PUTTING "MANAGEMENT" INTO YOUR REQUIREMENTS MANAGEMENT

PETER BAXTER
DISTRIBUTIVE MANAGEMENT

INTRODUCTION

While requirements engineering and requirements tools have become widely adopted, the number of software project failures attributed to poor requirements management remains high. Metrics [1] that focus on requirements engineering are widely available, yet often only the most advanced and mature organizations actually use them. Requirements metrics are key indicators of project scope, growth, stability and progress. Managers who use requirements engineering measures can spot trouble before a software project becomes a death march. By better managing requirements engineering, managers ensure that they deliver a quality product that completely satisfies customer expectations.
The most common requirements engineering challenges faced by typical organizations attempting to manage the software development process are:

- Most requirements engineering tools don't provide a way to establish the number of requirements to be developed, the baseline;
- Managers generally have no method for assessing requirements engineering progress against the baseline;
- Requirements engineering processes don't quantify the extent and impact of requirement changes on a product;
- Managers have no means to monitor and control the flow of requirements from development to design to coding and test;

Each of the challenges above is directly addressed when measurement is included as part of requirements management activities. Requirements measures provide project managers with the data needed to actively manage their projects. There are three key benefits that requirement measures provide. First, project managers initiate requirements measurement by establishing a plan or baseline, against which actual progress can be assessed. Second, with appropriate measurement data collection from a requirement repository, the project manager can review actual versus plan as well as other progress and quality indicators. Third, armed with up-to-date progress data, the project manager can take action before problems have a significant impact on the schedule or product, or both.

Requirements measures help managers prevent software project failures. This technique of using metrics is not new – what is a fresh step in the right direction is managers' expectations that they are more closely involved in monitoring and controlling the requirements process.

Mr. Baxter is actively involved in improving management and oversight through the use of performance measurement processes, techniques and best practices. Mr. Baxter is actively involved the leading organizations, including International Council on Systems Engineering (INCOSE), Practical Software and Systems Measurement (PSM), International Organization of Standards (ISO) Subcommittee, IEEE and others. At Distributive Management, Mr. Baxter directs solution delivery and has the privilege of working with the most mature and advanced companies in the world to improve and tune their management processes.

Software Projects That Failed Due to Poor Requirements Management
In a July 2005 IEEE article entitled "Why Software Fails – We Waste Billions Of Dollars Each Year on Entirely Preventable Mistakes", Robert Charette lists "Badly Defined System Requirements" as one of the primary causes of software project failure. He estimates that software failures have cost the US economy as much as $75 billion dollars over the past five years.
In 1995, a Government Accounting Office report entitled "Radar Availability Requirements Not Being Met" document the requirement failures of a project jointly developed by the U.S. Air Force, the Federal Aviation Administration and the National Weather Service.
In 1994, the Standish Group released The Chaos Study which cited "Incomplete requirements" as the number one impairment factor in failed projects. The number six factor is "Changing Requirements and Specifications".
In the 1993 article entitled "Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems" by Robyn R. Lutz of Jet Propulsion Laboratory, the root cause of 62% of the errors in safety-critical software was identified to be poor requirements.
In 1998, Robert Glass published the book, "Software Runaways: Lessons Learned from Massive Software Project Failures". The first reason cited reason for project failure is "Project Objectives Not Fully Specified."

WHAT HAPPENS WHEN REQUIREMENTS ARE NOT MANAGED

Study after study has identified poor requirements management as a key factor in project failures. There are countless studies spanning the last 35 years that cite poor requirements management as a primary or contributing factor in the failure and even cancellation of software development projects. The table above summarizes some of the more notable ones.

The inability to manage requirements leads to projects which do not deliver the functionality that end-users or customers expect or need. Frankly, after decades of failure and with the tools to actually manage requirements, we should have learned by now.

A TYPICAL REQUIREMENTS PROCESS

A requirements process first describes the purpose of a software product, and then refines that purpose into greater detail. Commonly, requirements are set out in one or more requirements documents, generically called work products. While there are special languages designed to express requirements, the natural, English language is still the most prevalent. Diagrams, such as the use case diagram, are becoming increasingly popular as a means to more efficiently convey requirements.

For small and medium-sized projects, typical requirement engineering activities define the high-level need for the system, refine the need into specific features and functions, and build lower level requirements for major components or functional areas. Requirements are also refined into software design requirements. Finally, test plans (which are requirements documents for testing) ensure that the requirements captured in the initial high-level requirements documents are satisfied.

Figure 1 depicts the typical flow of requirements through the software development lifecycle. There's a glaring omission from Figure 1 that should worry the astute reader – there is no apparent concern for managing the process!

In almost all requirements process diagrams the author has ever seen, the "management" of requirements management is missing and only the technical part is shown.

Figure 2 below shows the management part of requirements management within the context of actually performing the technical activities.

You can see that the manager will set a baseline and then monitor progress against this baseline. The software manager will use that information during the fact to change execution and ensure a successful completion.

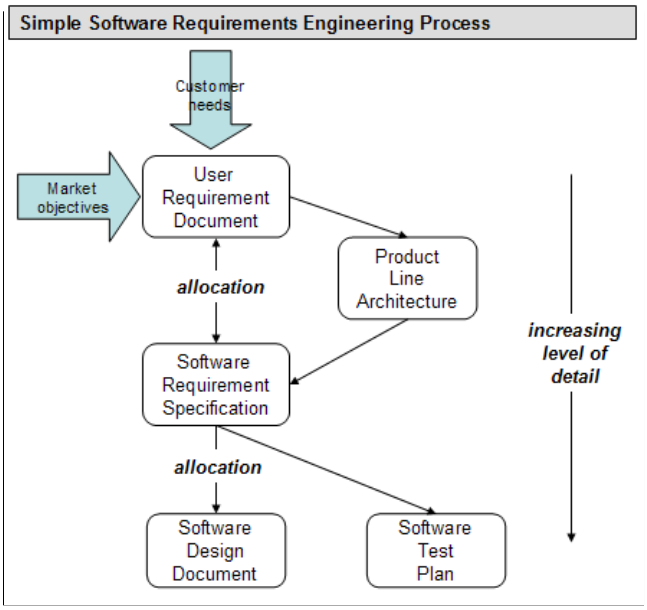


Figure 1 – typical flow of requirements through the software development lifecycle.

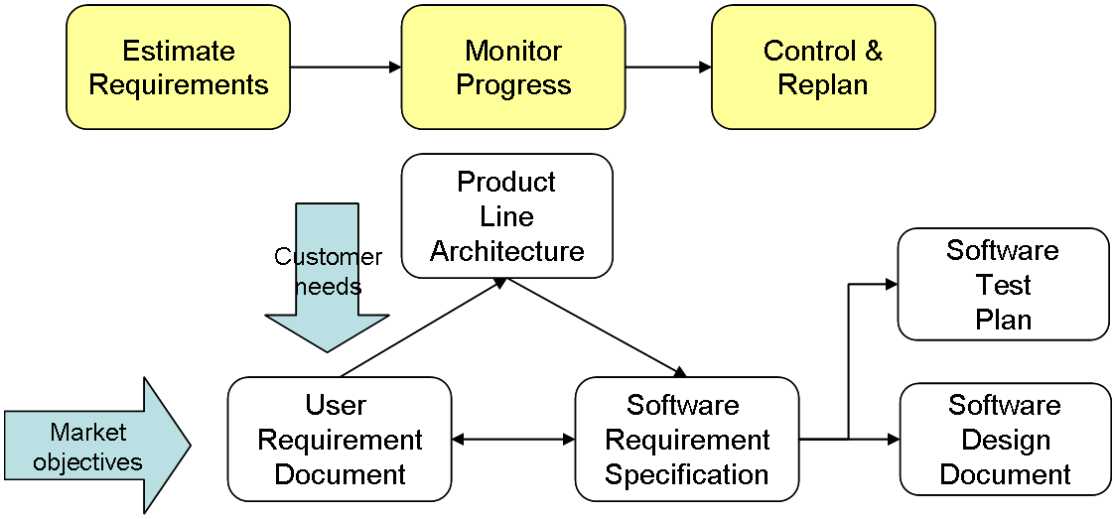


Figure 2 – revised requirements management incorporates real management.

Most modern requirements engineering tools provide an interface for extracting the progress data needed to manage. Once the data is extracted, it can be analyzed and compared against the baseline to create a status indicator, such as a green light or a yellow alarm. Because decisions are generally made using whatever information is readily accessible, requirements status is, ideally, delivered automatically to a stakeholder's desktop so they don't have to go dig it up themselves.

PRACTICAL REASONS TO MEASURE REQUIREMENTS

If you are still not convinced that managing requirements is essential for successful project completion, look through the reasons discussed in this section. When combined with the catastrophic failures presented earlier, these reasons demonstrate that requirements management is critical and essential – no project manager should start a project without requirements measures.

You Must Control the Requirements Creep

Requirements creep is a term for the adding of requirements after the project has been started, without proper consideration of the additional resources, time or risks those new requirements represent. In short, requirements "creep" into the software. Typically, project plans are based on an estimated number of software requirements. Once the functionality is estimated using those requirements, further estimates can be developed for staff, schedule, quality and other aspects of project execution. Because estimating requirements plays such a large part in developing the initial program plan, it is imperative to monitor that requirements are proceeding as expected.

Consider a simple scenario where you are developing 25% more requirements than you planned for – every life-cycle activity is likely to be over schedule and budget by at least 25%. One top-level requirement, when decomposed, expands to multiple requirements at the detail level. For example, one top-level requirement might decompose into five testing requirements. (This is called "requirements expansion" or sometimes "fan out".) So, if high-level requirements change by 25%, the project schedule and staff might be impacted by 5 times that amount or more!

It is advisable to monitor the requirements creep in each requirement work product. At a minimum, though, you should measure the system or top-level software requirements. When possible, you should also track the decomposition of system requirements into more detailed requirements.

Customer Satisfaction Depends On Requirements Delivery

All end-users, even those users of in-house software, will, justifiably, revolt if they are given software that does not perform to their expectations. The software team captures what the customer requires in a requirements document called the User Requirements Document (URD), and then monitors the development effort to ensure that all required functionality is present (and operational) in the software product. If the project manager does not know, absolutely and un-equivocally, that all user needs have been met before the product is shipped, the end-user is likely to be dissatisfied.

The challenge of customer satisfaction is difficult, even in the best of circumstances. Typically, there's a team of software developers, not experts in the domain of the end-user, who must interpret and translate user needs into a real software product. This in itself, is a challenging proposition. Being absolutely sure, through objective requirements measurement and management, that that the end-user requirements (as expressed in the URD) are complete is essential to end-user satisfaction.

While some end users can only complain (to their internal IT department for example), other end users can stop buying (or funding) a product and drive a company out of business if customer satisfaction becomes unacceptable.

Requirements Drive Effort and Schedule

The scope of your project is determined by the number of requirements to be developed. As you add requirements, you increase the amount of work required to complete the software product. Each added requirement means additional work to analyze and describe the requirement. Then, the added requirement must be refined into software design and code. Finally, the added requirement must be tested and documented. Throughout the process, quality assurance must ensure that the resulting product maintains the desired software quality goals.

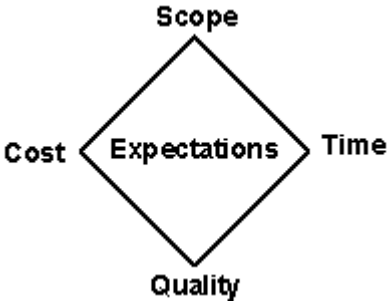


Figure 3 – changing requirements affect all aspects of the project.

The more requirements you add, the more your project will cost (to pay salaries) and the more time it will take (to implement the change). Additionally, you must address the quality of affected work products, such as URD, software design, software code, unit test and system testing. The relationship between scope, cost, time and quality is shown in figure 3 (from "Your Mission, Should You Choose to Accept It: Project Management Excellence" by David L. Hamil (PMP) of MESA Solutions, Inc.)

The Leading Software Framework (the CMMI[SM]) Requires It

Even if your organization hasn't chosen to specifically implement the SEI's Capability Maturity Model Integrated, it (the CMMI) is still an invaluable "playbook" that can help you improve areas of systems and software development. In the CMMI, there are maturity levels, 1 through 5 and requirement management appears early in the CMMI approach. Measurement has its own management process area and also plays a key role in the CMMI. One of the first things the CMMI focuses on is requirements engineering and requirements management.

The CMMI provides guidance for how to setup good and effective requirements management processes, including process planning, process activities, work products, characteristics of the work products, quality attributes, and associated measures. Even if you don't go through a formal assessment, the SEI web site (www.sei.cmu.edu) and the CMMI are valuable resources for improving software development processes.

GUIDANCE TO START MEASURING REQUIREMENTS

This section provides steps you can follow to start implementing requirements management in your organization.

Identify What You Need to Know

In order to control software development, managers need timely status information for decision-making. The information that managers receive should help them understand progress and prevent known or common issues from affecting the project. Unfortunately, instead of receiving useful information, managers generally receive just the information that the IT staff can get easily.

Identifying the correct information needs is essential for being able to monitor and control a software project. Relevant information needs answer questions or address risks that are relevant to the project manager.

Refer to figure 4 for the key places to look when identifying your information needs.

This paper provides the best practices for measuring requirements, but there are other areas that you may want to measure. Instead of measuring what is easy or been done in the past, you should first examine what your most essential management needs are, and then, like requirements, refine them into actionable measures and indicators.

Figure 4 – Primary Sources of Information Needs
September 2002 Software Technology Support Center's *Crosstalk*.

- Information Needs for Current Management Practices
- Measurements of "Requirements"
- Risks That Impacted Previous Programs
- Risks for the Current Program
- Measure What You Are Trying To Improve
- Software Quality
- Assumptions Used in Preparing the Project Plan
- Information Needed to Satisfy Organizational Policy
- Resources Consumed and Products Produced To Understand Process Performance

Establish a Requirements Plan or Baseline

Before committing all your resources and marching off towards a successful project, ask yourself "How much software functionality does our team have to develop?" In order to answer that question, you will need a plan, or baseline, of the number of requirements to be developed for the project. Ideally, you will create a baseline for every work product, but at a minimum you should create a baseline for the URD, and the system test plan.

The initial baseline is simply a monthly estimate of how many requirements you will have completed each month of your software project. This estimate is usually cumulative so that you can easily see the percent to be completed, the number and percent actually completed, and the relative time frame for completing them. During the project timeframe, the measurement process should deliver periodic updates of the progress against the requirements baseline.

Monitor Actual Versus Plan

As your software project proceeds, you must periodically extract requirements metrics from the applicable work products and generate the current "actual" status. The actual should be compared to the plan such that it is easy to see the difference between them.

Figure 5 on the following page shows the plan and actual number of requirements for the User Requirements Document along with a color-coded indicator (above the plot), making assessment straight forward and not subject to the scale/magnitude of the number of requirements.

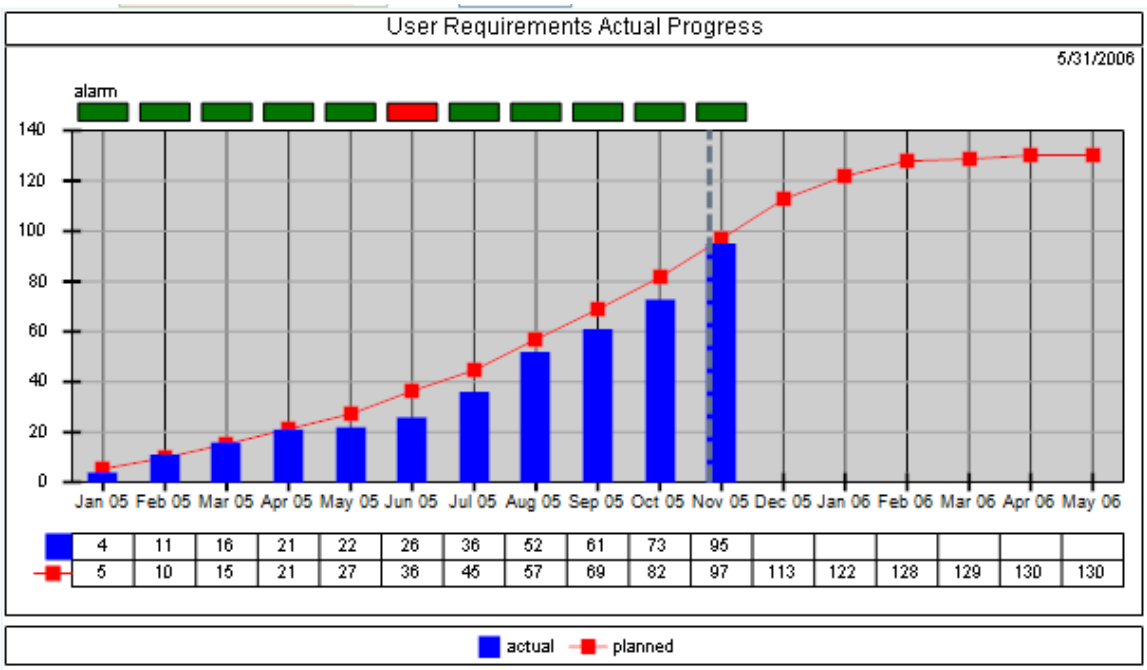


Figure 5 – number of requirements., planned and actual.

In the figure 5 above, notice that requirements growth is expected (as indicated by the plan line) to slow near the end of the project. By comparing the actual to the plan, we can easily see the difference. Also, notice that above the graph there's a color-coded indicator which displays red, yellow or green, depending on how significant the gap is between actual and plan.

Today's requirements management tools provide unsurpassed functionality and usability for the process of defining, analyzing and allocating requirements. They do not, however, provide a method for establishing a baseline plan, and then comparing plan to actual as the project proceeds. This is one reason for including measurement as part of requirements management – to capture the plan/baseline and then compare the actual data as it is collected.

Manage By Exception

Finding time is a constant challenge for the software project manager –time to review and approve work products, reviewing metrics/technical status, attend meetings, prepare team materials…there never seems to be enough. Modern measurement tools provide a method, called manage by exception, that can save valuable time when reviewing project status.

In a typical software company, you may have fifty (50) projects, each with 5 requirements documents, thousands of lines of code, and hundreds of staff. Reviewing each item manually and checking actual performance against the plan would be monotonous, error-prone and time consuming. Compared to the "eyeball management" approach, imagine if you could assign a business rule and color to each comparison, and then simply watch the colors for problems. Figure 6 on the following page depicts a set of requirements and project indicators, grouped into management categories.

Information Need View

Information Needs	Items	Actual	Target	Variance	Status	Trend
Requirements Progress	Concept Document	0	0	--		
	Software Design	0	0	--		
	Software Specification	0	0	--		
	System Specification	0	0	--		
	Test Plan	0	0	--		
Requirements Stability	Concept Document	0	0	--		
	Software Design	0	0	--		
	Software Specification	0	0	--		
	System Specification	0	0	--		
	Test Plan	0	0	--		

Figure 6 – requirements and project indicators grouped by management category.

Using management by exception, you are presented with color-coded status, evaluated from current data, for the high-level measurement areas (e.g. quality and risk). To review your projects, for example, you would instantly spot red or yellow status, then explore just those project areas.

Take Action

The primary goal of a project manager is to monitor and control their project to a successful completion. To "control" a project, the management must take action, i.e. change the current project execution to yield a more desirable result. While progress indicators act as "triggers" for a project manager, the project manager will need to review the current data and then take appropriate action.

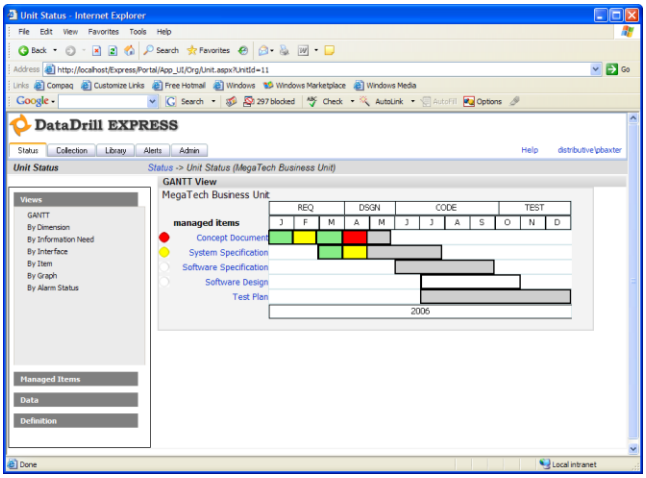


Figure 7 – information products for a software project.

To take action a project manager will need to get as complete a picture as possible of the current "state" of their project. Indicators and measures are inter-related, with some indicators being leading ones and some being trailing. For example, a requirements indicator may turn red because no progress was made in software development (but only after the software development failed to meet its targets!)

A sample set of information products for a software project is shown in figure 7. Notice that along with requirements, the project manager can review coding, cost, defects issues and other measures – in essence a complete picture of their project.

Inform Stakeholders

Everyone hates surprises in software development. Stakeholders, including the ones who fund or are ultimately responsible project success, especially dislike surprises. Scott Ambler (Software Development Magazine, September 2005) says this about stakeholders:

> ***Project stakeholders include anyone affected by the development or deployment of your system. This group incorporates direct or indirect users, managers of users, senior managers, operations staff, support (help desk) staff, developers working on other systems that interface with yours, and anyone else responsible for maintaining the system. To succeed, you must understand and then synthesize all of these people's requirements into a cohesive vision.***

You should take steps to keep stakeholders informed throughout the life of your project. In large projects, the list of stakeholders could include systems engineering managers, software managers, quality assurance groups, configuration management, customer representatives, program office members and others. Each type of stakeholders may require different a different set of reports, including status of requirements engineering activities.

RECOMMENDED REQUIREMENTS MANAGEMENT VISUALS

This section describes what requirements management might actually look like. One thing you should notice is that the basic graphing and display techniques are well-known – what is unique is that they are focused on the challenges of managing requirements. This section shows ways to show requirements status at a high-level, allowing you to manage by exception and spot trouble areas quickly. Then, key requirements monitoring information is shown through a set of sample graphs and indicators.

These visuals will help you ask for changes and improvements in your existing reports. In many cases, you may already have requirements reports or graphs which can be expanded to provide the visual techniques shown in this section. The samples were created with Distributive Management's DataDrill. See Section 7 for resources and tool for requirements engineering and management.

The Big Requirements Picture

As the size and complexity of your software product increases, the number of things to manage and control also increases. To stay ahead of potential problems, the software manager should have a concise and up-to-date big picture view of requirements engineering. This big picture view should provide a summary of the requirements management visuals suggested in subsections, 6.2 through 6.8.

A sample high-level requirement big picture is shown in figure 8 on the following page.

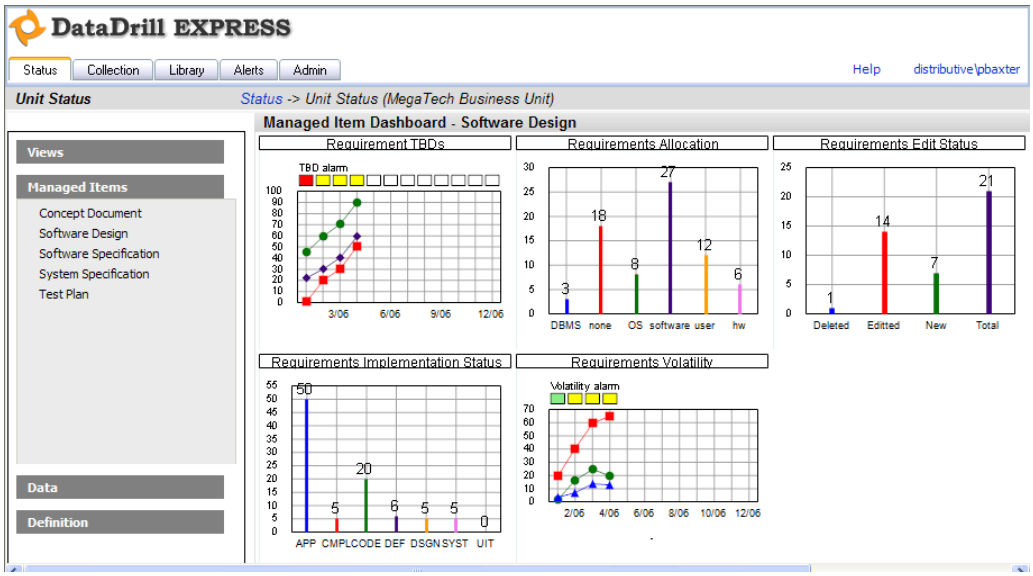
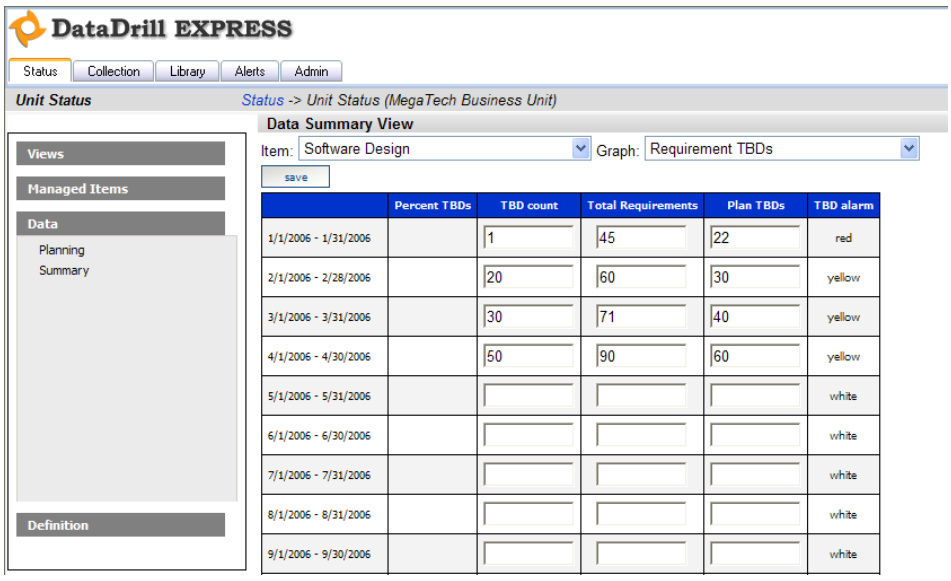


Figure 8 – high-level requirements status

The summary view shows each high-level requirements status as appropriate for the current project timeline. Easy-to-read gauges and color-coded indicators keep the team apprised of overall requirements progress and changes, and keeps an eye on how schedule is being affected.

Ideally, you would be able drill into more detailed information about its status, as shown here in figure 9.

The requirements visuals should be tailored to support the indicators, graphs and status information that your project managers need.



	Percent TBDs	TBD count	Total Requirements	Plan TBDs	TBD alarm
1/1/2006 - 1/31/2006		1	45	22	red
2/1/2006 - 2/28/2006		20	60	30	yellow
3/1/2006 - 3/31/2006		30	71	40	yellow
4/1/2006 - 4/30/2006		50	90	60	yellow
5/1/2006 - 5/31/2006					white
6/1/2006 - 6/30/2006					white
7/1/2006 - 7/31/2006					white
8/1/2006 - 8/31/2006					white
9/1/2006 - 9/30/2006					white

Figure 9 – detailed requirements status

Customer Input

One area where software projects often fall short is in the area of customer requirements – managing them initially and controlling changes to them. As a project progresses, customer requirements typically expand. This happens because of two constants in software development: 1) as we develop something, we learn more about it and are able to better articulate what we want; 2) over the course of time required to develop something, technology changes. Other reasons include a change of need by the end-user or an expansion of the level of automation needed. Whatever the reason, it is not always possible for a software project manager to say "no" to additional requirements. There are three aspects of measurement that can greatly help you manage the effects of customer input.

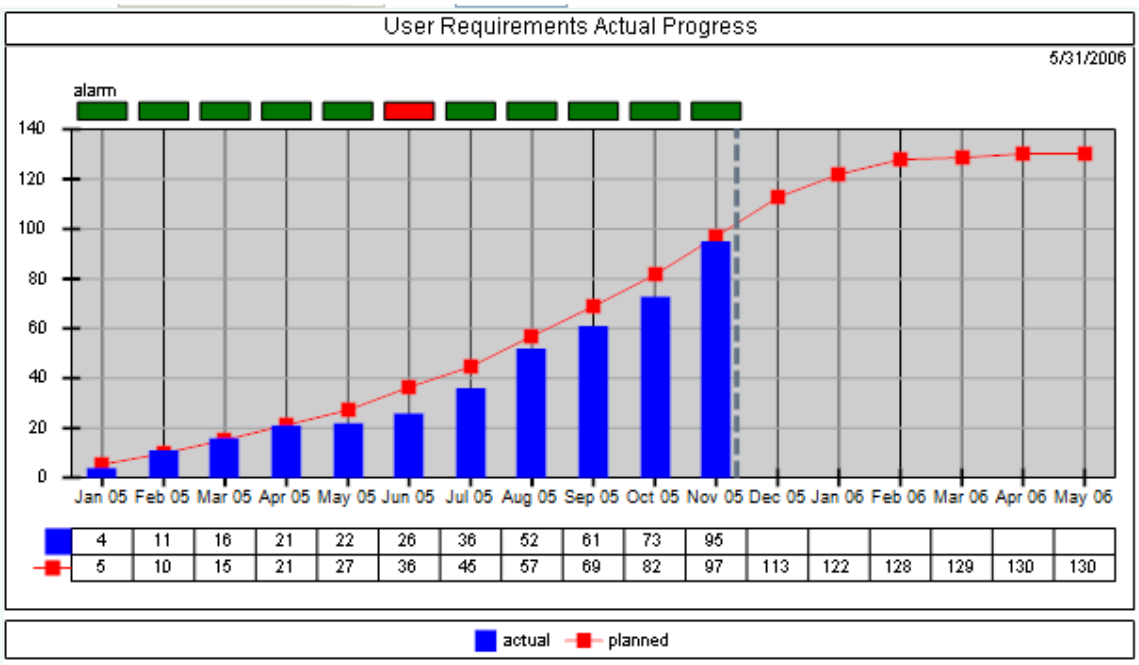


Figure 10 – User Requirements Actual Progress

First, by working with the customer and explaining to them that your schedule and resources are based on agreed-upon requirements, they will be aware that requirements changes are not free. (A common developer maxim is: good, fast or cheap, pick ay two.) Establish a period where you encourage customer feedback and collaboration on the high-level requirements. Once this is complete, you are then in a position to create, or in some cases refine (when you had to produce a schedule before knowing what the requirements are) the project plan, including schedule and resources.

Second, make sure you measure the progress of requirements through the lifecycle, so that you can show your customer the affect on schedule, cost, resources, quality, etc. of changing a requirement. Many times, a customer may not understand or appreciate the amount of work that has been performed and is therefore not concerned about changing a requirement. By having the hard data to discuss, you can turn the focus away from "we really need this" and to "how are we going to do this", where it belongs.

Third, once you agree to a customer change, you need to re-visit your plan and see if you need to assign more resources or change the schedule. The performance data collected to date will prove useful in quantifying the areas of impact.

Requirements Growth

Requirements growth is the number of requirements contained in each requirements document, and is typically counted on a weekly or monthly basis. If requirements was used as an estimation factor (as opposed to function points/functional size measures), then requirements growth would also include the requirements baseline. Requirements growth (the actual number of total requirements) and the requirements baseline (the planned number of requirements) are displayed, along with a calculation of the difference in percent between growth and baseline.

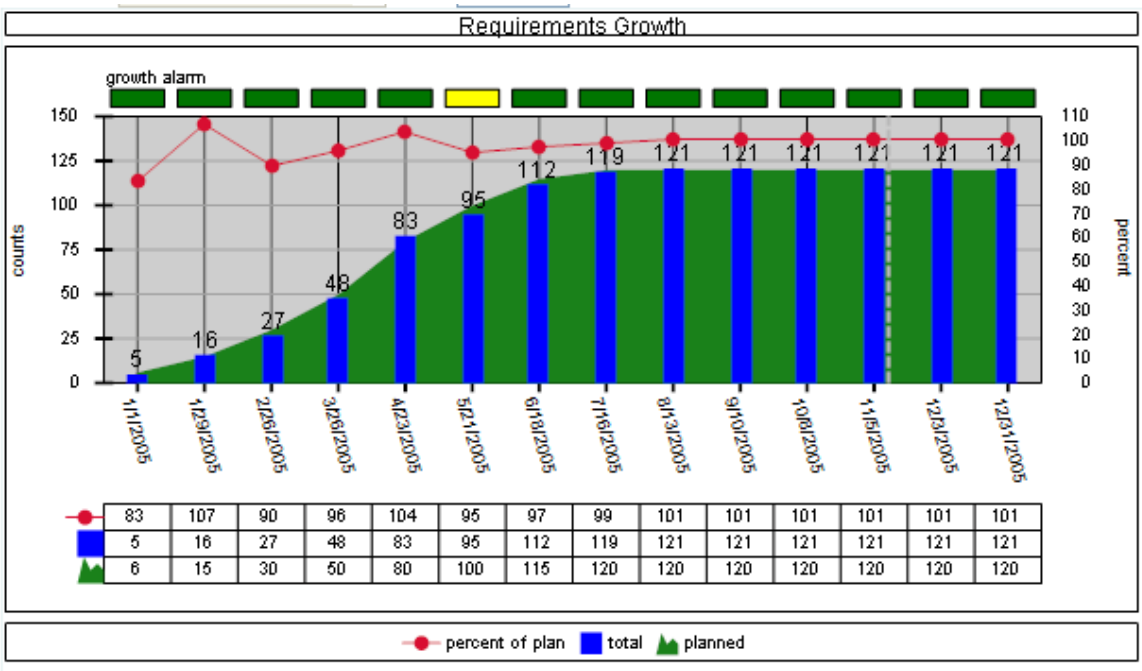


Figure 11 – Requirements Growth

In the figure 11, notice that the percent of plan is actually 101% at the end of our project, meaning we ended up with more requirements than we initially planned. In this case, we planned for 120 and actually had 121, which indicates a well-planned project. By monitoring this difference as the project proceeds, you have two primary options for dealing with a percent of plan over 100 percent. First, you can stop requirements creep by deciding not to develop the new requirements. Second, you may decide to include new requirements provided that you also assess whether to re-plan the project.

You should monitor requirements growth on a periodic basis, perhaps weekly or monthly. First, you want to make sure that requirements are actually growing, that is, the total number stays the same or increases each period. You want to make sure that growth is keeping reasonably close to your baseline. When growth lags, this represents under-performance by your team and is a cause for review.

You should also monitor requirements growth closely to make sure that new requirements are not included without review (i.e. "creep"). An easy condition to check for is that the requirements growth exceeds the baseline. You should monitor underperformance (growth less than baseline) to determine whether you have enough staff, or enough skilled staff working on the project. You should monitor over-performance (growth exceeds baseline) to see if you have unapproved requirements, or if staff are not developing sufficiently complete requirements. In both cases (under and over-performance), you should monitor whether the planned baseline is accurate. If not, then you should re-plan and move on.

Finally, you should review the requirements growth curve at the end of the project to check the accuracy of your initial estimated baseline requirements. For example, perhaps you did not accommodate requirements accepted after project start, so next time you will add 5% to the total. You should also look at the shape of the requirements growth and how well it tracked against your baseline. Did they match during all phases, at the beginning, only at the end? Understanding these simple differences is a great start to delivering on schedule next time.

Requirements Implementation Status

Managers need to know how well requirements are progressing through development activities. Instead of waiting until the last weeks of a project's schedule and then assessing how many of them are done (when it is too late to do anything meaningful about it), a software manager should know as soon as possible that requirements are not being completed. If this measure is used during the fact, then the manager can assign resources or re-plan the schedule. This measure is an indicator of the rate at which your requirements are moving through your development pipeline.

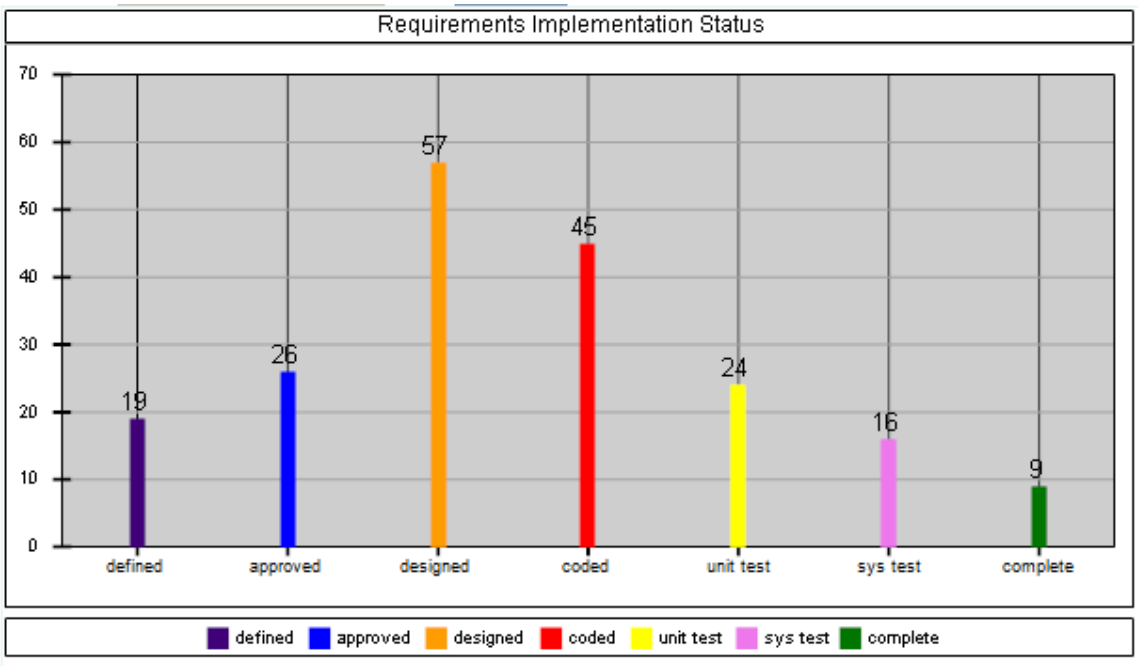


Figure 12 – Requirements Implementation Status

To track the status of implementation, you must measure the requirements moving through each development activity. If we consider each unique development activity (such as analyze customer needs, develop requirements, design software, develop code, etc) to be a state, then we can measure the number of requirements that are completed during each state. This measure forms the basis for a requirements implementation status indicator. A typical set of states for implementation status could be: defined, approved, allocated, designed, implemented, tested and verified.

A measure that shows the status of all requirements is essential in monitoring program status and acts as a scorecard to illustrate that requirements are being implemented. Early in the program schedule, ensure that requirements become defined, approved and designed as the system architecture is finalized. Near the end of the program schedule, you should see requirements move from implemented status to tested then complete status. While valuable in detecting "requirements volatility", this measure also supports monitoring efforts, configuration management and quality.

In the CMMI's Requirements Management process area, one of the work products identified for sub-process 1.3 "Manage Requirements Changes" is "requirements status." This measure is vital for controlling requirements during the development process.

Requirements Change Summary

Requirements change summary displays the type of requirements changes that are made each period. Change summary is useful in understanding the type of changes that are being made each period. Type of requirement change means "added", "edited" and "deleted". Remember that "deleted" requirements are no longer requirements so they cannot be counted unless the requirements tool provides a mechanism for listing the requirements that have been deleted.

As the project proceeds, you will expect that the rate of change decreases, as requirements activities are completed and software development and test assumes the majority of resources. During the end or a software project, individual requirement additions or changes, especially at the top level, can have tremendous negative impact on the software product.

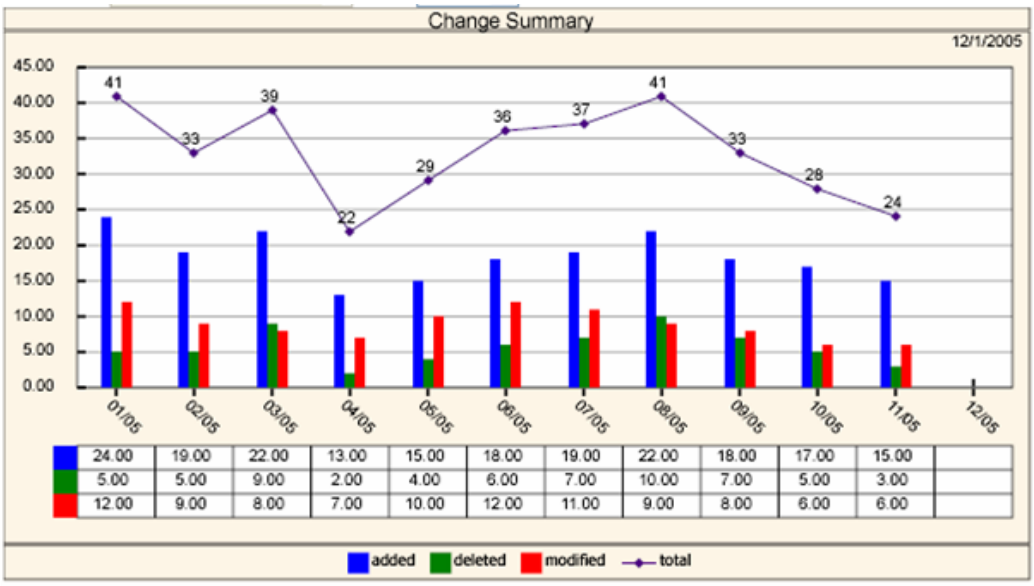


Figure 13 – Requirements Change Summary

A typical requirements change summary graph would contain added, edited and deleted requirements per month. These three might be added to form the total number of changed requirements. You would enter values for the limit of acceptable changed requirements per period. The graph would also contain the total number of requirements along with a color-coded indicator bar.

Requirements Allocation

Requirements allocation provides visibility into how many requirements are being implemented. Remember that a top-level requirement is usually allocated to another work product (such as a software design, software code/module or test plan). Eventually, all requirements are satisfied by being assigned to a work product that is actually developed or produced. Measuring requirements allocation shows how much work has been done on actually implementing requirements.

Each requirement in a requirements document is allocated to a work product. Requirements allocation measures the completion percent of that allocation. For example, all the requirements in a URD (top-level requirements document) must be allocated before the software product can be considered complete.

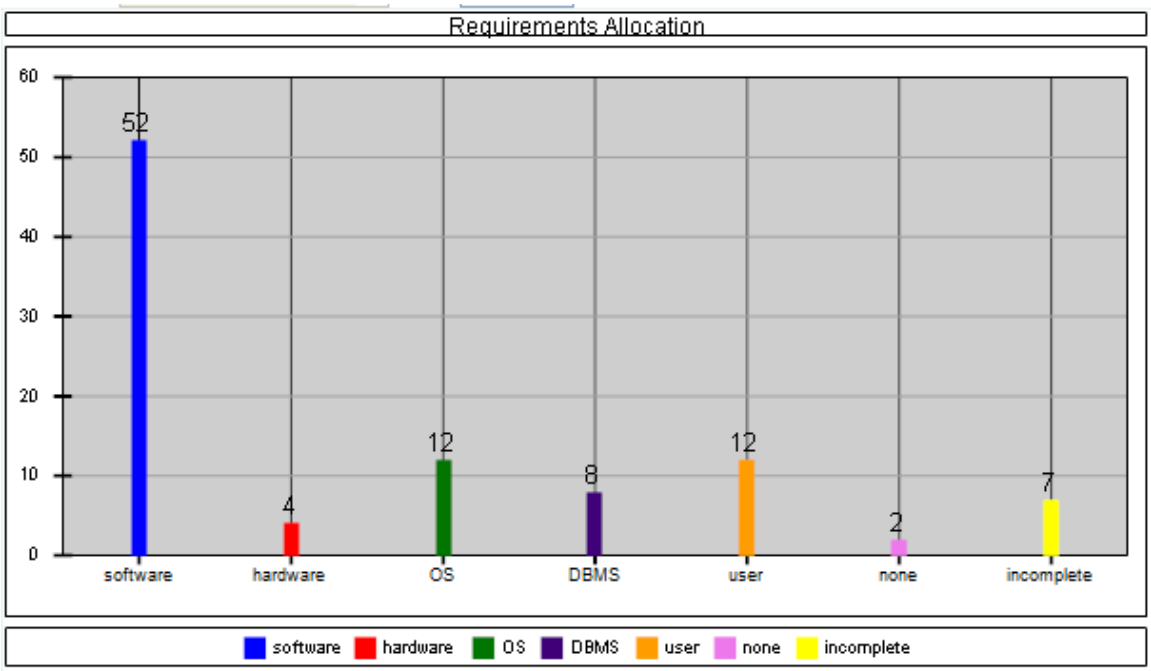


Figure 13 – Requirements Allocation

Notice in figure 13 that we can see where the requirements for this requirements document have been allocated, primarily software. In addition, we can also see that seven (7) requirements have not been allocated (shown in the “incomplete” bar). The number of incomplete is a potential risk since these must be implemented in one of the allocation targets.

Requirements allocation helps a manager in several key ways. By monitoring allocation, you are able to see how well analysts and requirements engineers are able to define and develop the software architecture. As requirements are defined and allocated, Requirements Allocation will show a nice pipeline of requirements being developed. When there are problems, the pipeline will develop lumps, as unallocated requirements "pile up" in one work product. Additionally, you want to ensure that the requirements allocation itself is reasonable. For example, if all URD requirements were allocated to software under development (and none to hardware, the operating system, supporting components like a database), this is cause for suspicion.

Requirements Volatility

Requirements volatility provides status related to the amount or percent of requirements which change each period. Volatility is of particular importance to a project manager since subsequent activities such as software design, development and testing depend on having stable requirements.

When requirements are volatile it means that these down stream activities must re-work their work product (e.g. code) to implement the changed requirement. Volatility at the beginning of a project is expected. However, at the end of a project a single change to a top level requirement could significantly delay a software product.

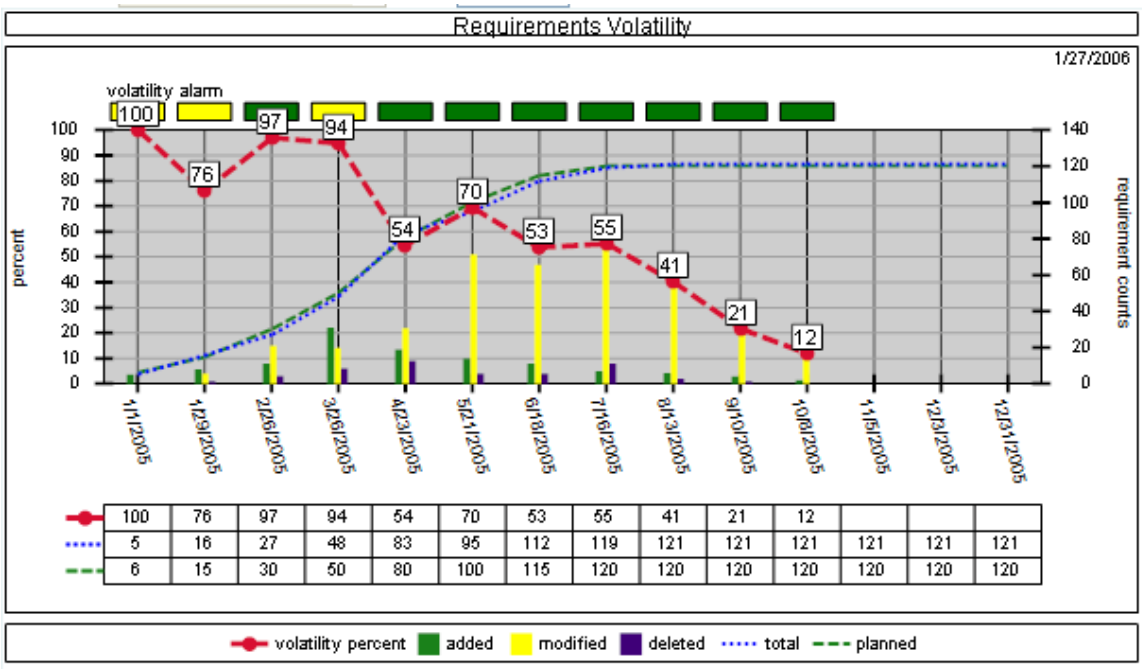


Figure 14 – Requirements Volatility

A converse to volatility is requirements stability. While volatility shows the percent of requirements that change each period, stability shows the percent of requirements that did not change each period. For example, if requirement volatility is 20% (of total requirements), then stability is 80% (of total). In most cases the graphs are interchangeable, such that any decision made with the volatility graph can be made with the stability graph.

Requirements TBDs

When requirement work products are first being developed, the engineer may not know the specific attributes of a requirement. As a placeholder for such a requirement, they might enter the text “tbd” (for “to be developed”) in their requirement tool. Requirement “tbd”s are useful for laying out the structure of a requirement work product, and for annotating where requirement definition is needed. Over time, the engineer replaces the “tbd” with actual requirement text, or removes it if it is un-needed.

From a management perspective, a “tbd” is serious potential problem (also called a “risk”) as it represents a requirement which has been identified but not defined (or implemented or tested). The following figure is a sample graph showing Requirement TBDs.

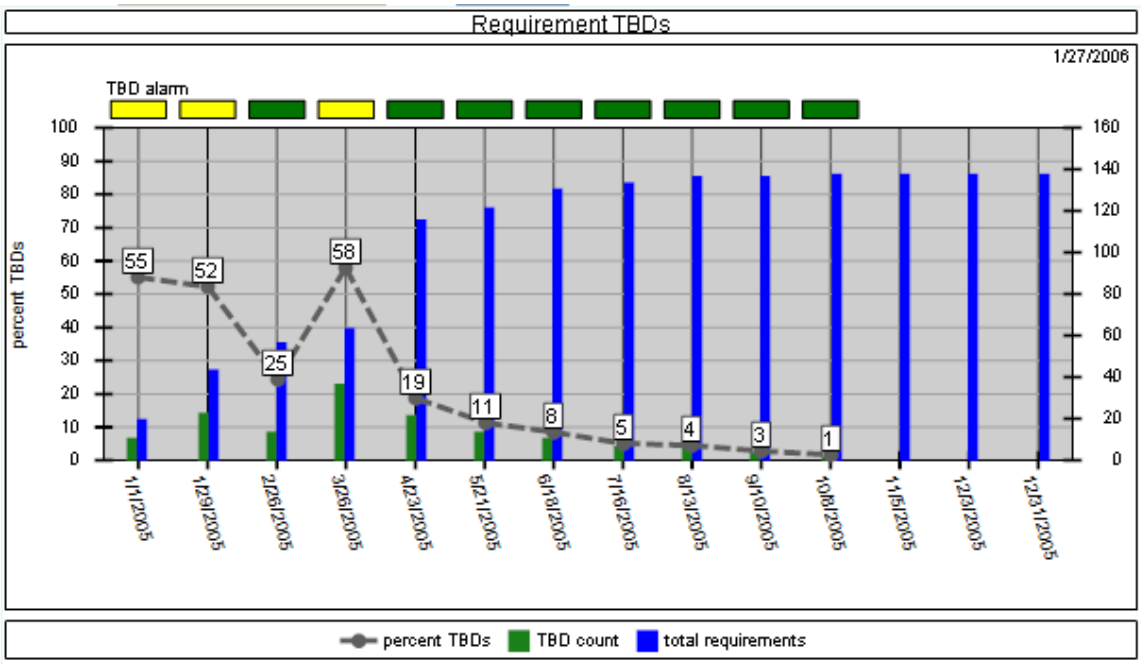


Figure 15 – Requirements TBDs

To manage requirement “tbds”, you need to monitor the number of them each period and ensure that the number goes to zero. You should ensure that requirement documents that contain “tbds” are not considered complete. And, you should make sure that peer reviews and inspection activities do not review “tbds”. Ideally, all “tbds” should be removed from a requirement document in time for the document to be reviewed before being sent to the next down stream activity. For example, you should remove all “tbds” from the Software Requirements Specification at least two weeks before attempting to close the Software Requirements phase.

Resources to Engineer and Manage Requirements

This section provides resources that you can use to find information about requirements engineering, requirements metrics and measurement, as well as requirements engineering tools. Commercial tools are available to support all aspects of requirements management.

The next table contains web sites which provide valuable information and guidance on all aspects of requirements engineering and management. Among the resources are process descriptions, checklists, quality criteria and other guidance.

Requirements Management Guidance on the Web	
Software Engineering Institute	www.sei.cmu.edu
Distributive Management	www.distributive.com/resources
Crosstalk Magazine from STSC	www.stsc.hill.af.mil/crosstalk
Scott Ambler's Web Site	www.ambysoft.com
Karl Weiger's Web Site	www.processimpact.com

The following table lists commercial tools that support requirements engineering. The leading tools provide a full range of training, services and support. Please contact Distributive Management to contribute other useful resources.

Company	Product	Web Site
Telelogic	DOORS	www.telelogic.com
IBM	Rational RequisitePRO	www.ibm.com/rational
Serena	RTM	www.serena.com
Borland	CaliberRM	www.borland.com
Steel Trace	Catalyze	www.steeltrace.com

SUMMARY

Hopefully, by this point in the document, you understand that measuring requirements engineering is critical, and that the methods for doing so are not difficult. As your organization matures and improves the requirements processes, you should make sure that the processes you create can also be managed. Even if you are not interested in a formal CMMI rating, the techniques described in this document can help you spot and correct project issues before your project becomes a death march.

CONTACT INFORMATION

For more information, please contact Distributive Management.

www.distributive.com

800.779.6306
540.891.8885 (fax)

Distributive Management
109 Olde Greenwich Drive, Suite 102
Fredericksburg, Virginia 22408

sales@distributive.com
info@distributive.com