Transforming software and systems delivery
White paper
May 2008

Rational. software

IBM

Tips for writing good use cases.

James Heumann, Requirements Evangelist,
IBM Rational Software

Contents

Introduction

Writing good use cases is more of an art than a science. And as with any art, there are no absolute rules for creating your masterpieces. Ultimately, use cases are about clearly communicating detailed information to a very diverse audience and reaching the goal of creating successful development projects.

Ivar Jacobson invented use cases. Maria Ericsson worked with Jacobson and was coauthor of one of his books. Kurt Bittner and Ian Spence also wrote a popular book on use cases. These pioneers and many people at IBM who have years of experience working with clients to develop good use cases have contributed to the technologies described in this paper. Writing a good use case isn't easy, but, fortunately, our experience can be your guide. The concepts and principles assembled here represent the works of many people at IBM, and they form a foundation of proven best practices. Many of the tips referenced in this paper are part of the IBM Rational® Unified Process® (RUP®) methodology; others are new and have not made it into RUP.

Understanding what a use case is (and what it is not)

Use cases have become very popular for outlining the business logic of development projects. This popularity, unfortunately, can also result in confusion and varying opinions about what they are and how they should be structured and written. Without a standard style and set guidelines for writing use cases, use case writers within your organization will write use cases their own way, possibly confusing readers and jeopardizing projects as well as the desired result of a quality solution that provides immediate business value.

Highlights

A use case is the story of how the business or system and the user interact.

Use cases are part of the Object Management Group (OMG) Unified Modeling Language (UML) standard. This standard tells us what the parts of the use case diagrams mean—the stick figures, ovals and lines—and it gives us the definition of a use case. But it doesn't tell us how to structure or write one. So we're left to read books or articles (like this one), to try to figure out the right way.

So, what is the right way? The best way is the way that works for you—without straying too far from the definition of what a use case is:

> *A use case is the specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system (UML 2).*

A less formal definition might be: A use case is a list of steps that specifies how a user interacts with the business or system while keeping in mind the value that this interaction provides to the user or other stakeholders. Simpler still: A use case is the story of how the business or system and the user interact.

Use cases are formal requirements that clearly define the resultant value.

A use case is not to be confused with a user story, a software system requirement formulated as one or two sentences in the everyday language of the user. Use cases are formal requirements with context and structure that clearly define the resultant value.

This still leaves use case writers with a big job—figuring out the best way to actually write use cases, given the significant ambiguity built into the definition. This paper's goal is to help you make the right decisions so that you can successfully do that job.

Another type of use case—the business use case—is used to develop models that describe how a business will interact with its customers and other external parties and provide value to these entities. This paper does not address business use cases and focuses exclusively on system use cases.

Highlights

Use cases transform `shall` statements into groups that provide observable value and context, organized from a user perspective.

Use cases are software requirements that specify functionality

People often talk about requirements and use cases, implying that they are different things. Use cases are simply a different method for organizing functional requirements; they are still requirements. Traditionally, requirements capture a system's functionality by using a long list of `shall` statements, sometimes numbering in the thousands. The goal of creating use cases is to transform these many `shall` statements into smaller groups that provide observable value and context, organized from a user perspective.

Use cases are about behaviors (at least, the output) that are observable to the user. There are other types of requirements besides functional ones, usually called nonfunctional or supplementary requirements. These requirements are normally also specified as `shall` statements, but they contain different information than what's in the use cases. They describe the necessary qualities of the system: how fast it has to be, how reliable it should be, how secure it must be and other qualities. Use cases need to decscribe what the system should do. They should express the interactive steps a user should take to achieve a valuable result.

Use cases describe both functionality and results.

A use case describes functionality but it also must describe a result. This is important, and it is addressed in the first definition that says use cases provide "value to one or more actors or other stakeholders of the system." One of the common issues observed in organizations that are just getting started with use cases is that they don't understand this focus on value. And, as a result, they end up writing many smaller use cases that don't add up to the full story. They don't focus on interactions that drive value and are of interest to the actors involved in the use case.

Highlights

This approach can often be seen in its impact on the testing phase of a project. If the test cases are based on use cases that reflect unit tests, rather than user acceptance tests, there are probably too many and they are too low level.

A common mistake is to confuse requirements with design specifications.

As an example, one client IBM has engaged with had about 10 people working on a project that was scheduled to last about a year. Initially, the organization had approximately 150 use cases, because there was a lack of understanding of their purpose. Upon deeper examination, it was determined that many of their use cases were really design specifications, not requirements, and others were too focused on interactions so minor that they didn't show any value to actor or stakeholder.

An effort was made to take a step back and focus on revising their use cases. After removing inappropriate cases and combining others, the client ended up with fewer than 10 use cases, each focused on complementary and comprehensive parts of the overall vision and showing significant value to each actor or stakeholder. For this client, it was an "ah ha" moment when the company's devleopers saw how these new, bigger, more complete use cases would be both easier to understand and of greater use in designing, developing and testing its system.

Tip: Don't forget the diagram

Diagrams provide a visual reference for the use case.

Use cases are represented graphically as ovals on use case diagrams, and they are also expressed as textual specifications (see figures 1 and 2). The text is definitely the essence of a use case model, and you can certainly benefit from writing the text without drawing the diagrams. However, the diagram can help communicate the requirements at a high level. It also helps stakeholders see what is in scope and what is out of scope. Actors are outside the system being developed, and use cases are inside. This is especially important for the use case survey diagram, which shows all actors and use cases on a single diagram.

Highlights

People are the most important use case element.

Tip: Human actors are the most important element

The stick figures employed in use case diagrams are referred to as *actors*. They indicate any person (or role), other system or perhaps even a device that is outside the system being built but that interacts with it. It is the human actors that represent the most interesting and difficult interactions for most systems, and the human/system interactions are where the real value of the use case comes into perspective. The functionality put into a use case will help actors do their job or task better—in a significant way.

Unless the system that you are building has a heavy dependency on other systems or devices, your time and effort are much better spent addressing the needs of human actors.

Tip: Go with the flow

Good use cases are not just paragraphs of clear and concise text. They have a structure that defines how the parts of the use case fit together. And they can be structured in many ways—referred to as a *use case style*.

Establishing a use case style makes writing and reading them faster and easier.

A common structural technique when creating use cases is to have one big list of steps from start to finish. This approach can cause issues, as there are conditional behavior and/or error processes that benefit by modularizing—breaking out each logical piece for a specific part of the use case. Modularizing makes it easier to write and to read, but only if the pieces can be fit together to depict the whole picture.

Highlights

Use cases should have a single main flow and multiple alternative flows.

A use case should have a single main flow and multiple alternative flows. The main flow (see figure 1) explains what happens when everything goes right and the use case accomplishes its goal. This is sometimes called the "happy path" scenario, and occurs most of the time. Sometimes use case writers try to have multiple main flows, but this can dilute the understandability of the use case because it makes the reader work harder to understand the true goal. So the most common scenario should be the anchor of the process, serving as a single main flow.

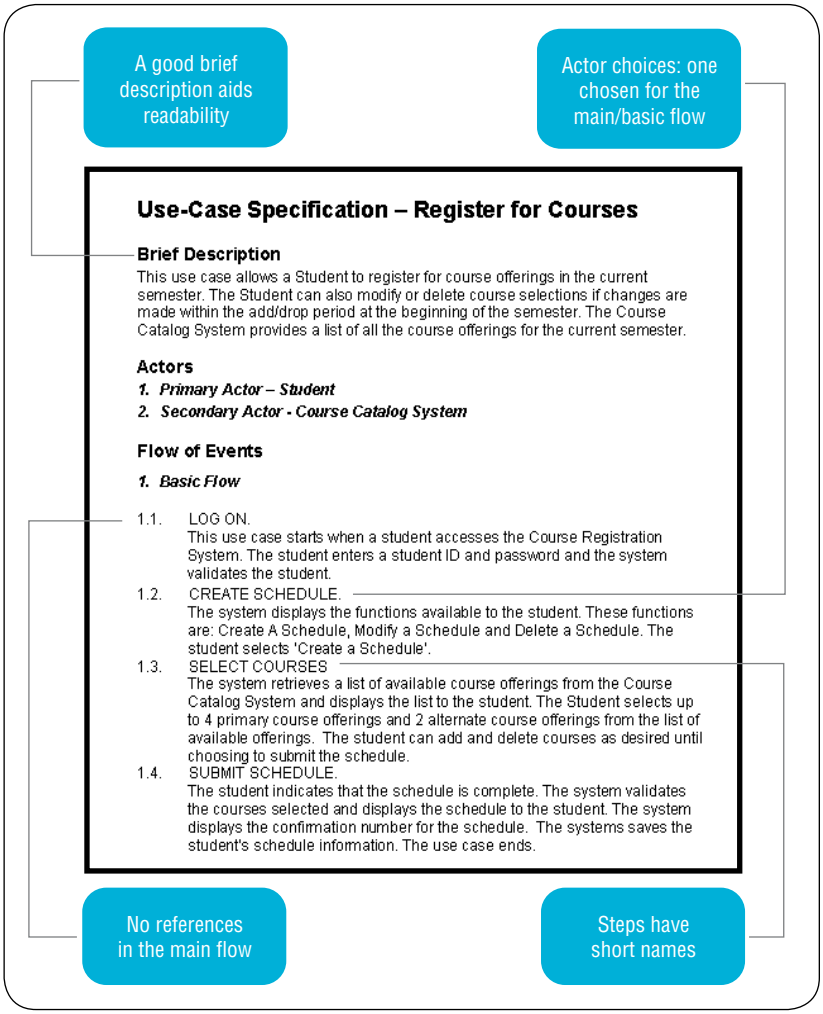

Figure 1: A well-written use case that tells how a student registers for courses.

Highlights

Alternative flows explain deviation from the main flow.

Alternative flows (see figure 2) explain what happens when something causes a deviation from the main flow. These deviations may or may not be labeled as exceptions or errors. Either way, they do not occur as often or are not as important as the main flow. In a use case document, each type of flow is listed in its own section. Sometimes alternative flows are broken into more detailed categories such as user errors and exceptions. Note that low-level error codes are beyond the scope of alternative flows.

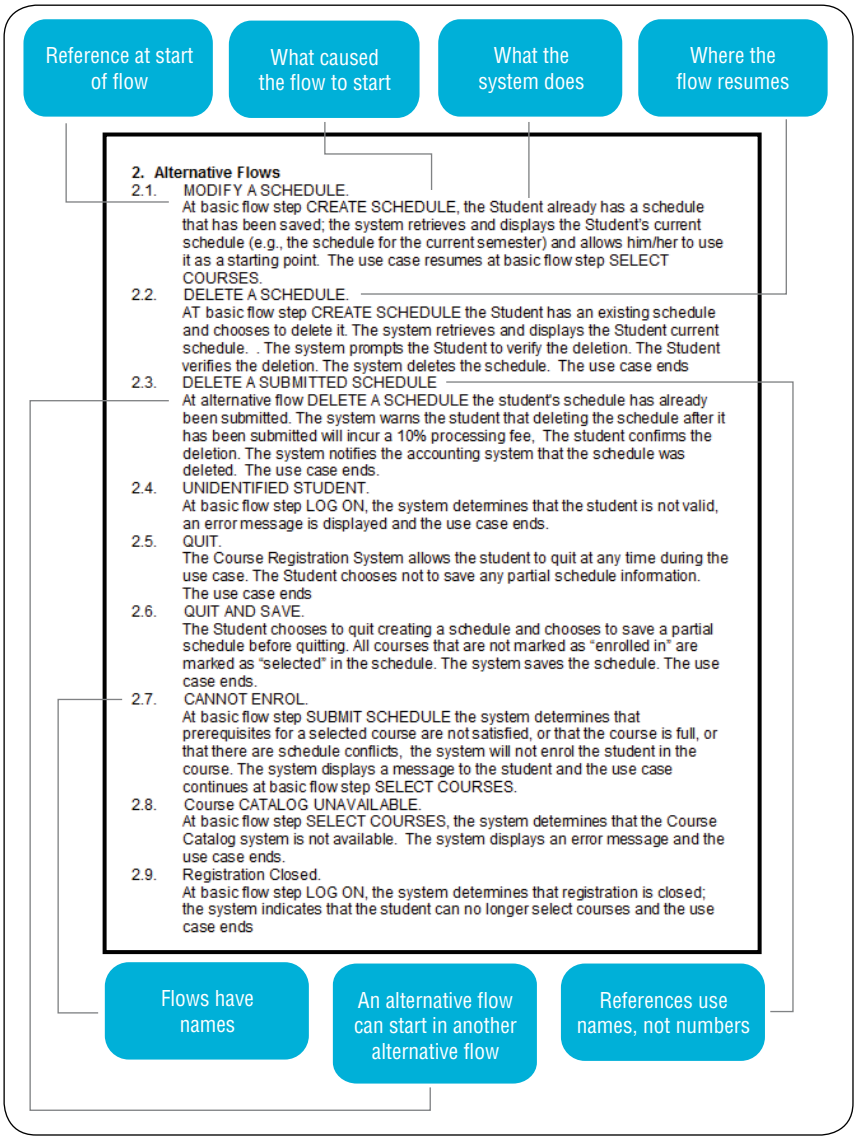

Figure 2: Alternative flows describe possible outcomes, when they start and when they end.

Highlights

Use cases are not design documents; they are requirements documents.

A use case with a single flow remains unfinished, because a good use case will always have multiple flows of events. Remember that use cases are supposed to explain to customers, users and development teams the functional requirements in enough detail to enable those users to either approve them or employ the use case to design the system. Understand that a use case is not a design document on how the requirements are to be implemented; rather, it's an indication of what needs to be built. Thus, a single flow is not enough to show the needed detail, or it may mean that there are many other smaller use cases that could be combined into others as alternative flows.

References allow users to reconstruct the whole story

It's best practice to modularize a use case by splitting it into multiple flows. But to be able to tell a whole story, you must also have some way to put the pieces back together again to illustrate the specific end-to-end scenarios. This is accomplished by using references, which essentially define the beginning and the end of an alternative flow by referring to a particular step in the main flow or another alternative flow (See figure 3). Making a use case easily understandable requires a consistent technique for using references.

References define the beginning and the end of an alternative flow.

Tip: Put references in the alternative flows, not the main flows

The main flow illustrates what happens when everything in the use case goes right and the actor succeeds in his goal. It includes a time-ordered set of steps (events) that explains what the system does and what the actor does to accomplish the use case. References are simple statements about where a flow starts and how and where it ends.

It is recommended that you not use references in the main flow, which would take the user to an alternative flow or to another use case if a specific condition occurs. Focusing on the happy path keeps the flow simple and easy to read, and it clearly communicates what happens when the use case succeeds. Some styles of use cases do use references in the main flow, which often show up as `if-then` statements and branch to many other places in the use case or outside of it. But this technique can be confusing, and it is recommended that all references occur in alternative flows.

Highlights

Alternative flows contain the references. (Alternative flows explain what happens when errors occur or when expected outcomes occur, yet they are not as important as the main flow.) This is where the importance of references comes in. Every alternative flow will start somewhere else in the use case. It may branch off of one of the steps in the main flow or extend from another alternative flow. And when an alternative flow ends, the use case will usually continue somewhere else.

References indicate what caused an alternative flow to start and what the system does in response.

Alternative flows explain what caused them to start and what the system does in response to the alternative flow. Not only do alternative flows contain references for where they start and where they resume, they also say what caused them to start and what the system does in response. Looking downstream in the development process, this information will be useful in writing test cases.

Tip: Scenarios tell the full story

Actors can't execute an alternative flow, but they can execute a scenario. The concept of a scenario can be confusing because people define scenarios in so many different ways. At IBM, we define them simply as the paths through a use case or the set of steps and flows that an actor might take—start to finish—through a use case to provide a result. Given the many possible alternative flows, there will be many scenarios, with many results—both positive and negative. The process of walking through scenarios and identifying these negative results provides great platforms for stakeholder conversations as teams try to satisfy the business requirements across all possible scenarios. Keep in mind that the complete set of scenarios tells the full story of the use case. It is this full set of scenarios that will be used by the designers and tester as the basis of their work.

Tip: Be careful with `if` statements

Readability can suffer if there are `if` statements in a flow, because it usually means multiple requirements.

IT professionals, particularly those with programming backgrounds, are familiar with `if` statements, and you often see such statements used in use cases. An `if` in a programming language specifies conditional behavior, and that's usually true in use cases as well. Problems arise when there are one or more `if` statements in a flow, because it usually means that you are specifying multiple requirements. This can be a negative for readability—it can be hard to follow the flow of multiple `if` statements in text. It's also a negative for designing and testing the system. It's best for both the reader and the project if you break each `if` into its own flow. It makes for more flows, but the trade-off is worth it.

Highlights

Selecting a single choice for an actor, and creating alternative flows for other choices, can simplify the main flow.

Tip: Specify actor choices

Actors often make choices in use cases. Consider the example of a university registration system being defined in a "Register for Courses" use case. In this example, a main flow step might ask the actor to: 1) create a schedule, 2) modify a schedule and 3) delete a schedule. Often use case writers will try to use `if` statements to show actors making choices, but for reasons indicated earlier, this may not be the best idea. A better alternative is to have all actor choices listed at the appropriate point and have one selected option addressed within the current flow and the others dealt with in alternative flows. This technique keeps each flow simple and reduces branching.

Tip: Kick out the CRUD

Often we see writers address actor choices by creating separate use cases for each choice. It's a tempting solution but also a dangerous one. The danger is the need to manage the exponential complexity that is often created, resulting in unnecessarily repetitive behaviors and, ultimately, use cases with minimal value.

Instead, try to focus on the thing the actor really wants to do, which is usually not about the actions of create, read, update and delete (CRUD).

In our university registration system example, there are the choices for 1) create a schedule, 2) modify a schedule and 3) delete a schedule. All are in the "Register for Courses" use case because the student doesn't really care about creating and modifying schedules; the student only wants to register. Taking the easy way out and listing all CRUD activities results in three or four times as many use cases as necessary and can quickly make the process complicated and hard to manage.

For an example of a use case model with a lot of CRUD, look to the example below from a project to update the database administration for a local government shelter program. If we take out all of the CRUD-type activities, what are the actual benefits to the business derived from all the effort to define this specific model?

Highlights

A poorly written use case has too many create, read, update and delete activities, reducing clarity.

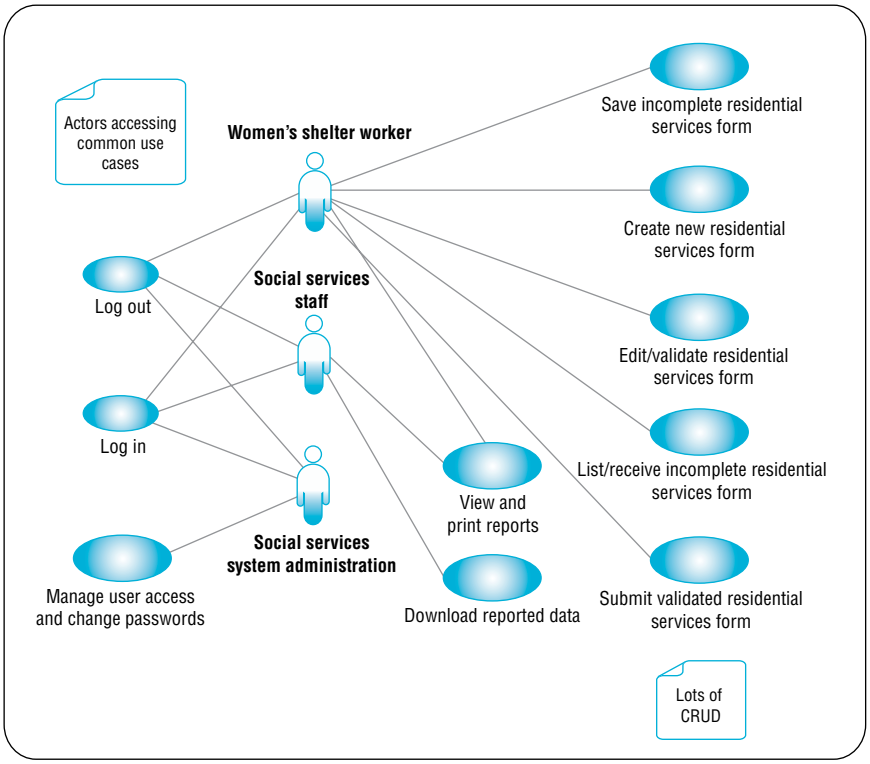

Figure 3: A poorly written use case has too many CRUD-type activities included.

Highlights

Removing CRUD creates a simpler document.

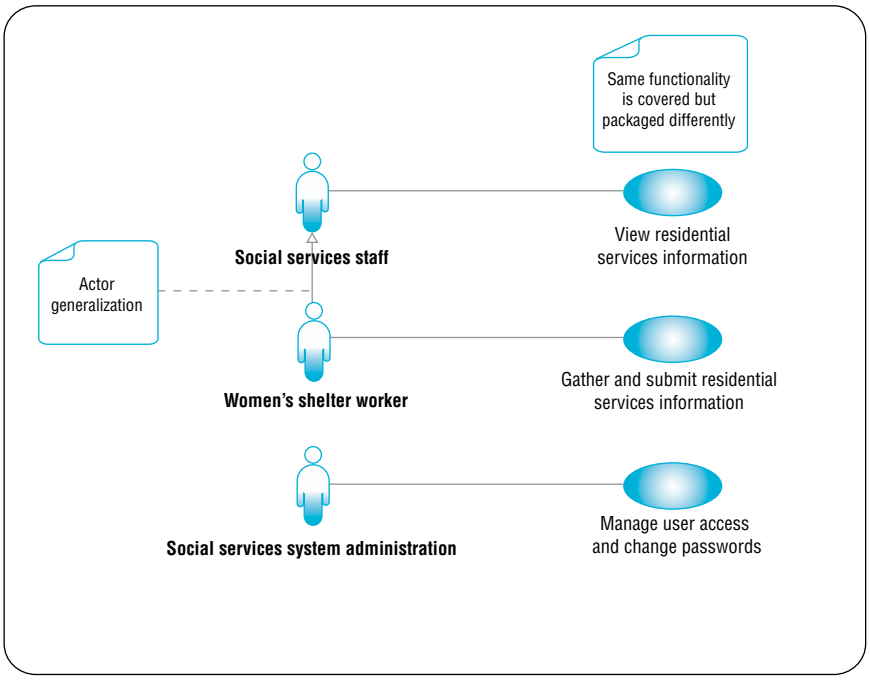

Figure 4: A well-written use case clearly indicates paths and value.

Sequencing the events in a flow is not always necessary to achieve clarity.

Tip: The sequence of events can be optional

We say that the steps in a use case are usually time ordered, but it should be considered whether the order of the steps is a requirement. For example, does the actor have to complete step four prior to completing step five? Usually the answer will be yes, but not always. It's a very simple thing, when developing the text of a use case, to clarify any temporal constraints and add comments such as `This step can occur in any order` or `This step can occur at any time before this step` to a line. The point is, you don't want to constrain the designers of the system by having implicit temporal requirements when they are not necessary.

Highlights

Don't forget you're writing use cases for the end user.

Tip: Use the correct level of detail

One of the most common questions is, "How much detail goes in a use case?" Remember, use cases are requirements. They aren't design documents, and they're certainly not user interface designs. One should never reference user interface elements, such as home pages, login screens or click buttons, in a use case.

Similarly, architectural details should be avoided. For instance, in the course registration example, a statement like `the schedule is saved in the SQL Server database` should not go in a use case. The part about saving the schedule is appropriate, but where it gets saved is an architectural detail that would have to be modified later if the database administrator decides that the schedule should be be saved in a different database.

Tip: Put yourself in the actor's shoes

Use cases are different than using `shall` statements. Since they do specify time-ordered steps, they create more context than stand-alone `shall` statements, which are not temporal. Because of this, they are more reflective of the intended user experience. Keep this in mind. If use cases are requirements (and they are), the designers and developers will be constrained by what they say. Remember this, take pity on the actor and write your use cases in a way that makes the system as easy as possible for the user.

Highlights

Agreement on use case structure and processes is essential to achieving quality and consistency.

Conclusion

When written well and with a consistent approach, use cases can be a great way to specify requirements in a way that is understandable to users and stakeholders and, at the same time, directly usable by the development team. But quality and consistency can only be achieved if there is agreement on use case structure and processes. So here's a final tip: Document the decisions you and your organization make about how to write good use cases. Create a use case style guidelines document or presentation, provide writers with plenty of examples of good use cases and then enforce the established guidelines.

Just as all of the effort put into a use case will (hopefully) pay off with a successful development project, the effort put into establishing standard practices and following these tips can result in better use cases that are valued by everyone involved in the process.

For more information

To learn more about how to write effective use cases and how IBM Rational software can help, please contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/rational/offerings/irm

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
05-08

Recommended reading:

Cockburn, Alistar, *Writing Effective Use Cases*, Addison-Wesley Professional, 2000.

Bittner, Kurt and Spence, Ian, *Use Case Modeling*, Addison-Wesley Professional, 2002.

Leffingwell, Dean and Widrig, Don, *Managing Software Requirements: A Use Case Approach*, Addison-Wesley Professional, 2003.

Contributors to this paper were Robin Bater, Yan Zhuo and Bruce Baron.

RAW14023-USEN-00