Removing Requirement Defects and Automating Test

Mark R. Blackburn, Robert Busser, Aaron Nauman

Organizations face many problems that impede rapid development of software systems critical to their operations and growth. This paper discusses model-based development and test automation methods that reduce the time and resources necessary to develop high quality systems. The focus is how organizations have implemented this approach of model-based verification to reduce requirements defects, manual test development effort, and development rework to achieve significant cost and schedule savings.

1 Introduction

Testing accounts for 40 to 75 percent of lifetime development and maintenance costs [Bei83; GW94]. Recent studies indicate that 50 percent of test failures are caused by requirement defects, and that these test failures typically result in 40 percent rework [NCS99]. Boehm and Basili report similar findings indicating that 40 to 50 percent of rework is avoidable. Further, they state that finding and fixing a problem late in the development process can be 100 times more expensive than finding and fixing it during the requirement or design phase [BB2001].

This paper describes a model-based verification approach that has been effective in locating and correcting requirement defects early in the development process, reducing manual test development effort, and reducing rework. The approach referred to as the Test Automation Framework (TAF) integrates various government and commercially available model development and test generation tools to support defect prevention and automated testing of systems and software.

1.1 Organization

Section 2 provides context by describing the concept of requirement defects, testable requirements, and some results achieved in applying the TAF. Section 3 describes how model analysis and model-based test generation reduce overall development time and effort. It describes how models clarify and formalize textual requirements and provide the basis for defect prevention and test automation. Section 4 summarizes an effective approach for organizational adoption, and also provides test engineer, design engineer, and manager perspectives to illustrate the positive impacts for a variety of stakeholders.

2 Context

2.1 Testable Requirements

Requirement defects occur in many forms. An incomplete requirement is open to differing interpretations, while a testable requirement must be complete, consistent and unambiguous. A testable requirement may include some implicit domain knowledge, but that knowledge must be known or documented within the organization ensuring the requirement is consistently understand

within the context of the system under test. Any potential misinterpretation of the requirement is a defect.

This paper focuses on another form of requirement defect referred to as contradictions or feature interaction problems. These defects arise from inconsistencies or contradictions within or between the requirements. These problems can be difficult to identify when requirements are documented in informal or semi-formal manners, such as textual documents. Often information related to contradictions span many pages of one or more documents and are introduced when more than one individual develops or maintains the requirements. Although rigorous approaches and manual inspections can assist in minimizing incompleteness and contradictions, there are practical limits to their effectiveness. These limits are related to human cognitive limits and are very dependent on the personnel involved. Modeling provides a means of formalizing the requirements. The discipline and structure of the modeling process helps eliminate incompleteness, and the resulting models provide a basis for tools to assist in detecting incompleteness and contradictions early in the development process. ***Requirement testability analysis*** is the process of refining and clarifying requirements through models using a combination of the process and automated tool analysis to develop defect-free requirements.

2.2 Defect Discovery

The effect of early defect discovery is illustrated in Figure 1 by the trend curve labeled "New." The rate of defect discovery increases early in the process, but quickly curtails. This is in contrast to the typical situation reflected by the trend curve labeled "Old," where defects are not identified until testing begins or after release when they are most expensive to fix. Defection prevention involves finding and correcting problems before they propagate to later development phases. Figure 1 also illustrates the conceptual savings associated with defect prevention as the decreased rate of defect discovery between the "New" and "Old" trend lines. Defect prevention is most effective during the requirements phase when the cost of correction is low. Industrial applications, described in Section 2.5, have demonstrated the TAF process directly supports early defect identification and defect prevention through the use of requirement testability analysis [Saf2000].

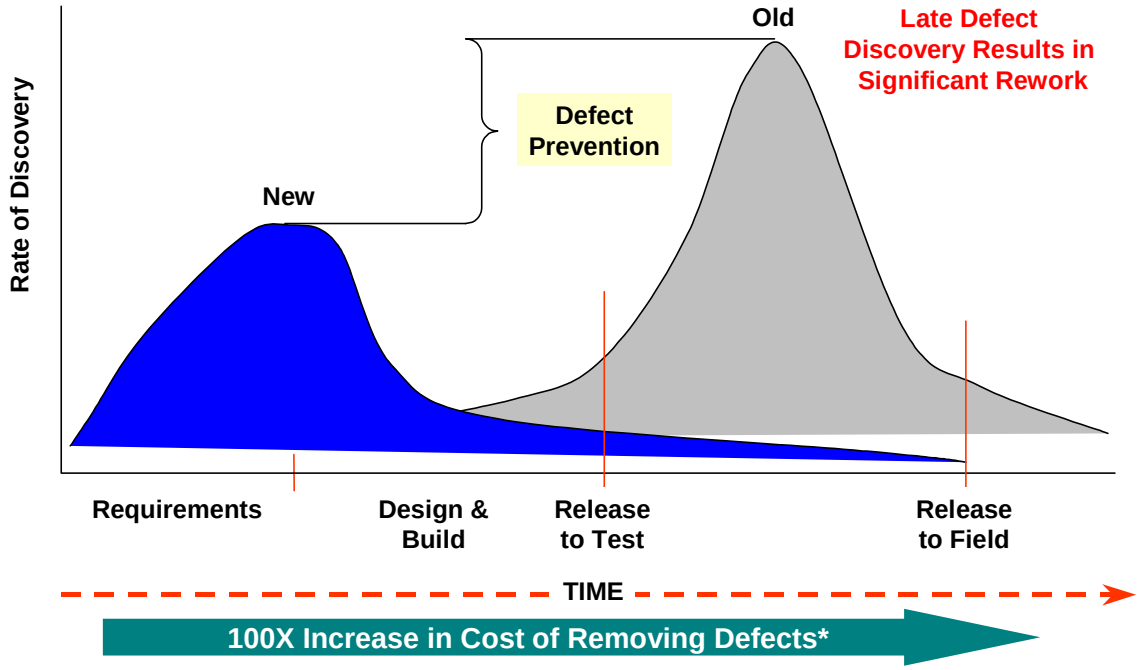
2.3 Requirement Validation

Requirement validation ensures captured requirements reflect the functionality desired by the customer and other stakeholders. Although requirement validation is not the focus of requirement testability analysis, it is supported. Requirement validation involves an engineer, user or customer judging the validity (i.e., correctness) of each requirement. Models provide a means for stakeholders to precisely understand the requirements and assist in recognizing omissions. Tests automatically derived from the model support requirement validation through manual inspection or execution within simulation or host environments.

2.4 Test Design Effort

The tasks related to test design are typically manual and error prone and can account for 60 percent of testing effort. Organizations have reported spending nearly 50 percent of their test effort developing and debugging test scripts. Automating the process of test design and test driver or script development

can result in significant cost savings and more effective testing. The TAF approach leverages the models used to support requirement defect analysis for automating test design activities.



Source*: Boehm, Barry. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1981.
Boehm, Basili. "Software Management." *IEEE Computer*, January 2001.

Figure 1. Early Defect Identification and Prevention

2.5 Applications and Results

The core capabilities underlying this TAF approach were developed in the late 1980s and proven through use in support of FAA certifications for flight critical avionics systems [BB96]. Statezni described how the approach supports requirement-based test coverage mandated by the FAA with significant life cycle cost savings [Sta99; Sta2000]. Safford presented results stating the approach reduced cost, effort, and cycle-time by eliminating requirement defects and automating testing [Saf2000]. Safford's presentation summarized the benefits:

- Better quality requirements for design and implementation help eliminate rework in those phases as well as during test
- Verification modeling can reduce the time normally spent in verification test planning by up to 50 percent
- Test generation from a verification model can eliminate up to 90 percent of the manual test creation and debugging effort
- Both the number of test cases and the phasing of their execution can be optimized, eliminating test redundancy
- A known level of requirements coverage can be planned, and measured during test execution

The National Institute of Standards and Technology (NIST) is assessing this approach as the basis for a methodology and supporting toolkit to automate major aspects of security functional testing [BBNC01]. The methodology recommends developing models of functional security requirements as the basis of automated test generation and execution. Experiments indicate the methodology provides a solution to the problem of functional security testing by increasing test coverage, while reducing time and manual effort. NIST and its sponsor have investigated other model-based test generation approaches, but found that they lack support for automated test execution. They cited the ability to integrate automated test generation with test driver generation mechanisms for multiple test environments as a key benefit of the TAF approach.

TAF has been used for modeling and testing system, software integration, software unit, and some hardware/software integration functionality. It has been applied to critical applications like telemetry communication for heart monitors, flight navigation, guidance, autopilot logic, display systems, flight management and control laws, airborne traffic and collision avoidance. In addition, it has been applied to non-critical applications such as workstation-based Java applications with GUI user interfaces and database applications. The approach supports automated test driver generation in a variety of open languages (e.g., C, C++, Java, Ada, Perl, PL/I, SQL), as well as, proprietary languages, COTS test injection products, and test environments.

3 Approach Overview

This section provides an overview of the TAF approach, starting with how it has been successfully applied in some organizations. Subsequent subsections describe modeling concepts, tools for creating and maintaining models, tools for automating test generation, tools for automating test execution and how these different aspects are integrated in the approach.

3.1 Process Flow and Roles

The conceptual process as rendered in Figure 2 identifies the typical organization roles: 1) a requirements engineer performs requirement analysis, 2) a designer/implementer develops system/software architecture, design and implementation, and 3) a test engineer performs verification, including testing, analysis and reviews, and some validation. Any person on the team may perform one or more roles. Requirements are typically recorded textually and are sometimes supplemented with graphics, tables or formalized models and algorithms. The requirements typically pass to the system designers and testers as documents that can include Software/System Requirement Specifications (SRS), function lists, or change requests.

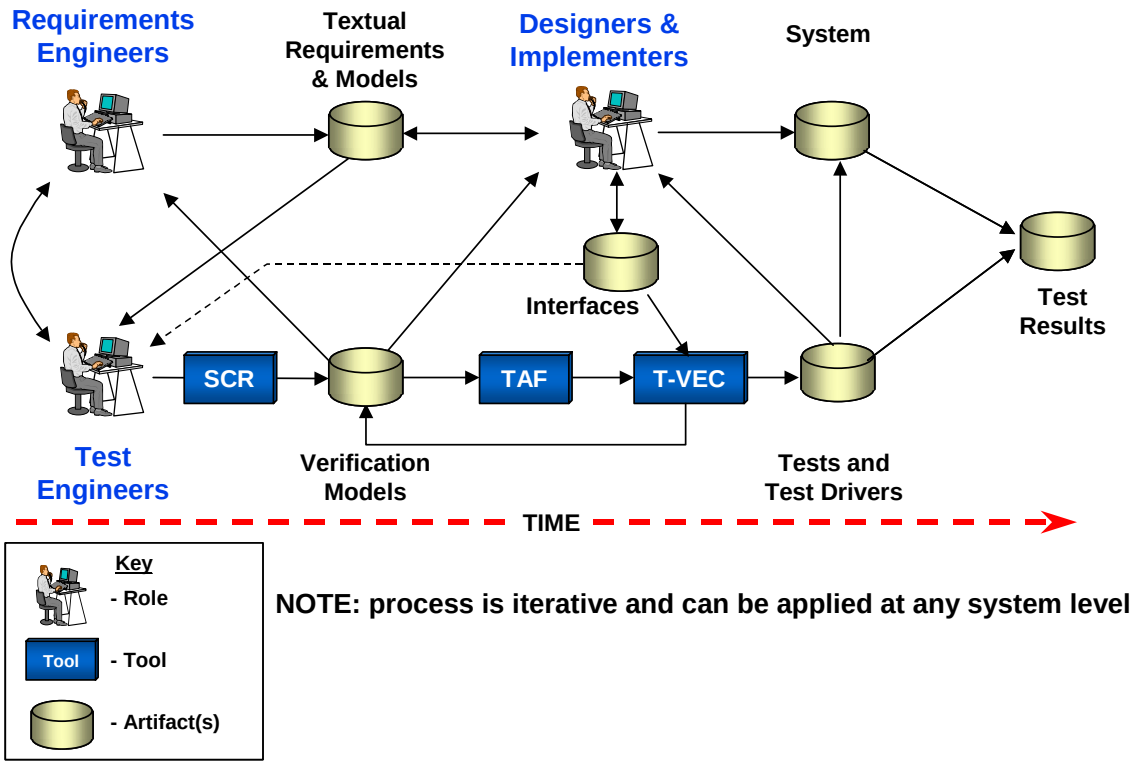


Figure 2. Process Roles and Flows

The key change to a typical process is the introduction of verification models. These models support automated means of identifying model defects and generating tests highly effective in verifying a system is consistent with the model. Figure 2 illustrates a specific process in which testers are involved in developing verification models. This approach has been effective in many organizations not already developing rigorous models. Other successful approaches have involved requirements engineers or designers using existing modeling tools or adopting new tools, such as MATRIXx, ObjecTime, or Statemate to develop models that support both development, validation and verification. This paper highlights a process in which testers develop models to support verification in SCR (Software Cost Reduction).

SCR, and the associated SCRtool developed by the Naval Research Laboratory, have been used in a variety of industrial applications to model system and software requirements [HJL96]. As reflected in Figure 2, the TAF translator transforms and expands the SCR specifications into a form supporting automatic test generation. T-VEC provides model analysis to detect requirement defects, as well as, generates test vectors, performs specification coverage analysis, and generates test drivers [BB96; BBF97].

3.2 Verification Model Development

A "pure" requirement model specifies the requirements in terms of *logical entities* representing the environment of the system under test, where as, a ***verification models*** specifies requirements in terms of the *interfaces* for the system under test; a design engineer typically defines the interfaces. This is analogous to the way a test engineer develops tests in terms of the specific interfaces as opposed to

logical concepts of the environment for the system under test. SCR is a table-based modeling approach, as shown in Figure 3 that models system and software requirements. The SCRtool is used to develop verification models through a process of requirement clarification.

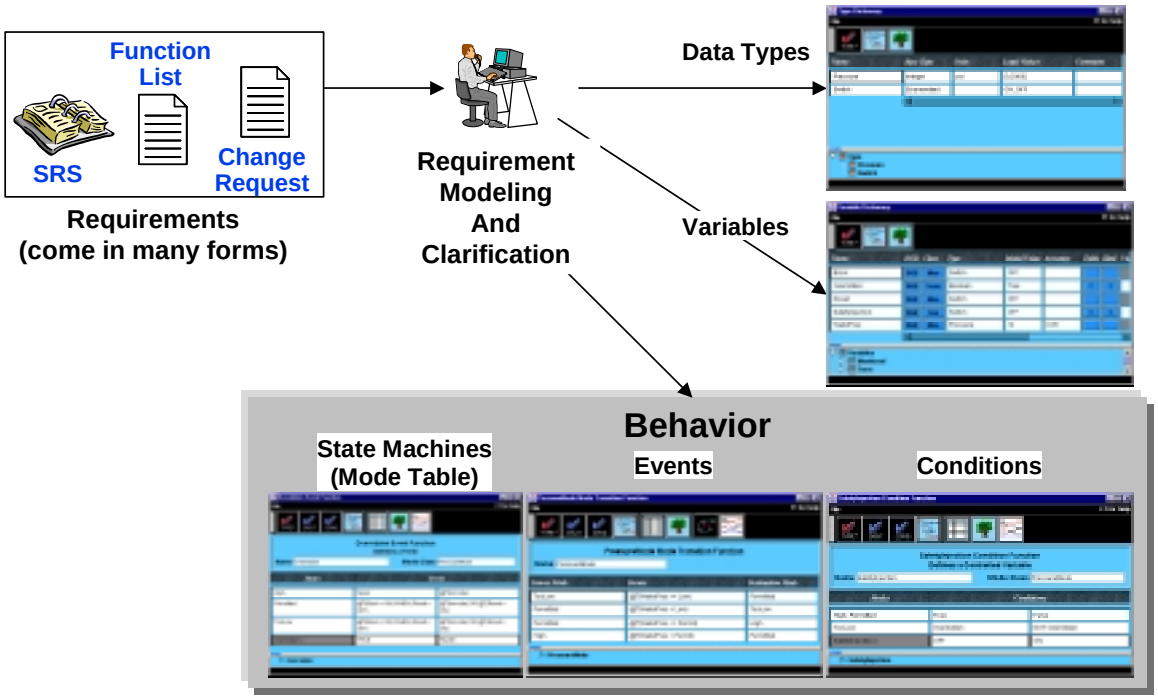


Figure 3. SCR Modeling Constructs

SCR represents systems inputs as monitored variables, system outputs as controlled variables and intermediate values as term variables. Variables are defined through primitive types (e.g., Integers, Float, Boolean, Enumeration) or user-defined types. Models are constructed from four model elements: modes, terms, conditions, and events. A ***mode class*** is a state machine, where system states are called system modes and the transitions of a state machine are characterized by guarded events. A ***term*** is any function of input variables, modes, or other terms. A ***condition*** is a **predicate** characterizing a system state. An ***event*** occurs when any system entity changes value. Each term and controlled variable must be defined using an event or condition table.

3.3 Model Translations

The TAF translator converts SCR models, which are composed of combinations of condition, event, and mode tables into test specification models as shown in Figure 4. For model analysis and test generation the model is "transformed" into a set of precondition/postcondition pairs referred to as ***test specification elements.*** As reflected in Figure 4, a test specification element includes a set of constraints on the inputs and a postcondition that defines the output as a function of the constrained inputs. The test specification element constraints are defined as a conjunction (i.e., logically ANDed) of Boolean-valued relations on the inputs (monitored variables or terms).

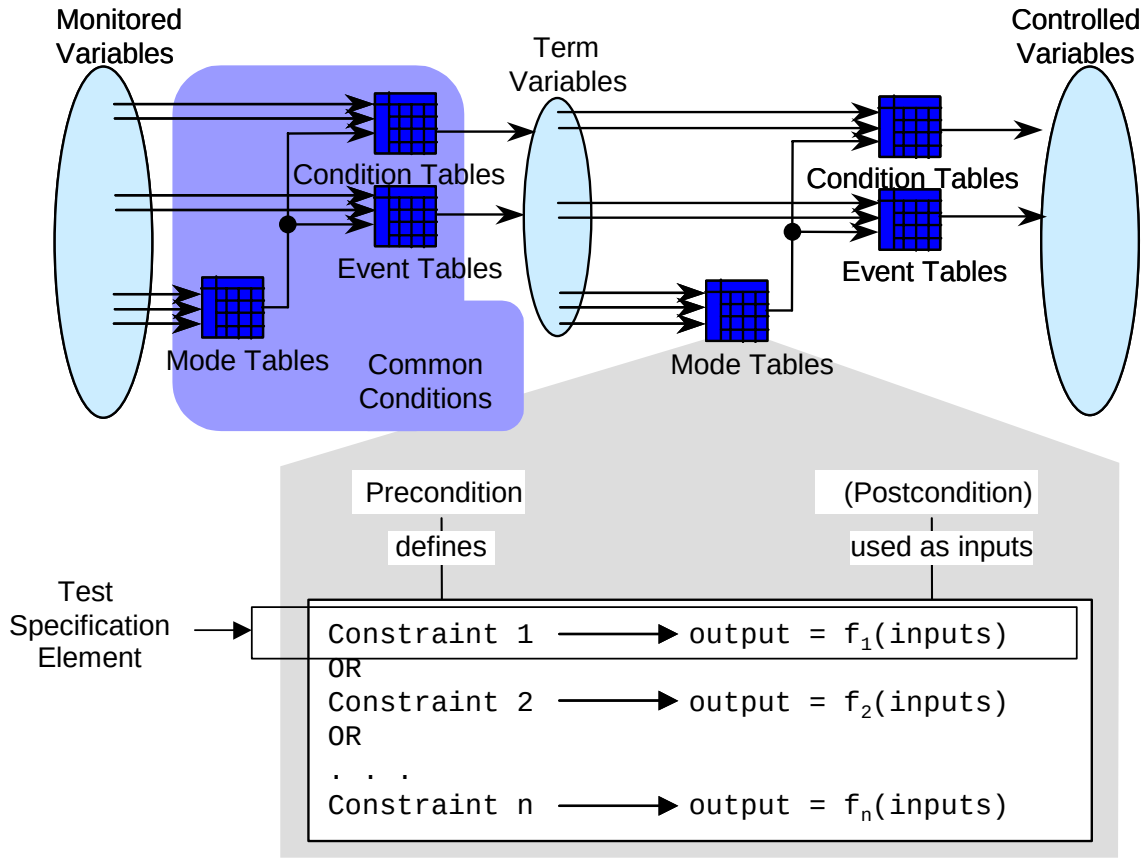


Figure 4. Representation of Test Specification Model

3.4 Test Generation and Defect Identification

Test vector generation attempts to produce a test vector for every test specification element. A ***test vector*** is a set of test input values that satisfy the input constraints, and an expected output value that is derived by evaluating the postcondition with the input values [BB96]. Informally, from a test generation perspective, a specification is ***satisfiable*** if at least one test vector exists for every specification element [BBF97]. If a test vector is not produced, then the specification probably contains a contradiction (a requirement defect).

The SCRtool can check consistency for individual tables, but most inconsistencies result from cross-table dependency relationships that are analogous to feature interaction problems. Therefore detect identification with TAF is a two-step process: 1) the test vector generator attempts to find a test for every test specification element, 2) a post-processing activity identifies test specification elements that have no associated test vector. The test specification elements are traced back to the requirements model to identify requirement defects.

3.5 Test Drivers, Execution and Results Analysis

Test driver generation automates the time consuming and error prone activity sometimes referred to as test script development. As illustrated in Figure 5, the test driver generator combines test vectors and a

test driver schema to produce a test driver and a file of expected test outputs. The test vectors describe the test data, while the test driver schema describes the generic test execution steps. A test driver schema describes a pattern that performs the steps necessary to execute a test case. The schema is defined once per test environment. The schema algorithm typically performs some type of initialization and then loops through each test to initialize outputs to something other than the expected value, loads inputs, calls the system under test, and then retrieves and stores the actual test outputs.

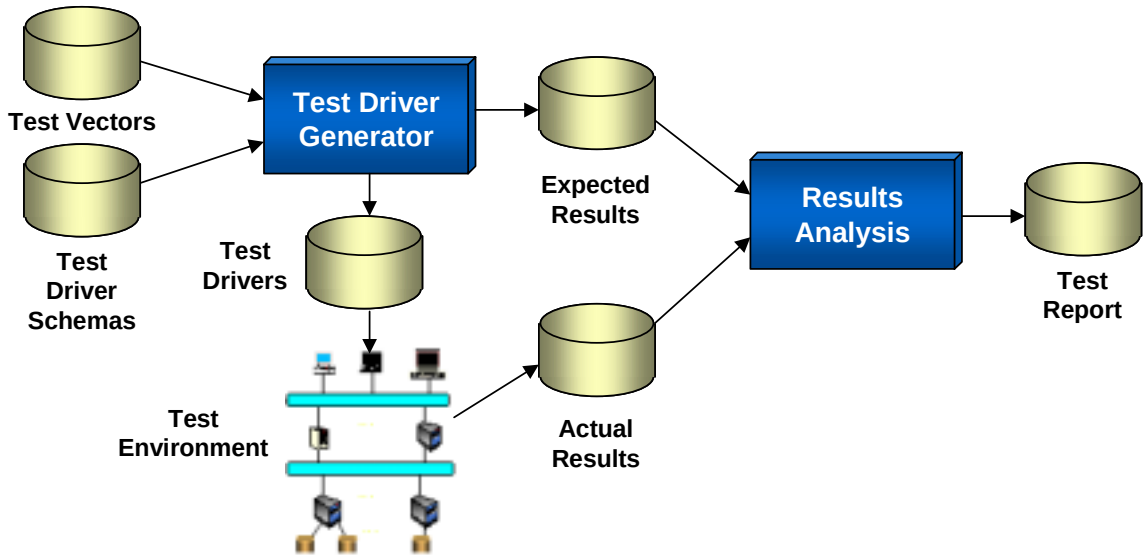


Figure 5. Automated Test Execution Process

Results analysis simply compares the actual results of test execution to expected test results as defined by the test vector expected outputs. A comparator utility supplied with the T-VEC tools supports automating the results comparisons while accounting for any numeric tolerances.

4 Organizational Adoption

The key changes to the organization and existing process necessary in adopting this approach are relatively minor. In Figure 2, the key organizational change involves using the test engineer to develop verification models early in the development process. Verification models are developed as requirements are acquired. These models are continuously analyzed. Defects discovered are fed back to the requirements engineers for correction. Later in the process, the models are used as the basis of test vector and test driver generation, rather than developing these artifacts manually. The following subsections provide rationale from different stakeholder perspectives to explain why this approach is adopted by organizations.

4.1 Test Engineer Perspective

Test engineers are willing to adopt the process because: 1) they are able to work early in the process as opposed to late in the process when schedule and budget are more critical and limited, 2) by working earlier in the process they have more impact on adding test hooks into the system architecture, 3) they are willing to use new tools to support their job, especially when the process for using the tools is similar to the existing manual process.

Test engineers have found that developing verification models using SCR is similar to test plan development using verification cross-reference matrices. Figure 6 provides an illustration of the relationship between requirements, which might come in the form of an SRS, function list, or change request. The traditional test planning activity determines how each function/action in the requirements is related to one or more conditions (or events). The tabular approach shown to the right is sometimes used to define the relationship between the functions and associated conditions. The test plan matrix is then used as the basis for developing the particular test sets.

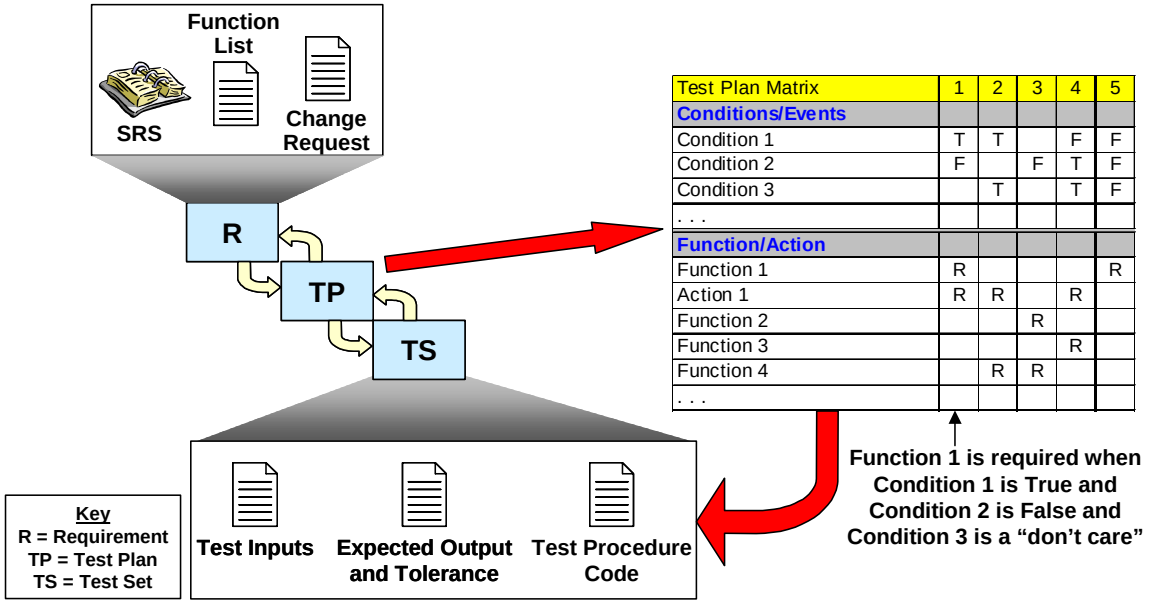


Figure 6. Traditional Requirement-Driven Test Planning

Figure 7 relates the concept of a tabular test plan to requirements modeled using SCR. SCR is a tabular approach, and testers have found it quite natural to model Conditions of a test plan as Boolean-valued terms, where the constraints on the inputs are defined using condition, event or mode tables. Similarly, the Functions of the test plan matrix can be modeled as an SCR table, and related to other tables that have specified the Conditional relationships. Once a condition or function is modeled it can also be reused. This systematic approach to requirement modeling, with planned reuse is what helped Safford's organization reduce verification test planning by 50 percent [Saf2000].

Use case testing is another popular approach that can be structured as a tabular test plan, as shown in Figure 6. A use case test is typically defined in terms of a precondition and postcondition. A precondition can be presented as a Condition/Event in the test plan matrix, while the postcondition can be represented as a Function/Action. A benefit of the test plan matrix is that commonality between truth-table relationships (i.e., conditions) is explicitly visible; this can help in better understanding the completeness of the tests for the required test combinations.

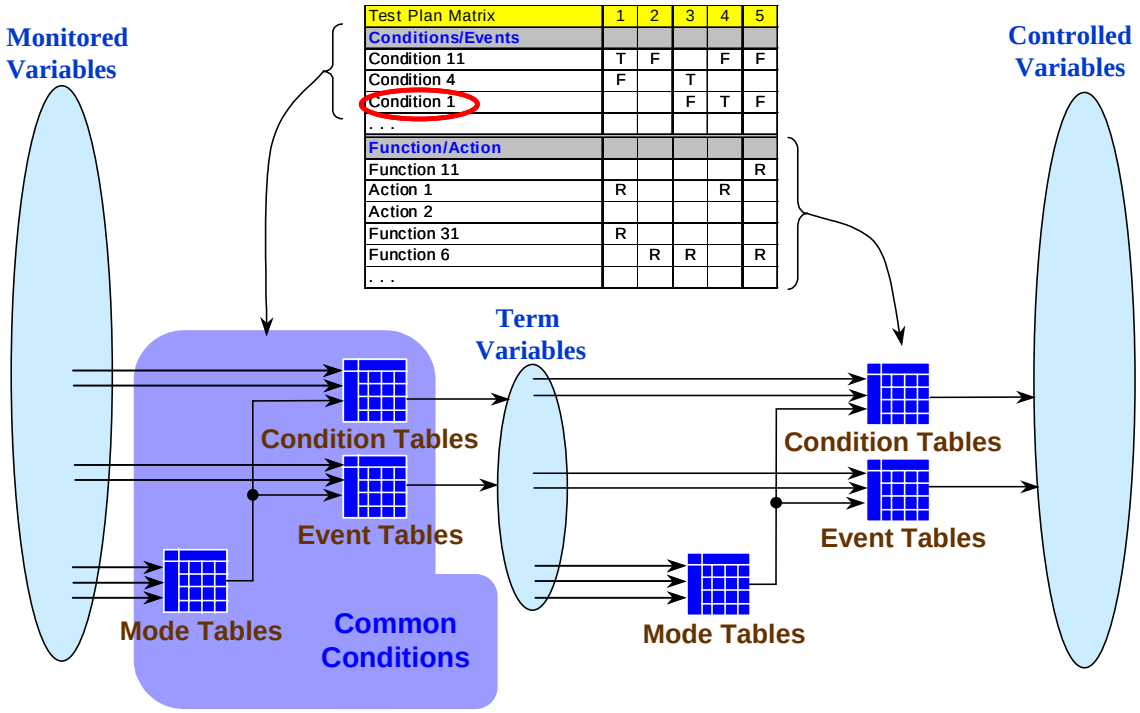


Figure 7. Relationship Between Test Plan Matrix and SCR Models

4.2 Design Engineer Perspective

Often design engineers are initially skeptical of this approach. They do not wish to complicate their designs to support testability. They are more supportive of the adoption once they realize that parallel testing and program development reduces program schedule risk allowing development and design to continue for a relatively longer period of time [Saf2000]. In practice, test engineers ask key questions of the design engineer that help address detailed issues (e.g., range constraints, sizing) earlier in the process, which helps avoid rework. In addition, the design engineer and implementer have early access to test drivers, which reduces the implementer's effort in creating test driver and test stubs to support unit testing and debugging.

4.3 Manager Perspective

Managers are willing to adopt the new process because developing verification models support requirement defect analysis and automated testing. Testing becomes continuous throughout the life cycle as verification models directly support automatic generation of test vectors and test drivers, reducing cost and schedule. As reflected in Figure 8, managers can use failure analysis supported by continuous testing as an objective measure of product goodness and releasability. Failures can be categorized to systematically address and resolve top priority problems. Correcting issues such as interface problems or feature interaction problems may be necessary for release, while correcting problems related to an experimental features may be deferred to later releases. Finally, this provides a basis for software reliability estimates by tracking the failures over time from the beginning of development as opposed to the start of a serial testing phase.

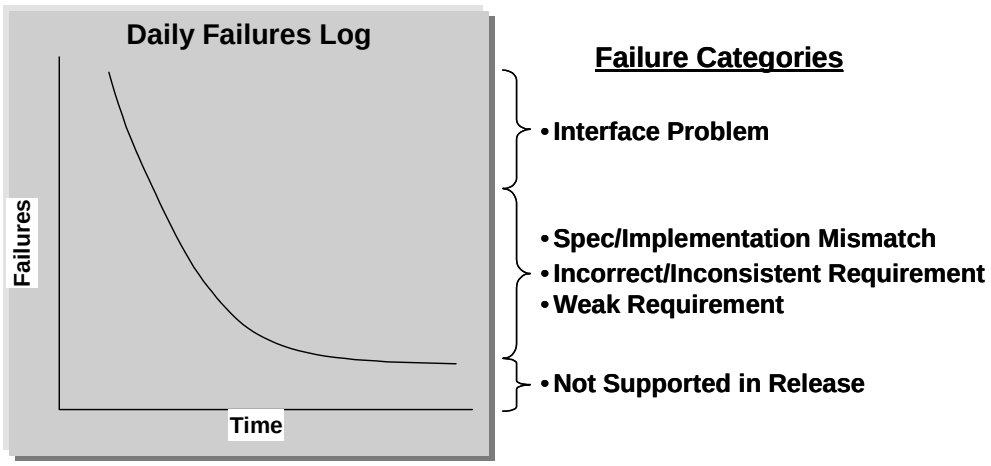


Figure 8. Management Through Failure Analysis

5 Summary

This paper describes a model-based verification approach that integrates commercially available model development and test generation tools to support defect prevention and test automation. Organizations have reported significant cost and effort savings by using this approach to reduce requirement defects, manual test development, and rework involved in developing and testing both software and systems. They found requirement modeling takes no longer that traditional test planning, while reducing redundancy and building in a reusable model library capturing the organization's key intellectual assets. Because testing activities occur in parallel to development efforts, they require less dedicated time at the end of the development cycle. Thus, the approach supports "relatively" longer development efforts without risk to the overall schedule. Defect prevention is a key benefit of the approach. It is achieved using model analysis to detect and correct requirements defects (e.g., inconsistency, ambiguities, feature interaction conflicts) early in the development process. The verification models enable automated test generation. This eliminates the typically manual and error-prone test design activities and provides measurable requirement-based test coverage. Organizations have demonstrated that the approach can be integrated into existing processes to achieve significant cost and schedule savings.

6 References

[Bei83] Beizer, B. Software Testing Techniques, New York, New York: Van Nostrand Reinhold, 1983.

[Bla98] Blackburn, M. R., Using Models For Test Generation And Analysis, Digital Avionics System Conference, October, 1998.

[BB96] Blackburn, M.R., R.D. Busser, T-VEC: A Tool for Developing Critical System. In Proceeding of the Eleventh International Conference on Computer Assurance, Gaithersburg, Maryland, pages 237-249, June, 1996.

[BBF97] Blackburn, M.R., R.D. Busser, J.S. Fontaine, Automatic Generation of Test Vectors for SCR-Style Specifications, In Proceeding of the 12th Annual Conference on Computer Assurance, Gaithersburg, Maryland, pages 54-67, June, 1997.

[BBNC01] Blackburn, M.R., R.D. Busser, A.M. Nauman, R. Chandramouli, Model-based Approach to Security Test Automation, Quality Week 2001, June 2001.

[GW94] Ghiassi, M., K.I.S. Woldman, Dual Programming Approach to Software Testing, Software Quality Journal, 3:45-58, 1994.

[HJL96] Heitmeyer, C., R. Jeffords, B. Labaw, Automated Consistency Checking of Requirements Specifications. ACM TOSEM, 5(3):231-261, 1996.

[Sta99] Statezni, David, Industrial Application of Model-Based Testing, 16th International Conference and Exposition on Testing Computer Software, June 14-18, 1999.

[Sta2000] Statezni, David. Test Automation Framework, State-based and Signal Flow Examples, Twelfth Annual Software Technology Conference, 30 April - 5 May 2000.

[Saf2000] Safford, Ed, L. Test Automation Framework, State-based and Signal Flow Examples, Twelfth Annual Software Technology Conference, 30 April - 5 May 2000.