAA staff and contractors. Corrections made by contractors
Additional training	Spring 1998	Training given by consultants. Students were FAA staff and contractors
Extensions and modifications to models	Summer 1998–Spring 1999	Done by FAA staff and contractors
Release of product to the field	Spring 1999	Fielding done by FAA staff and contractors
Modeling of other systems	Summer 1998–present	Done by FAA staff and contractors

that made logical sense from a human user perspective. Essentially, these could be viewed as the set of inputs that would be given during a session on the system. The level of granularity of the inputs was essentially on the 'mouse click' level. Pressing an enter button, typing in a file name, and selecting from a top-level menu were inputs to the system. Modeling at this level would ensure that functionality would not be missed and that important information would not be abstracted away. This also led to models being larger than would have been the case with more abstract inputs. The size of the models was somewhat mitigated and managed with a model composition utility that would allow small models to be created separately and then composed into a single large model. Tools were available to support the modeling, test generation, and test recording and evaluation efforts.

Upon completion of some initial model planning efforts, it was clear that a 'complete' model of system behavior would result in a model of over 65,000 states. This was not desirable for a number of reasons. First, even with a large number of tests, coverage of the model would be low. Although many of the states and transitions would be neither probable nor critical, the external perception of exercising maybe 10% of the testable functionality would be problematic. Secondly, many of the states were of extremely low probability and in low criticality areas of use. Expending a great deal of testing effort on those areas would have been an inefficient use of resources. Finally, even though this was a fair assessment of the size of the testing problem, there were insufficient resources available to do an effective test on this full extent of functionality. As a result, the decision was made to create a set of seven top-level models. The seven models corresponded to seven largely independent functional areas that the system would support. This made the resulting model development more easily manageable. However, the concern was expressed that separating behavior into seven models would preclude the interaction between these areas of functionality. This was determined to be low risk, since the functional areas were arguably independent. Nonetheless, there was a potential risk, as a result of the tradeoff between testing effort available and the size of the testing problem. Field data following project release show that problems from model interaction did not occur.

The seven major models were each created as compositions from a set of nearly 60 models. The sizes of these models ranged from five to forty states. These models corresponded generally to screens in the application system. Since every screen was modeled, each model was very granular, and in the major models resulting from composition of sub-models each major model had the full set of possible sequences that could occur within the scope of that major model. The major models each had between 117 and 312 states, with approximately nine times as many transitions as states. The models and their sizes are listed in Table 3.

Once the model structure was defined, the models were taken to the FAA customer to define probabilities. The FAA customer put probabilities on the two or three most or least likely transitions. The remaining probability density was uniformly distributed among other available transitions. This was believed to be sufficiently accurate, since one is not able to precisely predict future behavior. Once the models were completed, they were delivered to the FAA.

For actual testing, approximately 20 tests, consisting on average of 15–20 events, were generated and executed per major model. Tests were generated and executed by FAA and contractor staff. Failures and successes were recorded. All failures were reported to the contractors for correction. Based upon failures observed and the degree of state and arc coverage of the models achieved by the initial sets of 20 or so test cases, additional tests were generated. Once an entire suite of tests was executed failure-free, state coverage was achieved, and an acceptable level of arc coverage occurred, then the system was suitable for release to the field. Otherwise, the correction and re-test loop would continue. The type, severity, and point of occurrence of all failures were recorded.

The initial models were developed by consultants and given to the FAA. Subsequently, FAA and contractor staff extended of the models. Additionally, new models were

Table 3

Model	Number of states	Number of transitions
Model A	180	1263
Model B	117	1015
Model C	259	2264
Model D	136	916
Model E	264	1891
Model F	312	2990
Model G	138	1154

created (or adapted from existing models) to address other sub-systems by FAA and contractor staff. There were no reported problems where the models originally developed by the consultants were found to be difficult to use or re-use by the FAA and contractor staff.

6. Lessons learned

This section defines a number of the lessons learned from the project [2]. The authors believe that each lesson provides some important insight into the technology and its applicability to systems of this type. The lessons learned are divided into two sections: project lessons learned and broader insights.

6.1. Project-level lessons learned

- The size of the testing problem was made concrete and visible—The determination that a 'complete' state model would include over 65,000 states was an important observation. It not only drove the solution strategy to creating multiple models, but it also made concrete exactly how big the testing problem was for this project. Generally, it is not clear how a test is usefully measured before being performed. In some projects, one can estimate staff months, in other projects one estimates a number of tests that can be run. The size of the model (in states, arcs, or some combination thereof) is a way of concretely defining how much testing can be done to exercise the system. This measure is similar to a line of code or complexity measure as a rough assessment of the size of a code product.
- Test cases appeared sufficient—The starting suite of 20 tests (with an average of 15–20 events) per model appeared somewhat arbitrary. However, field use and tester experience/comfort demonstrated that the initial suites and any tests run subsequently to be sufficient. The testers noticed that the tests would exercise a broader scope of functionality than they might typically run (what one may call tester 'bias'). Additionally, the tests were covering a sufficient scope of functionality to support release decisions.
- Tests quite efficiently executed—The level of granularity of the tests made it easy for human test execution. In this case, the expected behavior was clear enough to the testers (a number of whom had air traffic controller experience) so that they could validate the tests easily, by direct observation. It was not necessary to generate expected outputs and place them with each test. The test cases, along with tester insight and project documentation, were sufficient to execute and validate. Projects that have a similar approach to test execution and validation should observe the same results. Projects that have complex validation criteria will see a smaller benefit.
- Elimination of paper—Test cases were generated by tool and displayed on a screen next to the running software. Errors were noted on the on-line test cases. As a result, all testing efforts were conducted on-line, which eliminated much of the need for paper. This finding was not at all predicted or expected prior to the project.
- Testing approach found effective—The testers observed many failures, which resulted in a number of faults being isolated and corrected. The resulting updated products were greatly improved in quality. The product has also exhibited high reliability in the field. As a result, the testing goals were achieved by the selected method.
- In this case, multiple independent models worked for the full set of behavior—Avoiding the single massive model by developing seven major models was a risk. The FAA traded off tractability of the test efforts with the risk that the areas not covered would be error prone. In this case, the FAA was correct and the approach did not negatively affect the project. However, these trade-offs must be carefully considered in the future if one is confronted with a similar situation. It may well be that partitioning the testing problem and ignoring interaction between the partitions may introduce risk. The insight from this project is that partitioning is possible, however it must always be done cautiously. In many projects, these sorts of trade-offs are made. Usage modeling approach allows the engineers to concretely see the full scope of functionality, allowing a more informed decision when prioritizing test.
- Found large number of requirements and 'what if' issues—When building the models, the testers uncovered a large number of situations where the requirements were incomplete. These often fell into the 'what if' situation, where a requirement defined behavior when a certain event occurred correctly, but did not define what would happen if the event occurred incorrectly. A large number of issues were fed back to the requirements team for resolution. These issues led to a number of changes in the requirements and in the code. Many of the possible scenarios had not been considered. The state modeling techniques support the uncovering of requirements ambiguities or omissions since the models are relatively concrete. When modeling, one tries to map a particular event from one state to another. The event is precise and the states are observable to a user. So the question of desired behavior can be precisely formed. This is much

more precise than the typical English text in a requirements document.
- Observed system reliability growth—The system evolved from the lowest reliability (in initial test) of a set being validated to the highest reliability. This was a result of failures being discovered and then removed. For that reason, the other systems, which did not use SUT, are now being modeled and tested with this approach. SUT appeared to test a broader scope of functionality than may have otherwise been tested, resulting in more failures revealed. The measure of 'lowest' reliability to 'highest' reliability is qualitative in this case; the testers believe that it is so. The results from field usage also reinforce this conclusion.

6.2. Broader insights

- Model based approach eliminated human bias—Testers often focus their testing efforts on what they believe is important. This may be boundary cases or bizarre behavior or normal use. By using SUT, the perspective that each individual had was encoded into the models. When tests were automatically generated, they represented the union of everyone's experience, resulting in an effective test. The alternative would be pockets of effective testing, each with different criteria. The model based approach supports encoding expertise and making it generally available (in this case in the form of tests).
- Can test 'forever'—If tests are being generated manually, one has only as many tests as have been created. If models are created, then tests are generated by the push of a button. The number of tests is equal to the number of tests requested at any one time. The modeling approach allows a large number of tests to be generated. If the model has a loop or cycle in it, then the number of possible tests is infinite. Instead of making the number of tests generated to be a direct proportion of the effort expended, modeling allows one to invest once in modeling but to be able to generate unlimited numbers of tests inexpensively. In larger projects and those with longer life cycles, model based test provides a significant savings in cost.
- Change in testing paradigm—The test mentality of the FAA team was changed. Generating tests one at a time (trying to 'beat' the developed system) is very different from building models representing system usage. The first is more of an implementation issue, while modeling is more of a design issue. This shift in perspective takes some time to effect, but allows testing issues to be understood in successive levels of refinement. Better mental 'building blocks' should lead to better project performance.
- Size and complexity of models relates to complexity of functionality—There is a strong relationship between model complexity and the complexity of the application implemented in software. A number of 'indicator' measures currently exist to quantify model complexity, as do measures to quantify software complexity. The relationship between the complexity of the application and the complexity of testing the application is an area for greater research and understanding.
- Uniform probabilities are good and mathematically neutral, when one is not sure of the usage—Probabilities were not assigned to every transition. This was not practical in this case, and would not be practical in many projects. One must remember, however, that the probabilities are intended to profile operational usage of this system in future field use. The probabilities are a mechanism for quantifying expected future usage. Since precise future use cannot be predicted, attempts for excessive levels of precision are most likely inefficient. The marginal gain in 'better' transition probabilities provides little benefit for effort expended in producing better operational profiles. This project was very effective in assigning some probabilities and leaving others to be uniformly distributed by the modeling tool. The technique has been used on other projects and should continue to be used.

7. Continuing activities

The positive accomplishments from the first pilot have resulted in continuing interest in using these methods.

- The FAA has maintained and re-used the models on an ongoing basis with successful results. The ability to save and re-run existing tests, as well as generate new tests almost at will has proven extremely beneficial to the testing process.
- The FAA has modeled an additional sub-system on their own (without help from consultants). The intention is to model additional sub-systems as time and resources are available.
- The business case for using SUT has become clearer in the FAA. Previously, test cases were extremely difficult to generate and maintain over time. Simple rearrangement of the computer–human interface would result in a tremendous amount of labor to re-write test procedures. SUT allows the process to be performed easily and effectively.
- Many of the quantitative benefits of the testing approach have yet to be leveraged. That information will be inserted into the tool to provide additional data for future projects.
- The FAA has also seen that Markov chains that are created in a granular manner result in huge state models. To manage the size of the state space, additional technologies and tools (such as the use of extended finite state machines using TestMaster from Teradyne-SST) are being considered. If the state space issue can be

addressed without negative side effects, then this testing approach can be done even more efficiently.

8. Conclusion

The FAA was able to use a usage-based testing approach to more effectively evaluate and accept contractor-developed software. The technology used was easy to adopt and provided better tests. Greater visibility in the testing approach and improved information in decision making were also concrete benefits. The result was improved testing and a higher reliability product being fielded. Failures were found in the laboratory and were removed before the software got to the field. Usage of the technology was also perceived positively by project staff.

Having achieved these results, the FAA will continue to use this testing approach to model and test other systems. The results are very positive, but the FAA will continue to improve what it does to more effectively fulfill its mission.

References

[1] B. Beizer, Black-Box Testing, Wiley, New York, 1995.
[2] A. Kouchakdjian, R. Fietkiewicz, Experience using statistical usage testing at the FAA, Presented at Quality Week 1999, San Jose, CA, June 1999.
[3] S.W. Sherer, A. Kouchakdjian, P.G. Arnold, Experience using cleanroom software engineering, IEEE Software (May) (1996) 69–76.
[4] S.W. Sherer, Process improvement based on cleanroom software engineering, Software Engineering Process Group Conference, Software Engineering Institute, Pittsburgh, PA, May 1996.
[5] G.H. Walton, J.H. Poore, C.J. Trammell, Statistical Testing of Software Based on a Usage Model, in: J.H. Poore, C.J. Trammell (Eds.), Cleanroom Software Engineering: A Reader, Blackwell, Cambridge, MA, 1996, pp. 331–344.