ReqsCov: A Tool for Measuring Test-Adequacy Over Requirements [*]

Matt Staats[1], Weijia Deng[1], Ajitha Rajan[1], Mats P.E. Heimdahl[1], Kurt Woodham[2]
[1]Department of Computer Science and Engineering, University of Minnesota
[2]L-3 Enterprise IT Solutions, NASA IV&V Facility

Abstract

When creating test cases for software, a common approach is to create tests that exercise requirements. Determining the adequacy of test cases, however, is generally done through inspection or indirectly by measuring structural coverage of an executable artifact (such as source code or a software model). We present ReqsCov, a tool to directly measure requirements coverage provided by test cases. ReqsCov allows users to measure Linear Temporal Logic requirements coverage using three increasingly rigorous requirements coverage metrics: naïve coverage, antecedent coverage, and Unique First Cause coverage. By measuring requirements coverage, users are given insight into the quality of test suites beyond what is available when solely using structural coverage metrics over an implementation.

1 Introduction

Currently, testing is the most common method of evaluating software correctness with respect to its required behavior. One general approach to creating test suites is to create tests that exercise the software requirements—black-box testing—where the system is tested without knowledge of the system's internal structure since software requirements are (generally) defined in an implementation-independent fashion [2]. When creating test suites to exercise requirements, we would ideally determine the adequacy of a test suite directly over our set of requirements; unfortunately, this is not currently done in practice.

To support adequacy measurements using requirements, we have developed *ReqsCov*, a tool to measure the requirements coverage provided by a test suite. The use of ReqsCov is similar to other tools measuring structural coverage over executable artifacts (e.g., tools measuring branch coverage and MC/DC coverage [3]), and allows developers to measure the adequacy of a test suite based upon how well the test suite exercises requirements, rather than (or in addition to) the level of structural coverage over the implementation achieved by the test suite. This paper presents *ReqsCov* and describes its functionality. Background information on the coverage metrics used and concept of requirements coverage in general can be found in [1] and [6].

[*]This work has been partially supported by NASA Ames Research Center Cooperative Agreement NNA06CB21A, NASA IV&V Facility Contract NNG-05CB16C, and the L-3 Titan Group.

2 Problem Overview

Black-box testing using requirements is a longstanding and common approach to testing software [2], to the extent that manual creation of black-box tests is common even in introductory computer programming courses. In practice, however, requirements are not used to *measure* the adequacy of a black-box test suite [6], as the automation to perform such a measurement is not generally available. Instead, the adequacy of a test suite is inferred by measuring structural coverage of an executable artifact, such as source code [2, 3] or a software model [1].

This presents a number of problems. First, if the executable artifact is incomplete (i.e., missing functionality) a test suite may provide good structural coverage while failing to expose the missing functionality. Conversely, if a test suite provides poor structural coverage, a developer must determine if (a) requirements are missing, (b) code exists in the executable artifact unrelated to the requirements, or (c) the set of tests is simply poorly constructed. Finally, this approach requires access to the internals of an executable artifact, thus forcing the executable artifact to be completed before the adequacy of the test suite can be determined.

A more direct method of evaluating the adequacy of a test suite is to directly measure its coverage over the software requirements themselves. This approach offers the ability to evaluate the adequacy of a test suite independently of the software system's structure and in an manner directly related the software system's requirements, thus avoiding the pitfalls listed above.

3 *ReqsCov* Description

ReqsCov exists in two forms: as a command line tool (written in Standard ML) and as an Eclipse plugin for the Eclipse IDE. Both forms offer the ability to measure the requirements coverage achieved by a test suite. Currently, three requirements coverage metrics are supported: *Unique First Cause* (UFC) coverage, *antecedent* coverage, and *naïve* coverage.

Our tool operates using three inputs: a test suite (given in a simple comma separated format), a set of requirements expressed as Linear Temporal Logic (LTL) [4] properties, and a list of the requirements coverage metrics the user wishes to measure. Our tool outputs two pieces of information: the coverage percentage for each coverage metric measured, and the list of test cases satisfying each test obligation. The list is expressed as an $n \times m$ matrix, with n test cases and m obligations. The n^{th} test case satisfies the m^{th} obligation if $\{n, m\}$ is 1; if the n^{th} test case does not satisfy the m^{th} obligation,$\{n, m\}$ is 0.

ReqsCov operates as follows. First, for each requirement, *ReqsCov* generates obligations corresponding to different paths satisfying the requirement, where each obligation corresponds to a different method of satisfying the requirement. The tool then measures how many obligations are covered by the test suite. The number of covered obligations versus the number of total obligations is used to generate the coverage percentage of the test suite. (Note that each coverage metric is measured separately.) These coverage metrics and the obligation generation process are fully explained in [6].

When using the Eclipse plugin, users can graphically select a set of LTL requirements, a test suite, and one or more requirements coverage metrics. The requirements coverage(s) achieved by the test suite is (are) then graphically displayed as a percentage, while the full list of obligations and the tests cases which satisfy them can optionally be saved to a file. The plugin also allows users to edit LTL requirements in Eclipse with syntax highlighting.

3.1 Linear Temporal Logic

We have chosen to work with requirements formalized as LTL properties as our primary industrial collaborator (Rockwell Collins Inc.) has found them to be a good match with how their natural language requirements are written. Thus, *ReqsCov* has been developed to measure requirements coverage of high-level LTL requirements; nevertheless, the tool could easily be extended to support requirements captured using other notations, provided a definition of requirement coverage corresponding to the notation exists. For example, we have used *ReqsCov* to develop coverage measures over a subset of Live Sequence Charts.

3.2 Generation of Obligations

Both UFC coverage and antecedent coverage require that tests exercise requirements in interesting (i.e., non-trivial) ways. For example, consider the requirement "$\mathbf{G}(a \rightarrow b)$." If a is never true in the test suite, this property is trivially satisfied, and thus the test suite does not truly test if this requirement holds. For UFC and antecedent coverage, we therefore generate an obligation that states that a must hold true in some test case. These obligations are generated as LTL formulas defining the nature of the test cases needed to satisfy the obligations; *ReqsCov* then uses these LTL obligations to measure the requirements coverage achieved by a test suite.

3.3 Coverage Over Finite Paths

LTL formulas, such as "$\mathbf{G}(a \rightarrow b)$", are formulated over infinite paths whereas test cases are by definition finite. Manna and Pneuli [5] examine the use of LTL over systems which finite (i.e., terminating) paths. They define two forms of LTL semantics: *strong* semantics which are simply standard LTL semantics, and *weak* semantics corresponding to weaker semantics useful for finite paths. Using these definitions, the above formula is always *false* under strong semantics, as we cannot state that it is true globally. However, under weak semantics the formula is *true* if in each state we are aware of the formula is true.

Intuitively, we can see that strong semantics, being defined over infinite paths, are not useful for our purposes. We therefore interpret LTL formulas using *weak* semantics. The rationale for this is fully explored in [6].

References

[1] Paul E. Ammann and Paul E. Black. A specification-based coverage metric to evaluate test sets. In *Proceedings of the Fourth IEEE International Symposium on High-Assurance Systems Engineering*. IEEE Computer Society, November 1999.

[2] Boris Beizer. *Software testing techniques*. Van Nostrand Reinhold, New York, 2nd edition, 1990.

[3] J.J. Chilenski and S.P. Miller. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal*, 9:193–200, September 1994.

[4] Orna Grumberg Edmund M. Clarke and Doron A. Peled. *Model Checking*. The MIT Press, 1999.

[5] Z. Manna and A. Pnueli. Temporal verification of reactive systems: Safety. Technical report, Springer-Verlag, New York, 1995.

[6] M.W Whalen, A. Rajan, and M.P.E. Heimdahl. Coverage metrics for requirements-based testing. In *Proceedings of International Symposium on Software Testing and Analysis*, July 2006.