Using Requirements Management to Speed Delivery of Higher Quality Applications

by Alan M. Davis and Dean A. Leffingwell

What's Wrong With Software Development?

The software industry has much to be proud of today. The worldwide software and services market generates over $221 billion and has become one of the most significant economic forces of our time. Technologically, the industry has pioneered extraordinary new developments that improve the productivity of companies every day: graphical user interfaces, relational databases, object-oriented programming, fourth generation languages, and the Internet to name a few.

But amidst all this progress there remains a disturbing undercurrent. Software development is still more art than science. While software as art has a certain aesthetic and emotional appeal, the fact is artistic endeavor has never been a reliable process. Like art, software is subject to bursts of creativity and individual genius rather than teamwork and engineering discipline. In addition, we are now using software to solve problems of unthinkable scope compared to only ten years ago.

In a recent article in Scientific American[i], our industry took a broadside for what the authors consider to be an extraordinary lack of progress in the science of software development. While many in our industry perceived this article as unfair, a recent study by The Standish Group[ii] confirms several key statistics we must begrudgingly acknowledge:

Over 30% of all software projects are canceled before completion
Over 70% of the remainder fail to deliver expected features
The average project runs 189% over budget and overshoots its schedule by 222%

These issues strike a major nerve within our industry because we all carry the scars of these efforts. Forty years after the invention of the subroutine, why are we still plagued by these failures? Most practitioners and researchers agree to these major causes:

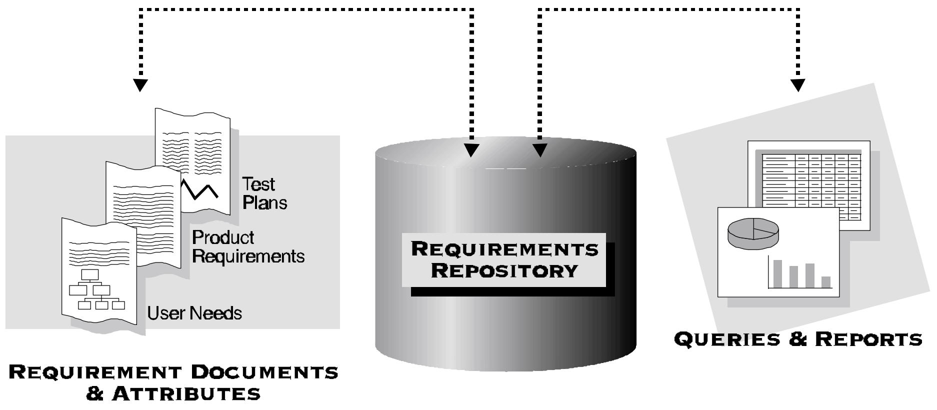
1. *Poor requirements management*. We forge ahead with development lacking user input and without a clear understanding of the problem we are attempting to solve.
2. *Poor change management*. Changes to requirements and other development products are inevitable; yet we rarely track them or understand their impact.
3. *Poor quality control*. We have poor measures for system quality, little knowledge of processes that affect quality, and no feedback to modify the process after witnessing the effects of a particular development strategy.
4. *Little control of schedules and costs*. Accurate planning is the exception while unrealistic expectations are the norm.

Fundamental to all of these problems is requirements management. After all, if we do not thoroughly understand the intended behavior of the system: 1) how can we specify that behavior? 2) how can we design the system? 3) how can we know if the system does what it is supposed to do? 4) how can we measure the level of quality achieved? and 5) how can we possibly predict the effort involved?

PART I. Introduction to Requirements Management

Simply put, a requirement is a capability or feature needed by a user to solve a problem or achieve an objective. Requirements management is *a systematic approach to identifying, organizing, communicating and managing the changing requirements of a software application.* A primary result of requirements management is a requirements specification which "defines and documents the complete external behavior of the system to be built."[iii]

A well implemented program of requirements management provides a central information repository. The repository includes requirements, attributes of requirements, requirements-related status information, and other management information pertinent to your company's environment. This repository is not only important to a project's success, it is a valuable corporate asset. Requirements and their associated attributes can be evolved, adapted, and reused, thus lowering the cost of later development projects.



Key Benefits of Better Requirements Management
Even in light of the above information, some will still argue, why waste time with this unnecessary step; why not proceed directly to implementation? Experience has shown that the benefits of effective requirements management include**:**

Better control of complex projects. Lack of knowledge of the intended behavior of the system, as well as requirements creep, are common factors in out-of-control projects. Requirements management provides the development team with a clear understanding of what is to be delivered, when and why. Resources can be allocated based on customer-driven priorities and relative implementation effort. And the impact of changes are better understood and managed.

Improved software quality and customer satisfaction. The fundamental measure of software quality is "does this software do what it is supposed to do?" Higher quality can result only when developers and test personnel have a concise understanding of what must be built and tested.

Reduced project costs and delays. Research demonstrates that errors in requirements are the most pervasive and most expensive errors to fix. Decreasing these errors early in the development cycle lowers the total number of errors and cuts project costs and time-to-market.

Improved team communications. Requirements management facilitates early involvement of customers to ensure the application meets their needs. A central repository builds a common understanding between the user community, management, analysts, developers and test personnel of project needs and commitments.

Easing compliance with standards and regulations. Most major standards bodies and regulatory agencies involved with software compliance and process improvement have a keen understanding of the requirements management problem. For example, the Software Engineering Institute's Capability Maturity Model (CMM) addresses requirements management as one of the *first steps* in improving software quality. DOD, FDA, and ISO 9000 standards and regulations also require companies to demonstrate maturity and control of this process.

It is clear that doing a better job of the above will save considerable time and money, not to mention reducing the career challenges that result from unsuccessful or partially successful software implementations.

The High Cost of Requirement Errors

There is strong evidence that effective requirements management leads to overall project cost savings. The three primary reasons for this are:

1. Requirement errors typically cost well over 10 times more to repair than other errors
2. Requirement errors typically comprise over 40% of all errors in a software project
3. Small reductions in the number of requirement errors pay big dividends in avoided rework costs and schedule delays

Studies performed at GTE, TRW, and IBM measured and assigned costs to errors occurring at various phases of the project life-cycle[2]. Although these studies were run independently, they all reached roughly the same conclusion: If a unit cost of *one* is assigned to the effort required to detect and repair an error during the coding stage, then the cost to detect and repair an error during the requirements stage is between *five* to *ten* times less. Furthermore, the cost to detect and repair an error during the maintenance stage is *twenty* times more. Figure 1 below illustrates a summary of the results.

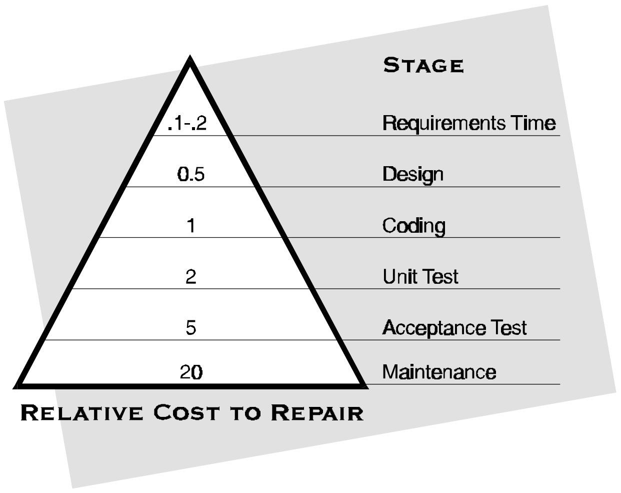


Figure 1- As much as a 200:1 cost savings results from finding errors in the requirements stage versus finding errors in the maintenance stage of the software life-cycle.

The reasons for this large difference is that many of these errors are not detected until well after they have been made. This delay in error discovery means that the cost to repair includes both the cost to correct the offending error and the cost to correct subsequent investments in the error which were made in later phases. These investments include the cost for redesign and replacement of code, cost for documentation rewrite, and the cost to rework or replace software in the field.

Requirement errors are the most common errors

These studies illustrate that errors made in the requirements phase are extremely expensive to repair. If such errors occurred infrequently, then the contribution to overall cost would not be significant. However, requirements errors are indeed the largest class of errors typically found in a complex software project. In a study of a US Air Force project by Sheldon[iv], errors were classified by source. It was found that requirements errors comprised 41% of the errors discovered, while logic design errors made up only 28% of the total error count. Other case studies back this result. For example, Tavolato and Vincena, quoting Tom DeMarco, report that 56% of all bugs can be traced to errors made during the requirements stage[4].

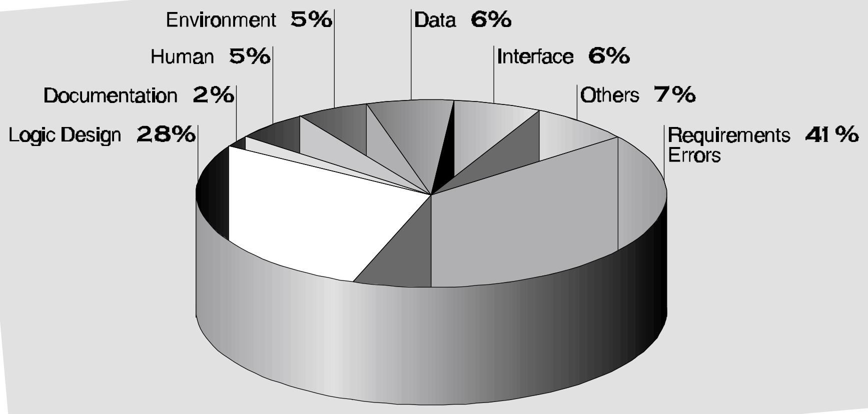


FIGURE 2- SOURCE OF ERRORS ON A U.S. AIR FORCE PROJECT

Cost savings through more effective requirements management

In a study performed at Raytheon, Dion reported that approximately 40% of the total project budget was spent in rework costs[5] . Other studies indicate that for the majority of companies today, rework contributes between 30-40% of total project costs. Because of their large number, and the multiplying effect, *finding and fixing requirement errors consumes between 70% - 85% of total project rework costs.*

Organizations like Raytheon have proven that improving their development processes, as measured by moving from CMM Level 1 to Level 3 (discussed later), can lower their cost of rework by up to 50-60%. Although "your actual mileage may vary," it is clear that even a small reduction in the number of requirements errors can result in significant cost savings.

But these hard costs exclude the intangible costs associated with a requirement error. Intangible costs include the lack of features that could have been delivered, loss of confidence in the project team, and lost and unrecoverable market share with its associated loss of sales revenue and profit. Taken together, these costs clearly demonstrate that a company cannot afford to ignore the benefits of better requirements management.

Requirements Management - The View from SEI's Capability Maturity Model

In November of 1986, the Software Engineering Institute (SEI), funded by the U.S. Department of Defense began developing a process maturity framework to help developers improve their software process[6] . In September of 1987, the SEI released a brief description of the process maturity framework which was later amplified in Watts Humphrey's book, *Managing the Software Process.*[7] By 1991, this framework evolved into what has become known as version 1.0 of the Capability Maturity Model, or CMM. In 1994, version 1.1 of the CMM was released.[8]

The CMM defines five levels of software maturity for an organization and provides a framework for moving from one level to the next. The CMM guides developers through activities designed to help an organization improve their software process with the goal of achieving repeatability, controllability and measurability. The CMM has gained considerable credibility in software intensive industries. Adherence to the CMM and corresponding improvements in software quality have significantly lowered the cost of application development within large commercial companies.[9] These saving have been primarily achieved by lowering the cost of rework[10].

As defined by the CMM, process improvement addresses "key process areas." These are activities that influence various aspects of the development process and resultant software quality. Table 4 illustrates the five levels of the CMM and their key process areas. Not surprisingly, Table 1 shows that one key process area that must be addressed to move from Level 1 to Level 2 is requirements management. The CMM's position is that requirements management is one of the *first steps* to achieving process maturity.

TABLE 1
Levels of the CMM With Key Process Areas

Level	**Key Process Areas**
1. Initial Ad hoc, even chaotic; success depends solely on individual heroics and efforts.	not applicable
2. Repeatable Basic project management to track functionality of application, and cost and schedule of project	**Requirements Management** Software Project Planning Software Project Tracking and Oversight Software Subcontract Management Software Quality Assurance Software Configuration Management
3. Defined The process for management and engineering is documented, standardized, and integrated, All projects use an approved, tailored version of the process.	Organization Process Focus Organization Process Definition Training Program Integrated Software Management Software Product Engineering Intergroup Coordination Peer Reviews
4. Managed Detailed measures of the software process and software quality metrics are collected. Both process and software products are understood and controlled.	Quantitative Process Management Software Quality Management
5. Optimizing Continuous process improvement is enabled by use of metrics and from piloting innovative ideas and technologies	Defect Prevention Technology Change Management Process Change Management

Requirements Management as Recommended by the CMM

The SEI's publication, "Key Practices of the Capability Maturity Model, Version 1.1"[11] describes the goals and activities of each of the key process areas within the CMM. According to this document, "The Purpose of requirements management is to establish a common understanding between the customer and the software project of the customer's requirements that will be addressed by the software project."

Under the CMM, software requirements must be understood, analyzed and documented. Further, the set of requirements serves as a primary input to the software development plan. The organization must make certain that all software plans and related activities are kept consistent with the documented requirements. In this way, *the requirements become a central project focus, and the management of both the requirements and the project itself are inextricably linked.*

Changes to requirements must be rigorously managed under the CMM. As requirements change, specification documents must be updated. The organization is also responsible for assuring that software development plans and activities are changed to remain consistent with the change in requirements.

The CMM provides suggested areas for measurement and analysis for each key process area. These measurements provide feedback for ongoing improvement of the key process area itself. Examples of measurements suggested for requirements management include:

REQUIREMENTS MANAGEMENT IS THE FIRST STEP TOWARD PROCESS MATURITY

1. Status of each of the allocated requirements
2. Change activity for each of the allocated requirements
3. Cumulative number of changes to requirements. Includes tracking of number of changes which are proposed, open, approved, and incorporated into the system baseline.

Clearly, management of these attributes is best handled by a data base that understands the requirements themselves, the relationships among them, and the attributes associated with each.

Requirements Management as Recommended by ISO 9000 and FDA GMPs

The ISO 9000 quality guidelines have achieved widespread support of North American, European and Asian manufacturers. Also, the US Food and Drug Administration (FDA) has proposed sweeping changes to the regulations governing the manufacture and sale of medical equipment. The FDA's proposed new GMPs (Good Manufacturing Practices) adopt significant portions of ISO 9000. In particular, Subpart C, Design Controls, is based heavily on ISO guidelines.

ISO 9000-3[12] "Guidelines for the Application of ISO 9001 to the Development, Supply, and Maintenance of Software", explicitly targets manufacturers of software and embedded software systems. Given the standard's perspective of supplier-oriented quality controls, it is not surprising to find a significant focus on requirements management. In particular, Subpart 5.3.1 states:

> *"In order to proceed with software development, the supplier should have a complete, unambiguous set of functional requirements. In addition, these requirements should include all aspects necessary to satisfy the purchaser's need. These may include, but are not limited to, the following: performance, safety, reliability, security and privacy. These requirements should be stated precisely enough so as to allow validation during product acceptance."*

The standard goes on to cover methods for agreeing on requirements, managing change and other aspects of requirements management. However, the above language clearly highlights two key issues: 1) There must be a complete set of both functional and non-behavioral requirements before development begins, and 2) The requirements must be detailed enough to support product validation.

In summary, requirements management is a key area which must be addressed to achieve and maintain compliance with the ISO 9000 standard, and thereby receive the quality benefits that the standard intends to provide.

Requirements are hard to understand and harder to specify. The wrong solution to this problem is to do a slipshod job of requirements specification, and rush ahead to design and code in the vain hope that:

1. Any system is better than no system.
2. The requirements will work themselves out sooner or later, or
3. The developers will figure out what to build as they are building it.

The right solution is to do whatever it takes to learn as many of the requirements as possible up front. Conduct interviews and focus groups, send out questionnaires, hold brainstorming and role play sessions. Review the bug reports and enhancement requests of the existing system. Understand what similar or competitive applications are doing to meet these needs. Finally, use prototyping.

There is nothing more useful than a prototype for reaching agreement on the user interface and the availability of specific functions. Not only do prototypes nail down requirements, they win the hearts and minds of the customer.

There are two types of prototypes: throwaway and evolutionary. *Throwaway* prototypes are built in a quick and dirty manner, are given to the customer for feedback, and are thrown away once the desired information is learned. A throwaway prototype is often based on a very short requirements document, perhaps a one-page list of bullets. But once the prototype has done its job, the requirements that were learned must be captured in a more robust requirements specification prior to full-scale development.

Evolutionary prototypes are built in a quality manner, are given to the customer for feedback, and modified to more closely approximate the needs of the user. This process is repeated until the features converge to the desired product. Because evolutionary prototypes evolve into a final product, you should shun development short-cuts. Instead, initiate development with a high-quality, fully traceable requirements specification.

Use throwaway prototypes when critical features or product architecture are poorly understood. Use evolutionary prototypes when the critical functions are well understood, but many other features are not. After every iteration, document the requirements you understand and plan to build a system to meet those requirements.

It's worth repeating, avoid the pitfall of not recording what you learn in a requirements specification. We have all heard the software developer say, "I have finished the design. All that's left is the documentation." This makes no sense, especially in today's team environments where success demands that requirements be clearly communicated. Can you imagine an architect saying, "I've completed the design of your new home. All that's left is to draw a picture," or a novelist saying, "I have completed the novel. All that's left is to write it"?[v]

If you expect the requirements to change significantly, that's okay; plan to build incrementally, but that is no excuse for doing a poor job of requirements specification on any one increment.

Part II. Making Requirements Management Work for You

Identifying Requirements - How Will I Know One When I See One?

Typically, requirements start out abstractly, e.g., “I need a system that controls elevators.” As exploration continues, they become more specific, they split, they recombine in new ways, and example requirements appear (especially when multiple cases exist). Eventually, we arrive at a set of very detailed requirements, e.g., “When the *up* button is pressed, the light behind that button illuminates within 1 second.”

Not only do requirements become more detailed, they become less ambiguous. Requirements begin their lives when first elicited from customers or users. Elicitation may occur using any of a variety of techniques such as interviews, brainstorming, prototyping, questionnaires, quality function deployment techniques, etc. Once captured, it is extremely important to maintain traces from each requirement to its more abstract predecessor requirements and to its more detailed successor requirements. Traceability aids in change management and is a fundamental component of quality assurance and sound requirements management.

When the final, most detailed requirements are arrived at, their containing document is called a *requirements specification*. This specification must be communicated and agreed upon by all relevant parties. It serves as the basis for design (it tells designers what the system is supposed to do) and for test (it tells testers what the system is supposed to do). Good requirements specifications exhibit the following characteristics[2].

1. *Lack of ambiguity*. It is unlikely your product will satisfy users’ needs if a requirement has multiple interpretations.
2. *Completeness*. Although it may be impossible to know all *future* requirements for a system, you should at least specify all known requirements.
3. *Consistency*. It is impossible to build a system that satisfies all requirements if two requirements are in conflict.
4. *Traces to Origins*. The source of each requirement should be identified. It may have evolved from the refinement of a more abstract requirement, or it may have come from a specific meeting with a target user.
5. *Avoids design*. As long as requirements address external behaviors, as viewed by users or by other interfacing systems, then they are still requirements, regardless of their level of detail. When a requirement attempts to specify the existence of particular subcomponents or their algorithms, it is no longer a requirement and has become design information.
6. *Requirements are enumerated.* Most requirements specifications enhance their readability by including auxiliary types of information that are not requirements per se. This information includes introductory paragraphs or sentences, summary statements, tables, glossaries, and so on. Actual requirements contained in the document should be somehow easily discernible, whether by unique font, identifying label, or other highlighting.

A complete list of principles to adhere to when performing requirements specification appears in Chapter 3 of Davis’ book *201 Principles of Software Development.*[14]

Writing Your SRS - Getting Off to A Good Start

Many important documents exist within your development project: descriptions of user needs, design documents, test plans, etc. But one particular document, the software requirements specification or SRS, is a primary concern of the software developer. The purpose of this document is to define the complete external characteristics of the system to be built. It defines all the behavioral requirements (e.g. this system shall do A when the environment does B) and nonbehavioral requirements (e.g. the system shall have an availability of 99.9%). While standards are by no means a panacea, an organization that adopts a standard for the SRS achieves several benefits:

- The standard serves as a checklist of things to be addressed so nothing is left out,
- It helps readers quickly locate and review requirements, and
- It shortens the learning curve for new requirements writers and other members of the project team.

Numerous software specification standards can be used as a starting point in drafting an SRS. One that provides a good deal of guidance and flexibility is the IEEE/ANSI 830-1993 "IEEE Recommended Practice for Software Requirements Specifications". There are many other standards that can be adopted to suit your needs. A good resource is the compilation by Thayer and Dorfman[13] . They have reprinted 26 different requirements specifications under one cover including national, international, IEEE, ANSI, NASA and U.S. Military standards.

What is important is that your document outlines encourage accuracy, consistency and a short learning curve. The ANSI IEEE 830 serves as a good starting point for an SRS. Then, based on usage, you may find it beneficial to modify the standard and turn it into a corporate standard that better matches your company's specific processes and culture.

Selecting Requirements From Your Documents

Requirements documents contain a variety of information that are not actual requirements of the system. For example, introductions, general system descriptions, glossary of terms and other explanatory information. While important to an understanding of the requirements, they do not constitute requirements to be fulfilled by the system.

To ease communication of requirements, and allow requirements management, writers should label those portions of text, graphics, or embedded objects that must be implemented and subsequently tested. Ideally, the actual requirements will be left "in situ" (Latin for "in its original place") rather than stored in multiple places. That is, they can be edited and maintained in the project documents even after they have been selected as individual requirements. This makes it easier to keep project documentation up-to-date as requirements change.

Organizing Your Requirements

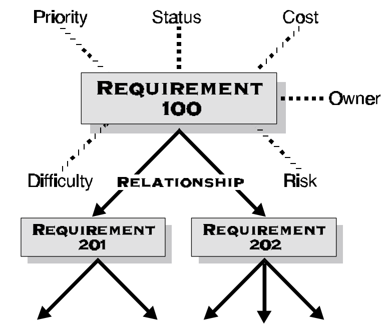
Whether following a recognized standard or your own, there will be a section devoted to specific requirement descriptions. Let's say we have isolated 500 requirements. Rather than documenting them as a long list of bullets, we should find some way of grouping them to aid in their understanding. We recommend organization by[2]:

- **mode of operation**
- **class of user**
- **object**
- **feature**
- **stimulus**
- **mixing any of the above**

Applications that have clearly defined states (powered up, error recovery, etc) could group their requirements under their corresponding mode of operation. Systems that have a significant number of diverse users might be best organized by class of user. For example, a specification for an elevator control system could be organized into three major subsections: passenger, fireperson, and maintenance person. This provides a logical way to group specific requirements so that they can be reviewed and understood by each class of user.

Other applications may best be suited to organization by feature - that is, highlighting the features and their intended behaviors as viewed by the user. Yet others, for example an air traffic control system, which is rich in real world objects, may best be organized by grouping the behaviors of objects in the system. This approach may also be well suited to software organizations that have adopted object technology as their development paradigm.

Managing Requirements with Attributes



REQUIREMENT ATTRIBUTES ARE A RICH SOURCE OF MANAGEMENT INFORMATION

All requirements have attributes whether we recognize them or not. These attributes are a rich source of management information that can help you plan, communicate and track your project's activities throughout the life-cycle. Each project has unique needs and should therefore select the attributes that are critical to its success. Here is a sample:

Customer Benefit - All requirements are not created equal. Ranking requirements by their relative importance to the end user opens a dialogue with customers, analysts and members of the development team.

Effort- Clearly, some requirements or changes demand more time and resources than others. Estimating the number person-weeks or lines of code required, for example, is the best way to set expectations of what can or cannot be accomplished in a given time-frame.

Development Priority- Only after considering a requirement's relative customer benefit, and the effort required to implement it, can the team make feature tradeoffs under the twin constraints of a project's schedule and budget. Priority communicates to the entire organization which features will be done first, which features will be implemented if time permits, and which are postponed. Most projects find that categorizing the relative importance of requirements into *high*, *medium* and *low*, or, *essential*, *desirable* and *optional* is sufficient, although finer gradations are certainly possible.

Status- Key decisions and progress should be tracked in one or more status fields. During definition of the project baseline, choices such as *Proposed, Approved, and Incorporated* are appropriate. As you move into development, *In Progress, Implemented, and Validated* could be used to track critical project milestones.

Authors- The individuals (or teams) responsible for the requirement should be recorded in the requirements database. This might be the person responsible for entering the text or the person responsible for identifying the need.

Responsible Party- This is the person responsible for ensuring the requirement is satisfied.

Rationale- Requirements exist for specific reasons. This field records an explanation or a reference to an explanation. For example, the reference might be to a page and line number of a product requirement specification, or to a minute marker on a video tape of an important customer interview.

Date- The date a requirement was created or changed should be recorded to document its evolution.

Version of Requirement- As a requirement evolves, it is helpful to identify the version numbers (and history) of requirements changes.

Relationships to Other Requirements- There are many relationships that can be maintained between requirements. For example, attribute fields can record:

1. the more abstract requirement from which this requirement emanated,
2. the more detailed requirement that emanated from this requirement,
3. a requirement of which this requirement is a subset,
4. requirements that are subsets of this requirement,
5. a requirement that must be satisfied before this requirement is satisfied, etc.

It is especially important to maintain linkages from requirements to all development products that emanate downstream from them. By providing these links, one can easily ascertain the impact of any changes, and quickly determine development status (which should be an attribute of those downstream entities).

The above list is not exhaustive. Other common attributes include *Stability, Risk, Security, Safety Release Implemented, Functional Area,* etc. Whatever method is used to track them, attributes should be easily customized to adapt to the unique needs of each team and each application.

Requirements Traceability - Ensuring Quality and Managing Change

Requirements traceability is explicitly required in most U.S. Department of Defense software contracts, and is typically practiced by manufacturers of all high reliability products and systems. In the healthcare industry, requirements traceability is governed by the proposed changes in the Good Manufacturing Practices Regulation. However, most companies outside of these industries do not routinely practice requirements traceability.

Traceability is a link or definable relationship between two entities. Those who use requirements traceability find that it provides a level of project control and assured quality that is difficult to achieve by any other means. At Abbott Laboratories, where traceability was instituted in 1987, they like to say "you can't manage what you can't trace"[15] This makes intuitive sense, if for no other reason than assuring full requirements test coverage is virtually impossible without some form of requirements traceability.

Traceability helps manage change and costs for all types of projects[15]

Benefits of Requirements Tracing. In its simplest terms, requirements tracing demonstrates that software does what it was supposed to do. The key benefits of this process include:

- Verification that all user needs are implemented and adequately tested.
- Verification that there are no "extra" system behaviors that cannot be traced to a user requirement.
- Understanding the impact of changing requirements.

Implementing Requirements Traceability

Figure 3 shows a sample hierarchy of project documents. In this example, the product requirements document is the "source" of all requirements. In other examples, the source document could be a user needs document, system specification, etc.

A hierarchical relationship between two documents in the figure is an indication that interrelationships may exist among specific elements in those documents. For example, the relationship shown between the product requirements document and the software requirements specification implies that any specific product requirement could be satisfied by one or more software requirements. Similarly, any software requirement may help to satisfy one or more product requirements. Clearly, a product requirement with no related software requirements or hardware requirements will not be satisfied. And vice versa, a software requirement with no related product requirements is extraneous and should be eliminated.

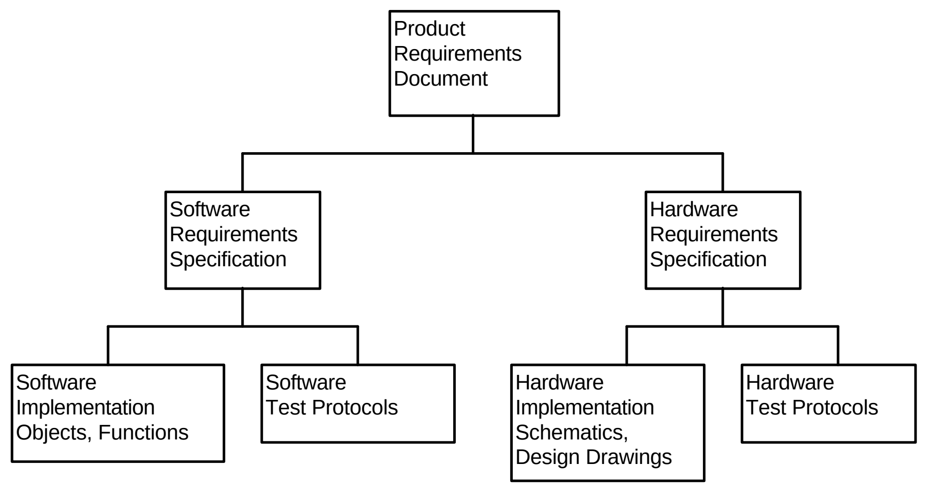

Product Requirements Document
Software Requirements Specification
Hardware Requirements Specification
Software Implementation Objects, Functions
Software Test Protocols
Hardware Implementation Schematics, Design Drawings
Hardware Test Protocols

Figure 3- Example Document Hierarchy

In addition to establishing document relationships to support traceability, you will need to employ some form of system to maintain links between individual items within the hierarchy. This could be done by embedding links and identifiers directly within the document, or by using a separate spreadsheet or database that manages the linkages outside the document. There are advantages and disadvantages to each of these approaches. A new class of requirement management tools automatically maintain traceability links. Ideally, this capability is integrated in the same tool that manages and manipulates the documents and their individual requirements.

Change Management

Traceability provides a methodical and controlled process for managing the changes that inevitably occur as an application is developed. Without tracing, every change requires that documents be reviewed on an ad hoc basis to see what, if any, elements of the project require updating. Because it is difficult to establish if all affected components have been identified, changes tend to decrease system reliability over time.

With traceability, management of a change can proceed in an orderly fashion. The impact of a change can now be understood by following the traceability relationships through the document hierarchy. For example, when a user need changes, a developer can quickly identify which software elements must be altered. A tester can pinpoint which test protocols must be revised. And managers can better determine the potential costs and difficulty of implementing the change.

Requirements Reporting - Easing Management Reviews

A requirements repository gives managers a powerful tool for tracking and reporting project status. Critical milestones are more easily identified. Schedule risks are better quantified. Priorities and ownership are kept visible. Querying the repository can quickly uncover facts that provide answers to important questions, such as:

> **“How many requirements do we have on this project? How many are high priority?”**
>
> **“What percentage of the requirements are incorporated in the baseline? When will they be implemented?”**
>
> **“Which requirements changed since the last customer review? Who is responsible for the changes?”**
>
> **“What’s the estimated cost impact of the proposed changes?”**

High-level reports aid management reviews of product features. Requirements can be prioritized by customer need, difficulty and cost to implement, or by user safety considerations. These specialized reports help managers better allocate scarce resources by focusing attention on key project issues. The net result is that *managers make better decisions* and thereby improve the outcomes of their company's application development efforts.

Conclusion

Software development is one of the most exciting and rewarding careers of our time. Unfortunately, many of us carry the scars from applications that missed expectations. It is common for applications to overshoot their schedule by half, deliver less than originally promised, or be canceled before release. To keep pace with rising complexity and increased user demands, we must begin to mature the ways in which we develop, test and manage our software projects. The first step in this advancement is improved requirements management.

Requirements management provides a "live" repository of application requirements and their associated attributes and linkages. This repository establishes an agreement on exactly what the software is supposed to do. It provides a wealth of information that can be used to manage and control your projects. Your quality will improve, and the software you build will better fit your customer's needs. And with requirements data available to all members, team communication is greatly improved.

Start now. You can cut project costs significantly by catching requirement errors early. Try to write down, in plain English, all the requirements of your current project (*hint:* if this difficult, or wasn't already done, ask why). Compile these requirements in a suitable, short report and share them with your customers and peers. Get their feedback. Are any requirements missing? Incomplete? Wrong? There's a good chance the answer is yes to all three. If it's still early enough to correct these errors, you've saved a bundle. If it's too late, ask what is it about your process that
could change and propose a first step.

What You Can Do

1. Continue educating yourself on the benefits of requirements management. Secure training. Read. We've included a suggested list in the bibliography.

2. Explore and utilize the new tools that are making requirements management easier.

3. Adopt a personal strategy to better communicate the requirements you own.

Suggested Reading

Software Requirements - Objects, Functions, & States, Davis, Alan M., Englewood Cliffs, NJ Prentice Hall, 1993

Exploring Requirements - Quality Before Design, Gause, Donald C., and G. Weinberg, New York, NY Dorset House Publishing, 1989

For information on how to order these books, or for addresses of the available internet forums discussing requirements management, please contact Rational Software Corporation, 4900 Pearl East Circle, Suite 106, Boulder, CO 80303, phone (303)-444-3464, fax (303)-444-3413, e-mail: information@requirement.com

Bibliography

[1]Gibbs, W., *Software's Chronic Crisis,* Scientific American, September 1994.

[ii] *CHAOS,* The Standish Group International, Inc., Dennis, MA, 1994

[iii] Davis, A., *Software Requirements, Objects, Functions, and States*, Englewood Cliffs, NJ., Prentice Hall, 1993.

[iv] Sheldon, F. et al, *Reliability Measurement from Theory to Practice*, *IEEE Software, 9,4* (July 1992)

[4] Tavolato, P., and K. Vincena. "A Prototyping Methodology and Its Tool." In *Approaches to Prototyping,* R Budde et al., eds., Berlin: Springer-Verlag, 1984. pp. 434-46

[5] Dion, R. "Process Improvement and the Corporate Balance Sheet," *IEEE Software, 10,4* (July 1993),pp. 28-35

[6] Paulk, M., et al, "Capability Maturity Model , Version 1.1", *IEEE Software, 10, 4*(July 1993), pp. 18-27.

[7] Humphrey, W.S., *Managing the Software Process*, Reading Mass,Addison Wesley, 1989.

[8] Paulk, M., et al, "Capability Maturity Model for Software, Version 1.1," Software Engineering Institute, Pittsburgh, PA, SEI-93-TR-024

[9] Humphrey, W., et al., "Software Process Improvement at Hughes Aircraft," *IEEE Software, 8*, 4 (July 1991), pp. 11-23.

[10] Dion, R. "Process Improvement and the Corporate Balance Sheet," *IEEE Software, 10,4* (July 1993),pp. 28-35

[11] Paulk, M., et al, "Capability Maturity Model for Software, Version 1.1," Software Engineering Institute, Pittsburgh, PA, SEI-93-TR-024

[12] ISO 9000-3, "Guidelines for the Application of ISO 9001 to the Development, Supply, and Maintenance of Software", Int'l Organization for Standardization, Geneva, 1991

[v] Davis, A., *201 Principles of Software Development*, New York, NY. McGraw-Hill, Inc., 1995

[14] Davis, A., *201 Principles of Software Development*, New York, NY: McGraw Hill, 1995.

[13] Dorfman, M., and R. Thayer, *Standards, Guidelines and Examples of System and Software Requirements Engineering*, Los Alamitos, CA: IEEE Computer Society, 1991.

[15] Watkins, R., and M. Neal, Why and How of Requirements Tracing ," *IEEE Software, 11,4* (July 1994), pp. 104-106.