Test-Driven Specification: Paradigm and Automation

Edward L. Jones
Florida A&M University
Tallahassee, Florida
850-599-3050 USA
ejones@cis.famu.edu

ABSTRACT
This paper introduces *test-driven specification*, whereby the specification process is aided by the use of test cases. We also introduce an automated tool, the test-driven specification assistant (TDSA), which supports this approach. Test cases reveal specification anomalies such as incorrectness, incompleteness and ambiguity. Specification-based test coverage criteria are applied to reveal deficiencies in the set of test cases. Decision tables are used as a lightweight specification language capable of modeling black-box and Mills' state box specifications.

Categories and Subject Descriptors
D.2.4 **[Software Engineering]**: Programming environments – *interactive environment.*

General Terms
Design, Verification.

Keywords
Specification-driven testing, decision table, model-based testing, state box, functional coverage.

1. INTRODUCTION
Test-driven development, popularized by Extreme Programming, engages the developer in the dual role of tester and developer. When combined with pair programming, evidence shows improved software quality [10]. Quality improvements can result from a number of factors, including the positive effects of pair programming, the focus on testing before coding, or the type of application being developed.

This paper introduces *test-driven specification*, in which the specification process is aided by the use of test cases. We also introduce an automated tool, the test-driven specification assistant (TDSA), which supports this approach. Test data are used to reveal specification anomalies such as incorrectness, incompleteness and ambiguity. Specification-based test coverage criteria are applied to reveal deficiencies in the set of test cases. We seek the same type of synergism between specification and testing as is suggested by test-driven development.

ACM SE'06, March, 10-12, 2006, Melbourne, Florida, USA

The choice of specification language or notation is important. Despite current advances in formal methods in software engineering, widespread use has not yet occurred. *Lightweight* formalisms, which provide intuitive and visual representations, are preferred. In this paper, we use the decision table as the lightweight specification language of choice. In addition to being intuitive and graphical, decision tables are capable of modeling simple black-box functional behavior as well as more complex, state-based behavior like Mill's state box specifications [5].

Test-driven specification is also motivated by Hoffman's *specification-by-example* approach described in [7], "Prose + Test Cases = Specification." They faced the problem of providing clear and accurate documentation for API software. Their solution was to provide prose descriptions augmented with test cases to improve the precision of the prose description. Although formal specifications provide precision and the potential for automated analysis, few developers are skilled in writing or reading them.

A motivation of this work is the belief that the power a specification notation or model is not realized unless the model can be given "life", i.e., be made dynamic so that the software developer can interact with the model. Interaction is essential to the development of a high quality model, and should be employ representative usage examples to communicate intent. Interaction tools facilitated the adoption of Parnas' SCR methodology [4].

We propose that the model be made executable and capable of emulating the cause-effect/stimulus-response intents encoded in the model. The executable model, when presented with representative inputs (stimuli), produces responses specified in the model [2]. An automated toolset provides analysis capabilities such as those found in model checking, e.g., the ability to detect anomalies in the model, and those found in testing tools, e.g., computing coverage measures for a given set of test stimuli. Our approach is unique in its objectives – to unite specification and testing – and in our desire to provide open source tools to a broad community of users.

2. AN EXAMPLE
We include an example of a state-based specification (Figure 1), and show the evolution of the decision-table specification and test set. We discuss specification anomalies – incompleteness, and ambiguity and incorrectness, and give examples of test data that reveal these anomalies. We also show how the decision table based coverage criterion [1,13] is used to reveal deficiencies in the test data set.

Figure 1. Black-Box Schematic for Open Garage Door

Specification for Garage Door Keypad: The garage door opener requires the sequence of keystrokes 2-6-3. Once this sequence is entered, the door opens, and all subsequent keystrokes are ignored. An audible beep is emitted for each keystroke except the keystroke that completes the open sequence, for which the audible message "DOOR OPENING" is emitted.

The TDSA toolset includes a library of utilities along with the generator tool, tdsaGEN. Given a decision table dtb_P for a program P, tdsaGEN generates a test driver and a self-instrumenting executable decision table class. The instrumentation gathers data from which three reports are generated: the *test coverage* report identifies the rules satisfied by each test case; the *specification anomalies* report identifies missing or ambiguous rules (one test case satisfies multiple rules); and the *test oracle* report, which shows, for each test vector, the expected values of the response variables and state variables. The specification of oracles need not be computational, but may instead be annotation describing the succinct sequence of responses expected from the system.

3. RELATED WORK

As a simple formalism, decision tables have an intuitive appeal and are suitable for reasoning about the cause-effect behavior of software. Studies have demonstrated that decision tables are an effective specification notation, especially as specifications grow in size and complexity [3,9].

Vanthienen and others have done much to advance the use of decision tables in knowledge engineering, artificial intelligence, and decision support systems [6,11,12]. Vanthienen's Prologa tool automates the creation, verification, and optimization of decision tables. Because decision tables are models, they can be malformed or otherwise exhibit anomalies such as redundancy, ambiguity or incompleteness. Prologa and the other decision table tools place certain restrictions on the size and style of decision table supported and assumptions about how to handle the combinatorial explosion of rules.

Although there are limitations to the applicability of decision tables, typical software contains many components whose behavior can be specified as a decision table. Textbooks on software testing routinely introduce decision tables as a technique that can be used as the basis for test case design [1].

4. ONGOING AND FUTURE WORK

The TDSA toolset establishes an important link between specification, which occurs early in the software lifecycle, and testing, which occurs at the end. The toolset can improve both the quality of test data and the quality of the specification used to develop the software and to create test cases.

In the future, we will consider the use of random test data generation to drive the specification refinement process. We are extending this work to the specification of object-oriented software as collection of models (for methods) that share a common (object) state. We are also investigating interfaces to pre/post-condition and state-based specifications. Finally, we are working to integrate TDSA into an Eclipse environment.

5. ACKNOWLEDGMENTS

This work was partially supported by NSF Minority Institutions Infrastructure Grant #0424556, and by a grant from the Medtronic Foundation.

6. REFERENCES

[1] Binder, R.V., *Testing Object-Oriented Systems: Models, Patterns, and Tools*, Addison-Wesley, 1999.

[2] Clark, M., "Support for Automated Specification-Based Testing Using Executable Decision Tables," *Proceedings of ADMI 2005*, Rincon, Puerto Rico, Oct. 13-15, 2005, 16-25.

[3] Glora, N., G., Pu, H. and Rom, W.O. Evaluation of Process Tools in Systems Analysis, *Information and Technology 37*, 2 (1995), 119-126.

[4] Heitmeyer, C., Using the SCR* Toolset to Specify Software Requirements, *Proceedings of the 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques*, 1998, 12-13.

[5] Hevner, A.R. and Mills, H.D., Box structured methods for system development with objects, IBM *Systems Journal Volume 32*, No 2 (1993), 232-251.

[6] Hewett, R. and Leuchner, J., Restructuring decision tables for elucidation of knowledge, *Data & Knowledge Engineering 46* (2003), 271–290.

[7] Hoffman, D. and Strooper, P., Prose + Test Cases = Specification, *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 34'00)*, 239-252.

[8] Jones, E.L., Automated Support for Test-Driven Specification, *Proceedings of the 9th IASTED Conference on Software Engineering and Applications (SEA2005)*, November 14-16, 2005, Phoenix, Arizona, 218-223.

[9] Lagenwalter, D., Decision Tables – An Effective Programming Tool, P*roc. 1st SIGMINI Symposium on Small Systems*, 1978, 77-85.

[10] Maximilien, E.M. and Williams, L., Assessing Test-Driven Development at IBM, *Proceedings of the 25th International Conf on Software Engineering*, 2003, 564-569.

[11] Vanthienen, J. and Dries, E., Illustration of a decision table tool for specifying and implementing knowledge based systems, *International Journal on Artificial Intelligence Tools 3* (2) (1994) 267–288.

[12] Vanthienen, J., Dries E. and Keppens, J., Clustering Knowledge in Tabular Knowledge Bases, *Proceedings of the 8th International Conference on Tools with Artificial Intelligence (ITTAI '96)*, 1996, 88-95.

[13] Zhu, H., Hall, P. and May, J., Software Unit Test Coverage and Adequacy, *ACM Comp. Surveys 29*, 4 (1997), 364-42.