Testing and Evolutionary Development

Dr. Anneliese von Mayrhauser
Computer Science Department
Colorado State University
Fort Collins, CO 80523

1. The Evolutionary Testing Problem

As software use changes and evolves, the software must change and evolve with it. Traditionally this meant

* determine the current operational profile and excerpt a set of test cases for it.
* match it with the currently existing test cases and determine which are still valid
* delete the invalid ones and add test cases that represent new, enhanced or modified software capabilities.

Operational test cases and system test cases can comprise a huge number of tests. This makes it desirable to identify those test cases that do not have to be rerun vs. those that do. On the other hand, this can be impractical, especially for large scale systems. It might therefore be more cost effective to try and rerun the whole battery of tests again. With evolutionary development this can carry a hefty price-tag. Thus, it is very desirable to have a tool and a methodology to describe tests, their purpose (as evidenced in the operational profile) and how classes of changes will affect tests, create the need for new ones, and make old ones obsolete.

Additionally, tests happen all through the software development life cycle. They usually are related. For example, an aspect of the operational profile may be reflected in a particular design module. Hence, one or more test cases exist that test the proper functioning of that design module. These relationships exist and should be preserved and properly modified as software evolves. This poses challenges. The next sections will describe a classification of tests and the design of a test analysis and modification tool that could make testing evolving software easier.

2. Operational Profile and Software Evolution

Most of what has traditionally been called software maintenance is really adaptive and perfective in nature and falls under evolutionary development. Changes are made to software because requirements change and thus the operational profile. The first task is then to identify which requirements change and what type of change it is. There may be three types:

* adding requirements
* deleting requirements (probably rare)
* modifying existing requirements.

Associated with requirements are sets of test cases that test each requirement. These are either described via the operational profile or the acceptance test (sometimes the two are in fact equivalent). Requirements and test cases can be related through a two-way link, i. e. each requirements phrase is associated with one or more test case identifiers. An example of such a tool is BUSTER from AT&T Bell Laboratories [ARCH90].

When new requirements are added, each new requirement must have at least one new test case associated with it that tests the new requirement. Thus a new link is established between requirements and (a set of) test cases. Individual test cases and packages or suites of test cases may be named.

When an existing requirement is deleted, its test cases can be deleted also. This, however, presupposes that one test case only tests one requirement. This is hardly a parsimonious approach to testing. Thus it must be determined, whether a test case that has been identified as a candidate for removal, is connected with another requirement that is not being deleted. A two-way link will enable such a cross-check. This establishes a many-to-many relationship between requirements and test cases.

When an existing requirement is modified, all test

cases are listed that test this requirement. Then we must determine which of them will have to change, which can stay as is, and which need to be replaced by new ones. Thus test cases are subject to the same three change operations:

* delete a test case
* add a test case
* modify a test case.

When a test case tests more than one requirement, the following rules apply:

* test cases that need modifications due to a change in a requirement lose their link to the modified requirement (but not their links to other requirements that may exist). The modified requirement is linked to a new test case that reflects the modifications in the requirement. Test cases that need to be replaced are handled the same way.
* no change for test cases that need no modifications.

This approach provides a minimal solution to updates for test cases that are requirements driven. It is by no means complete nor will it solve all problems. Note that we have taken an approach that is similar to some configuration management tools. We describe evolution (change) in terms of type of change and its resulting effect on the configuration items (in this case requirements and test cases). To implement this concept, we need a requirements "editor" and a test case "editor" as well as a browsing mechanism that allows us to navigate through requirements and test information and find the spot where changes need to be made.

3. Dealing with Size Complexity.

Large software systems may have thousands of requirements and test cases. The simple, linear, unstructured approach to requirements and associated test case changes is no longer realistic. It would be too hard to find what we want to change, navigation becomes difficult. The answer to handling the complexity of changes to requirements and test cases lies in structuring them and using the structure to navigate. We also must avoid information overload for the regression tester. Thus structure and browsing facilities become indispensable.

3.1. Structuring Requirements

This is not a new idea, but should be exploited for regression testing purposes. Requirements Diagrams provide levels of requirements for systems and subsystems down to the actual function level ([VONM90]). Let us call this lowest level of requirements an "atomic requirement". This structure can be captured and used to structure and name sets of requirements, making navigation easier. The advantages of capturing the levels of refinement of an abstract requirements diagram as a named hierarchical set of objects are

* functionally related requirements objects are clustered. This is a simpler, but similar approach to [HSIA88], but as we will see, no less powerful.
* test case suites can be associated with a chosen level of abstraction. This provides the basis for requirements "editing" and associated test case "editing" at the highest applicable level of abstraction.
* we can provide for inheritance properties of requirements and test cases via appropriate rules with this structure.
* the requirements representation for testing purposes relates naturally to the requirements representation of abstract requirements diagram tools. A user does not have to translate between the requirements structure the tools provide and the requirements structure for testing purposes: it is the same and can be captured from the Requirements Diagram tool's internal representation.

So far, we are not yet dealing with qualitative requirements such as security, performance, adaptability, maintainability, performance, etc. (for a complete list and discussion of qualitative requirements see [VONM90], Chapter 4). Commonly, these requirements apply to more than one atomic requirement. Thus we should associate such qualitative requirements as qualitative characteristics with the highest level of the named requirements objects to which they apply. We suggest to use an attribute vector to represent these qualitative requirements for a requirements object at a particular level of abstraction. This provides the following:

* the ability to represent several instances of an attribute associated with a particular qualitative requirements type. For

example, a performance requirement for computing taxes in a tax preparation system may specify that data and computation must fit into 1 Megabyte of storage, that response time for editing tax data must be under 2 seconds while compilation of a tax form must not take longer than 35 seconds. Here we find 2 types of performance attributes, storage requirements and response time requirements. For type response time 2 values exist, for two different collections of requirements (note that the functional requirement "editing" and "tax form compilation" are not at the atomic level).

* these attribute vectors can then be associated with test suites: if the attribute value is present, this means that a "capability" exists to execute the associated test suite. Thus we end up having a structure analogous to a capability based architecture ([WULF81]). It has been suggested ([VONM90]) that they are a natural extension to the concepts of abstract data types, and, when associated with rules or policies are very flexible. This is a very useful structure for requirements from a testing point of view as it makes it possible to use different testing criteria or strategies within this structure (modifying rules or policies).

Editing operations on the requirements structure are fairly easy:

* adding new requirements requires finding the proper level of insertion and adding a connection between the new requirement and its next higher level of abstraction. To illustrate impact we can highlight or otherwise mark a path from the insertion through the abstraction hierarchy to the top.
* deleting requirements deletes the subtree for which the requirement to be deleted is a root, as well as the connection between the deleted requirement and the next higher level of abstraction. Impact illustration is represented by a path up the abstraction hierarchy.
* modifying requirements involves changes in the text associated with the requirement, refining a requirement (adding a subtree), or editing the qualitative requirements via changes to the attribute vector (see below). Impact illustration marks a path up and down the abstraction hierarchy.

Editing operations for the attribute vector involve the following:

* deleting a qualitative requirement at a specific level of the abstraction hierarchy deletes the value of the attribute slot at that level. Since values are inherited at all lower levels of abstraction, this also removes the deleted qualitative from all lower level functional requirements that belong to the cluster. Test cases associated with testing the deleted qualitative requirement are removed, unless they possess links to other functional or qualitative requirements (links).
* adding a qualitative requirement involves assigning a value (and possibly a type distinction) for the appropriate attribute. Whether new test cases have to be defined depends on whether existing test suites can be reused (e. g. a test suite testing functional requirements can also be used to test performance requirements).
* modifying a qualitative requirement. This usually involves modifying the attribute value in the attribute vector. Associated test cases may need no change at all (e. g. when response time must be less than 20 seconds is changed to response time must be less than 15 seconds). Some changes in qualitative requirements require test case modifications or the writing of new test cases (e. g. when the requirements states that software must be capable of handling 100 transactions per hour and that is increased to 120 per hour. This may require rewriting test cases so that at least 120 transactions per hour are generated). We will discuss ways to represent other situations when we discuss test case structuring.

3.2. Structuring Test Cases.

Analogous to requirements, test cases can be functional or qualitative. A test case is associated with a runnable script (inputs plus anything that is necessary to actually execute the test case), expected output, test case identifier, and structuring information. We will also later

introduce test case attributes that will allow us to use testing and regression testing rules. For now, primary concern is how to structure test cases and how this structure changes with editing operations on the test case identifiers and the test case structure. This test case structure is hierarchical and subtrees can be named (i. e. suites of test cases). The hierarchical structure represents a "consists of" relationship. Test cases for qualitative requirements are represented with a test suite vector containing a reference to a test suite that tests the appropriate qualitative requirement in the attribute vector slot of the corresponding requirement. We also need a test suite table that contains definitions for and references to all named collections of test cases and test suites.

Editing operations on the test cases:

* deleting a test case or a named set of test cases: remove the test case from the test case table and (through double links) from the test case structure and all test suite vectors. If the link from the deleted test suite to a subset name is the last one, remove the subset. This latter operation may be delayed to an occasional "garbage collection" operation that marks and deletes test case definitions.

  A delete operation for a test case may have been the result of a delete operation for a functional or qualitative requirement. A test suite associated with a functional requirement object that is deleted is marked for deletion (the link between them is removed). If the test suite has no other links to requirements, it is removed and links to other test suites are removed also (it may have been part of another test suite or it may be connected to named subsets as in T consist of t1 and t2). Note that we will find rules in the test rule section that may result in a test modification rather than a test deletion. These will be discussed separately. Here we are solely concerned with a delete operation for test objects.

  Deleting a qualitative test object (test case or test suite) usually is the effect of deleting a qualitative requirement. It is easily accomplished by removing the name of the test object from the corresponding slot of the test suite vector.

  Implementing the test case structure is possible using a doubly linked list structure. In the current scenario (all test case change operations are triggered by requirements change operations), this is not really necessary as long as we keep a counter for each test suite/test case name that records how many links still point to it. This makes it easy to determine whether the test entity should be removed (nothing points to it any more) or not.

* adding one or more test cases. This usually happens due to changes in the Requirements structure and may involve the following:
  - add test case identifier to a set of existing ones (a new named subtree)
  - for a test case (suite) that tests a qualitative requirement, add its name in the appropriate slot(s) and establish links.

* modifying one or more test cases. Content changes happen in the appropriate test script itself. Since we are concerned here more with the structure and how it facilitates change, we do not as yet provide support for this. However, for impact analysis purposes, we can mark the changed test case(s) for regression test to indicate that change actually took place and the suite should be rerun. Since test cases are referenced by their names, no test case structure changes are necessary.

  When test case changes affect the test case structure (e. g. a suite is used to test a qualitative requirement in addition to a functional requirement), appropriate links are generated and entries are made into the attribute vector (see add, delete functions).

4. Rules for Test Case Update

4.1. Regression Test Rules - Requirements Driven.

4.1.1. Requirements Deletion

We assume that most changes in test suites come about as a result of changes in test cases. Not all changes in requirements simply result in the

corresponding change operation for the test objects associated with it. For example, the deletion of a qualitative requirement, let's say a performance requirement, may result in no test case deletion at all (if all tests from a performance test are also used for functional tests as well). Thus, whether or not a test will be deleted depends on whether it is currently used for a purpose other than testing the (deleted) requirement. This would be apparent through the number of links to the test case. Conversely, a deletion of a functional requirement may involve the deletion of qualitative tests as well as functional tests (namely those associated with qualitative requirements for a stated function). When functional tests are also used for qualitative purposes this may simply delete the same test cases. However, stress tests are not always used for testing functional requirements. Then we indeed have two delete operations, one for the functional, one for the qualitative aspects of the requirement being deleted and the test cases associated with both.

Impact analysis determines which test cases need to be rerun. We can use the requirements or the test case structure to assess impact. In the requirements structure we marked impact of a requirements change as a path up the abstraction hierarchy to the highest level abstraction of the requirements cluster (subtree). A maximal regression test set would identify all test suites for the marked requirements structure. The rationale for such a rule is that a requirement modification changed a requirement not just at the level where it occurs, but also provided a modified abstraction to the next higher level which in turn then provides a modified abstraction to the requirements level above and so forth. Thus a maximal rule like regression testing all test suites that are associated with the path upwards from a requirements modification makes sure that the requirements cluster associated with a changed requirement still works.

A more minimal rule would require to select only those test cases that are associated with the level of abstraction where the requirements change happened and those test cases for the level immediately above. This still regression tests the changed requirements cluster, but not as extensively.

We can also use the modifications in the test case structure itself to determine regression test suites. If a test suite changed at a given level in the structure, the modified suite gets used for regression testing. The rationale for this rule is that it is important to make sure the new test suite works. Also, a test suite is identified as a meaningful collection, because it tests some underlying abstraction, functional or qualitative. Therefore the test cases are related and if one changes, the whole test suite, and the code they test could be impacted.

Other test rules beyond the requirements and test case structure require a formal specification for test cases to state purpose and intent, even design principles. Without them, interpretation of the 'why' of test case change and therefore a more detailed set of rules for regression test configurations is not possible.

4.1.2. Requirements Addition

When requirements area added (functional or qualitative) new functional and qualitative test cases must be added to test them. Sometimes functional requirements tests can also be used for qualitative tests, they simply are interpreted differently (see earlier remark on performance vs. stress tests). The impact of adding new requirements follows a path up the abstraction hierarchy. Whether it is minimal (includes next higher level only) or maximal (all the way to the top) depends on how conservative regression test must be. Both functional and qualitative requirements should be retested. At higher levels the content of a test case may need modification. This holds for functional and qualitative requirements.

4.1.3. Requirements Modification

All test cases, functional or qualitative, that are associated with a modified requirement must be checked for necessary modification. This may for example involve changing the required response time value (if that is the nature of the requirements change). The impact of a requirements change means that at least the next higher level of abstraction is affected. Thus, depending on whether a minimal or maximal approach is to be followed, the next or all higher levels in which the modified requirement is a part must be regression tested. There is also an impact down to the next lower level of requirements, thus they need regression testing, too. In addition, test case content may need changes.

4.2. Rule Structure versus Requirements/Test Structure

While we have some rules for identifying a suite of regression test cases, these are not the only possible rules. therefore, a tool should be able to work with various sets of rules, indeed be able to accommodate rules that reflect new insights into the regression testing process. To provide this flexibility, we propose a test structure that separates the requirements and test editor from the rules mechanism and determines a regression test suite based on a set of given rules which may change.

First, we separate the requirements and test (structure) editor from other tool components. Thus editing operations on the requirements, test cases and their structure do not overlap with regression test definition. Analysis of the changes in requirements and test structure identifies dependency information that can be used as input to testing rules and regression test definition. The next level of tool identifies the rules to be used for definition of tests that form a regression test suite. This structure separates concerns and also provides a possible interface to an existing tool for requirements analysis ([VONM89]). This tool records requirements in the structure discussed above, performs a benefit analysis, and identifies candidates for successive versions based on priorities and evolutionary changes. Thus the regression testing mechanism described here complements the earlier tool. Further, a similar approach to regression testing and the representation of test structure vs. content vs. rules is taken in [JEON91] for white box testing as part of a Maintenance Toolkit ([VONM91]). More work is still required to be able to use a truth maintenance system for updating black box regression test suites as was done for the white box testing in [JEON91]. We also have an experimental test case language with a YACC based parser for black box testing ([BARE90]).

5. Conclusions

This paper presented a structure based approach to representing requirements and test cases in an evolutionary context. Types of changes and their impact on regression test suites was discussed. When combined with a hierarchical browser with zoom-in/out capabilities this mechanism can provide significant help in updating and identifying regression tests. Formal specifications, or even more detailed classification of test case intent can make rules for test suite identification and update more precise. More work is needed in this area.

References:

ARCH90 Archie, K. C.; "Environments for Testing Software Systems", AT&T Technical Journal, March/April 1990, p.65-75.

BARE90 Bareiss, C.; von Mayrhauser, A.; "Parser for a test case language", Illinois Institute of Technology, AvM-12-1990.

HSIA88 Hsia, P.; Yaung, S. H.; Jiam, S., H.; "Another Approach to System Decomposition: Requirements Clustering", COMPSAC, 1988, p. 75-82.

JEON91 Jeon, T.; von Mayrhauser, A.; "Architecture for Knowledge-Based Regression Testing", accepted, Second Great Lakes Computer Science Conference, Oct. 17-19, 1991.

VONM89 von Mayrhauser, A.; Johnson, D.; "A Requirements Analysis Tool for Evolutionary Software Development", Illinois Institute of Technology Technical Report, AvM-12-1989.

VONM90 von Mayrhauser, A.; "Software Engineering: Methods and Management", Academic Press, 1990.

VONM91 von Mayrhauser, A.; "AMT -the Ada Maintenance Toolchest", Triada 1991, Oct. 23-25, 1991, San Jose, CA.

WULF81 Wulf, W., A.; Levin, R.; Harbison, S.; "HYDRA/C.mmp: An Experimental Computer System", McGraw-Hill, 1981.