A MANAGEMENT METHODOLOGY FOR TESTING SOFTWARE REQUIREMENTS

K. W. Krause and L. W. Diamant

TRW
DEFENSE AND SPACE SYSTEMS GROUP
Redondo Beach, California 90278

This paper describes the methodology employed on the System Technology Program involving a set of disciplined management techniques, coupled with an automated tool for tracking the moving requirements and software baselines, to provide snapshot visibility into the various stages of the requirements testing activities at any point in time. Included is a description of the requirements identification activity and the allocation of requirements to three levels of testing as the first part of the methodology during the test planning phase. The use of the tool and the techniques to provide visibility and control during the test execution phase is traced from test case definition, through test procedure generation, acceptance criteria measurement and final reporting of requirements testing.

INTRODUCTION

In an ideal software development world, the software requirements are frozen prior to any testing. Once it has been demonstrated that requirements have been satisfied, they stay satisfied in this perfect world. No software project experiences these ideal situations. Requirements are never perfect and must be updated based on new discoveries. Furthermore, the software configuration baseline must be updated to correct problems discovered in testing. Thus, defining the status of requirements testing requires the tracking of at least two moving targets: the requirements baseline and the software baseline.

Effective test management requires current and accurate knowledge of the planning and the status of requirements testing for proper control of the test program and the quality and schedule of the product. The methodology for tracking the two baselines is essential to providing this visibility. Such a methodology was developed, implemented and proved effective for the Army's System Technology Program (STP).[1]

TRW, as the software subcontractor to McDonnell Douglas Astronautics Company (MDAC), is in the final steps of delivering the STP software to support the missions at Kwajalein Missile Range (KMR) designed to validate the resolution of key high technology issues related to the defense of ballistic missiles against threatening reentry vehicle target complexes. The real-time mission software consists of 212,000 MLI developed from more than 10,000 requirements. Starting in March 1972, the mission software was developed, tested and delivered in three major increments or capability releases: January 1977, September 1977, and July 1978, respectively. The unqualified success of the KMR tests to date speaks well for the reliability of the software and therefore the success of the requirements test program.

This paper describes the methodology used to manage the test program with emphasis on a tool, the Test Evaluation Matrix (TEM), used to plan, provide visibility into status, and to control the requirements test program. This methodology matured as the program objectives evolved from a prototype terminal defense system (Site Defense) to one of technology validation (STP).

BACKGROUND

Early in the program the planning and management of requirements testing were more straightforward. Since the Site Defense System was to be a prototype development program, all software requirements were to be tested on the delivered configuration of software using formal, approved test procedures. Therefore all requirements testing (validation) was to be performed by an independent test team as part of the established software development methodology.[2]

In 1975 the program was so redirected that objectives consisted of resolving key technology issues (e.g., object discrimination). The program name was changed to STP reflecting the change from a prototype development to an R&D program.[1] This had a profound effect on the software test program. It was now possible to accept more risk, cutting back test resources due to the revised objectives of the KMR missions.

Thus, the software test program was redefined. It was decided that requirements could be satisfied at the unit and subprocess levels of testing, rather than just the process level. Once satisfied they did not have to be individually retested on the final configuration. Requirements needing

system hardware or special test conditions different from those nominally expected at KMR would be identified for special off-nominal analyses.

It became clear that the implementation of this redirection would be a management challenge in the planning and control of a well integrated test requirements program at the unit, subprocess, and process levels of testing. The refinement of the management methodology described in this paper evolved throughout the testing of the three capability releases (CR1, CR2, and CR3). That the application of this methodology was successful is borne out by the successful satisfaction of all mission objectives on all thirteen missions flown with the installed software to date.

REQUIREMENTS IDENTIFICATION/ALLOCATION

Two fundamental elements of the test planning process needed to be addressed as early in the software development life cycle as possible:[3] (1) identification of the testable unique requirements (those within the scope and capability of the test program and not repeated elsewhere in the specification), and (2) their allocation to test levels.

Two factors combine to increase the difficulty of defining the precise requirements to which the system must be validated. The first is the manner in which the requirements specifications are written. STP test management implemented procedures to force shredding out (identifying each unique requirement) of numerous embedded (e.g., multiple parts) requirements from within any given requirement identifier. This allowed each unique requirement to be addressed from the viewpoint of whether or not it could be tested. It is probable that implementation of the Software Requirements Engineering Methodology (SREM)[4] will alleviate this problem in the future. The second factor which increases the difficulty of identifying the uniquely testable requirements relates to the real world issue of the moving baseline. As test planning was accomplished, changes to the requirements baseline were being made either as a result of performance changes or as a result of completed testing of previous releases. These changes to the baseline underscored the need to maintain clear traceability of requirements implemented and tested.

The Test Evaluation Matrix (TEM) constructed by the test requirements analysts is a report generated from an information management tool used to explicitly record plans, status, and results of the test program throughout the various phases. The TEMs (Table 1) show the traceability from detailed specification requirements to design allocations and to the test procedures allocated to verify those requirements. The "Paragraph No." refers to the exact specification designation containing the requirement. Under "Design Solution" there are three subheadings, "Function," "Task," and "Routine." These entries define where in the software hierarchy the requirement is implemented. This is referred to as design solution traceability. The next block of entries is grouped under "TEST LEVEL ALLOCATION" and includes "Unit," "Subprocess" and "Process" headings. Reflected under these headings are designations of initial allocations to test levels (e.g., DLT, SLT, PT) and completed allocations to actual test procedures (e.g., P1941, U:1, T1401). The "VERIFICATION AUTHORITY" block reflects test status (i.e., does the software satisfy the requirement, Yes/No) and where the results are documented. The "CHANGE NOTICES" column is used to designate specification changes impacting the particular requirements paragraph. The final column marked "METHOD" reflects the agreed upon method of testing where I is testing by inspection, ANAL is testing by analysis, SIM is testing via computer simulation, and T is testing under actual environmental conditions. As a planning tool the TEMs are used to identify the allocation of requirements testing of a given software release.

Several updates of the TEM occur as the software product matures through its development phases. The preliminary version of the TEM, published at the start of unit test, is used:

1. To establish the testing baseline for requirements to be tested in the particular capability release test program.
2. To identify the design solutions to each requirement.
3. To baseline those requirements which are being tested at the unit level.
4. To provide a preliminary estimate of those requirements to be tested at the subprocess and process levels, and to define those requirements which are either deferred or not testable.

During the subprocess and process level test periods, the TEMs are updated to show:

1. Actual unit test case identification for those requirements verified at the unit level.
2. Subprocess and process level procedure identifications associated with requirements planned to be verified at these levels.

At the completion of testing for a capability release, the TEMs are further updated with final test information to form a basis for test reporting.

ALLOCATION OF REQUIREMENTS TO TEST LEVELS

Requirements, if properly allocated, can be sufficiently tested at the unit level, the subprocess level, and/or the process level of testing. Sufficient testing for STP was defined to be the degree of testing required to verify software operability in its intended use at KMR. Unit level testing verifies computational algorithms and logic by means of code inspection and simulation; subprocess level testing integrates the major components and verifies interfaces and software interoperability; process level testing

TABLE 1

EXAMPLE - TEST EVALUATION MATRIX

PARAGRAPH NO.	DESIGN SOLUTION			TEST LEVEL ALLOCATION			VERIFICATION AUTHORITY	CHANGE NOTICES	METHOD
	Function	Task	Routine	Unit	Subprocess	Process			
3.2.2.1.03.3.1	ODD1	TDRP	TUPK3	DLT					I
3.2.2.1.03.3.2	ODD1	TIRP	TRVCP		SLT	PT			ANAL
3.2.2.1.03.3.3	ODD3	TDRP	TINSL			P1941	YES - BLK 9 FINAL RPT	SCN 3	SIM
3.2.2.1.03.3.4	ODD3	TDRP	TUPK3			P2931	NO - BLK 9 FINAL RPT (ONLY PARTIALLY TESTED - SCENARIO DID NOT STIMULATE FULL RANGE OF PARAMETER)		T
3.2.2.1.03.3.5	ODD4	TIRP	TANGL	U:1	T1401		YES - UNIT DEVELOPMENT FOLDER, BLK 4 FINAL RPT		SIM
3.2.2.1.03.3.6	ODD5	TIRP	TANGL			P1942	NO - FAILED - DR 12236 (TO BE RESTESTED AFTER PROBLEM RESOLVED)		SIM

verifies satisfaction of requirements and evaluates software performance. Assumptions must be made concerning adequacy of testing for each level since in none of these levels is it normal that the full operational environment be used. At the process level, one must infer that the operational hardware satisfies its interface and operational specifications, and, in some cases, that the physical and target environment is properly simulated. In addition, at the subprocess level, it may be necessary to infer that, in the absence of components of the complete process, these missing components meet specifications. At the unit level, in addition to the above assumptions, it may be necessary to infer that certain units which interface with the unit containing the requirement being tested satisfy their specifications. Since fewer assumptions are verified by actual tests, the degree of risk increases for requirements to be tested sufficiently at lower test levels. However, the risk is reduced and the assumptions become more valid when the aggregate of all requirements testing is taken into account.

These assumptions and associated risks were considered in developing the criteria for allocating requirements to their appropriate test levels as described below.

The following criteria are used by STP analysts to allocate release-unique requirements to the unit, subprocess, or process levels of testing:

1. If the requirement is contained in a single routine or task, it is allocated (in most cases) to the unit level for test.
2. If the requirement is significantly affected by interfaces with other tasks (even though it satisfies (1) above), it cannot be tested sufficiently at the unit level and must be allocated to either the subprocess or process levels.
3. Requirements associated with port-to-port (entry/exit) timing are allocated to process level testing.
4. If a requirement is satisfied by more than one application software task, it can be allocated to either the subprocess or process level. If the requirement is limited to data transfer between tasks, then it is allocated to the subprocess level test. However, if the requirement relates to a performance measurement it is allocated to the process level test.
5. If a requirement needs full simulation to be satisfied, it is allocated to the process level test.
6. If the requirement cannot be sufficiently tested at one of the three levels (e.g., operational hardware required), it is identified for customer's disposition/ system level testing.

Each requirement is analyzed in the context of the above allocation criteria to select the level best suited for test. Confidence testing (repetition of comprehensive test cases to insure non-degradations due to modifications/changes), retesting (repeat testing due to a previous failure), test drivers required, analysis required, post

processors required, and configuration control of the software are included in this determination. Requirements are allocated to the lowest level of testing at which they can meet the test sufficiency guidelines defined later in this paper. If all of the requirements cannot be tested sufficiently at this level, the requirements are broken down and portions allocated to different levels of testing. The allocation of requirements to test levels is documented by the TEMs.

This allocation procedure is not a static process. Requirements continue to be reviewed and analyzed throughout the test program. If an assessment is made that an initial allocation is either incorrect (e.g., does not meet criteria for test sufficiency) or incomplete (e.g., test did not demonstrate requirement sufficiently), the allocations are revised and test procedures adjusted.

REQUIREMENTS TESTING

Proper selection of test scenarios is key to test effectiveness, particularly at the process level, due to the resource and schedule tradeoffs that must be made. Thus it was necessary to select test scenarios which:

1) satisfied test objectives
2) were representative of KMR test missions
3) exercised requirements allocated to that level of testing.

Test cases were derived from these scenarios by selecting the target configuration and user options necessary to drive the subprocess or process through the desired paths. Once the test scenarios were defined and approved by MDAC and requirements allocated to the test levels, the process level test procedures were generated. The test procedures describe each step required to execute the test, analyze the results relative to acceptance criteria, and determine whether the requirements were satisfied. Acceptance criteria derived from the requirements were explicitly defined in the procedures.

To allocate requirements to individual test procedures, it was necessary to identify all threads or entry-exit paths through the software which the test case would execute. Then the requirements which would be executed by the threads were identified and allocated to the appropriate procedure.

Once this process was completed for each procedure a very important milestone from a visibility and control viewpoint was achieved -- the first planned confirmation that the testing of requirements was attainable with those test cases. These data (the mapping from requirement to test procedure and test case) were entered into the TEM data base and reported. More importantly, once all test procedures were defined, the remaining requirements not mapped to procedures were identified for disposition and possible reallocation. If these requirements could not be tested within the scope of the defined test cases or hardware configuration they were identified to MDAC/TRW management for disposition. An assessment of the effect on KMR operability was made to determine whether these requirements warranted a new test scenario or should be allocated to the system test level (hardware/software integration testing).

The sample TEM shown in Table 1 depicts examples of requirements allocated to design solutions and test levels and also requirements which have completed testing and reflect their status. Examples of completely and partially tested requirements are shown with a requirement that failed testing initially and required a design modification and retest.

TEST SUFFICIENCY CRITERIA

The establishment of specific criteria for testing an individual requirement sufficiently must be based on the acceptance criteria derived from the requirement itself, the judgment of the tester, and the guidelines listed below:

1. Each requirement is assessed for test sufficiency on an individual basis by the tester or test director/conductor. That is, it may be assumed that all interfacing software has satisfied its requirements and the specifications are complete, compatible, and correct. (This assumption is actually an inference based on integration, regression, and evaluation testing results.)

2. If more than one test is necessary to satisfy the requirement stated in the specification, the requirement is assessed as tested insufficiently (partially) until all portions necessary to satisfy the acceptance criteria have been met.

3. Hypothetical cases (except where specified as requirements) need not be included in the determination of sufficiency of testing the requirement unless, in the judgment of the test director, risk, cost, and schedule tradeoff justify testing.

4. Error condition requirements testing is subject to the same sufficiency criteria as other requirements.

5. At the subprocess and process levels, if the requirement test meets its acceptance criteria for the data base and scenario values baselines, the requirement is judged to be tested sufficiently. At the unit level, "range of variable" testing is a necessary condition for test sufficiency (see 7. below).

6. If test results show that a requirement is satisfied under one set of conditions, but, in the judgment of the tester, other conditions/paths must be exercised to sufficiently test the requirement, he will do so provided such conditions are in scope for site operability.

7. A necessary condition for testing message requirements is that the message must be unpacked and its data for each field compared against the content specified by the requirement for those conditions. If other conditions in scope for software testing to the functional interface are necessary for complete satisfaction of the requirement (e.g., other field options), they are exercised and analyzed similarly to establish test sufficiency. Range of variables is defined as a nominal and minimum and maximum values for the parameter(s) in question, as well as singular points and discontinuities.

8. External drivers may be in scope at the unit and subprocess levels of testing to establish sufficiency, but shall not be used for process level testing. If, in the judgment of the tester, such drivers or instrumentation distort the software execution to such an extent that reasonable doubt exists that the software may satisfy the requirement without these techniques, this is reported as a qualifier on the sufficiency of testing.

9. A necessary condition for test sufficiency is that documentation of the test results and conclusions is sufficient for an independent technical reviewer to assess whether the requirement was test sufficiently. This documentation need not be all-inclusive in that references to back up documentation may be included provided such documentation is available for access by others, e.g., Unit Development Folders,[5,6] Test Data Folders, etc.

10. In case of breakage of a routine (i.e., a routine is opened to add more capability or changes existing code), this routine and its allocated requirements are tested at the unit level as a new routine.

TRACEABILITY

Documentation of test results provides the means for tracing the completed requirements tests back to the original specifications. STP test management requires a precise means for assessing the status of test by determining where a requirement is allocated to the design solution; at what level(s) is it allocated for test; what specific test procedure(s)/test case(s) exercise the code; what method of test is employed; is the verification sufficient; what, if any, discrepancy reports were issued against the code during the test phases; and which reports documented the test results. The response to these issues allows test management to assess where the problem areas were, what the impact of proposed solutions was, and when testing was adequately completed.

The Test Evaluation Matrix served as an excellent recording device to provide "snapshots" of this data in summary form. This management tool was utilized to maintain a dynamic test baseline which allowed for tracing changes to the requirements baseline while maintaining test status traceability. At any given time, a computerized information management system provided management with a snapshot of completion data for any level of test performed (Table 2). Reallocation of requirements to different design areas or test levels was maintained in this on-line system and provided the audit capability to insure that specified requirements were implemented and tested.

The TEMs capability for expanded traceability/allocation appears to be well suited for inclusion in the SREM[4] tools (e.g., REVS).

TEST REPORTING

STP project policy is that a requirement is not tested unless substantiated by documentation. The documentation must be sufficient for an independent reviewer to conclude whether or not the requirement was sufficiently tested. The documentation should identify any qualifications which prevent such a conclusion as well as identify which embedded requirements, if any, have not been sufficiently tested. The TEMs were employed as the means for tracing the documentation which substantiated sufficiency of testing.

FAILED REQUIREMENTS

Some requirements testing failed their initial attempt at verification. In most cases a change to source code (Quick Fix) or data values was created and applied against baselined software to provide an interim real or work-around solution identified on a Discrepancy Report (DR). These modifications were treated as software changes and required retesting and verification. In most cases when the quick-fixes were approved and implemented by the Configuration Control Board they were allocated back to their original test level. In some instances the affected requirement was reallocated to a higher level testing to reduce the risk of introducing unforeseen degradation of related software. The status of these failed requirements was identified as "failed" in the TEM data base until sufficient verification occurred and the DR was closed out. The TEM data base was then updated to report this completed verification.

CYCLE UPDATES

TRW's software development procedures called for planned block updates to the end item software to accommodate new capabilities, specification changes, and corrections to errors. These are defined as cycle updates. Management closely monitors these updates because they have the possibility of invalidating the results of previous tests, specifically requirements which had been satisfied by testing of a previous configuration of the software. The TEMs, coupled with a good confidence test program, have provided an effective method of managing requirements testing in the presence of cycle updates.

TABLE 2
STP REQUIREMENTS TESTING SUMMARY

		CR1	CR2	CR3	TOTAL
1)	NO. OF REQUIREMENT ENTRIES	5881	502	4151	10534
2)	NO. OF TESTABLE REQUIREMENTS	3603	477	2089	6169
3)	NO. OF REQUIREMENTS REFERENCED BY MORE THAN ONE SPEC.	847	9	82	938
4)	NO. OF UNIQUELY TESTABLE CRITICAL REQMTS	2796	468	2007	5271
5)	NO. OF COMPLETELY TESTED REQMTS	2277	406	1603	4286
6)	NO. OF PARTIALLY TESTED REQMTS	303	23	127	453
7)	NO. OF REQMTS ALLOCATED BUT NOT YET TESTED	0	0	110	110
8)	% TESTED (5 + 6 + 7)/4	92.3%	91.7%	91.7%	92.0%

The TEM reports, reissued for each cycle update, identify any changes in requirements as well as the requirements which were modified and therefore retested. New requirements are allocated to the appropriate level of testing as described above. Modified requirements are generally retested at the level to which they were originally allocated. Again, these allocations and the ultimate test results are recorded and published in the TEMs to provide the traceability and accountability required for proper planning and control.

Confidence testing addresses the problem of assuring that the conclusions drawn from previous tests (that the requirements are satisfied) remain valid in the presence of cycle updates to the code. The low risk/high cost solution to this objective is to repeat each of the tests. A significant cost savings was realized with moderate risk by implementing a plan whereby key test cases are exercised and compared against previous results for each cycle update. Differences are analyzed and explained based on the code differences between cycles. Thus if there are no unexplained differences, it is inferred, at moderate risk, that previous results remain valid and the requirements tested are still satisfied.

CONCLUSION

None of the management techniques described is new or unique; most are derived from applying a logical solution to a well defined problem. Nevertheless, the combination of these techniques, that is, the methodology, has been shown to be highly effective for the STP software. Based upon the complete success of the test missions at KMR to date (100 percent software successes based on mission objectives), the software reliability record is outstanding. The software test costs resulted in a substantial reduction from the original program's target costs as well as the often quoted industry average of 40 percent[7] and even below TRW's historical average of 34 percent.[8] The increased risk inherent in the methodology appears well justified based upon the software performance in the field. It is believed that the methodology will be effective for other software development programs where the end product is delivery of large scale, real-time programs.

REFERENCES

[1] R. G. Schluter, "Experience in Managing the Development of Large Real-Time BMD Software Systems," AIAA/NASA/IEEE/ACM Computers in Aerospace Conference, Los Angeles,CA,Oct-Nov 77.

[2] R. D. Williams, "Managing the Development of Reliable Software," Proc.Internat'l Conf. on Reliable Software, Los Angeles,CA, 21-23 Apr 75.

[3] F. J. Mullin, "Considerations for a Successful Software Test Program," AIAA Computers in Aerospace Conf, Los Angeles, CA, Oct-Nov 77.

[4] C. G. Davis and C. R. Vick, "The Software Development System," IEEE Transactions on Software Eng., Vol. SE-3, No. 1, Jan. 77.

[5] F. S. Ingrassia, "The Unit Development Folder (UDF): An Effective Management Tool for Software Development, DATAMATION, Vol.24,No.1,Jan.78

[6] E. A. Goldberg, "Applying Corporate Software Development Policies," Proc. of AIAA Third Software Management Conf., Washington,D.C., Dec.77.

[7] B. W. Boehm, "Software and its Impact: A Quantitative Assessment," DATAMATION, Vol.19, pp. 48-59, May 73.

[8] L. H. Putman and R. W. Wolverton, "Quantitative Management: Software Cost Estimating," IEEE Catalog No. EHO 129-7, COMPSAC '77 Meeting, Chicago, Ill., Nov. 77.