Requirements Management Practices for Developers

By Catherine Connor and Leonard Callejo

Rational Software White paper
TP559, 03/02

Rational®
the software development company

Table of Contents

Introduction

As a developer, are you frequently asked to incorporate "minor" changes in your code to improve the existing system? Do you feel that these requests are coming from everywhere? Are you often working from incomplete or inaccurate specifications? Are you often unclear on what the requirements are actually trying to convey? Do you sense that the requirements are not truly known and therefore are a constant moving target? Do you see yourself at the tail end of a whip constantly reacting to customer fickleness?

If you answered "yes" to any of these questions, keep reading - there is hope. As a developer, requirements management (RM) may not be something you think much about, compared to design and implementation techniques. In fact, many believe that RM is the responsibility of the analyst on your project team and that the developer's involvement in this process is limited to receiving requirements specifications. On projects where the role of the developer is reduced to simply being a recipient of requirements, you will often find a lot of rework occurring. This can lead to missed deliverable dates, and a reduction in the scope of the project, resulting in a sub-par deliverable to the client. This all ultimately leads to an unsatisfied and unhappy client, which negatively affects the morale and frustration on the project team. On successful projects, developers play a larger critical role in the RM practices. This expanded role empowers developers to control what they are developing, minimize rework, and finally, deliver the right solution to the customer.

The goal of this paper is to illustrate the important RM role developers play on successful software project teams. Additionally, the paper will provide RM implementation tips that developers can apply to make their teams more successful in delivering the right solution to their customers.

Why Should Developers Care About Requirements Management?

As a developer, you may think "I don't have time for requirements!" or maybe "RM doesn't involve me!". If that is the case, then how does your team define the system capabilities that you are building? Somehow, someone decides and defines what you will be responsible for delivering. Whether requirements are in this person's head, written on paper napkins, informally discussed in hallways or formally written in requirements documents, every software development team is engaged in some form of requirements management. They may not often realize it because their RM practices might be very informal. Regardless, all software teams are engaged in RM to some degree. The question is to which degree should your team formalize its RM practices?

To understand the importance of developers' involvement in RM, it helps to reflect on the purpose of RM. The purpose of RM is to establish a common understanding between the customer and the software team and encompasses finding, documenting, organizing and tracking changing requirements. This common understanding with the customer is the basis for planning and managing the project. If the team fails to effectively manage requirements, it will hurt the team's ability to deliver on key milestones and thereby discount the accuracy or effectiveness of the project plan. This often results in development and testing resources being wasted working on the wrong thing. Developers play a crucial role in RM not only because they build the software from the requirements, but also because they have the power to prevent the team from starting with incomplete or ambiguous requirements. Maximizing requirements completeness and clarity is a turning point in ensuring the entire team, including testers and documentation writers, will build a quality system in the shortest amount of time.

The degree of RM formalism will vary from project to project, team to team, and is directly linked to how much risk your team is willing to take at not delivering the right software solution. The less formal the RM process is, the more risk your team takes on not delivering software that will satisfy your customer. Hopefully, the degree of looseness in a project's RM process is a concise decision taken by measuring the pros and cons of RM formalization. Some of the most common arguments for adopting a loose RM process include: it will allow us to develop faster, we can better adapt to the changing market, and we don't need formal requirements documents to know what we are supposed to create. Unfortunately, these are the exact arguments that actually end up coming back to haunt a project team and further supports the need to carefully analyze the degree of RM formalism that is required for the project to be a success. Fundamentally, the RM practices should yield: 1) requirements that are clearly understood by all team members, 2) control of changing requirements to keep a team on track for delivering the right solution, and 3) effective communication to keep the entire project team on the same page.

There are cases where adhering to a very formal RM practice may be overkill. For instance, if your team is tasked with building a video game, enhancement requests and changes to the requirements may occur at such a fast rate that formally following a traditional change control process, which involves approval by a Change Control Board (CCB), may actually be detrimental to the success of the project. In this case, the change control process could actually inhibit developers' creativity and act as a bottleneck to the delivery of the software. However, even in this instance, the project team would benefit from utilizing some RM techniques such as storyboards or prototypes to validate game ideas prior to committing to development and delivery.

At the other extreme, there are cases where adherence to very formal RM practices is highly desired. For example, if your project team is tasked with developing software to run a medical device that automatically administers the exact, correct amount of medication to a human patient based on certain conditions, your team should adopt a highly formal RM process to ensure that the correct system is developed. The risk of making a mistake with a requirement in this situation could lead to the loss of human lives.

So how does RM impact a developer like you? Well, studies like the Standish Group reports (see References section) have shown that requirements errors are the most expensive errors to fix because they become magnified the longer they go uncorrected. These errors become increasingly more difficult to correct as you move further along the software development lifecycle and actually have a snowball effect. If you start from a wrong requirement, or a requirement changes midstream in the development lifecycle, your design becomes invalid, which will ultimately result in expensive architectural rework, your validation tests will be wrong, your user documentation will be inaccurate, and so on. Ultimately, this will all lead to more time spent fixing problems that should not have occurred in the first place.

But you're a developer, not an analyst. Isn't RM just for analysts? It is true that the person, or people, representing the customer on your team will be the most involved in RM. Analysts are typically responsible for creating the requirements. However, the quality of the requirements does not rest solely on the shoulders of the analyst. Remember, it is critical that the entire project team have a complete, clear understanding of the requirements. In order to achieve this type of quality, the developer must be involved early on to help clarify and drive out any ambiguity that may be inherent in the first version of the requirements. Developers provide an implementation perspective that enhances the quality of the written requirement, as well as increases the opportunity for successful development of the solution. It is with the developers that the "rubber meets the road"; developers are responsible for turning concepts into reality. Therefore, the sooner developers are involved in clarifying requirements, the greater chance the project team has at successfully delivering the right software solution.

Developers should also care about proper RM because it can simplify their life. Quality assurance (QA) or quality engineering (QE) and documentation teams are more effective when working from quality requirements as opposed to poor requirements that leave them trying to figure out what to test and interrupting developers with questions. Additionally, maintenance activities are reduced to focusing on actual implementation defects in the system, as opposed to defects that result from the original requirements being unclear and incomplete. Better requirements ultimately lead to better quality software that enables developers to focus on forward-thinking enhancements.

Software Problems Linked to Poor Requirements Management

Let's look at some of the problems you may be facing, connect them to their potential root causes in poor RM practices, and propose RM remedies for each of these problems.

Problem 1: Constant Change

How often do sales or marketing people ask you to incorporate a customer-requested change into your code? It often appears to be just a "minor" change and it seems reasonable. And of course you want your code to be successful, so you graciously try to accommodate the request by working extra hours to incorporate the change. These constant interruptions impact your effectiveness at designing and delivering quality software. It is simply harder to concentrate and stay focused. But more devastating to the team is the impact accepting these casual requests has on the project plan and overall software quality. It is constant interruptions like these that cause a project plan to expand beyond its original intent, or what is commonly known as "scope creep". These interruptions sidetrack a team from being able to deliver on its initial target and results in delivering software that does not meet the customer's expectations.

The requirements root cause of this problem is a project team's inability to properly assess the impact of the change or enhancement request before actually accepting it.

By putting developers in a situation to respond to these direct inquiries, the project team risks failing to properly assess the entire impact of accepting the change. Consequently, if the developers oblige the request, the change, which often at first glance can appear to be minor, causes a ripple effect of necessary change on other parts of the code. Also, the ripple effect transcends the code and moves into other areas of the project team. For example, if the change has not been effectively communicated to the testers, they may not be building the necessary validation scripts to test the change and thereby risk compromising the quality of the software. Furthermore, even though the change might require only minimal effort for a developer to implement, the impact on QA/QE or documentation might delay the entire project schedule. Because of ripple effects such as these, it is essential that the entire team be on board with changes that are occurring.

Because of these dependencies, it is important for developers to delegate an acceptance of change to the person on the team who has a high level and complete visibility on the project deliverables. To decrease the risk of destabilizing the project, you only want to incorporate changes that have been approved by the team so that you know you are working on agreed upon functionality, and that everyone is on board to test and document the functionality.

Remedy: Controlled Requirement Change by Assessing Impact of Change

Good RM practices prevent casual changes to be made without being evaluated. They treat change requests as first class citizens to ensure that all potential changes are reviewed and the impact of accepting the changes is assessed, before any changes actually impact a developer's work.

Requirement specifications should only change when a group representing, at a minimum, the customer, the development team and the testing team, has approved the changes. That group is often referred to as a CCB. This group's responsibility is to evaluate every requested change from a customer point of view, from a development point of view and from a testing and documentation point of view. Depending on the type of application, training and support staff may need to be involved as well.

Why is it necessary to involve all these people? Because each representative brings a different perspective to the table when analyzing the potential impact or cost of implementing any given change request:

- The *development representative* is responsible for assessing all of the areas of the software design that the change request impacts, as well as the subsequent level of effort required to implement the change. That representative often requires individual developer participation in analyzing the impact of the change request.

- The *QA representative* must assess the feasibility and level of testing effort associated with accepting the proposed change. One should always remember that a change might be easy to incorporate in code but might be very difficult, at times impossible, to test.

- The *customer representative* provides the business perspective and ensures that the change does not detract the project from its original business objectives, and provides any additional information about the change to help the CCB understand the customer needs.

- Lastly, it is always safe and recommended to include *representatives from training, support, and documentation* on the CCB, since change requests often have an impact on these areas as well.

To ensure the effectiveness of the evaluation process, each member of the CCB should be responsible for finding an adequate replacement in case of absence so that assessing change requests is never biased. For those rare occasions where all parties are not represented, change requests should be deferred for later evaluation.

Good RM practices will protect a developer from being asked to change things on the fly. The changing aspect of requirements, from the time you write them down to the time you deliver your software, is often the single most difficult thing to deal with on a project. The Standish Group in its 2001 Extreme Chaos report states: "Changing requirements are as certain as death and taxes". Mastering the management of changing requirements is critical to the success of a project, and it is a practice that directly impacts the lives of developers.

The advent of iterative development in place of a traditional waterfall approach is of great help to manage requirements change. In traditional approaches, requirements specifications were frozen at the start of the project and changes could not be

incorporated until after the software was released. The only way changes could make it into the release was to go directly to developers and ask them to change their code, an approach as we have seen above which destabilizes the project by failing to assess the true impact of the change on the overall project. Iterative development allows software teams to split the work into iterations where requirements are stable and not changeable during the iteration. At the start of the next iteration, the team gets a chance to update the requirements specification to incorporate any change that was submitted and accepted in previous iterations. In iterative development, specifications only change at the start of iterations, not during the iteration. This allows you to concentrate on a firm set of requirements during the iteration time. When the next iteration comes, you may be assigned a set of defects to work on as well as new or updated requirements.

Problem 2: Being in the Dark about Changes that Affect Your Work

When requirements change, how are you informed of these changes? Is your manager informed of accepted changes? Often developers work on obsolete requirements because someone forgot to inform them of a change that impacts their work.

Even though in practice QE/QA and documentation are more often in the dark than developers, it often happens that a change that indirectly affects your work as a developer is not timely communicated to you. The analyst on your team may discuss alternatives with a particular developer to gauge the impact of a change before accepting it, but fail to realize the impact it might have on your work.

Remedy: Requirement Change Notification

Good RM practices ensure that you are proactively informed when a requirement that impacts your work is changed. It also readily informs you of the person you should contact to make sure you are working on the latest requirements.

How do you know when a change in requirements impacts your work? Unless someone on your team tracks the relationships between requirements and design elements that were created to fulfill these requirements, it is very hard to quickly assess the impact a change in requirements has on your design. Good RM tracks dependencies between requirements and design artifacts so that as soon as a requirement changes you can pinpoint which part of the design is affected by the change.

Because requirements are moving targets, a key benefit of RM is the ability to track whether requirements are implemented. This is referred to as traceability. There are various levels of traceability and unfortunately many ways to abuse the use of traceability. The role of traceability is to give you some assurance that the functionality you committed to your customers will indeed be implemented (done by developers) and work as specified (validated by testers). To ensure requirements are implemented and to assess the impact of requirement change on design, requirements should be traced to design elements. To ensure requirements are validated and that test validation is updated when a requirement changes, requirements should be traced to test artifacts, typically test cases.

What about tracing requirements to source code? While this seems appealing at first (e.g., if I change a requirement, I know which piece of code must be updated), the reality is that code is more dynamic in nature than requirements. You may start with one design, and then optimize that design. The new design will likely result in code changes, but it will still comply with requirements. Maintaining traceability links takes time, and software teams never have enough time. So you should use traceability judiciously. The true value of traceability is in tracing requirements to both design and tests. There are many ways to write code to fulfill a specification and the end users will not care what code you write as long as it meets their needs (and requirements are the vehicle to express these needs unambiguously to the entire software team). An exception to this is if you work on systems that are audited by standard bodies, like the FDA, that mandate traceability from requirements to code, sometimes even lines of code. Even in this case though, you should not be tracing requirements to code while building the software. You only need to report that the *final* product requirements trace to code and that no piece of code was built without fulfilling a requirement. Managing traceability links between requirements and source code can be a full time job with not much reward to it.

Problem 3: Coding Against Incomplete Requirements Specifications

Among the top RM challenges your team faces is the ever-present ambiguity in the first version of requirements documents. When was the last time you read a requirements document and felt confident that you had a solid understanding of what you were expected to build? Most requirements documents do not contain sufficient information for developers to design their part of the system, leaving them to “interpret” what the users want.

Despite analysts' best efforts to gather and document requirements thru RM workshops, joint application development (JAD) sessions, interviews, or focus groups, developers often have many questions after their initial read of the requirements. This is true no matter how much the analysts are subject matter experts. Often, developers simply provide a different perspective that may have been overlooked by the analyst. For example, developers often raise exception situations that a user of the system may encounter that the analyst failed to recognize. Therefore, it is imperative that developers and analysts work closely together to refine the original requirements specifications before starting design. Developers should feel comfortable raising issues and concerns about the requirements specifications to the analyst, who should then provide answers, contacting customers directly if necessary.

Remedy: Detailed and Unambiguous Requirements Specifications

Good RM practices recognize that the first version of the requirements specification needs to be reviewed by the project team, and that developer questions about that specification are expected. Therefore, it is essential that developers are given sufficient time to review and raise any questions about the requirements. In turn, the responsible requirements analyst should answer any questions the engineers raise, and document the clarifications in the requirements documents as appropriate. For project planning purposes, time should be allocated for this review process and in particular, time should be allotted to allow time for the analyst to review unknown issues and questions directly with the customer for more details. Therefore, requirements specification reviews should be the norm rather than the exception on any project. Lastly, it is essential that all developer issues be addressed before design begins. If thorough requirements specification reviews are compromised, your project team runs the risk of allowing engineers to introduce unwanted assumptions into the design of the system.

Another effective practice that can be utilized, in addition to reviews, is the use of use cases to express functional requirements. Use cases describe a system's behavior when interacting with the outside world by documenting system functionality as a sequence of steps from a user's perspective. That perspective provides both, software teams and their customers, a common understanding of the expected behavior of the system to build. By minimizing the risk of requirement misunderstandings, use cases also improve the clarity of requirements. For instance, in projects using use cases, all too often the analyst will simply gloss over major use cases (usage scenarios of how the systems will be used) and believe that there are no major issues. However, once the developer begins to flush out the details of the use case by identifying all alternative flows (alternative usage scenarios in the event that something goes wrong), the real issues begin to surface. If a developer is not doing this elaboration work *before* design, that design will need to be reworked many times to incorporate all user scenarios. Establishing a key communicative relationship between the developer and the analyst in that elaboration process often determines the success of the final product delivery.

However, it must be noted that not all requirements can be expressed in use cases. For instance, reliability and performance requirements are better expressed in a declarative form (e.g., "the system shall..."), but use cases do provide a mechanism to document the user experience with the system from a complete perspective. By focusing on the user point of view, use cases also avoid the undesirable problem of introducing design elements in requirements specifications and thereby free designers from unwarranted design constraints.

Problem 4: Coding Against Obsolete Requirements Specifications

Before you can start designing and writing code, you need to understand what you are expected to deliver, and the analyst on your team is in charge of validating that your understanding is inline with the customer needs that have been identified. That is the role of the requirements specification document. It must ensure that everyone on the team is on the same page regarding the system that needs to be developed to resolve the specific problems that the customer has raised. Given the role of the requirement specification document, it is critical that developers can easily locate the document, and more importantly locate the most current version of it. This may seem trivial, but in practice it has proven to be quite tricky. Often times, specifications get reviewed on an ad hoc basis, decisions are made and assumptions are drawn. Unfortunately, in such situations, the requirements do not always get updated in a timely manner, and furthermore, the updated requirements do not always get redistributed to the team. Yet another scenario might be that the updated requirements do get redistributed only to have them lost amidst the clutter of email that is in everyone's inbox. Ultimately, keeping the requirements current and providing access to them should be standard practice for your team.

Remedy: Up-to-Date Specifications Readily Available

Good RM practices clearly identify a central repository for managing your requirements specifications so that everyone always has access to the latest version of the requirements. Rather than relying on email timestamps, you, as a developer, can be assured that you will not mistakenly start with an obsolete version of the requirements documents and waste time and effort. Furthermore, communicating to the team (via email, phone, meeting, etc.) that the requirements have been updated is simply a good, healthy practice.

Six Tips to Improve Requirements Management in your Organization

In this section, we provide simple and effective guidelines for developers to empower their team to prevent the software problems discussed above.

Tip 1: Participate in Establishing a Change Request Process

This may seem daunting but it is actually quite easy to achieve. Simply put, any requests for change should be validated before being accepted.

- The first step in establishing the change request process is to determine how your team will collect and manage the change requests. The simplest method is to create a standard paper form that everyone can fill out and submit via email or in person. A more robust method would be to employ some degree of automated tool support.
- Next, you need to determine how you will store these requests. Will it be keeping the paperwork in a three-ring binder or will you use some sort of electronic database? Use of a commercial defect and change tracking tool (like Rational® ClearQuest®) greatly simplifies managing change requests by allowing the submission of change requests over the Web, requiring no local software installation, and allowing the CCB to perform queries to triage change requests and determine which ones to accept.
- Then your team needs to decide how often the CCB should meet to review change requests and determine the criteria to use when determining which ones will be implemented.

Tip 2: Enforce the Established Change Request Process by Just Saying NO

You, as a developer, play a critical role in successfully establishing a change request process for your team. The more you accept and implement ad hoc change requests from the typical sources (e.g., sales people, customers, senior management, etc.) without having the CCB assessing them, the further you push your team away from controlling the never-ending feeling of constant change. Also, this will only increase the number of ad hoc requests that come directly to you because the requestors will know that you are the one they can count on to implement the features that they want.

In order to help control the constant change that impacts your team, you must learn to say “no” to your requestors and to direct them toward your established change control process. This may seem easy enough, but often this can lead to high-pressure situations. For example, often times, the most successful sales person can be very influential and convincing and will often be the instigator of many ad hoc requests. The sales person will apply pressure by stating things like, “I can close the XYZ account if you add that feature” or “I have been losing deals lately to the competition because we don’t have the ABC feature”. No matter what the arguments appear to be, you and your development team have to exhibit strong discipline and politely direct these requestors to your established change control process. The change in these requestors’ behavior will not occur over night, but, with some patience and dedication to this process, it will change over time.

Tip 3: Establish and Participate in Requirements Specification Reviews

Requirement specification reviews are a simple and effective way to validate your understanding of the requirements. As a developer, you know that whenever you receive a set of requirements you have many questions because some of the specifications are unclear or ambiguous. Rather than assuming what is intended and increase your team’s risk in not delivering what the customer is asking for, as well as creating rework for yourself and your team, your team’s project plan should allocate time for periodic requirements specification reviews. These reviews do not need to be terribly formal with set rules and procedures, rather they should be an open forum where specific pieces of the requirements specifications are openly discussed with the assigned development team to ensure that a solid understanding has been gleaned from the requirements.

As a developer, you should be an active participant in these review sessions and you should come prepared with initial preliminary designs or concepts based on your interpretation of the requirements. This process is highly iterative in nature in that multiple sessions are typically required before development teams are able to establish a solid understanding of the

requirements and are in a position of moving into design. Also, through these iterations, it is the responsibility of the analyst on your project team to ensure that all changes are properly captured and documented. Before you and the other developers move into design, the entire team must reach consensus that everyone has a solid understanding of all the requirements.

Furthermore, the iterative nature of the requirements specification reviews provides a self-checking mechanism for the team to ensure the quality of both, the software requirements and preliminary designs. By keeping both of these artifacts in synch, the project team will move forward together and increase its chances of delivering a successful solution. Consequently, time should be allocated thru the design phase for requirements specification reviews. Often times, it is during design that many developers discover more ambiguities in the requirements specifications and need further clarity. Here again, a requirements specification review is in order with the results of the meeting being clearer requirements.

Tip 4: Require a Glossary

A glossary document is a simple yet powerful way to remove ambiguity from requirements specifications and avoid misunderstandings and should be owned, developed and maintained by the analyst. Due to time constraints, team members may not be aware that they have different interpretations of the same requirement. The glossary does not need to list every term used in the requirement specifications, but it should definitely include any terms that potentially have high ambiguity. A glossary helps remove ambiguity by accompanying requirement specifications with definitions of key terms. If terms in the glossary have specific and important relationships (e.g., in building a financial application, a *customer* can only have a set number of *accounts*, and no more than two customers can share the same account) you may want to supplement the glossary with a domain model which visually depicts the agreed upon relationships between *customer* and *account*, as the software will need to respect them.

Any member of the project team can suggest terms to be added in the glossary by the analyst, but the entire team must agree upon all the definitions for the terms. As a developer, you must be sure to clearly understand the key terms that impact your area of the system that you are responsible for developing.

Tip 5: Insist on a Project Vision to Help Provide Context Around Your Solution

To help gain understanding of the application of the software you will be building, your project leader or the analyst on your team should be documenting the specific business problems your team will be tasked to solve with this project. Because of time pressure, a lot of software teams do not take the time to analyze the business problems they will be solving. Rather, they focus on the specific requirements that they are responsible for and move into design as quickly as possible. Without an understanding of the business problems, developers run a great risk in developing something that their customer does not want.

Think about a model car kit for a moment, completely contained in the nice box with the nice picture of the completed car on it. You open it up and you have instructions and a bunch of pieces and parts that need to be glued together. If you simply dive into the instructions and start building your model car you will not end up with the beautiful car that is pictured on the box. You'll end up with something rather plain and bland. However, if you studied the picture on the box and then developed the car using the instructions, you would end up with the correct end product; something that is very pleasing to you. In this analogy, the picture of the car on the box served as the vision and the instructions were the requirements. Similarly, on software projects, teams need to be acquainted with the vision in order to build the right solution for the customer. As a developer, before you dig into the requirements and start designing, review your project's vision document. If it does not exist, raise this issue to your project leader or analyst and insist on them developing one, before you get started. You may save yourself from a lot of time being wasted building useless software.

Tip 6: Adopt Use Cases to Illustrate the Functionality of the System

Express functional requirements in the form of use cases to better understand how users will use the software. By creating usage stories of the software, use cases help flush out the requirements details and prevent developers from having to do a lot of guesswork. Use cases are unique at providing a format that expresses requirements for technical and non-technical folks. They create a universal description that expresses what the software should do for users. By avoiding the traditional user and system representations of requirements, which often leads to a disconnect between the less technical analyst and the more technical developer, use cases give software teams a better chance to agree on and understand the expected system functionality.

Furthermore, use case storyboards are a great alternative method to build a user interface prototype to validate requirements. A **use-case storyboard** is a logical and conceptual description of how the functionality described in the use case will be presented by the user interface. Use-case storyboards are used particularly to understand usability requirements. They represent a high-level understanding of the user interface and are much faster to develop than the actual user interface. The use-case storyboards can thus be used to discuss various user interfaces before it is prototyped, designed, and implemented.

Writing good use cases takes practice and Rational provides Webinars and various papers on the Rational Edge (http://www.therationaledge.com) on writing good use cases. Furthermore, for existing Rational customers, there is a wealth of information on the Rational Developer Network.

Leveraging Tools to Automate the Requirements Management Process

The biggest help that can be provided to a software team when it comes to RM is process guidance. An RM tool by itself can help, but only to the extent that the process that it automates is good. Furthermore, an RM tool can only help automate parts of the process. For example, no tool can replace the need for communication between team members. To be successful, first focus on your organization's RM practices and be sure that those are sound and work well. Then, and only then, your team can benefit from using an RM tool.

Rational provides a complete RM solution, including process guidance in the Rational Unified Process® (RUP®), tool automation with Rational RequisitePro®, and public or on-site classes from Rational University (http://www.rational.com/university/index.jsp). Rational also offers various forms of onsite Professional Services through local customer teams.

Requirements Management Best Practices

Best practices are documented in several books (see the reference section of this document for a recommendation). However, just-in-time access to guidelines is often the most practical way to help ensure guidelines are followed. That is why Rational publishes online software development process guidelines in the RUP in the form of a navigable and searchable Web site. These guidelines include roles and responsibilities for managing requirements effectively. You can get a glimpse of the RUP at http://www.rational.com/products/rup/index.jsp.

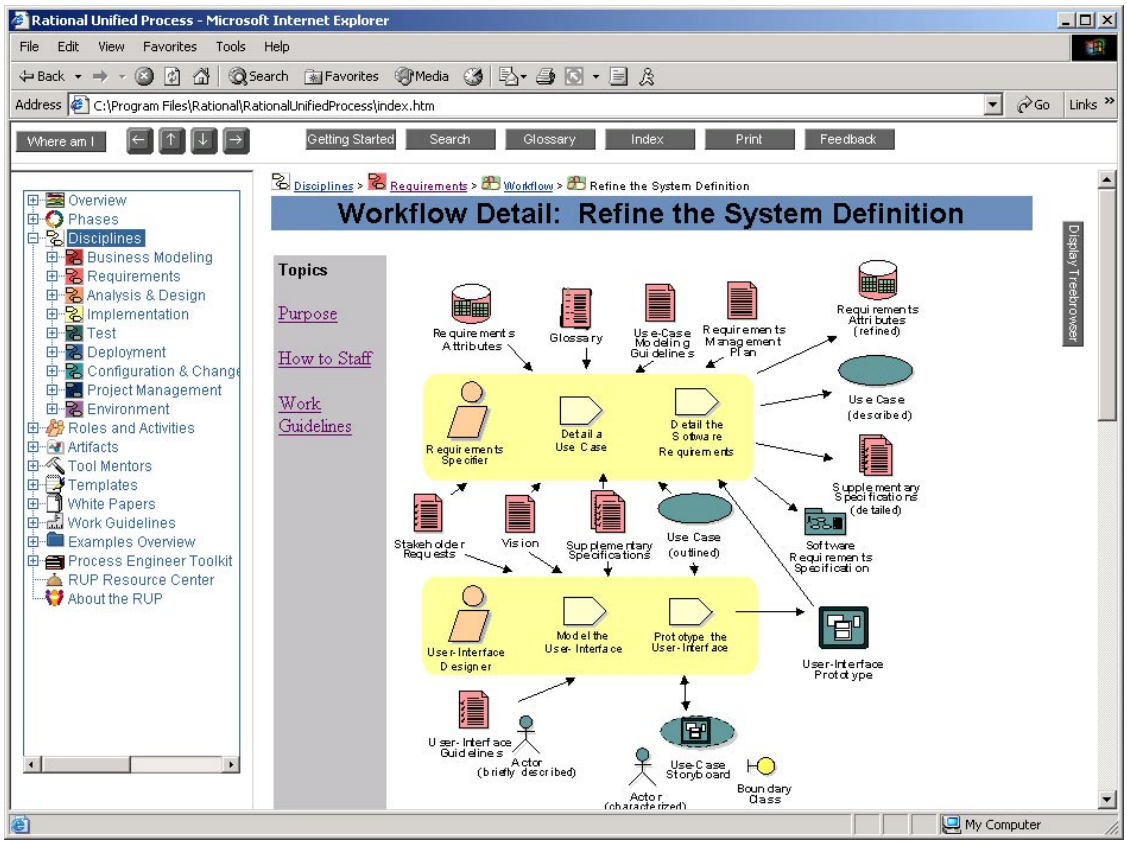

Figure 1: RM best practices in the Rational Unified Process

The RUP also includes tool mentors, which spell out exactly how to implement RM best practices when using Rational RequisitePro.

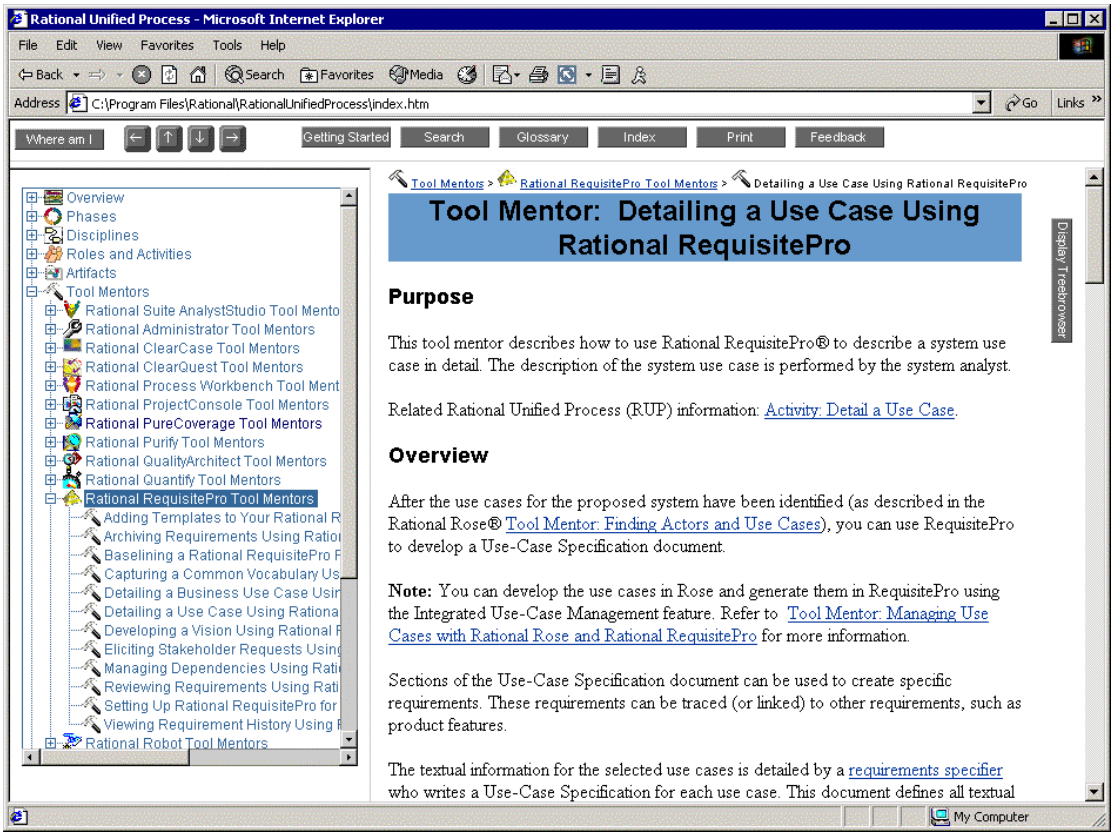

Figure 2: Rational RequisitePro Tool Mentors – tool guidance in the Rational Unified Process

A Requirements Management Tool for the Developer

By automating parts of the RM process, Rational RequisitePro provides the following benefits to developers:

- Fast access to the most current requirements specifications
- Controlled requirements changes
- Quick access to the glossary when reviewing specifications
- Complete audit trails of requirement creation and changes
- Immediate identification of requirement change impact on design
- Use case specification templates

Most developers find Rational RequisitePro the easiest requirements management tool to use because it integrates with Microsoft Word, which is likely the format in which they already receive requirements specifications.

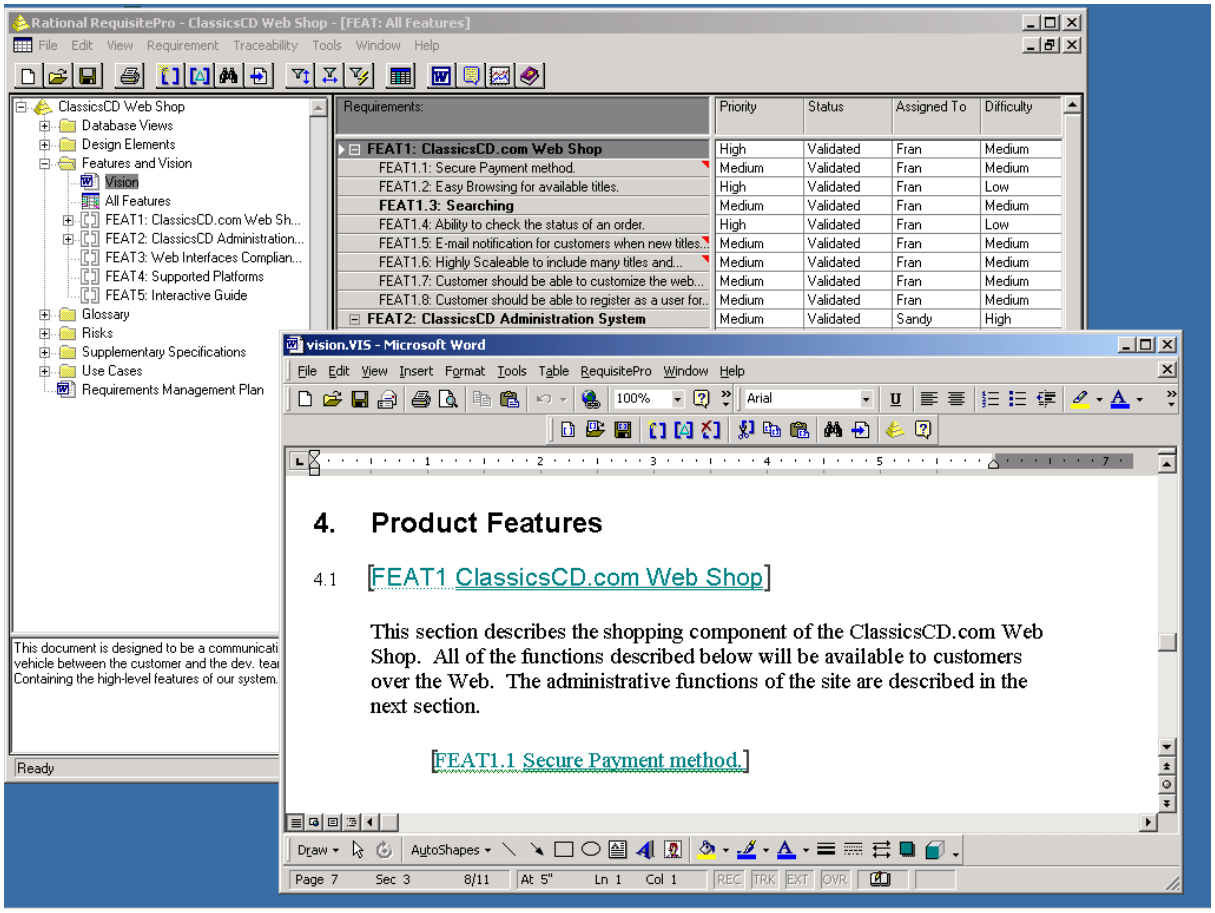

Figure 3: Rational RequisitePro – tight integration of a requirements database with Microsoft Word

To get a glimpse at Rational RequisitePro, you can view the following demo recordings:

- Overview demo at http://www.rational.com/products/reqpro/index.jsp
- Detailed demos at http://www.rational.com/tryit/reqpro/seeit.jsp

Access to Requirements from Development Tools

Above and beyond providing easy access to requirements, Rational RequisitePro is also integrated with Rational's development environments: Rational XDE™ Professional: Microsoft® .NET Edition and Rational XDE™ Professional: Java™ Platform Edition (www.rational.com/xde). Rational RequisitePro also integrates with Rational's visual modeling tool, Rational Rose (www.rational.com/rose), to provide fast and up-to-date access to requirement specifications directly from your environment of choice.

These integrations provide for seamless navigation from use cases diagrams, documented in either Rational XDE or Rational Rose, to the use case specifications stored in Rational RequisitePro. With these integrations, RequisitePro maintains use case specifications as true requirements documents ensuring that you get all the benefits of managed requirements while documenting the software use cases.

For more information on these integrations, see the use case management white papers on the Rational Web site (www.rational.com/products/reqpro/whitepapers.jsp).

Conclusion

We hope this paper has proven to you that developers must play an active role in implementing effective requirements management techniques for software projects to succeed and to minimize chaos in developers' lives.

By refusing to incorporate changes that are not approved, along with questioning the ambiguity and completeness of requirements documents *before* you start designing software, your daily rework and frustration challenges will be reduced, and you will help your team deliver software that actually solve customers problems, which is the key to succeeding in the software industry.

With the added pressure to deliver software faster, it is that much more important to get the requirements right from the start because you may not be given a chance to fix defects caused by poor requirements. Because of the impact requirements have on the rest of the development lifecycle – they define what to design, what to test and what to put in the user manual - requirements errors have been shown to be the most costly errors in software projects. Yet once you know what to watch for, requirements errors are relatively easy to avoid. We hope this paper has shown you what to watch for.

References

The references below can support you in helping your teams understand the value of RM and adopting good RM practices.

- Achieving ROI with Rational Requirements Management Tools, an IDC-sponsored study at http://www.rational.com/products/whitepapers/431.jsp
- Understanding and Implementing Stakeholder Requests: The Integration of Rational ClearQuest and Rational RequisitePro at http://www.rational.com/products/whitepapers/414.jsp.
- Standish Group – Extreme Chaos Report 2001 at http://www.standishgroup.com/
- Managing Software Requirements – A Unified Approach, Dean Leffingwell and Don Widrig.
- Virtual demos of Rational RequisitePro are available at http://www.rational.com/tryit/reqpro/seeit.jsp.

If you are a Rational customer on current maintenance, you can register for the Rational Developer Network at http://www.rational.net. The Rational Developer Network provides a wealth of information for software developers, including requirements management, as well as Java and .NET developer process guidance.

Rational®
the software development company

Dual Headquarters:
Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Toll-free: (800) 728-1212
E-mail: info@rational.com
Web: www.rational.com
International Locations: www.rational.com/worldwide