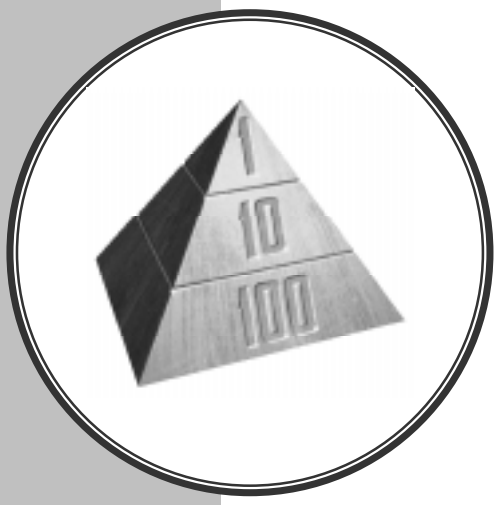
RATIONAL
SOFTWARE

Applying Requirements Management with Use Cases



Roger Oberg
Leslee Probasco
Maria Ericsson

Technical Paper TP505

Roger Oberg is Vice President and General Manager for Rational Software's requirements management products and on the Board of Directors of two start-up software companies. He has 16 years of high technology marketing and engineering management experience in office automation systems, software development tools, and telecommunications, and a B.A. in Economics from the University of Michigan.

Leslee Probasco is the Product Manager for Rational University's requirements management professional education courses. She is a trainer and consultant with experience teaching requirements management, project management, TQM, CMM and ISO 9000.

Leslee has twenty years of software management and development experience in the areas of military and aerospace systems, telecommunications, oil and gas, and electrical engineering. Leslee has a M.S. in computer science from the University of Denver and a B.S. in Management Science and Operations Research from Colorado State University.

Maria Ericsson is Senior Technical Consultant for Rational Software, located in McLean, Virginia. Maria is part of the team developing the Rational Unified Process, and the co-author of the textbook The Object Advantage: Business Process Engineering with Object Technology. Maria has a M.S. in Engineering Physics from the Royal Institute of Technology in Stockholm, Sweden

Applying Requirements Management with Use Cases

by Roger Oberg, Leslee Probasco and Maria Ericsson

If you are new to or somewhat familiar with requirements management and are interested in requirements process improvement, this paper offers a framework with which to develop your own approach.

USE CASES AND SOFTWARE REQUIREMENTS SPECIFICATIONS (SRS)

To make requirements management workflows more meaningful to readers, the authors have chosen specific document types and other requirements management artifacts from Rational Software's Unified Process™ and the industry-standard Unified Modeling Language, both of which recommend a use-case-driven software engineering process.

Therefore, a use-case-driven approach for specifying software requirements is described here. These workflows may also be used with the more traditional Software Requirements Specifications (e.g., IEEE standards) in place of or in addition to use-case models and use cases.

Software and System Development in the Age of Process

For most software and system* development teams, the 1990s have been process-intensive when compared to the more freewheeling days of the past. Standards for measuring and certifying effective software development process have been introduced and popularized. Many books and articles on software development process and related material on business process modeling and re-engineering have been published. Increasing numbers of software tools have helped define and apply effective software development process. The global economy's dependence on software accelerated in the decade, enabling development processes and improving system quality.

So how do we explain the high incidence of the software project failure today? Why are many, if not most, software projects still plagued by delays, budget overruns, and quality problems? How can we improve the quality of the systems we build as our businesses, national economies, and daily activities become increasingly dependent on them?

The answers, as always, lie in the people, tools, and processes applied to our profession. Requirements management is often proposed as a solution to the ongoing problems of software development, yet relatively little attention has been focused on improving the practice of this discipline.

This paper presents the elements of an effective requirements management process and highlights some of the obstacles to its successful implementation.

---

* Requirements management applies equally to software-only projects and to projects in which software is only a part of the end result or not included at all. For convenience, the paper will hereafter use the term "system" to mean any or all of these things. However, it is the abstract nature of software development, alone or in combination with hardware, that complicates requirements management and is therefore the primary focus of the paper.

Why Manage Requirements?

Simply put, system development teams who manage requirements do so because they want their projects to succeed. Meeting their project's requirements defines success. Failing to manage requirements decreases the probability of meeting these objectives.

Meeting your project's requirements defines success.

Recent evidence is supportive:

- The Standish Group's CHAOS Reports from 1994 and 1997 established that the most significant contributors to project failure relate to requirements.[1]
- In December 1997, Computer Industry Daily reported on a Sequent Computer Systems, Inc. study of 500 IT managers in the U.S. and U.K. that found 76 percent of the respondents had experienced complete project failure during their careers. The most frequently named cause of project failure was "changing user requirements."[2]

Avoiding failure should be sufficient motivation to manage requirements. Increasing the probability of a successful project and other benefits of managing requirements may be equally motivational. The Standish Group's CHAOS report further established that managing requirements well was the factor most related to successful projects.

What is a Requirement?

The first step towards understanding requirements management is to agree on a common vocabulary. Rational defines a requirement as "a condition or capability to which the system [being built] must conform." The Institute of Electronics and Electrical Engineers uses a similar definition.

Well-known requirements engineering authors Merlin Dorfman and Richard H. Thayer offer a compatible and more refined definition that is specific – but not necessarily limited – to software:

A requirement is a condition or capability to which the system must conform.

"A software requirement can be defined as:

- A software capability needed by the user to solve a problem or achieve an objective.
- A software capability that must be met or possessed by a system or system component to satisfy a contract, specification, standard, or other formally imposed documentation."[3]

What is Requirements Management?

Since requirements are things to which the system being built must conform, and conformance to some set of requirements defines the success or failure of projects, it makes sense to find out what the requirements are, write them down, organize them, and track them in the event they change.

Stated another way, ***Requirements Management is:***

- ***a systematic approach to eliciting, organizing, and documenting the requirements of the system, and***
- ***a process that establishes and maintains agreement between the customer and the project team on the changing requirements of the system.***

This definition is similar to Dorfman and Thayer's and the IEEE's "Software Requirements Engineering." Requirements Engineering includes elicitation*, analysis, specification, verification, and management[4]. All of these activities are incorporated in the definition of requirements management presented here and taught by Rational Software. The difference lies mainly in the choice of the word "management" rather than "engineering." Management is a more appropriate description of all the activities involved, and it accurately emphasizes the importance of tracking changes to maintain agreements between stakeholders and the project team.

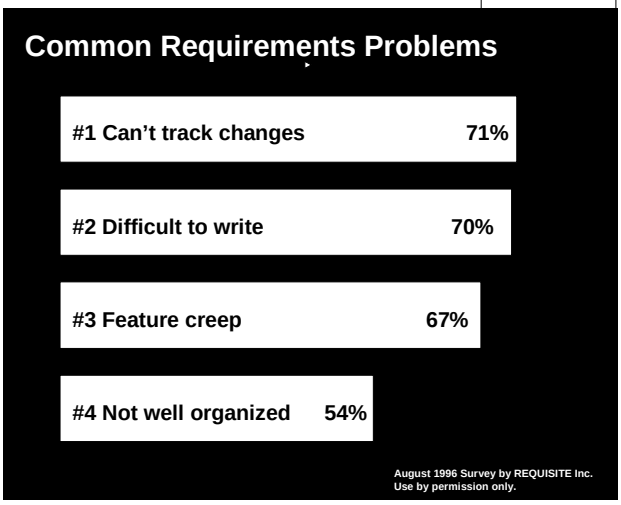
The Problems of Requirements Management

So what might be difficult about a process intended to ensure that a system conforms to the expectations set for it? When put into practice on real projects, difficulties come to light. Figure 1 displays the results of a 1996 survey of developers, managers, and quality assurance personnel. It shows the percentage of respondents who experienced the most frequently mentioned requirements-related problems.

A more comprehensive list of problems includes:

- Requirements are not always obvious and have many sources.
- Requirements are not always easy to express clearly in words.
- There are many different types of requirements at different levels of detail.
- The number of requirements can become unmanageable if not controlled.
- Requirements are related to one another and to other deliverables of the process in a variety of ways.
- Requirements have unique properties or property values. For example, they are neither equally important nor equally easy to meet.
- There are many interested and responsible parties, which means requirements need to be managed by cross-functional groups of people.
- Requirements change.
- Requirements can be time-sensitive.

Figure 1



When these problems are combined with inadequate requirements management and process skills, and the lack of easy-to-use tools, many teams despair of ever managing requirements well. Rational Software has developed the expertise to instruct teams in requirements management skills and process. In addition, Rational Requisite®Pro is an accessible tool for automating effective requirements management.

* For those unfamiliar with the term "elicitation," it is defined as the set of activities that teams employ to elicit or discover stakeholder requirements.

Requirements Management Skills

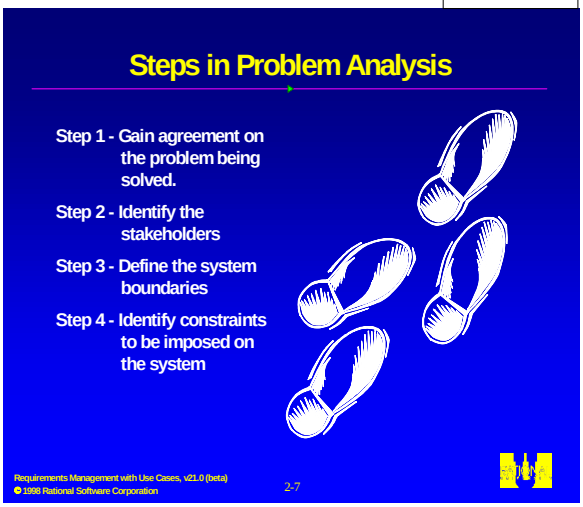
To resolve the problems mentioned above, Rational encourages the development of ***key skills***. These skills are presented below in what appears to be sequential order, but in an effective requirements management process they are applied continuously in varied order. Here they are presented in the sequence one would likely apply to the first iteration of a new project.

Key Skill 1: Problem Analysis

Problem analysis is conducted to understand business problems, target initial stakeholder needs, and propose high-level solutions. These acts of reasoning and analysis find "the problem behind the problem."

During problem analysis, agreement is gained on a statement of the real problems and the stakeholders are identified. Initial solution boundaries and constraints are defined from both technical and business perspectives. If appropriate, the business case for the project analyzes return on investment that is expected from the system.

Figure 2



Key Skill 2: Understanding Stakeholder Needs

Requirements have many sources. They may come from anyone with an interest in the outcome of the project. Customers, partners, end users, and domain experts are some sources of requirements. Management, project team members, business policies, and regulatory agencies can be others.

It is important to know how to determine who the sources should be, how to get access to those sources, and how to elicit information from them. The individuals who serve as primary sources for this information are referred to as "stakeholders" in the project.

If you are developing an information system to be used internally within your company, you may include people with end-user experience and business domain expertise in your development team. Very often you will start the discussions at a business-model level rather than at a system level. If you are developing a product to be sold to a marketplace, you may make extensive use of your marketing people to better understand the needs of customers in that market.

Techniques for eliciting requirements include interviews, brainstorming, conceptual prototyping, questionnaires, and competitive analysis. The result of requirements elicitation is a list of requests or needs that are described textually and graphically, and that have been given priority relative to one another.

Key Skill 3: Defining the System

To define the system means to translate and organize the understanding of stakeholder needs into a meaningful description* of the system to be built. Early in system definition, decisions are made on what constitutes a requirement, documentation format, language formality, degree of requirements, request priority and estimated effort, technical and management risks, and scope. Part of this activity may include early prototypes and design models directly related to the most important stakeholder requests.

---

* We use the word "description" rather than "document" to avoid the perceived limitation inherent in the common use of the latter. A description may be a written document, electronic file, a picture, or any other representation meant to communicate system requirements short of the system itself.

The outcome of system definition is a description of the system that is both natural language and graphical. Some suggested formats for the description are provided in later sections.

> **Principle 55**
> **Write Natural Language Before A More Formal Model**
>
> If you write the formal model first, the tendency will be to write natural language that describes the model instead of the solution system. Consider the following examples:
>
> TO MAKE A LONG DISTANCE CALL, THE USER SHOULD LIFT THE PHONE. THE SYSTEM SHALL RESPOND WITH A DIAL TONE. THE USER SHOULD DIAL A "9". THE SYSTEM SHALL RESPOND WITH A DISTINCTIVE DIAL TONE…
>
> THE SYSTEM CONSISTS OF FOUR STATES: IDLE, DIAL TONE, DISTINCTIVE DIAL TONE, AND CONNECTED. TO GET FROM THE IDLE STATE TO THE DIAL TONE STATE, LIFT THE PHONE. TO GET FROM THE DIAL TONE STATE TO THE DISTINCTIVE DIAL TONE STATE, DIAL A "9."
>
> Note that in the latter example, the text does not help the reader at all.
>
> **- Alan M. Davis, 201 Principles of Software Development, 1995**

Key Skill 4: Managing the Scope of the Project

The scope of a project is defined by its requirements. Managing project scope to fit the available resources (time, people, and money) is key to managing successful projects. Managing scope is a continuous activity that requires iterative or incremental development, which breaks project scope into smaller more manageable pieces.

Using requirement attributes, such as priority, effort, and risk, as the basis for negotiating the inclusion of a requirement is a particularly useful technique for managing scope. Focusing on the attributes rather than the requirements themselves helps desensitize negotiations that are otherwise contentious.

It is also helpful for team leaders to be trained in negotiation skills and for the project to have a champion in the organization, as well as on the customer side. Product/project champions should have the organizational power to refuse scope changes beyond the available resources or to expand resources to accommodate additional scope.

Key Skill 5: Refining the System Definition

With an agreed-upon high-level system definition and a fairly well understood initial scope, it is both possible and economical to invest resources in more refined system definitions. Refining the system definition includes two key considerations: developing more detailed descriptions of the high-level system definition, and verifying that the system will comply with stakeholder needs and behave as described.

The descriptions are often the critical reference materials for project teams. Descriptions are best done with the audience in mind. A common mistake is to represent what is complex to build with a complex definition, particularly when the audience may be unable or unwilling to invest the critical thinking necessary to gain agreement. This leads to difficulties in explaining the purpose of the system to people both inside and outside the project team. Instead, you may discover the need to produce different kinds of descriptions for different audiences. This paper includes suggested formats for detailed natural language, formal text, and graphical descriptions. Once the description format is established, refinement continues throughout the project lifecycle.

Key Skill 6: Managing Changing Requirements

No matter how carefully you define your requirements, they will change. In fact, some requirement change is desirable! It means that your team is engaging your stakeholders. Accommodating changing requirements is a measure of your team's stakeholder sensitivity and operational flexibility – team attributes that contribute to successful projects. Change is not the enemy, ***unmanaged*** change is.

A changed requirement means that more or less time has to be spent on implementing a particular feature, and a change to one requirement may have an impact on other requirements. Managing requirement change includes activities such as establishing a baseline, keeping track of the history of each requirement, determining

which dependencies are important to trace, establishing traceable relationships between related items, and maintaining version control. As Figure 3 illustrates, it is also important to establish a change control or approval process, requiring all proposed changes to be reviewed by designated team members. Sometimes this single channel of change control is called a Change Control Board (CCB).

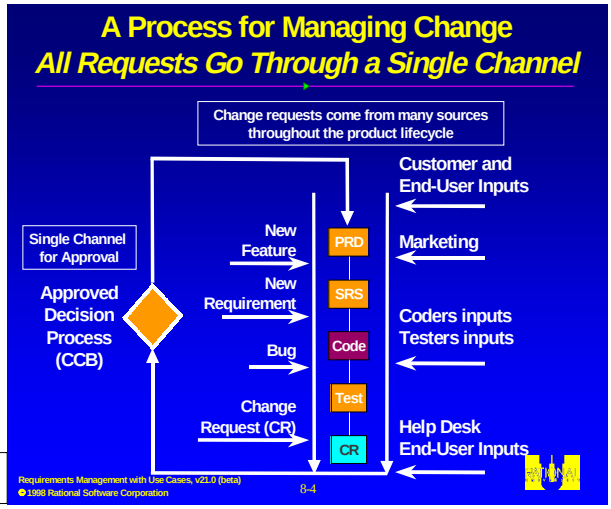


Figure 3

Important Requirements Concepts

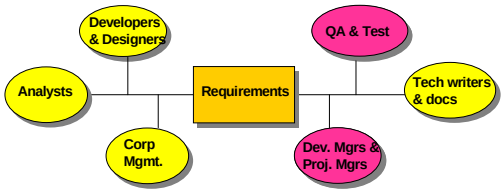
To apply requirements management skills to a project, certain requirements management concepts are useful for everyone on the project to understand. They include the following:

Requirement Type. The larger and more intricate the system, the more types of requirements appear. A requirement type is simply a class of requirements. By identifying types of requirements, teams can organize large numbers of requirements into meaningful and more manageable groups. Establishing different types of requirements in a project helps team members classify requests for changes and communicate more clearly.

Usually, one type of requirement can be broken down, or decomposed, into other types. Business rules and vision statements can be types of high-level requirements from which teams derive user needs, features, and product requirement types. Use cases and other forms of modeling drive **design requirements** that can be decomposed to **software requirements** and represented in analysis and design models. **Test requirements** are derived from the software requirements and decompose to specific test procedures. When there are hundreds, thousands, or even tens of thousands of instances of requirements in a given project, classifying requirements into types makes the project more manageable.

Cross-Functional Teams. Unlike other processes, such as testing or application modeling, which can be managed within a single business group, requirements management should involve everyone who can contribute their expertise to the development process. It should include people who represent the customer and the business expectations. Development managers, product managers, analysts, systems engineers, and even customers should participate. Requirements teams should also include those who create the system solution – engineers, architects, designers, programmers, technical writers, and other technical contributors. Testers and other QA personnel should be counted as important team members.

Figure 4



Often, the responsibility for authoring and maintaining a requirement type can be allocated by functional area, further contributing to better large project management. The cross-functional nature of requirements management is one of the more challenging aspects of the discipline.

Traceability. As implied in the description of requirement types, no single expression of a requirement stands alone. Stakeholder requests are related to the product features proposed to meet them. Product features are related to individual requirements that specify the features in terms of functional and non-functional behavior. Test cases are related to the requirements they verify and validate. Requirements may be dependent on other

requirements or they may be mutually exclusive. In order for teams to determine the impact of changes and feel confident that the system conforms to expectations, these traceability relationships must be understood, documented, and maintained. While traceability is one of the most difficult concepts to implement in requirements management, it is essential to accommodating change. Establishing clear requirement types and incorporating cross-functional participation can make traceability easier to implement and maintain.

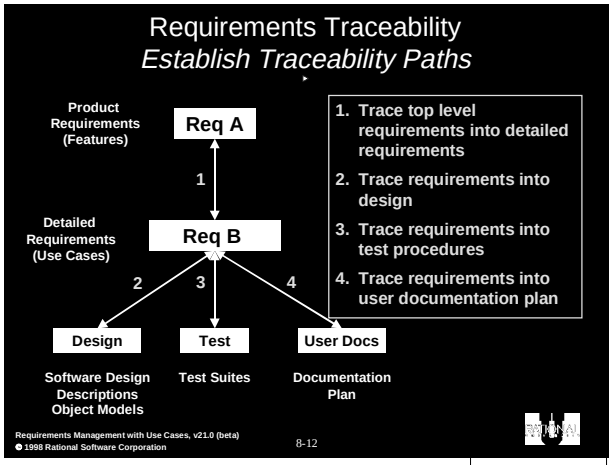


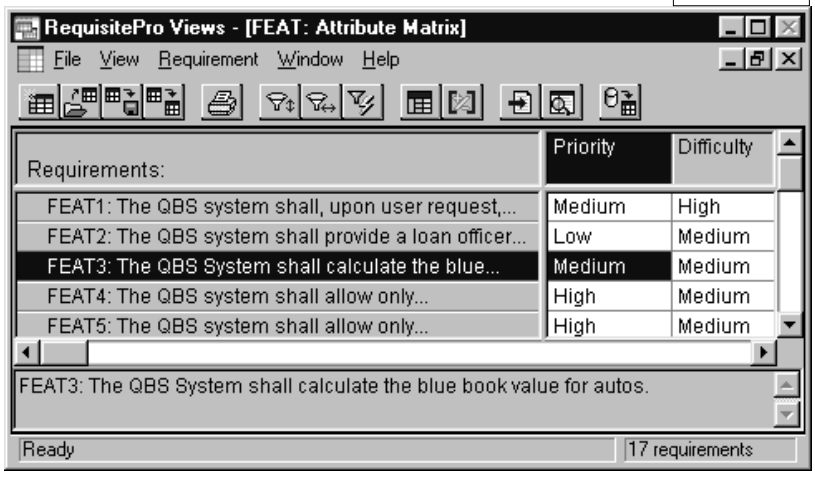
Figure 5

Multi-Dimensional Attributes. Each type of requirement has attributes, and each individual requirement has different attribute values. For example, requirements may be assigned priorities, identified by source and rationale, delegated to specific sub-teams within a functional area, given a degree-of-difficulty designation, or associated with a particular iteration of the system. To illustrate, Figure 6 displays attributes for a Feature Requirement Type from a Learning Project in the Rational RequisitePro requirements management tool. As implied by the title of the screen, the requirement type and attributes for each type are defined for the entire project, ensuring usage consistency across the team.

In Figure 7, instances of feature requirements are displayed for a specific project in Rational RequisitePro. Note that even without displaying the entire text for each requirement, we can learn a great deal about each requirement from its attribute values. In this case, its priority and difficulty – no doubt assigned by different members of the team – will help the team begin to scope the project to available resources and time, taking into account both stakeholder priorities and a very rough estimate of effort reflected in the difficulty attribute value. In more detailed types of requirements, the priority and effort attributes may have more specific values (e.g., estimated time, lines of code, etc.) with which to further refine scope. This multi-dimensional aspect of a requirement, compounded by different types of requirements – each with its own attributes – is essential to organizing large numbers of requirements and to managing the overall scope of the project.

Figure 6

Figure 7



Change History. Both individual requirements and a collection of requirements have histories that become meaningful over time.

Change is inevitable and desirable to keep pace with a changing environment and evolving technology. Recording the versions of project requirements enables team leaders to capture the reasons for changing the project, such as a new system release. Understanding that a collection of requirements may be associated with a particular version of software allows you to manage change incrementally, reducing risk and improving the probability of meeting milestones. As individual requirements evolve, it is important to understand their history: what changed, why, when, and even by whose authorization.

Putting Requirements Management to Work

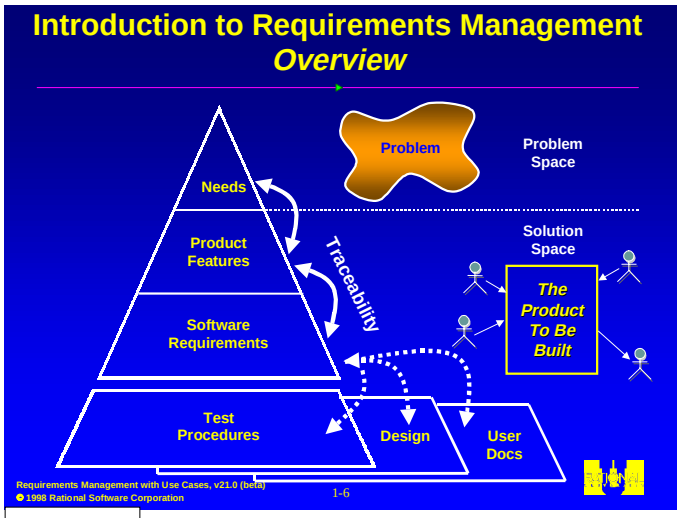


Figure 8

Requirements management employs the key skills and concepts presented above to identify and resolve the problems successfully.

To build a system that truly meets customers' needs, the project team must first define the problem to be solved by the system. Next, the team must identify stakeholders from whom business and user needs are elicited, described, and prioritized. From this set of high-level expectations or needs, a set of product or system features should be agreed upon.

Detailed software requirements should be written in such a form as can be understood by both the customers and the development team. We have found that using the language of the customer to describe these software requirements is most effective in gaining the understanding and agreement. These detailed software requirements are then used as input for the system design specifications as well as for test plans and procedures needed for implementation and validation. Software requirements should also drive the initial user documentation planning and design.

To facilitate this, the project team should:

- Agree on a common vocabulary for the project.
- Develop a vision of the system that describes the problem to be solved by the system, as well as its primary features.
- Elicit stakeholders needs in at least five important areas: functionality, usability, reliability, performance, and supportability.
- Determine what requirement types to use.
- Select attributes and values for each requirement type.
- Choose the formats in which requirements are described.
- Identify team members who will author, contribute to, or simply view one or more types of requirements.
- Decide what traceability is needed.
- Establish a procedure to propose, review, and resolve changes to requirements.
- Develop a mechanism to track requirement history.
- Create progress and status reports for team members and management.

A Few Words about Documents

The decision to describe requirements in documents deserves some thought. On one hand, writing is a widely accepted form of communication and, for most people, a natural thing to do. On the other hand, the goal of the project is to produce a system, not documents.

Common sense and experience teach that the decision is not whether but *how* to document requirements. Document templates provide a consistent format for requirements management. Rational's RequisitePro offers these templates and the additional feature of linking requirements within a document to a database containing all project requirements. This unique feature allows requirements to be documented naturally while being made more accessible and manageable in a relational database.

These essential requirements management activities are independent of industry, development methodology, or requirements tools. They are also flexible, enabling effective requirements management in the most rigorous and the most rapid application development environments.

Requirements Management Workflows

Requirements management can follow an infinite number of domain-specific paths. The following approach prescribes six detailed workflows that apply to each of the key requirements management skills but can be applied to any domain.*

Workflow: Problem Analysis

In the **Problem Analysis** workflow, the primary activity is ***vision development***. Output from this activity is a ***vision document*** that identifies the high-level user or customer view of the system to be built. The vision expresses initial requirements as key features the system must possess in order to solve the most critical problems. The ***system analyst*** has the primary role in this workflow. The system analyst should have problem domain expertise and an understanding of the problem, and should be able to describe a process that he or she believes will solve the problem. Active involvement from various project stakeholders is required.

To begin managing dependencies, features should be assigned attributes such as rationale, relative value or priority, and source of request. As the vision develops, the analyst identifies users and systems (the actors) of possible use cases. Actors are the first element of the use-case model, which will define the system's functional and non-functional technical requirements.

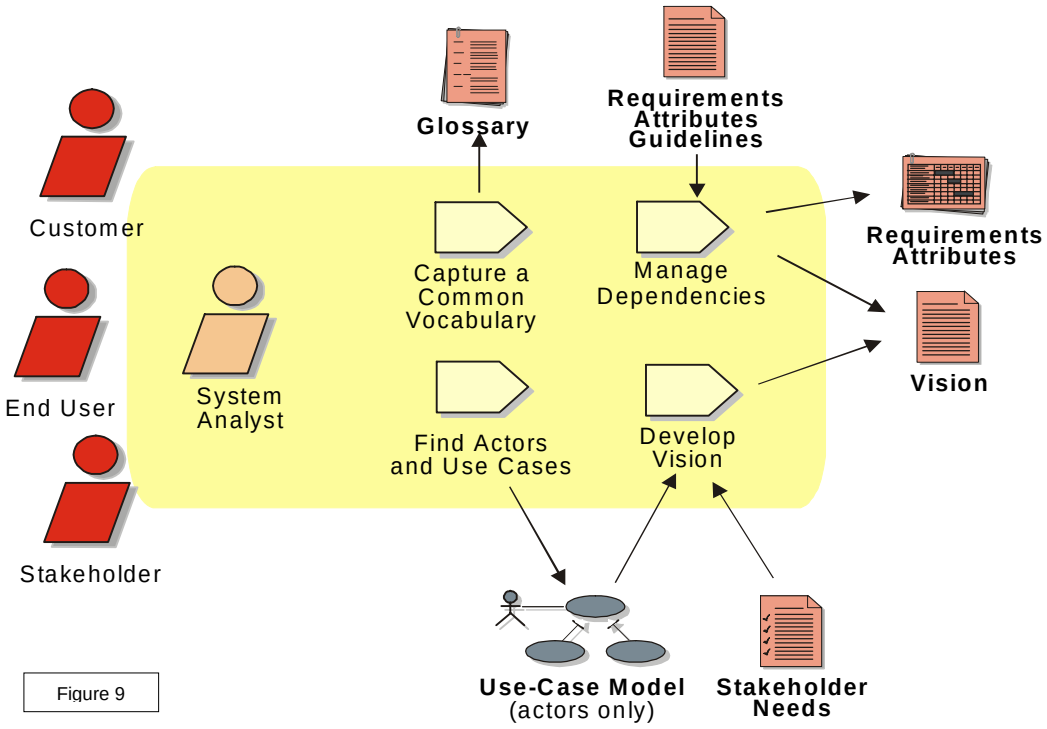


Figure 9

* The following workflow diagrams are from the Rational Unified Process Requirements Workflow. The workflows are expressed in terms of workers, activities and artifacts (input or output). The accompanying text in this paper describes each workflow briefly, in the hopes of stimulating your thoughts and interest in improving your requirements management process. More information on the Rational Unified Process can be found at www.rational.com.

The Activities in Problem Analysis

Initiation: One or more stakeholders who perceive a problem will initiate the workflow.

One or more system analysts in a development team conduct a session to help the initial stakeholders describe the problem they want to have solved. The elements of the vision document are organized in the following table:

Problem	define the problem
Affected Stakeholders	list the stakeholders affected by the problem
Impact	describe the impact of the problem
Successful Solution	list some key benefits of a successful solution

The problem statements succinctly explain the purpose of the project. Problem analysis stimulates further investigation into all stakeholder needs and the initial business case including compelling benefits and roughly estimated costs. In parallel with defining problem statements, you should also compile a glossary by keeping track of commonly used terms and agreeing on their definitions.

Use-Case Model Introduction

A use-case model consists of actors, use cases, and relations among them. *Actors* represent everything that must exchange information with the system, including what are typically called users. When an actor uses the system, the system performs a *use case*. A good use case is a sequence of transactions that yields a measurable result of value for an actor. The collection of use cases is the system's complete functionality.

Jacobson I., Christerson M., Jonsson P., Overgaard G., Object-Oriented Software Engineering – A Use Case Driven Approach, Addison Wesley – ACM Press, 1992

Problem analysis also identifies the main system actors. Actors are users of the system or of any other system that will exchange information with it. At this stage, problem analysis should briefly identify some obvious ways that the actors will interact with the system. Descriptions should be oriented towards business process rather than system behavior. For example, a budgeting program may allow one type of actor to "Create departmental budget," while another actor will be able to "Consolidate departmental budgets." The system analyst may later break them into additional use cases that align more meaningfully with specific system behavior. For example, "Create departmental budget" could result in system use cases such as "Import spreadsheet information" and "Create budget views."

The problem analysis session described above is often performed more than once, maybe with different stakeholders, and intermingled with internal development team sessions. The system analyst who conducted the meeting with the stakeholders will lead a session with members of the development team to envision a technical solution to the problems, derive features from the initial stakeholder inputs, and draft the vision description, the first definition of the system to be built. To facilitate understanding of the proposed solution among the initial stakeholders, the system analyst may use modeling tools or manual drawing techniques to complement the vision description.

The initiating stakeholders are consulted at multiple points to help refine the problem description and constrain the number and scope of possible solutions. Stakeholders and system analysts manage dependencies in this workflow by negotiating the priority of key features and gaining a general understanding of the resources and effort needed to develop them. While priority and effort/resource estimates inevitably change, managing dependencies early establishes an important pattern that continues throughout the development lifecycle. It is the essence of the scope management and an early predictor of project success.

After several drafts, the vision reaches a point when the team must decide whether to invest in additional requirements elicitation. By the same time, the business case approval process has been initiated separately. Although not addressed further in this paper, the business case describes:

- the context (product domain, market, and scope),
- the technical approach,
- the management approach (schedule, risk, objective measure of success),
- and the financial forecast.

Workflow: Understanding Stakeholder Needs

If the initial vision justifies additional investment, the **Understanding Stakeholder Needs** workflow begins in earnest. The key activity is *eliciting stakeholder needs.* The primary outputs are collections of *prioritized stakeholder needs*, which enable refinement of the vision document, as well as a better understanding of the requirements attributes. Also, during this workflow you may start discussing the system in terms of its use cases and actors. Another important output is an updated glossary of terms to facilitate common vocabulary among team members.

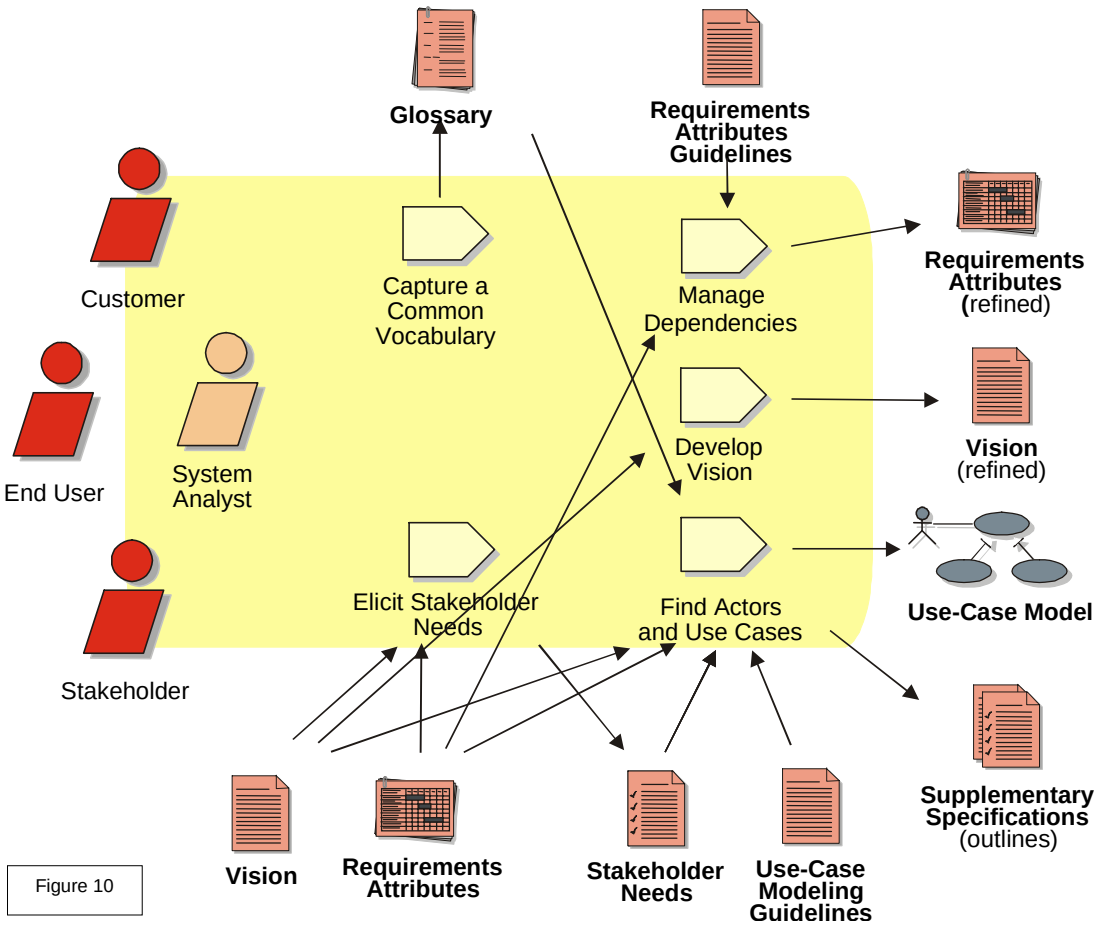


Figure 10

Activities in Understanding Stakeholder Needs

The system analyst and key stakeholders identify additional stakeholders and elicit their needs via interviews, workshops, storyboards, business process use cases, and other techniques. One or more system analysts facilitate these sessions. Requirements are among the most useful elicitation techniques. The process includes users, help-desk personnel, business owners, testers, and others who have a stake in the outcome of the proposed project. Stakeholder needs are often ambiguous, overlapping, and even extraneous. In addition to formal elicitation results, stakeholder needs may be expressed in well-formatted documents, defect and enhancement requests from databases, or e-mail and groupware threads. System analysts record, categorize, and prioritize stakeholder needs .

Understanding Stakeholder Needs: Where "Delighting Customers" Begins

Stakeholder needs are a type of proposed requirement captured as much as possible in the language and format of the submitting stakeholder. Unlike subsequent requirement types that are usually authored by process-educated and technically proficient project team members, stakeholder needs are often expressed poorly. They are duplicated or overlap. They can be expressed on anything from slips of paper to enhancement-request databases.

The analyst (or team representing the analyst role) must review them all, interpreting, grouping, perhaps retyping (without rewriting), and translating them into features in the vision description. Depending on the rigor applied in your development and the availability of tools, traceability between some or all stakeholder needs and features can be applied to help stakeholders understand how their needs were taken into account.

Demonstrating serious concern for eliciting and satisfying stakeholder needs by applying the Understanding Stakeholder Needs Workflow can be critical to establishing stakeholder confidence in your team's abilities.

Based on a better understanding of stakeholder needs, the system analysts in the development team refine the vision document, paying special attention to the product position statement. In two or three sentences, this statement establishes the compelling value of the project. The statement should include intended users, the problems it solves, the benefits it delivers, and the competitors it replaces. All team members should understand this project theme. System analysts also update the glossary to facilitate common understanding of terms.

Key stakeholders are consulted at multiple points to negotiate priority of new features derived from understanding stakeholder needs and gain a current understanding of resources and effort needed to develop them. As with problem analysis, managing dependencies in this workflow helps manage scope. It also establishes traceability between stakeholder needs and vision features so stakeholders can be sure their inputs were considered.

Workflow: Defining the System

The **Problem Analysis** workflow and the **Understanding Stakeholder Needs** workflow create early iterations of key system definitions, including the vision document, a first outline to the use-case model, and the requirements attributes. The **Defining the System** workflow *completes the description* of the system-level requirements with the addition of new actors, use cases, and supplementary specifications.

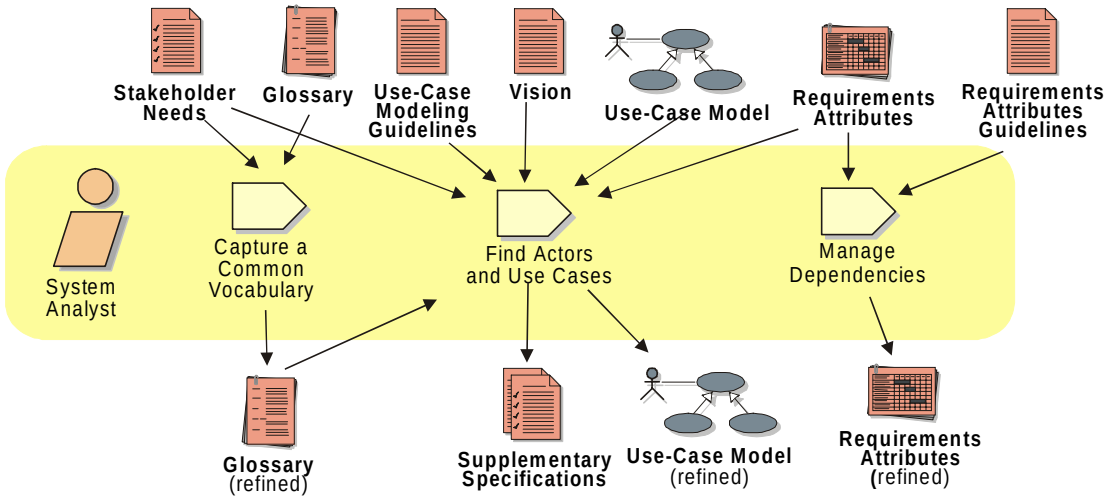


Figure 11

Activities in Defining the System

The glossary is updated to reflect current understanding about the terms used to describe features and the use-case model.

The system analyst uses the refined vision to derive and describe the use cases that elaborate on the system's expected behavior. The use-case model serves as a contract between the customer, the users, and the system developers. It defines expectations for system developers and helps customers and users to validate that the system will meet these expectations.

The system analyst describes requirements that do not fit well in use cases in a supplementary specification. Usability, reliability, performance, and supportability requirements often end up here. It should be noted that many non-functional requirements of these types are specific to a single use case. It is better for use case authors to place these requirements in the use case specification itself (see the **Refining the System** workflow), leaving the supplementary specification for *global* non-functional requirements.

In this workflow, the system analyst creates attributes for the supplementary requirements (such as priority and related use cases). In addition, the system analyst adds and updates attribute values for the initial and new use cases.

Finally, the system analyst manages dependencies by tracing important user needs and critical features to related use cases and supplementary specifications.

Workflow: Managing Scope

Identifying most actors, use cases, and supplementary specifications allows the system analyst to apply priority, effort, cost, and risk values to requirements attributes more accurately. This better understanding also enables the architect to identify the architecturally significant use cases.

Experience teaches that the keys to managing scope successfully are the well-considered attribute values assigned to stakeholder needs, use cases, and supplementary specifications; and regular, open, and honest interaction with representative stakeholders.

The *iteration plan*, developed in parallel by project and development management, first appears in the **Managing Scope** workflow. Also known as a development plan, the iteration plan defines the number and frequency of iterations planned for the release. The highest risk elements within scope should be planned for early iterations.

Other important outputs from the **Managing Scope** workflow include the initial iteration of the *software architecture document*[*] and a *revised vision* that reflects analysts and key stakeholders' increased understanding of system functionality and project resources.

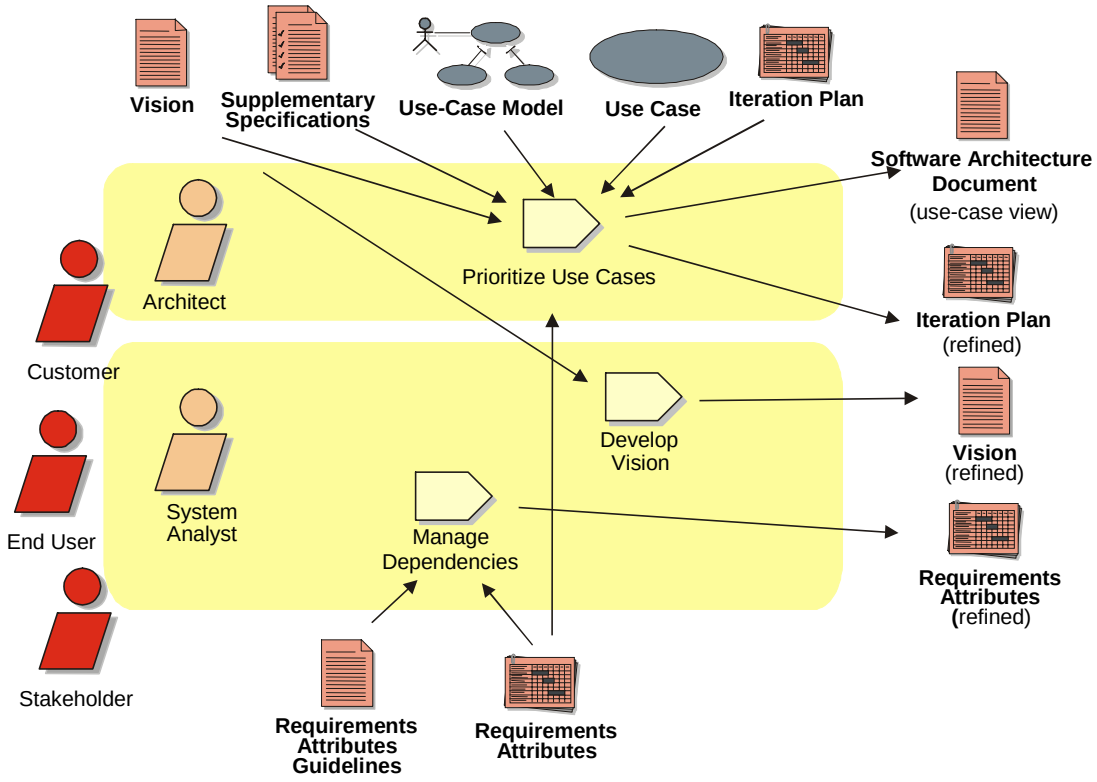


Figure 12

[*] Like the business case earlier and first issue of the iteration plan, the software architecture document is not an artifact of requirements management workflows, although it is related and is part of the Rational Unified Process. It is not the subject of this paper.

Activities in Managing Scope

Architects prioritize use cases for their risk coverage, architectural significance, and architectural coverage. While the system may be defined with many use-case and supplementary specification requirements, only a subset of use cases are usually critical to good system architecture. With prioritized use cases, architects refine the iteration or development plan and model a use-case view of the system architecture in tools such as Rational Rose.

System analysts manage dependencies by refining requirements attributes for features in the vision. They also refine requirements in use cases and supplementary specifications. System analysts ensure that appropriate traceability* exists for stakeholder needs, features, use-case requirements, and supplementary specification requirements.

System analysts negotiate revised project scope and vision with key stakeholders. This step is among the most important in the entire project. For the first time the **breadth** of knowledge about the proposed system is available to make serious commitments on requirements, project resources and delivery dates. At the same time, it must be understood that these requirements will change as the **depth** of knowledge increases. If dependencies have been managed in the previous three workflows, this step is much easier, and future changes will be easier.

Managing scope to match available resources is successful only if stakeholders and development team members view this step as a natural progression – not an ambush on users' expectations or an attempt to blackmail the organization for more time and money. This workflow will need to be repeated at major milestones in the project to assess whether new insight into the system and its problems requires change to the scope. While committed requirements, budgets, and deadlines are hard to change, an in-depth understanding of prioritized use cases, supplementary specifications, and early system iterations inevitably lead to scope reconsideration.

Once again, it is critical that the project team engages in habitual scope management before reaching the refinement stage. Representative stakeholders must understand and trust that their priorities and interests are taken seriously during increasingly difficult scope negotiations. By the time system requirements are refined, only important requirements remain to be negotiated or modified. Unless effective scope management habits have been established, your project may be doomed as a "death march" – a hopelessly over-scoped project moving inexorably towards delays and cost overruns.

* The term "appropriate traceability" is deliberate. See the inset text on Traceability later in this paper.

Workflow: Refining the System

The **Refining the System** workflow assumes that system-level use cases have been outlined and actors have been described, at least briefly. Through managing the project scope, the features in the vision have been re-prioritized and are now believed to be achievable by fairly firm budgets and dates. The output of this workflow is a more in-depth understanding of system functionality expressed in **detailed use cases**, **revised supplementary specifications**, and **early iterations of the system** itself[*].

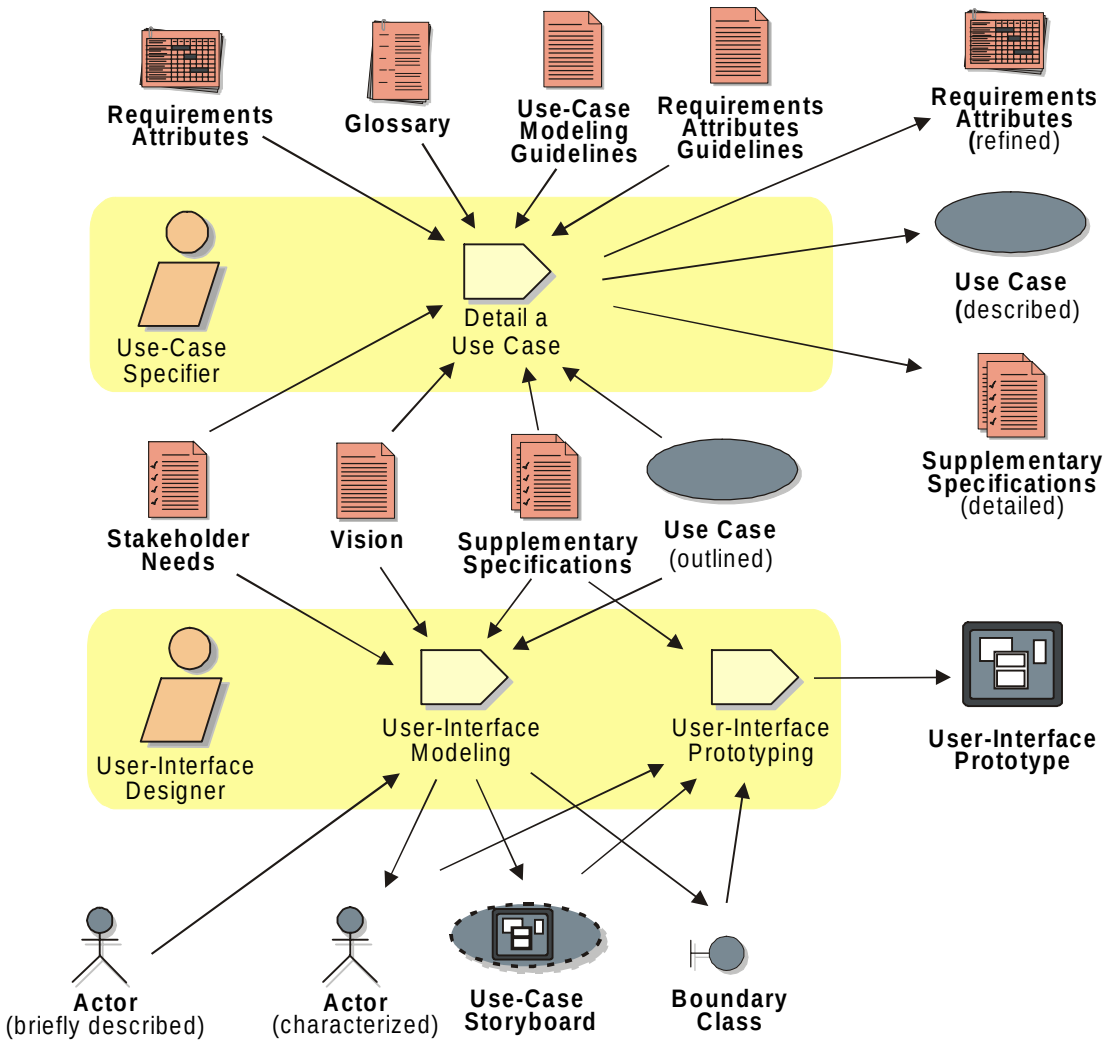


Figure 13

[*] Obviously, not all systems will have user interfaces and not all early iterations will include GUI elements. We use them here only as an example of an early iteration. Other examples includes prototypes, models, and storyboards.

Activities in Refining the System

The ***use-case specifier*** details the definition of the flow of events, pre- and post-conditions, and other textual properties of each use case. To minimize the effort and enhance the readability, it is advisable to use a standard document format or a use case specification template, to capture textual information about each use case. Creating well-thought-out use case specifications is critical to the quality of the system. This specification development requires a thorough understanding of the stakeholder needs and features related to the use case. It is desirable to have several members of the project team (such as software engineers) participate in creating the use cases.

In parallel, the use case specifier revises the supplementary specification with additional requirements that are not specific to the use case.

The ***user-interface designer*** models and prototypes the user interface of the system. This work is highly correlated to the evolution of the use cases.

The use-case specifier and the system analyst revise the effort, cost, risk, and other attribute values for each requirement that is understood better.

The result of this system refinement is submitted to another round of the **Managing Scope** workflow. Once you know more about the system, you may want to change the priorities.

Workflow: Managing Requirement Change

Like the **Managing Scope** workflow, the **Managing Requirements Change** workflow should be applied continuously. The output of this workflow can cause modification to every artifact, which requires effective communication with all project team members and stakeholders.

In this workflow we ***introduce additional artifacts*** that are affected by requirements workflows. Changes to requirements naturally affect the system models that represent them in the analysis and design workflows. Requirement changes also affect tests created to validate the proper implementation of the requirements[*]. Traceability relationships identified in the process of managing dependencies are the keys to understanding these impacts.

Another important concept for Managing Requirements Change Workflow is ***requirement history tracking***. By capturing the nature and rationale of requirements changes, reviewers (anyone on the software project team whose work is affected by the change) receive the information needed to respond to the change properly.

Traceability

Much is made of traceability in the requirements field. Many promote the virtue of tracing individual customer requirements to each related specification, test, model element, and ultimately source code files. Certainly, some traceability is the key to successful requirements change management.

Be forewarned, however, that all forms of traceability require an investment to set up and maintain during the life of the project. Like all investments, traceability has diminishing points of return depending on your specific situation. This paper emphasizes the value of tracing between types of requirements. This is a good place to start and can be automated by tools such as Rational's RequisitePro. We believe you will find some level of requirements traceability to be a good investment.

[*] As in earlier examples, these artifacts are part of the Rational Unified Process but are not the subjects of this paper.

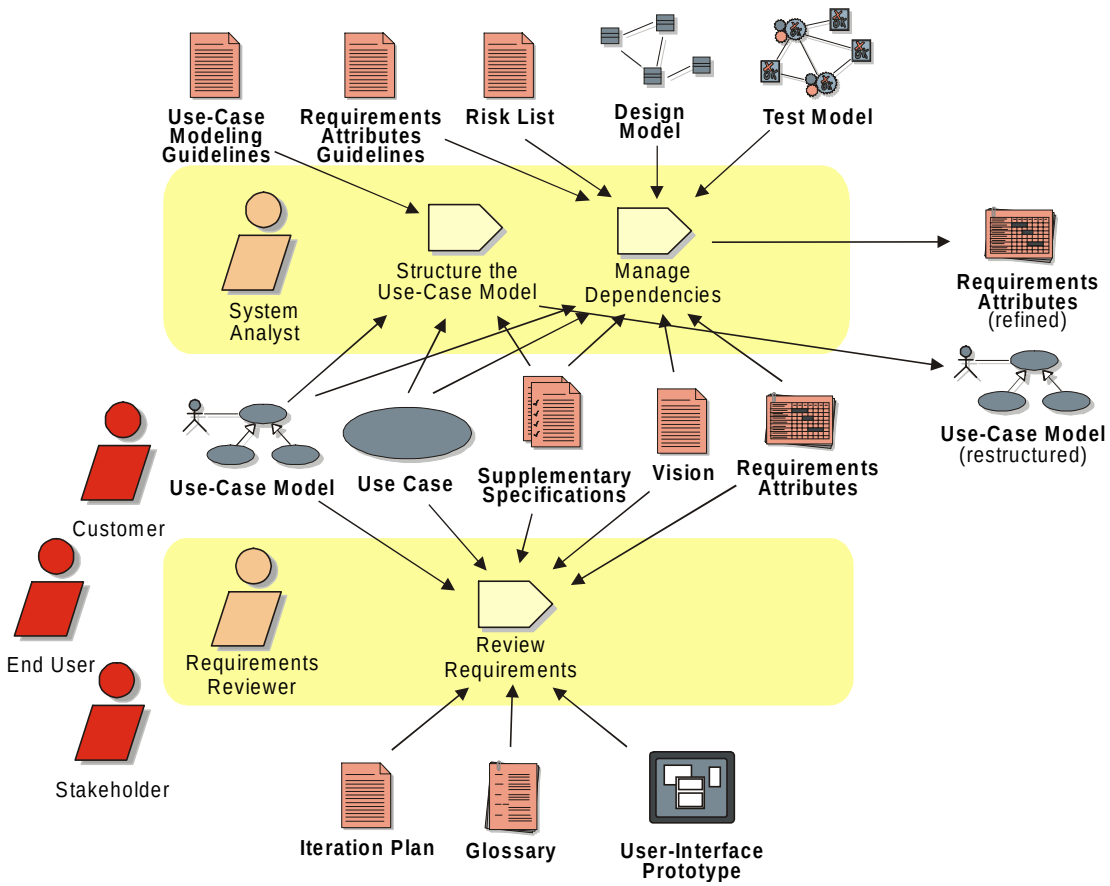


Figure 14

Activities in Managing Changing Requirements

Changes to requirements are initiated by any stakeholder or project team member for an infinite number of reasons.

The system analyst initiates a review activity, assimilating all change requests, and classifies them as:

- defects in implementation that do not affect requirements,
- modifications to existing requirements of some type, and
- new stakeholder needs or enhancement requests.

Once classified, the proposed changes to requirements are assigned attributes and values as described in other requirements workflows.

In reviewing changes, the system analyst presents proposed prioritized requirement changes to a ***change control board*** comprised of representative stakeholders and project team members. Scope modifications that exceed resources should be rejected or elevated to stakeholder representatives that are empowered to approve required changes to date and budget commitments.

The change control board approves or rejects changes to requirements.

The system analyst communicates requirements changes to requirements authors or makes changes directly to requirements in the vision, use cases, or supplementary specification documents.

The ***requirements reviewers*** (developers, testers, managers, and other project team members) evaluate the impact of changes to requirements on their work by reviewing requirement history. Finally, they implement the change and make appropriate changes to related requirements for which they have authority.

Summary

The need to manage requirements is not new. So, what makes the preceding information worth considering now?

First, if your projects are not regularly satisfying customers, meeting deadlines, and staying within budget, you have reason to reconsider your development approach. If in doing so, you determine that requirements-related problems are undermining your development efforts, you have reason to consider better requirements-management practices.

Second, the requirements management practices summarized in this paper embody the collective experience of thousands, and are the well-considered opinions of a number of individuals who have spent years working with customers in the field of requirements management*. We suggest that this overview of their contributions -- and the more thorough presentation of them made in Rational's Unified Process[5] -- represent a "best practice" in requirements management. You will not find more proven advice on requirements anywhere.

Finally, in the past two years Rational Software, a leader in the business of producing effective software development solutions, has taken on the challenge of requirements management and emerged with the tools to automate this difficult task. The chronic, pervasive problems of requirements management are solvable. And that, ultimately, may be the best reason to start practicing excellence in requirements management today.

* The authors would like to acknowledge the direct and indirect contributions of Rational Fellow Ivar Jacobson, and those of Dean Leffingwell, Dr. Alan Davis, Ed Yourdon, and Elemer Magaziner. Most importantly, we appreciate the Rational Software customers who have applied and improved these practices in hundreds of development projects.

End Notes

[1] *CHAOS*, The Standish Group International, Inc., Dennis, MA, 1994, 1997

[2] Computer Industry Daily, December 12, 1997

[3] Dorfman, M. and R. Thayer, Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1997 pp. 79

[4] Dorfman, M. and R. Thayer, Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1997 pp. 80

[5] Rational Unified Process™, Rational Software Corporation, Cupertino, CA, 1998

RATIONAL
SOFTWARE

Corporate Headquarters
18880 Homestead Road
Cupertino, CA 95014
Toll-free: 800-728-1212
Tel: 408-863-9900
Fax: 408-863-4120
E-mail: info@rational.com
Web: www.rational.com

Australia +61-2-9419-0100
Belgium +32-16-46-24-11
Brazil +55-11-829-7585
Canada 613-599-8581
France +33-1-30-12-09-50
Germany +49-89-628-38-0
India +91-80-553 8082/9864
Japan +81-3-5423-3611
Korea +82-2-556-9420
The Netherlands +31-23-569-4300
New Zealand +64-4-568-9591
South Africa +27-12-663-5677
Sweden +46-8-703-4530
Switzerland +41-1-445-36-00
Taiwan +886-2-720-1938
UK +44-1344-462-500

Lit No. TP505 Rev. 10/98. Subject to change without notice.