Automated Requirements Management – Beware HOW You Use Tools
An Experience Report

Theodore Hammer
NASA GSFC
Bld 6 Code 300
301-614-5225
Thammer@pop300.gsfc.nasa.gov

Lenore Huffman
SATC/Unisys
Bld 6 Code 300.1
301-286-0099
Lhuffman@pop300.gsfc.nasa.gov

Abstract

At NASA and across industry, with multiple release projects, requirement storage is a volatile, dynamic process. The skill with which a project maintains, keeps current, tracks, and traces its set of requirements affects every phase of the project's software development life cycle - including maintenance. The ability to effectively manage requirements influences, months and/or years before project completion, how, when, and how expensively completion will take place. The Software Assurance Technology Center (SATC) at NASA Goddard Space Flight Center is working to continually evaluate the requirement activities of a multi-billion dollar project. This task requires that the SATC evaluate the project's tools, specifically the requirement database management tool.

The objective of this paper is to identify a major failing in the use of requirement management tools that causes loss of data, loss of data integrity, and loss of tool functionality. It will also show that bringing engineers and database designers together to define tool use is mandatory; and that each discipline's experience and expertise is required for successful tool implementation. Using experiences from a NASA project, we will demonstrate some potential risks when a requirement management tool is incorrectly used, and how this fatal flaw plants the seeds of requirement management destruction and consequent project overruns. This information will benefit any project considering or using requirement management tools.

Key Words: Requirements and Requirement Management Tools

I. Introduction.

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.

The requirement management tools available today require a high degree of knowledge not only in the potential application of the tool but also in the actual use of the tool base itself. Successful application of a requirement management tool implies knowledge in requirement development and also knowledge in database design and application. [Marconi] Unfortunately, requirement engineers are often given the task of designing the structure of the requirement database and this can lead to the fatal flaw of the inexperienced designing complex procedures thus causing adverse effects on the functionality of the database and the integrity of the data.

A common failing has been the manner in which tools are used to support requirement management processes starting at the initial development phase. This paper will discuss some capabilities of requirement management tools that, if not developed properly, can have a negative influence on the

software development process. We will use a large NASA software development project to demonstrate the problems resulting from the misapplication of requirement management tools when the features are not fully understood and properly activated. In our description, we assume that the reader has little or no knowledge of database theory and application, but does have intimate knowledge of requirements and their management. The tool we discuss is a commercial product, and for the purposes of this paper, we call it the Requirement Management Tool (RMT). Not all of its capabilities are discussed in this paper, only those which have the greatest effect on our project's ability to manage its requirement set. These features are ones generally found in requirement management tools and are described in the generic sense – that is, if another tool under consideration has these utilities, then by definition it is a requirement management tool. Consequently, using another requirement management tool for this project would not have solved any of the problems discussed. It is the project *modus operandi* which is faulty. And further, should a tool without these capabilities be used, for example - a simple relational database without the specific software layer to manipulate requirements, the design flaws would remain and the requirements would simply be stored and not managed. The key to effective requirement management is to have engineers and database engineers work together to understand tool capabilities and to establish a proper fit of the tool with the desired requirements management process.

II. Development Environment.

At NASA and across industry, projects are built in multiple builds, each having a specific set of requirements linked to a particular build. Project X[1], used for demonstration in this paper, has 4 major builds spanning 5 years of development. At NASA, requirement definition starts 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 several levels of decomposition to produce Allocated Requirements, called "Level 2"; these requirements are also high-level and change should be minimal. Project X's development started at this requirement level. Level 2 requirements are then divided into subsystems and a further level is derived in greater detail; hence, "Level 3 Derived Requirements." Generally, contracts are bid using this level of requirement detail. 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. [NASA] There is also a bi-directional tracing between Level 3 requirements and Level 4 requirements. This traceability is critical to ensuring that the completed system contains all functionality specified at each level and that no additional functionality creeps into the final system, increasing risks, cost, and time to complete. This is shown in Figure 1 below. [Hammer]

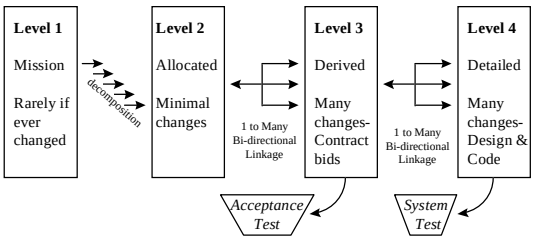

Figure 1. Requirement Expansion & Tracing

III. Requirement Management Tool.

A Requirement Management Tool (RMT) has specific database capabilities that are needed for effective management of the requirements. Requirements must be tracked as the level of detail is increased and as they are assigned to builds. This is a volatile, dynamic process as requirements are frequently re-allocated between different builds. Some important considerations to the requirement management profession are requirement storage, retrieval (in formats needed by a variety of users), traceability from one requirement level to the next, modifiability (e.g. enhancements), expansion, consolidation, inheritance of descriptors, history, parentage and consistency. The RMT provides for all these considerations through its utilities and/or transparent system processes as well as its inherent powers of relational design. How well the RMT accomplishes these tasks is dependent upon how carefully the tool is configured at the very beginning of the project. [Marconi] Each decision to include, exclude or to not thoroughly understand the full powers of the tool affects the project throughout its lifetime. Attempts to correct initial configuration errors after project start-up only build upon and magnify the effects of the original decisions.

One of the first decisions the requirement management team must make, and often the decision

[1] Anonymity of all SATC projects is guaranteed so project names must be omitted.

with the largest consequences, is the structure of the RMT for the project. The RMT uses the concept of classes as a unit for requirement storage and manipulation. A class roughly corresponds to a relation in a relational database[2] and possesses all the relational powers available to it. Requirements found in multiple documents may reside within that class, so that the delimiting factor for inclusion in a class is a document's level within the software development phase hierarchy. For example, many documents may describe the system at Level 2, i.e., systems, interface, networks, etc. All the requirements for this level are imported into the class and reside initially in the 'document' form. Each individual requirement then may be rigorously analyzed by first breaking it down into its simplest components so that each component is examined on its own merit. Language which duplicates components broken down from other requirements may now be identified and the duplication eliminated. With this identification, redundancy is minimized and the specification is in its simplest form; each Level 2 requirement is now ready to be 'expanded' into the requirements which implement it at the next level. For example, the requirement "The box shall provide protection to its contents and form a barrier to unwanted traffic." can be broken into two requirements: (1) "The box shall provide protection to its contents." and (2) "The box shall form a barrier to unwanted traffic." Once individual requirements from the source document have been decomposed in this way, duplicate requirements are identified and with this identification are combined into one requirement; redundancy is minimized and the specification is in its simplest form. The pictorial representation of this concept is shown in Figure 2. [Marconi] One benefit among many other benefits derived from this process is that project teams are able to make more accurate estimates of the scope of work from this simplified list of requirements.

Once Level 2 requirements are prepared in this manner, Level 3 class describes each Level 2 requirement in more detailed terms so that functionalities are 'packaged' as units which may be more cleanly manipulated in implementing the Level 3 requirements. Two additional classes would contain the test information, one for acceptance tests and one for integration tests. This minimum number of classes requires minimum links to other requirements and tests in other classes in order to process queries quickly and to perform updates easily and time conservatively.

[2] A relational database is basically multiple flat tables with each row a record and each column a field; A relation is one table.

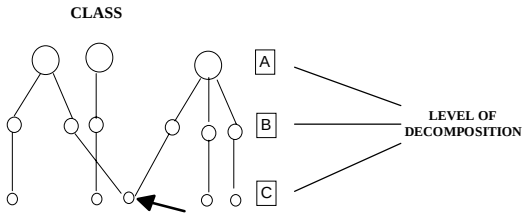

Figure 2. Requirement Expansion & Focus

Box A in Figure 2 represents the requirements as they are taken from the document; Box B represents the expansion of each requirement into its simplest form; Box C represents the final and simplest form of the entire level after requirement duplication has been eliminated. The bold arrow represents the point at which a duplicate requirement is focused into a single requirement. The history of these particular actions on this set of requirements remains within the system and is retrievable at any time. All those activities that altered the requirements from their original form trace the requirement evolution from its initial state to its final state within that class. The map by which a requirement acquired its current form is available for inspection at any time; additionally the history and attributes are retained with the requirement as it evolves.

Once the requirements at a specific level are packaged into their simplest form, they are now ready to be expanded into their next development cycle phase.

IV. Project Implementation.

Project X's focus in establishing a requirement management process was influenced by project organization. The project established a system management office for managing the high level requirements, a design group for managing the design requirements, integration groups for system testing, and acceptance test groups for the acceptance testing of the system (see Figure 3). With test identified within each test group, emphasis now shifts to testing-by-build. That is, instead of all IT (integration testing) and AT (acceptance testing) tests residing in their own class, tests were further subdivided by build - A, B, C - so that test classes now are labeled: ITA, ATA, ITB, ATB, ITC, ATC, etc. This again made sense at the time since one organization was responsible for one build in one type of test.

The established requirement management process was to have each group responsible for the data in its domain without consideration for how one data set related to another data set. As a result, the requirement management tool was set up to have a class for each of the organizational elements without

regard to a.) how the requirements were being managed in each class and b.) whether the requirements were relating cleanly to requirements or tests of other classes. Figure 4 shows the database concept that was developed for the project.

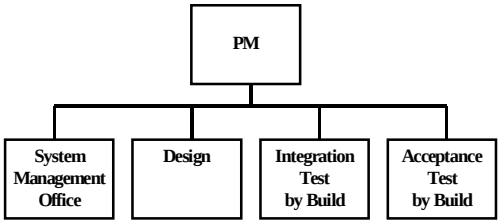

Figure 3. Project Organization

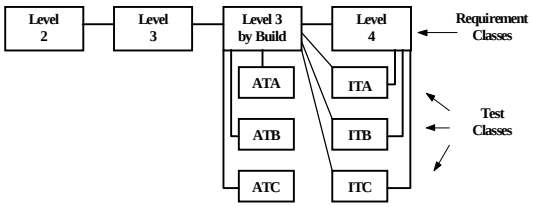

Figure 4. Project X's RMT Database Schema

As can be seen there is a class for each of the organizational elements: Level 3 requirement class for the system management office, another Level 3 requirement class which simply assigns builds to each complex requirement, Level 4 requirements class for design engineering, and test classes by build for both integration and acceptance testing. (Note that for the Level 3 requirement class, an entire class is duplicated to another class simply for the purpose of assigning builds.) This produced an unnecessary or inaccurate compartmentalization of each set of requirements or test cases. [Hansen] The 'expansion' traces between requirements, and between requirements and test cases were then established between the many classes. This seemed a reasonable approach which provided an easily understood database schema, and a sense that each class was under the control of the appropriate organizational element.

The use of the RMT in the manner selected by the project, while appearing reasonable on the surface was actually fraught with the fatal flaw of inexperience that ultimately worked against the project. Specifically, multiple classes mirrored organizational structure instead of a single class existing for each development phase. This was simple solution but a deadly mistake. This is much like using a screwdriver to open the car door. The surface problem is solved, opening the door; but more serious problems have been introduced by that act. (e.g. broken lock).

With this multiple test class and undecomposed requirement class approach, there was a natural tendency for the organizations to "improve" the data schema definitions assigned to them. Though there was a centralized configuration management office responsible for control of the database schema, each individual organization dictated to the CM Office so that there were many customers with many classes requesting many changes in isolation from each other; control was not effectively administered. This naturally led to losses in data integrity and prevented access to or use of important information about the requirements. Some information became specific to a particular organization which should have been available to all project organizations; other data became degraded and useless when it was no longer maintained.

Because multiple classes were implemented at the test-by-build level, fields were duplicated to each of the test classes, common information then became self-contained within each class. However, confusion developed between the test organizations as to which one was responsible for populating common data. The project started with a set of naming conventions, but these were corrupted as each organization customized its classes *ad libitum*. Also, each organization interpreted the meaning of the common fields in different ways. In all of these cases it was not obvious to the configuration management office that there was a problem. The number of classes with similar fields helped to obscure a problem in entering data and worked against rigid compliance with a published data dictionary. [Hansen] This all lead to inconsistent data entries and prevented effective data mining. [Chen]

Another very significant problem that developed with the selected schema was the necessity of using a significant number of linkages to individual requirements between individual classes. This was necessary since each level of requirement was populated to a separate class, but requirements were never decomposed into their lowest structures so that specific attribute values could be independently assigned. No class manipulated its requirements into its simplest form. Problems of redundancy abounded; many (as high as 200) lower level requirements were then drawn from a single, complex requirement. In effect, this meant that no actual requirement expansion had really taken place since it was difficult to determine, for example, which particular part of a Level 3 requirement was implemented by any given set of Level 4 requirements. However, the project did claim a (pseudo-)decomposition simply because Level 3's were in their own class, Level 4's in theirs and a varying number of links were made between the two

classes . In addition, test class structure differed from the requirement class structure to now reflect testing by build. Whereas all the requirements were dumped wholesale into a class and 'requirement expansion' was done across classes using many links, the test classes divided up the test cases first by test type, integration or acceptance, and then by build organization resulting in each test class having minutely defined organizational ownership. This structure required a complex network of linkages between the classes in order to establish the traceability between the different kinds of data (see Figure 5). Complete traceability became difficult. Responsibility for traceability became blurred and changes within a class caused the breaking and re-establishment of linkages. This also lead to a loss of history associated with changes in the data and important information about the evolution of the system requirements. Lastly, the approach taken to establish a significant number of linkages between individual and complex requirements of the classes lead to degraded performance in the tool. The tool was specifically designed to use a single class for all requirement manipulation at a single level. Inter class links were reserved for infrequently changed relationships. As a result, the project's schema established its main, and highly changeable data with linkages that had high tool processing overhead.

In practice, the extremely large nature of this project was not completely understood from the beginning. Had the true size been understood, appreciation for the capabilities of a requirement management tool would have been apparent also. This project averaged 1500 requirements at Level 3 and 6000 requirements at Level 4; the total number of single links between these two classes averaged 19,000 links. However, even these numbers are misleading, as was comprehension of what manipulating this number of requirements truly required. As testimony to this, we found that within each single Level 3 requirement anywhere from 2 to 30 sub-divisions of the requirement might be included. For example: requirement #450 in the database would be counted as one requirement, yet within requirement #450 there is a list of partitions to that requirement which were designated as a, b, c. d, e, f. In reality then, the Level 3 requirement document was describing a minimum of six requirements were only one was counted in the database. At a minimum the project should have decomposed this requirement into the six parts before performing any expansion into the Level 4 requirements. Additionally, where a few links to this one requirement might trace requirement expansion, now 6 times that many links existed between this one large requirement and its next level of implementation.

Due to the multiple class approach, links tracing requirements to tests also became extensive and conflicting. Since the project decided to organize the database schema along the lines of the organization, it was necessary to provide the traceability of requirements to requirements and test case to requirement by connections between many classes. The Level 3 by Build requirements were pseudo-decomposed into a set of design requirements at Level 4, as discussed above. This produced unnecessary links and complicated relationships further so there were many more links to each Level 2 and Level 3 requirement than desirable, as stated previously. There was also a stipulation within the project to have traceability between the system test cases (IT) and both Level 3 and Level 4 requirements. This resulted in a complex, undocumentable traceability relationship between the system test cases and the two different levels of requirements. The acceptance test case classes also had undesirable number of links with the high level requirements since it was difficult to understand which part of a test tested which part of a requirement. The tool selected was not designed to efficiently support this kind of usage. Most RMTs are designed to use minimal classes and effect decomposition within a class and not between classes. The traceability between classes should be used in areas having little change, since the breaking and re-establishment of links between classes is a complex process.

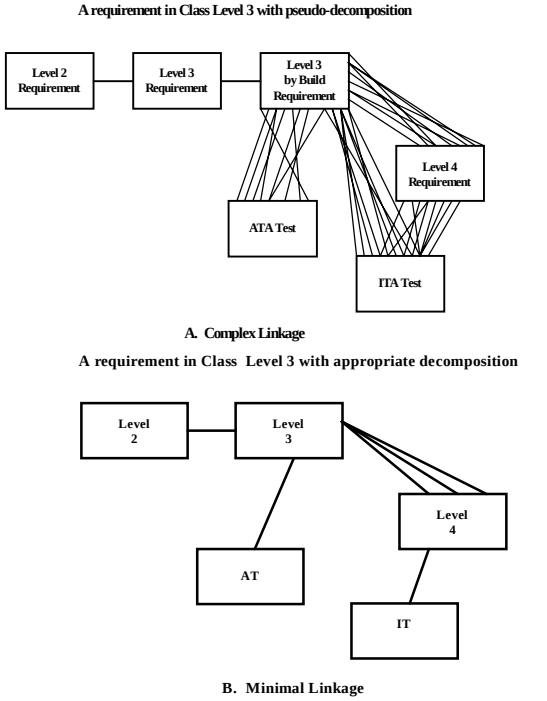

Figure 5. Class Design Affects Link Design

As the population of data in the classes grew, so did the number of links - exponentially. The poor linkage design and the independent, undecomposed class schema caused enormous time delays when updating; major updates took hours and keeping the database current became a formidable task. The developer for the RMT was called in to assist in resolving the problem but could not without totally redesigning the schema and linkages. The alternative solution, tried for a few months, was to take monthly snapshots and to update without keeping track of previous link information, hence loosing all historical data.

The project finally had to adopt a process whereby the requirement manipulation was done in a spreadsheet outside the requirement management tool. When the update was completed in the spreadsheet the data was then loaded back into the requirements management tool. The RMT became ineffective and essentially a distributed storage medium only, not because of tool limitations but because of inappropriate implementation. A tool that should have decreased costs of requirement maintenance and increased reliability was having exactly the opposite effect on the project. The mentality of text based requirement processing using a powerful Requirement Management Tool left the project in a more of a shambles than if the tool had not been employed at all.

V. Lessons Learned.

The lessons learned from Project X's implementation of the RMT, relating to the management and development of the RMT are very simple.

1.) Database designers and requirement engineers must work as a team. Database designers or personnel familiar with database design and use need to be in the design and implementation of the schema for the RMT. Requirement engineers are needed to match the structure to the requirements but basic principles of database design cannot be ignored. The project chose the methodology described to fit the implementation process as it perceived it at the time. Because the project is so large with many internal and external organizations and groupings involved with individual parts of the implementation, it made sense at that time to assign a class to each organization without using the powers of requirement manipulation an RMT grants to its classes. What resulted was not only the drawing of a schema with no central philosophy to govern design, but also the preparation of schema pieces which were not internally consistent. An iterative process is needed such that tool capabilities become fully integrated into the requirements process. Requirement engineers should not assume that all tool capabilities are equal. An effective dialogue between requirements engineers and database engineers is needed in order to ensure a marriage of tool and requirement management process; one controlling group needs to have total overseer capacity of the RMT. Project X did have such a group, but it was overwhelmed and over ruled. Hence, each organizational group responsible for a class became a separate ruling body with RMT management fragmented and contradictory; thus leading to chaotic, untrustworthy data.

2.) Some education and a change in thinking is mandatory when a project brings last generation comprehension to next generation tools. Management cannot assume that old technology has simply been repackaged and renamed. If the tool has powerful capabilities, those capabilities must be understood before proceeding.

Problems that arise need to be addressed in the overall schema; patches just destroy the integrity of the structure and lead to additional problems.

IV. Conclusion.

The project demonstrated a common failing in the implementation of a requirement management tool. The assumption was that a tool could be selected that provided capabilities that matched the needs of the project process for requirement management. The project then adapted the tool to the project processes. There was no further consideration as to the ramifications of using the tool in that manner. The approach should have been used to select a tool based on the overall requirement management process and then refine the details of the process. To enable capture and use of the tool's powers, the refinement of the process examines how the tool can be used in support of the process. Such details are geared toward maximizing the effectiveness of the process once the tool has been integrated with it.

A key to this approach is a teaming arrangement between requirement engineers and database engineers. The requirements engineers would be the experts in the process and be able to explain the objectives of the process. The database engineers would be experts in the tool and be able to explain the ways in which the tool would be able to satisfy the objectives of the process. In this way, specific details could be laid out as to how the tool would be used, populated, and controlled to achieve the maximum benefit from the tool. It should not be assumed that a

high level correlation between process and tool is sufficient, a detailed understanding of the tool and its capabilities and use within the requirements management process is necessary to assure success.

References

[1] 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

[2] Fuggetta, Alfonso, "A Classification of CASE Technology", Computer, Vol 26, No 12, 12/93.

[3] Hammer,T., "Measuring Requirement Testing", 18th International Conference on Software Engineering, 5/97.

[4] Palmer, James, "Traceability", Software Engineering, 1996.

[5] Hansen, Gary W., Hansen, James V., Database Management and Design, Prentice Hall, 1992.

[6] Marconi, Requirements and Traceability Management, Marconi Systems Technology, 1994

[7] NASA, *Software Assurance Guidebook* (September 1989).