Requirements, Testing, and Metrics

Linda H. Rosenberg, PhD
Unisys/GSFC
301-286-0087
Linda.H.Rosenberg@pop300.gsfc.nasa.gov

Theodore F. Hammer
NASA GSFC
301-286-7475
thammer@pop300.gsfc.nasa.gov

Lenore L. Huffman
Unisys/GSFC
301-286-0099
Lhuffman@pop300.gsfc.nasa.gov

Key Words: Testing, Requirements, Metrics

ABSTRACT

The criticality of correct, complete, testable requirements is a fundamental tenet of software engineering. Also critical is complete requirements based testing of the final product. Modern tools for managing requirements allow new metrics to be used in support of both of these critical processes. Using these tools, potential problems with the quality of the requirements and the test plan can be identified early in the life cycle. Some of these quality factors include: ambiguous or incomplete requirements, poorly designed requirements databases, excessive or insufficient test cases, and incomplete linkage of tests to requirements. This paper discusses how metrics can be used to evaluate the quality of the requirements and test to avoid problems later.

Requirements management and requirements based testing have always been critical in the implementation of high quality software systems. Recently, automated tools have become available to support requirements management. At NASA's Goddard Space Flight Center (GSFC), automated requirements management tools are being used on several large projects. The use of these tools opens the door to innovative uses of metrics in characterizing test plan quality and assessing overall testing risks. In support of these projects, the Software Assurance Technology Center (SATC) is working to develop and apply a metrics program that utilizes the information now available through the application of requirements management tools. Metrics based on this information provides real-time insight into the testing of requirements and these metrics assist the Project Quality Office in its testing oversight role. This paper discusses three facets of the SATC's efforts to evaluate the quality of the requirements and test plan early in the life cycle, thus preventing costly errors and time delays later. Data from NASA projects are used to support and clarify the concepts discussed.

1. Introduction

The National Aeronautics and Space Agency (NASA) is increasingly reliant on software for the functionality of the systems it develops and uses. The Agency has recognized the importance of improving the way it develops software, and has adopted a software strategic plan to guide the improvement process. At NASA's Goddard Space Flight Center (GSFC), the Software Assurance Technology Center (SATC) and the project Quality Assurance Office are working together to develop and apply a metrics program that utilizes the information available in the requirements phase of the software development life cycle. Metrics based on this information provides insight into the testing of requirements; this information assists the Quality Assurance Office in its project oversight role.

Requirements development and management have always been critical in the implementation of software systems—engineers are unable to build what analysts can't define. Recently, automated tools have become available to support requirements management. The use of these tools not only provides support in the definition and tracing of requirements, but it also opens the door to effective use of metrics in characterizing and assessing testing. Metrics are important because of the benefits associated with early detection and correction of problems with requirements. Problems that are not found until testing are at least 14 times more costly to fix than if the problem was found in the requirement phase.[2]

The first group of testing metrics activities that will be discussed involve the development of a tool and its application early in the life cycle in order to assess the quality of requirements. This paper describes application of

the Automated Requirements Measurement (ARM) tool. ARM parses the text of the requirements into identifiable units in order to evaluate potential words or phrases that may affect their testability. Because both the software acquirer and the software provider must understand and contractually agree to the requirements, specifications are usually written in natural language. The use of natural language to prescribe complex, dynamic systems has at least two severe problems: ambiguity and inaccuracy. Many words and phrases have dual meanings which can be altered by the context in which they are used. Defining a large, multi-dimensional capability within the limitations imposed by the two dimensional structure of a document can obscure the relationships between individual groups of requirements.

Once requirements are written, methods for ensuring that the system contains the functionality specified must be developed. The next group of testing metrics activities we investigate relate to the test plan links between test cases and the requirements. Using NASA project data we will look at the linkage of the requirements, the relationship between unique requirements and unique tests, and the ratios of test to requirement links. In this section of the paper we discuss three efforts to evaluate the quality of the test plan while still in the requirements phase.

And finally, this paper investigates the quality of the requirements management database schema as it relates to cleanliness of data and the ease with which requirement and testing metrics can be obtained. In the preparation of the database that houses the requirements and tests, both the requirement segment and the test segment must be designed with the identical schema design philosophies; this to enable evaluation of the test plan links to requirements as they are entered into the database. This paper briefly discusses a requirements database schema that supports comprehensive evaluation of requirements driven testing.

There are no published or industry guidelines or standards for these testing metrics—intuitive interpretations, based on experience and supported by project feedback, are used in this paper. NASA project management has reacted favorably to these metrics and has used the analysis results to mitigate perceived risks. The SATC continues working on methods to mathematically validate the intuitive guidelines. The objective is to assist project management in producing high-quality requirements and test plans while identifying and minimizing project risks.

2. NASA Development Environment

In order to demonstrate how metrics can provide the insight needed to get the requirements right, data from two large NASA projects will be used. While the projects must remain anonymous, a general understanding of the development environment is necessary. These projects are implementing large systems in multiple incremental builds.[1] The development of these builds is overlapping, design and coding of the second and third builds having been started prior to the completion of the first build. Each build adds new functionality to the previous build and satisfies a further set of requirements.

The definition of requirements for this system started with the formulation of System Level Requirements, referred to as “Level 1” requirements. These are mission-level requirements for the spacecraft and ground system; they are at a very high level and rarely, if ever, change. Level 1 requirements then undergo decomposition to produce Allocated Requirements, called “Level 2”; these requirements are also high-level and change should be minimal. Project development starts at this requirement level. (We will not discuss Level 1 or Level 2 requirements.) Level 2 requirements are then divided into subsystems and a further level is derived in greater detail; hence, “Level 3 Derived Requirements.” Each requirement in Level 2 traces to one or more requirements in Level 3. This is a bi-directional tracing, with Level 3 requirements refocusing into Level 2 requirements. The Detailed Requirements are found in “Level 4” requirements; these requirements are used to design and code the system. There is also a bi-directional tracing between Level 3 requirements and Level 4 requirements. To verify the requirement, two stages of testing are used. System Tests are designed to verify the Level 4 requirements and then Acceptance Tests are to be used to verify the Level 3 requirements.[8]

[1] Various names are used—deliveries, releases, builds—but the term *build* will be used in this paper.

3. Requirement Specification

The importance of correctly documenting requirements has caused the software industry to produce a significant number of aids to the creation and management of the requirements specification documents and individual specifications statements. However very few of these aids assist in evaluating the quality of the requirements document or the individual specification statements themselves. The SATC has developed a tool to parse requirements documents. The Automated Requirements Measurement (ARM) software was developed for scanning a file that contains the text of the requirements specification. During this scan process, it searches each line of text for specific words and phrases. These search arguments (specific words and phrases) are indicated by the SATC's studies to be an indicator of the document's quality as a specification of requirements. ARM has been applied to 56 NASA requirement documents. Seven measures were developed, as shown below.

1. Lines of Text - Physical lines of text as a measure of size.
2. Imperatives - Words and phases that command that something must be done or provided. The number of imperatives is used as a base requirements count.
3. Continuances -Phrases that follow an imperative and introduce the specification of requirements at a lower level, for a supplemental requirement count.
4. Directives – References provided to figures, tables, or notes.
5. Weak Phrases - Clauses that are apt to cause uncertainty and leave room for multiple interpretations measure of ambiguity.
6. Incomplete – Statements within the document that have TBD (To be Determined) or TBS (To Be Supplied).
7. Options - Words that seem to give the developer latitude in satisfying the specifications but can be ambiguous.

It must be emphasized that the tool does not attempt to assess the correctness of the requirements specified. It assesses individual specification statements and the vocabulary used to state the requirements; it also has the capability to assess the structure of the requirements document. [10]

The results of the analysis of the Level 3 and Level 4 requirements are shown in Table 1 with the comparison to other NASA documents.

56 DOCUMENT	**LINES OF TEXT** - Count of the physical lines of text	**Imperatives** - shall, must, will, should, is required to, are applicable, responsible for	**Continuances** - as follows, following, listed, inparticular, support	**Directives** - figure, table, for example, note:	**Weak Phrases** - adequate, as applicable, as appropriate, as a ninimum, be able to, be capable, easy, effective, not limited to, if practical	**Incomplete** - TBD, TBS, TBR	**Options** - can, may, optionally
Minimum	143	25	15	0	0	0	0
Median	2265	382	183	21	37	7	27
Average	4772	682	423	49	70	25	63
Maximum	28459	3896	118	224	4	32	130
Stdev	759	156	99	12	21	20	39
Level 3 FOS	1011	588	577	10	242	1	5
Level 4 FOS	1432	917	289	9	393	2	2

Table 1 : Textual Requirement Analysis

We are especially concerned with the number of weak phrases since the contract is bid using Level 3 and acceptance testing will be against these requirements. It is also of concern that the number of weak phrases has increased in Level 4, the requirements used to write the design and code and used in Integration testing.

An easy way to compare Project X with other NASA documents is to normalize by lines of text, shown in Figure 1.

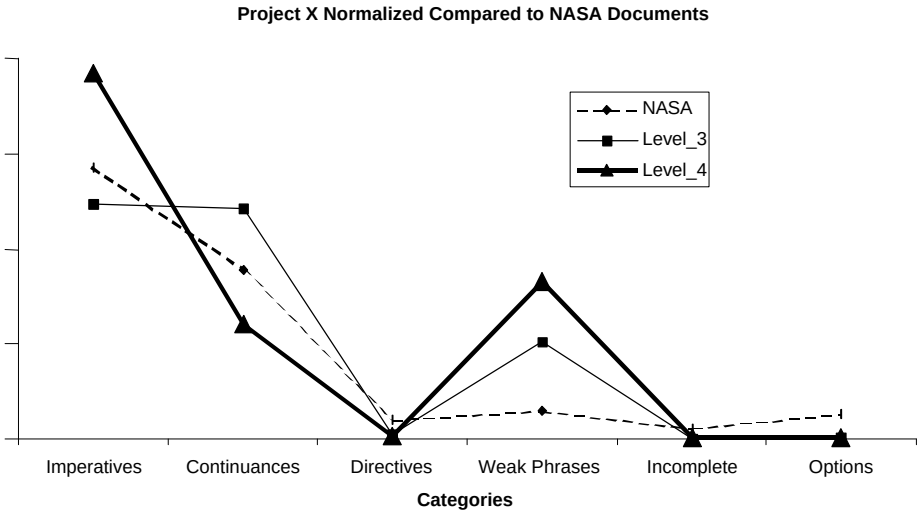

Figure 1 : Project X Normalized and Compared to NASA Documents

From Figure 1 it can be seen that Project X documents are terse without excess extraneous information and with few continuances or directives. This is probably a result of the requirements being analyzed from a database as opposed to a textual document where additional text would be expected. Project X does have a very high number of weak or ambiguous phrases as discussed previously, but few incomplete and optional phrases.

4. Testing Characterization

Once requirements are written, methods for ensuring that the system contains the functionality specified must be developed; this section of the paper discusses three efforts to evaluate testing in the requirements phase. To validate the requirements, test plans are written that contain multiple test cases; each test case is based on one system state and tests some functions that are based on a related set of requirements.[8]

In the total set of test cases, each requirement must be tested at least once, and some requirements will be tested several times because they are involved in multiple system states in varying scenarios and in different ways. But as always, time and funding are issues; while each requirement must be comprehensively tested, limited time and limited budget are always constraints upon writing and running test cases.[2] It is important to ensure that each requirement is adequately, but not excessively, tested. In some cases, the requirements can be grouped together using criticality to mission success as their common thread; these *must* be extensively tested. In other cases, requirements can be identified as low criticality; if a problem occurs, their functionality does not affect mission success while still achieving successful testing.[1,8,9] In order to ascertain the point at which testing benefits become marginal, the SATC developed a third set of metrics based on data available in a requirements management tool using the information design discussed in the previous section.

These metric analyses use the linking information of the requirements to the tests in three ways. The first is to verify that each requirement is tested at least once. The next two analyses characterize the depth and breadth of

[2] For simplicity in this paper, we will use the term *test case* to refer to any type of test for verification and validation of requirement functionality.

the test plan. It is expected that each requirement will be linked to multiple test cases, and that each test case will test multiple requirements.[1,8,9] Data from Project Y is used to demonstrate the metrics application and interpretation in this section.

4A Test Coverage

The first objective is to verify that each requirement will be tested; the implication is that if the software passes the test, the requirement's functionality is successfully included in the system. This is done by determining that each requirement is linked to at least one test case.[6] A query such as those shown below would result in data that could be displayed in a graph shown in Figure 2.

Query: How many requirements in Level 4 Build 1 are linked to a test case?
Query: How many requirements in Level 4 Build 2 are linked to a test case?
Query: How many requirements in Level 4 Build 3 are linked to a test case?

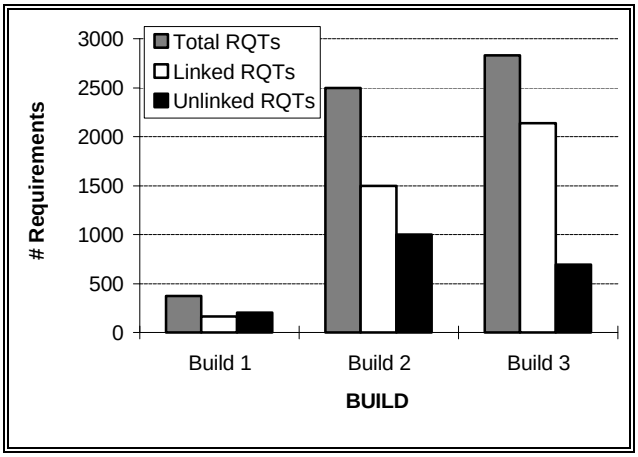

Figure 2: Level 4 Requirement Linkage to Tests

Build 1 appears greater than 60% unlinked due to database problems, the database was not created until after Build 1, hence most of the data from Build 1 was not entered. Build 2 is currently testing Level 4 requirements, but 40% of the requirements are not linked to any test. This data indicates that there is no way to verify whether the functionality of 1,000 requirements is included in the system. It may be possible to link some of the requirements to existing test cases with minimum modification to the test data. If new test cases must be developed, budgetary problems may be created and the testing schedule must be increased. In all cases, further investigation of the missing links is warranted.

For Build 3, just starting the coding phase (coding continues for approximately 10 additional months), only 25% of the requirements are not linked to test cases. This situation needs to be monitored on a monthly basis but is not one for major concern at this time.

4B Test Span

This activity characterizes the test plan and identifies potentially insufficient or excess testing. Requirements are usually tested by more than one test case, and one test case usually covers more than one requirement.[6] Since each test costs money and takes time, the obvious questions are how may requirements are covered by one test, and how many tests cover only one requirement. On the other hand, if requirements are insufficiently tested, functionality may not be verified. The metrics for this analysis are in two parts because of the bi-directional linkage between the requirements and tests. Each direction yields different information. Counting the number of unique tests used for a requirement indicates that requirements at both ends of the graph may have too

much or too little testing. Counting the number of unique requirements tested indicates the exclusivity of the testing.[3] Due to space limitations, we will demonstrate only one direction - unique requirements per test to identify excessive or insufficient testing.

Figure 3 shows an expected profile of unique requirements per test case based on data from NASA projects [5].

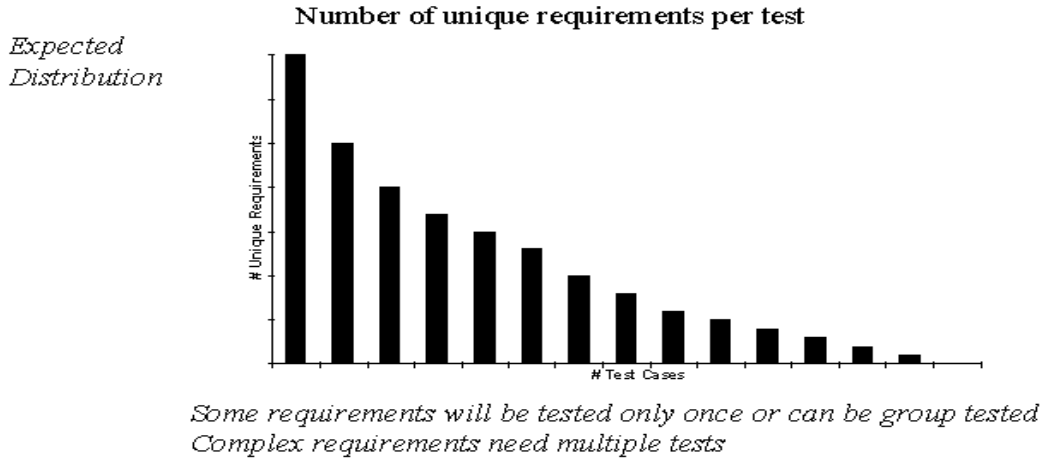

Figure 3 - Test Program Characterization Tests per Requirement

This profile shows that there is an expectation that there will be a large number of requirements tested by only one test case, and that there will be some number of requirements that will be tested by a multiple number of test cases. It is expected that the upper bound of multiple test cases will range in the tens. This makes sense, as more complicated requirements may require different test cases to thoroughly verify all aspects of the requirement. However, there is a limit on the number of test cases. As the number of test cases increases the difficulty of verifying the requirement also increases. This difficulty arises due to the complication in data analysis, understanding the results of the multiple tests cases, and understanding the impact of multiple test case results on the verification of the requirement.

Number of tests per requirements counts the number of unique tests associated with each test. A program query such as the one below might be used.

Query: How many requirements are tested by Test A.1? (Acceptance test, Test1)
Query: How many requirements are tested by Test A.2? (Acceptance test, Test2)
Query: How many requirements are tested by Test A.3? (Acceptance test, Test3)
. . .

The data is then summarized to count the number of unique requirements evaluated by a given test.

This data was compiled for Build 3 Level 4 requirements and is graphed in Figure 3 with bars. The profile curve (solid line) was derived to identify outliers - areas where testing may be insufficient or excessive. The X-axis is the Number of Unique Requirements and the Y-axis the Number of Test Cases.

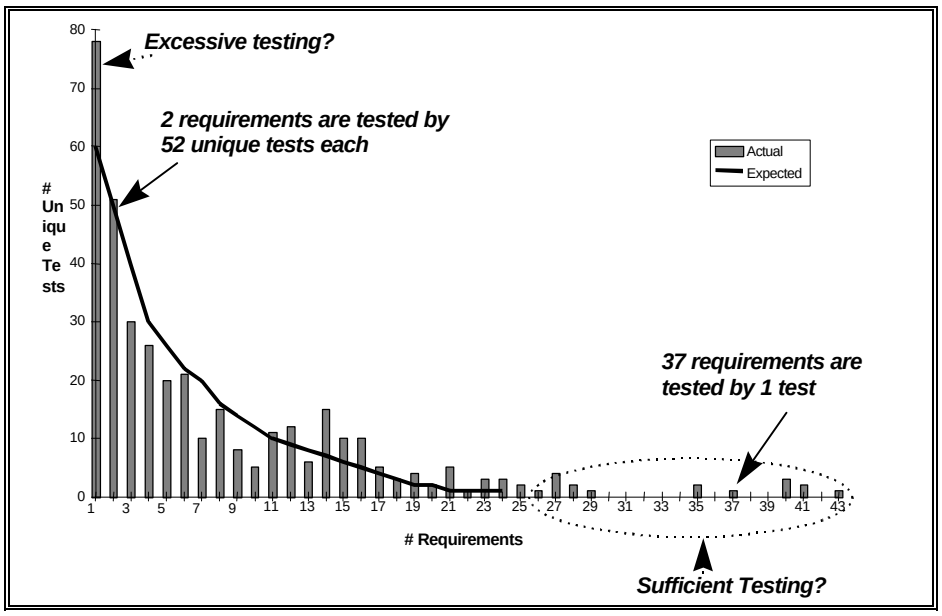

Figure 4: Sufficiency of Testing Plan

The analysis in Figure 3 shows the dilemma of structuring a test program. A testing criterion is to have a one to one relationship between tests and requirements. In this way the validation of requirements is isolated to single tests and so are easily verified. The problem with large systems is that a one to one relationship between tests and requirements will cause a large test program to be developed that will have a huge number of test cases. It will be too costly and too time consuming to complete, due to the number of test cases, the amount of test data required, and the large number of test sessions needed to execute all of the tests. Therefore a balance must be obtained where a one to one relationship between requirements and test cases is developed for critical requirements, but less critical requirements are tested in groups based on system states or functional threads.

4C Test Complexity

Figure 3 indicates there may be excessive testing scheduled for some Build 3 Level 4 requirements due to the large tests per requirement ratio seen on the left hand side of the chart. The next step in evaluating the requirements testing is to investigate the testing magnitude through the complexity of the linkages. One way this can be done is to look at the number of requirements, the number of linkages, and the number of tests. Recall, each link is a connection between a requirement and a test. This data presents a third view of the data previously presented. Figure 4 shows this raw data for Build 2 and Build 3, Level 3 and Level 4 requirements.

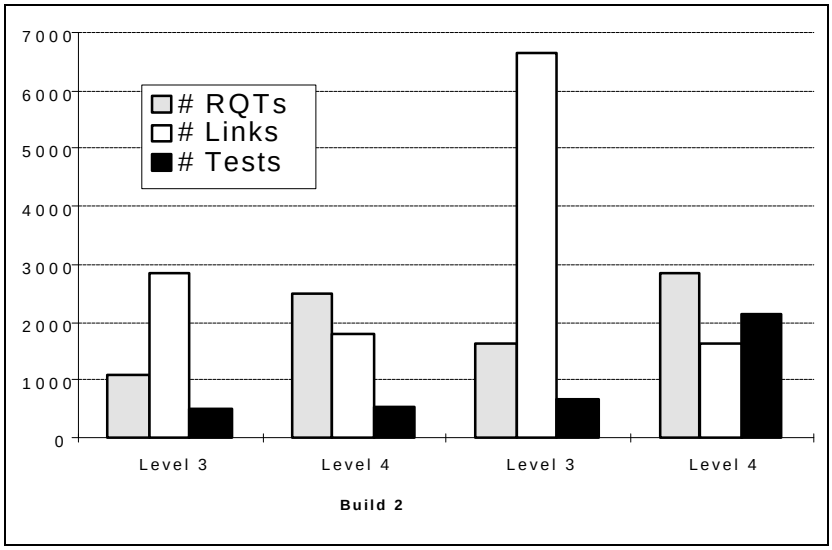

Figure 5: Requirements, Links, and Tests

Looking at Figure 4, it appears the number of links for Level 3 may be disproportionate. Table 2 shows the actual ratios.

Build	Level	Ratio Links to Requirements	Ratio Links to Tests	Ratio Requirements to Tests
2	3	2 : 1	5 : 1	2 : 1
3	3	4 : 1	10 : 1	4.5 : 1
2	4	0.5 : 1	3 : 1	4.5 : 1
3	4	0.75 : 1	4 : 1	1.25 : 1

Table 2: Ratio Links to Requirements

Figure 4 and Table 2 show a number of different perspectives of the test information. First look at the ratio of the number of links to the number of requirements summarized in Table 2 below. In the Level 3 requirements for both Build 2 and Build 3, there are at least two links for each requirement. This means that on average, each requirement is tested by at least two different tests. For Level 4 requirements however, there is less than one link for each requirement. This indicates there are some requirements that are not linked to any test, hence their functionality may not be verified.

Another way of viewing the data is to look at the total number of links to the total number of tests - are there too many? These ratios are also contained in Table 2. Looking at Level 3 requirements for Build 3, the graph in Figure 4 is reinforced by the ratios in Table 2, that show a ratio links to tests of 10 to 1. This indicates that although the number of tests seems adequate for Build 3 (Figure 4), the complexity of the test program is too high; that is, the linkage between requirements and tests is complex (Table 2).

As stated, when metrics indicate a potential problem, further investigation is needed. The tentative conclusion is that the test plan for Build 3 Level 3 requirements is too complex, but the last column in Table 2 indicates the ratio of requirements to tests for Build 3 Level 3 requirements is not out of line. It is likely that the number of tests is sufficient, but the number of links is excessive and may need to be decreased, thus decreasing the complexity of the tests. But while the last column in Table 2 resolves one concern, another is raised. Looking at the ratio of tests per requirement for Build 3 Level 4 requirements, there is a one to one ratio. This indicates a very large number of tests for this Build and Level, suggesting a potential risk of failing to complete testing within schedule and budget. These factors may all be pointing to poor test case design, or, they may simply be a lapse in procedures for requirements and test case management.

As discussed previously in this paper, there are no guidelines for these metrics since they are in research infancy; in evaluating the data in Table 2, we were looking for inconsistencies based on experience.

The objective of an effective verification program is to ensure that every requirement is tested, the implication being that if the system passes the test, the requirement's functionality in included in the delivered system [1,3]. An assessment of the traceability of the requirements to test cases is needed. It is expected that a requirement will be linked to a test case, and may well be linked to more that one test case as shown in Figure 6 [5,6].

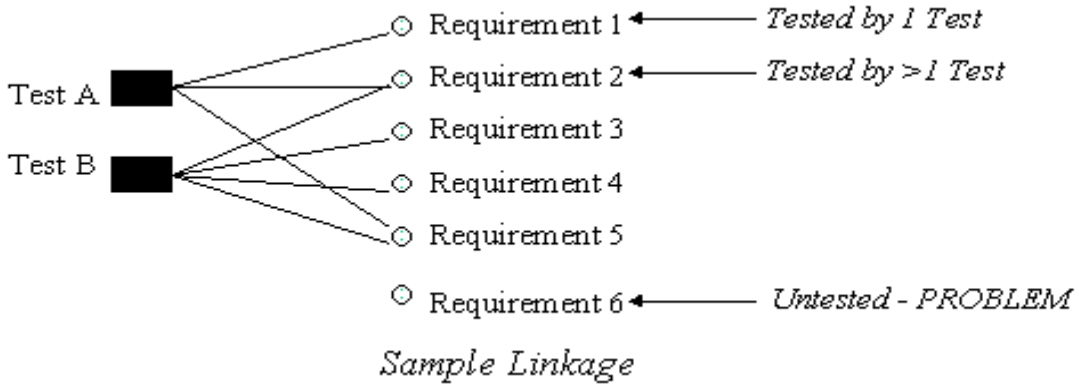

Figure 6 - Requirement Verification - Trace to Test Linkage

The important aspect of this analysis is to determine which requirements have not been linked to any test cases at all.

Figure 7 shows that the traceability of requirements to test cases for Project Y around the CDR time frame for Build 2. The information was extracted from the requirements management database used in support of the development effort. The profiles show several problems.

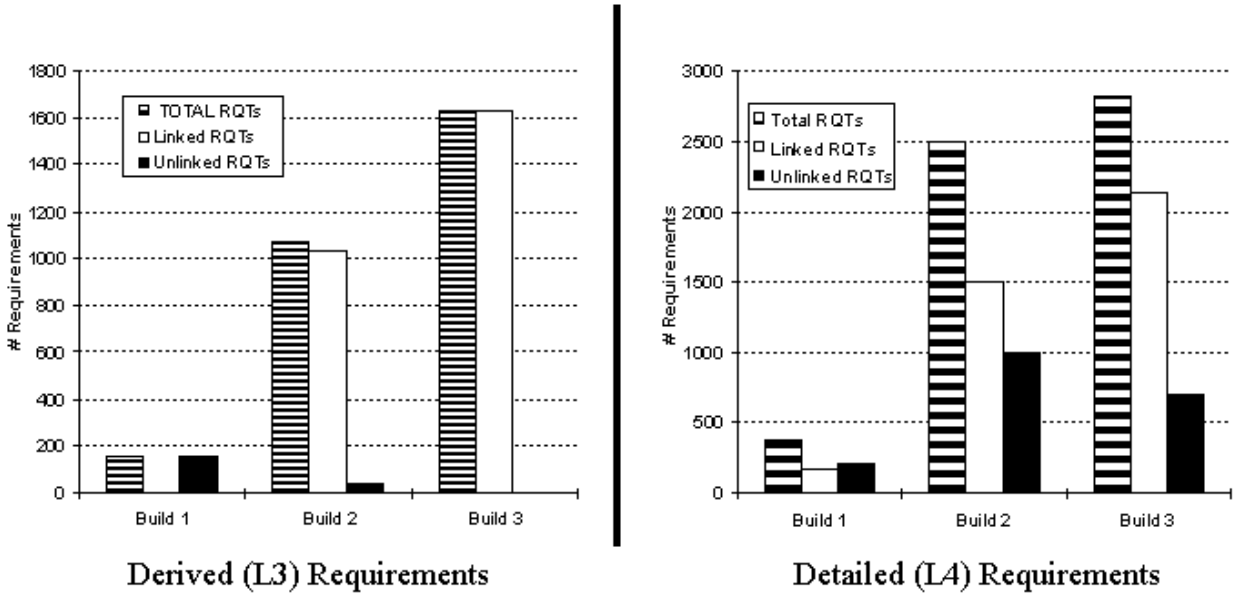

Figure 7 - Requirement Verification Trace to Test

First, the requirements management tool was not used effectively early in the project life cycle. This explains the poor traceability between the requirements and test cases for Build 1. Secondly, there seems to be a mix up in the test priorities by the implementer. The test program for Build 3 is further along than that for Build 2, when it is Build 2 that will be developed and tested before Build 3. Resources may have been inappropriately allocated to the development of the test program for Build 2. Lastly, the test program for the Level 4 requirements is behind that for the test program for the Level 3 requirements. Again, this is backwards. The first tests to be executed will be that for the Level 4 requirements, the system tests, and after that tests for the Level 3 requirements will be executed, the acceptance tests. An explanation for this problem may be found is a previously presented metric. Remember the metric showing the push of Level 4 requirement from Build 2 to Build 3. This movement of requirements from Build 2 to Build 3 may well be the cause of the lack of traceability of requirements to test cases. The test case developers may be having difficulty in keeping up with the changes in requirements resulting in a number of requirements in each build without a link to a test case.

5 Requirement Management

The use of tools to aid in the management of requirements has become an important aspect of system engineering and design. Considering the size and complexity of development efforts, the use of requirements management tools has become essential. The tools which requirement managers use for automating the requirements engineering process have reduced the drudgery in maintaining a project's requirement set and added the benefit of significant error reduction. Tools also provide capabilities far beyond those obtained from text-based maintenance and processing of requirements. Requirements management tools are sophisticated and complex – since the nature of the material for which they are responsible is finely detailed, time-sensitive, highly internally dependent, and can be continuously changing. Tools that simplify complex tasks require skill and a thorough understanding of their capabilities if they are to perform effectively over the lifetime of a project [7].

There are many requirement management tools to choose from. These range from simple word processors, to spreadsheets, to relational databases, to tools designed specifically for the management of requirements such as DOORS (Quality Systems & Software - Mt. Arlington, NJ) or RTM Requirements Traceability Management (Integrated Chipware, Inc. - Reston, VA). The key to selecting the appropriate tool is the functionality (See Table 3 for a comparison of tool capabilities) provided and the capability to develop metrics from the data, secondary contained in the tool.

	Word Processor	**Spreadsheet**	**Relational Database**	**Requirement Tool**
Document config. mgt	X		X	X
Document preparation	X			X
Function decomposition			X	X
Report preparation			X	X
Requirement allocation		X	X	X
Requirement config. mgt		X	X	X
Requirement expansion			X	X
Requirement importation				X
Requirement simplification				X
Requirement storage	X	X	X	X
Requirement traceability			X	X
Test coverage/adequacy			X	X
Metrics			X	X

Table 3 - Requirement Repository Capabilities

The metric capability of the tool is important. It should be noted that most of the metrics presented in this paper to demonstrate how to do requirements the right way were developed from the data contained in a requirement management tool. Table 4 shows a comparison of the metric capability associated with the different tools. Clearly the relational database and requirements management tool provide the capabilities needed to effectively support the management of requirements.

	Word Processor	Spreadsheet	Relational Database	Requirement Tool
Document size	X			
Dynamic changes over time				X
Release size	X	X	X	X
Requirement expansion profile			X	X
Requirement types	X	X	X	X
Requirement verification			X	X
Requirement volatility	X	X	X	X
Test coverage			X	X
Test span			X	X
Test types	X	X	X	X

Table 4 - Requirement Repository metric Capabilities

The selection of a tool is only part of the equation. A thorough understanding of the tool capabilities and the management processes that will use the tool is necessary. The tool should not be plugged into the management processes with no thought as to the impact on the tool capabilities. Adjustments may be needed in the management processes and employment of the tool to bring about an efficient requirements management process.

6 Conclusion

It is generally accepted that requirements are the foundation upon which the entire system is built. And that requirement verification and validation is needed to assure that the functionality representing the requirements has indeed been delivered. However, all too often requirements are not satisfied, leading to a process of fixing what you can and accepting the fact that certain functionality will not be there. A better approach is to get the requirements right the first time, complete, concise and clear, that will provide the implementer a clear blue print with which to build the system. This is not done by magic but through the application of tools and metric analysis techniques in the areas of requirement specification, requirement verification and requirement management

The use of an automated tool to track requirements and their test cases has opened the door to the use of new requirements and testing measures. The ARM tool can be used to point out requirements that may be ambiguous or otherwise poorly worded and thus subject to testing problems. Since the data base that contains the requirements can be repeatedly analyzed, quality trends can be tracked and partial sets of requirements can be monitored and investigated.

The same database contains the test data, which allows new measures that characterize a test program in terms of its structure and complexity, and to assess whether all requirements are verified by test cases. The use of an automated tool for requirements management is essential for gaining insights not otherwise available. The metrics presented here were the result of many different attempts to display and use the data. Key to this analysis was access to the requirement database.

Based on the work done to date, four conclusions can be reached:

- Requirement metrics assist in identifying potential project risks
- Metrics are available in the requirement phase to assess test plans

- Multiple metrics are needed for comprehensive evaluation
- Metric collection is cheaper, faster and more reliable with requirement management tools

REFERENCES

1. Beiser, B. *Software Testing Techniques* (1983), Van Nostrand Reinhold Company.
2. Boehm, B. *Tutorial: Software Risk Management* (1989), IEEE Computer Society Press.
3. Hammer, T., Huffman, L., Wilson, W., Rosenberg, L., Hyatt, L., Requirement Metrics—Value Added in *Proc. Third IEEE International Symposium on Requirements Engineering*, (Annapolis MD, January 1997) IEEE Computer Society Press.
4. Hammer, T., Rosenberg, L., Huffman, L., Hyatt, L., Measuring Requirements Testing in *Proc. International Conference on Software Engineering* (Boston MA, May 1997) IEEE Computer Society Press.
5. Kitchenham, Barbara, Pfleeger, Shari Lawrence, Software Quality: The Elusive Target, *IEEE Software 13, 1* (January 1996) 12-21.
6. Marconi Systems Technology, *RTM User's Manual* (1994).
7. Software Technology Support Center, *Software Test Technologies Report* (August 1994).
8. Wilson, W., Rosenberg, L., Hyatt, L., Automated Quality Analysis of Natural Language Requirement Specifications in *Proc. Fourteenth Annual Pacific Northwest Software Quality Conference*, (Portland OR, October 1996).
9. Brooks, Frederick P. Jr., No Silver Bullet: Essence and accidents of software engineering, *IEEE Computer, vol. 15, no. 1,* April 1987, pp. 10-18.
10. Hammer, T., Huffman, L., Rosenberg, L., Wilson, W., Hyatt, L., "Requirement Metrics for Risk Identification", Software Engineering Laboratory Workshop, GSFC, 12/96.
11. NASA, *Software Assurance Guidebook*, NASA Goddard Space Flight Center Office of Safety, Reliability, Maintainability, and Quality Assurance, 9/89.
12. Wilson, W., Rosenberg, L., Hyatt, L., "Automated Analysis of Requirement Specifications", Fourteenth Annual Pacific Northwest Software Quality Conference, 10/96.
13. Hammer,T., "Measuring Requirement Testing", 18th International Conference on Software Engineering, 5/97.
14. Hammer,T., "Automated Requirements Management – Beware How You Use Tools", 19th International Conference on Software Engineering, 4/98.
15. Hansen, Gary W., Hansen, James V., Database Management and Design, Prentice Hall, 1992.
16. Chen, M., Han, J., Yu, P. "Data Mining: An Overview from a Database Perspective", IEEE Transactions on knowledge and Data Engineering, Vol 8, No. 6, 12/96