

As seen in
**Software Test
& Performance**



Chain of Tools

By Rob Soosaar

Software development is far from immune to trends, buzzwords and fads, but all of that window dressing is

driven by a need to do more and to do it faster, better and first. One area that has been receiving a lot of attention lately is requirements management (RM).

In its rudimentary form, requirements management is a chain, a series of links between the target audience for a deliverable and the people who construct it. The better you can organize your requirements and describe what you want, the more likely it is you'll get it. And if you can't articulate exactly what you want, you may not know the breadth and depth of what's technically possible.

Every successful innovation in engineering is backed by effective prioritization and organization of requirements in

the chain, no matter how the approach may differ. Over my many years in software development, I've experienced the full range of methodologies, spanning what I call the "ultra-lightweight University research style" (where you're lucky to start with notes scribbled on the back of an envelope) to heavily prescribed U.S. military-governed MIL-STD-2176 projects, and everything in between. Each approach has its merits and weak points.

The industry has spent considerable effort examining the front-end process of eliciting "a good set of requirements." Many resources, books and experts have contributed to this field, so I'll spare you

a methodology debate here. But reality intrudes: Regardless of how good you are at gathering and defining those requirements, I guarantee that throughout the life of your project, you'll encounter the inevitable: changes, amendments, additions or deletions.

In this article, I'll explore some of the crucial elements of requirements management and offer some suggestions for dealing with the most common challenge: change. I'll investigate how to handle the valuable requirements that you've collected. And I'll also tell you some tips on improving bidirectional

Rob Soosaar is director of solutions development at MKS Inc. He has also worked as senior software engineer at Raytheon developing next-generation air traffic control and secure satellite communication systems.

communication throughout the links of the requirements chain.

Feature Fundamentals

The first link of the requirements chain is a *feature*—a high-level notion for a capability. Think of your new PDA's packaging that lists its features: MP3 player, spreadsheet, word processor and so on. These high-level features get specified into something much more tangible in the development process, which become *requirements*.

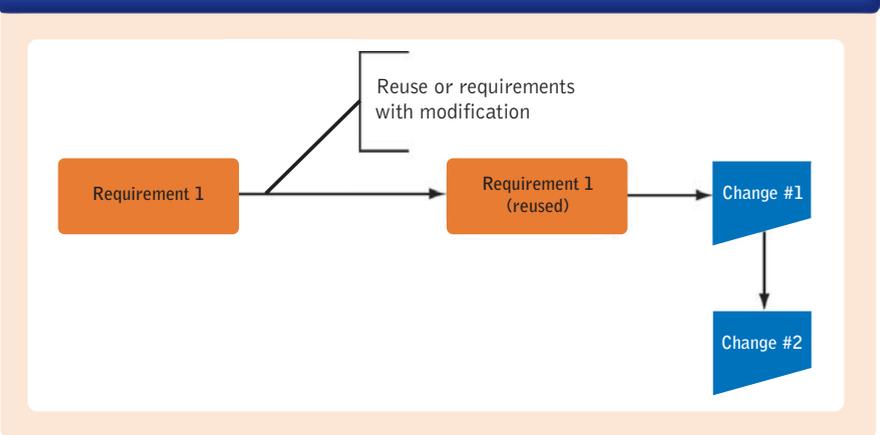
When specifying requirements for a word processor in a PDA, for example, you'll likely end up with many, perhaps hundreds of subrequirements as all the expected word-processor capabilities are documented. Changes to requirements are indicated by issuing change requests. You may have multiple types of change requests depending on your environment and whether you're dealing with features, requirements or downstream implementation activities. As complexity

STRENGTHENING THE CHAIN

Requirements management is made up of a series of links between developers and the end users:

1. Anticipate and plan for requirements to change through the project life-cycle.
2. Store the requirements as they're collected in a central repository that can be readily accessed.
3. Monitor the relationship between requirements, how they were implemented and the subsequent issues that have arisen as a consequence.
4. Make sure engineers and analysts are talking by creating an effective workflow to flag issues for resolution, assignment, review and cycling back to implementation.
5. Keep teams informed of updates to requirements and changes to validation-planning activities.
6. Integrate your implementation process with requirements management so that the overall change impact is well understood by all teams.
7. To foster organizational learning, use a system that offers a contextual audit trail of the decision-making process and supporting research.

FIG. 1: COMPLEXITY OF REQUIREMENTS REUSE



builds, you must consider some key questions: What are the effects and relationships of those change requests? Does an engineering change request have an impact on the originating feature?

Don't Reinvent—Recycle!

As you move down the chain, once you're presented with the high-level features or business objectives, you construct a set of requirements to define a system to deliver the requested features. This activity can be challenging and tedious. Sometimes you start with a completely blank slate and define all the requirements from scratch. But more likely, you look for other existing components that fit the bill, or at least provide a good starting point. In this scenario, you may end up reinventing bits of technology that you've already built, perhaps requiring only slight modifications.

Borrowing or reusing preexisting requirements isn't that simple (see Figure 1). You're usually not looking for a single requirement, but a chain of related requirements, so simple text searches return many results that must be further searched. In some industries, searching for reusable requirements may be relatively simple, as the number of individual components is small. More likely, however, you'll end up sifting through hundreds or thousands of possibilities.

You'll lose a considerable amount of your investment in the front-end process if, after creating your requirements, you have a difficult time locating them when needed.

In cases where organizations haven't established consistent requirements definitions and management processes—and where they've failed to communicate these practices to project stakeholders—the searching exercise becomes too complex and frustrating. Ultimately,

these organizations may give up on requirements reuse and define a new set, reinventing the wheel and expending valuable and often scarce resources.

Your tools and processes play a critical role in effective requirements reuse. They can help you answer some critical questions: Can you determine how effectively the previous year's model met the market's needs? If you found problems, were they at the requirements level or did they stem from poor implementation? How do you find this information? Once you find it, you should track and manage the change process to those requirements and have the stakeholders validate the modifications, as shown in Figure 1.

Utilizing a system that provides a repository that not only allows you to extract what you're looking for but that also supports and fosters a culture of consistent communication of your RM practices is critical. Some of the more sophisticated requirements-creation tools offer mechanisms for defining keywords and defining advanced searches. Ultimately, however, the burden is on you to establish a consistent, standardized way of communicating requirements data to the system in anticipation that someone may want to reuse it in the future.

Stale Spell Checker

To illustrate our next links in the requirements chain, let's take a brief detour to a sample project. You're working on the requirements for a new model of PDA that needs a word processor. Last year's model, though slightly different, included a word processor, and it seems easy enough to just grab that set of requirements that you invested so much time developing. That's all well and good, but on closer

examination, you realize that the market is now demanding additional capabilities from that word processor, including a grammar checker that complements the spell checker.

As you sift through the reports, you realize a core user complaint was the spell checker's ineffectiveness. How could a spell checker be the root of so much trouble, as it seems like a fairly standard capability that's been in use for several releases of the PDA? The requirements hadn't changed, but something on the engineering side must have.

To find out where things fell apart between the requirements and how they got implemented for this particular capability, you need to explore the relationship between the requirements, how they got implemented and the subsequent issues that arose as a result of the implementation. The success of your project relies on your ability to extract this valuable information quickly, allowing you to make informed decisions about how to assign, further refine and/or implement requirements.

Assigning and Refining

In these links of the requirements chain, the real work starts and things get interesting. The analysts have done their job, and the requirements are now allocated to individual engineers or teams for implementation. Requirements will be fulfilled by a variety of means:

- Allocate hardware components from other vendors
- Acquire third-party software
- Develop in-house software
- Develop supporting artifacts: documentation, training and so on

The number of people and tools involved in this process expands rapidly. Engineers examine the requirements closely for the system and plan the detailed design activities that come downstream. They also identify inconsistencies, conflicts and missing requirements. These issues need to be flagged for resolution, assigned, reviewed and cycled back for implementation again.

As a project lead or engineer, you may ask yourself the following questions: How do I know if all of our requirements have been allocated for implementation? Do we use our own requirements system for tracking this phase or are we transitioning to the

use of one or more other tools?

This workflow can be daunting for a large project employing hundreds or thousands of engineers. Tools such as word processors or spreadsheets are commonly used for requirements definition, yet some organizations have adopted more formal definition tools or languages like UML for modeling the needs of the system.

In addition, the quality assurance team gets involved. They plan how to validate the requirements once they've been met with the final deliverable. They probably use yet another system for performing their activities.

fosters communication between engineers and analysts.

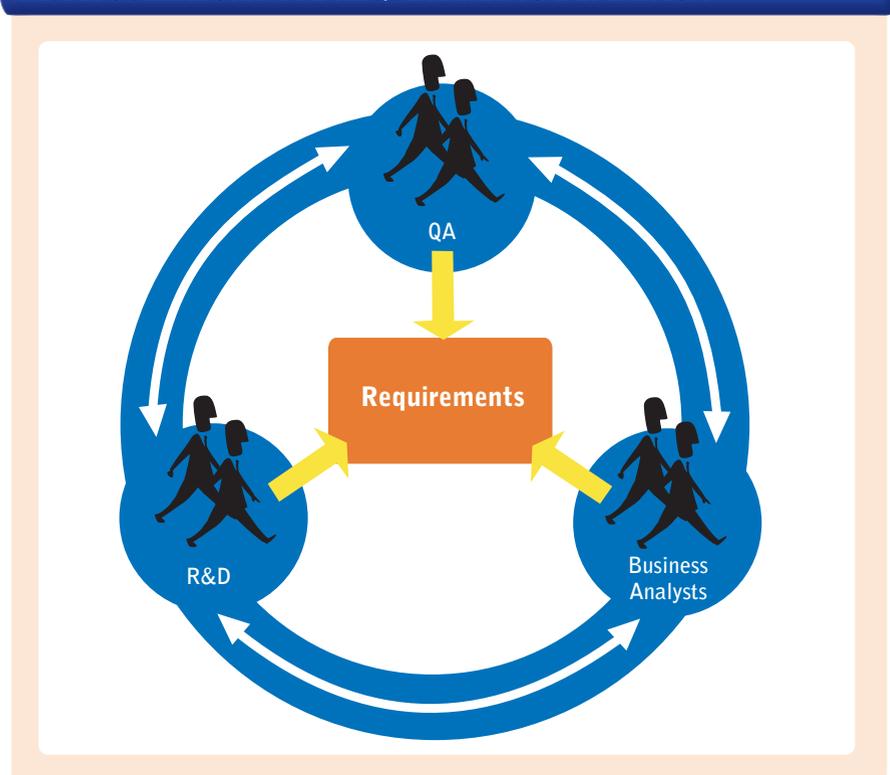
Implementation Options

In the next link, implementation, the rubber hits the road and everyone begins doing detailed design and implementation work. Hardware is developed or acquired, software is written and prototypes are constructed.

In implementation, some of the more subtle inconsistencies in the requirements come to light. What may not have been obvious in the written description becomes evident.

Several things can happen at this

FIG. 2: COLLABORATIVE REQUIREMENTS DEFINITION



Ensuring that all project stakeholders are informed of updates to the requirements and incorporate changes to the validation planning activities is critical to the success of your project.

An adequate workflow system that tracks all the open issues and ensures resolution is the first line of defense in a successful project. Figure 2 depicts all facets of the organization sharing involvement and responsibility in the requirements definition process. Eradicating, or at least minimizing, requirements problems at this stage will save a lot of time, money and effort. The key thread throughout this set of activities is creating a system that

point. The engineers may just make a change and then carry on getting the job done, or they may stop work pending refinement of the requirements. Still worse, they may do nothing and ignore the problem.

The first option, however, is most likely to happen in practice. Engineers are pragmatic and want to get the system working. They're under pressure to meet project schedules, and delays are not well tolerated. Based on the engineer's judgment, minor changes are performed to get the system running and a note is made to have requirements updated, but the note gets lost among all the other implementation notes. This is the most com-

mon scenario in an engineering organization when the requirements begin to diverge from the actual implementation.

There are a couple of problems with this. With the change not being linked back to the original requirements, it's unlikely that the information will get propagated out to the testing group for validation. This mess may get cleared up once test failures are reported and the inevitable argument about who's right ensues between the engineers and QA department. A single engineer's seemingly innocent change on one project may have ripple effects across several other projects because the requirement was shared, along with the resulting implementation. The cost in wasted time and delayed schedules is obvious.

Waiting for requirements to get updated is time-consuming and frustrating for engineers, especially if several layers of bureaucracy and multiple tools and systems are involved. This results in costly delays, but is probably the best course of action.

What we're looking at up until now falls into the more traditional waterfall development model. The problems are amplified in a more agile methodology, where requirements are commonly based on prototyping and dynamic evaluation of technologies.

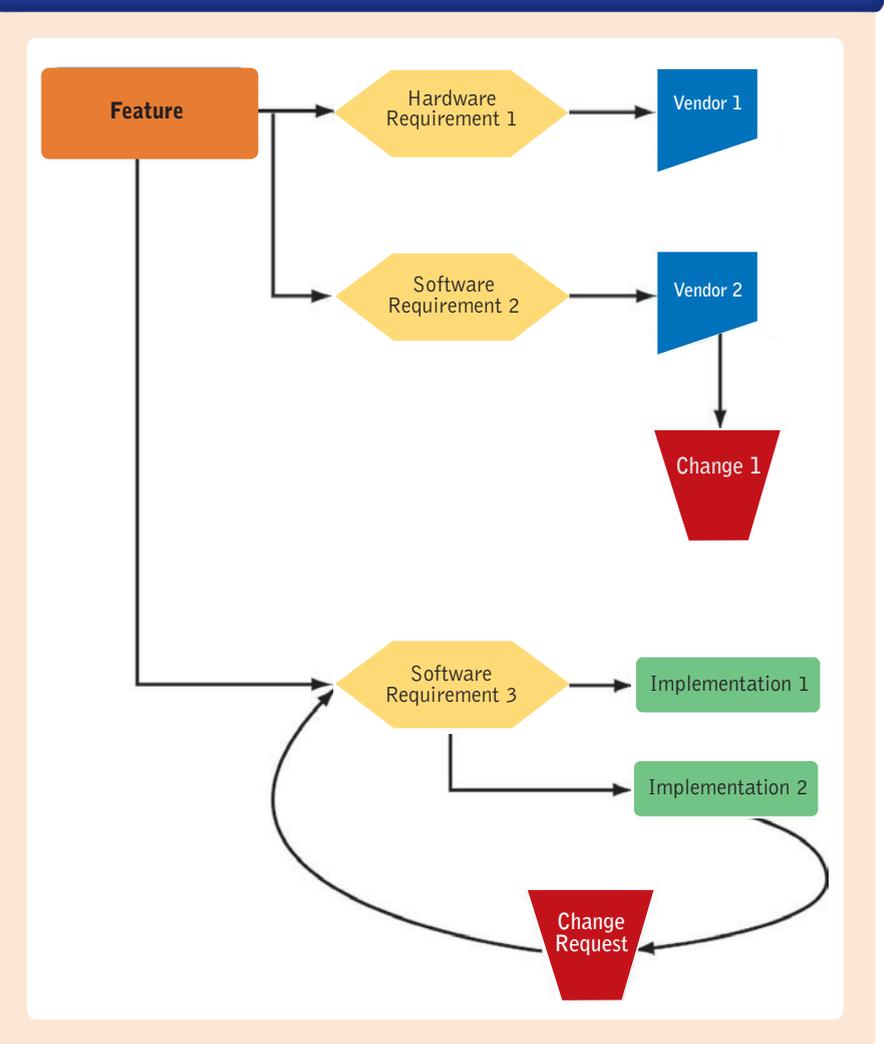
As a result, requirements in agile environments are often vague and ill-defined. A highly efficient communication feedback loop is more important than ever, as is transparency between the tools used for each of the functional areas of the process.

Dealing with Decision

For the final link in the requirements chain, let's examine the decision-making process. As an example, we'll return to our detour: adding a revised spell checker to the PDA word processor. An engineer may be doing some reading in a trade journal and discover an alternative to building one in-house by licensing it from a third-party vendor.

The catch? Well, the requirements don't exactly match the capabilities that can be delivered by the third party. The engineer knows that the vendor-supported approach makes more sense in terms of initial cost, ongoing maintenance and technical

FIG. 3: DECISION-MAKING AUDIT TRAIL



risk. In addition to amending the requirements to match the capabilities of the third-party offering, factors or licensing costs, and legal and technical review all must be accounted for in the decision-making process.

If all decision makers could interact with the same workflow system, act on the same fundamental information and work in parallel, decisions like this one can be made quickly. However, several systems are at play in an organization with a limited ability to share details and context about the request at hand.

Ensuring your system provides a contextual audit trail of the decision-making process and supporting research is important. People's memories of the decision-making processes tend to be short. A decision to choose the third party may have been the right one based on information available at the time, but 12 months later, vendor circumstances may have changed and questions raised about due diligence. A good audit trail will help with organi-

zational learning and may identify places where workflow improvements can be made in the future.

The key point here is integration among all the workflow and communication systems used across a project. Making decisions without all the necessary information and context can be dangerous and costly.

Figure 3 depicts the decision-making cycle, revealing that the organization learning process will continue to build in the repository database as long as the ideas and decision-making steps are captured by the requirements management system.

As your own project's requirements chain responds to change, feature definition, reuse, a clear approach to assigning and refining requirements, and an organized decision-making process can be useful tools to manage workflow and implementation at a task level. In this way, the organization can harness the efforts expended and move ahead efficiently and confidently.