A technical discussion on defining requirements for legacy system re-engineering
July, 2003

Rational. software

IBM

Requirements Discovery for Legacy Systems

James S. Heumann
Requirements Evangelist
Rational Software
IBM Software Group

Executive Summary

Just as with “green field” development, legacy transformation projects must identify and manage functional requirements in order to succeed. On the surface this might seem unnecessary: is the legacy system itself not an adequate specification for the new solution? But “requirements on the hoof” in the form of an existing system are much less useful to designers, developers, testers, project management, users and business partners than an agreed-upon set of managed requirements, especially on what is likely to be a multi-year effort.

Particularly for large systems, it is essential to have an intermediate representation of the functionality in the form of managed requirements, which serves to describe and verify the behavior of the new system. Requirements derived from the current system are also essential to ensure that all important features are implemented.

Often, legacy transformation projects evolve from one approach to another. For example, a business may wish to rapidly port an existing system to a new hardware and operating system platform, then web-enable it, and then re-architect and gradually replace the original code base piece by piece with new software, new object-oriented modules, and new middleware.

Managed requirements greatly reduce risk on legacy transformation projects by providing a means to cope with this continuous change in the project scope, features, and priorities. By providing a common reference point for communication, requirements further support effective scheduling and parallel development.

Even on comparatively straightforward projects like web-enabling a legacy application, requirements will help a development team or systems integrator understand much more quickly what the application should do. Requirements also mitigate the risk of scope creep: after all, if you don’t know what you’re building, how do you know when you’re done?

This paper discusses a proven and efficient strategy for deriving requirements for legacy transformation projects by “discovering” them in the existing system and

describing them in the form of use cases.

About IBM Rational Software

IBM Rational, formerly an independent company and now one of the IBM® software brands, offers a comprehensive software development solution based on the three imperatives above. The IBM Rational platform combines software engineering best practices, market-leading tools, and expert professional services, all of which drive rapid and continuous improvement in software development capability for on demand businesses.

In addition, IBM Rational offers more than 20 years experience in promoting and delivering integrated and open software systems, both of which are key characteristics of the on demand operating environment:

Integrated — IBM Rational has contributed considerable thought leadership and expertise in the areas of Service-Oriented Architecture (SOA), enterprise and software architecture, and heterogeneous platform support.

Open — IBM Rational has a long history in developing and supporting the goals of open computing. This includes development of the Unified Modeling Language (UML), now a standard for modeling applications, database design, and business processes. IBM Rational has promoted and participated in the development of a wide variety of open computing standards. It offers support for major programming languages and operating platforms, and it provides an extensive set of application programming interfaces for third-party tool interoperation.

Thousands of companies around the world have realized the benefits of the approach advocated by IBM Rational. Their processes are results-oriented, the

Contents

artifacts they produce are well-designed and reusable, and they are working at higher levels of capability now required by the on demand era.

Introduction

Today's state-of-the-art business solutions are tomorrow's legacy systems. Any production application—whatever its age, complexity, platform, language or database management system—can be considered a legacy system, depending on an organization's current needs and future plans.

While it has no standard definition, the term legacy is most commonly applied to systems that:

- *Are difficult or risky to integrate, modify or extend in the face of changing business conditions;*
- *Entail high costs to maintain the software and/or the production environment;*
- *Cause dissatisfaction among end users due to poor performance, lack of integration, outdated interfaces, etc.*

Legacy systems embody significant assets that are worth reusing in whole or in part. Many legacy systems represent years of accumulated business experience and continue to support critical, day-to-day business processes. Nevertheless, time spent maintaining outmoded applications drains scarce resources that could be applied towards new, strategic initiatives.

Whatever modifications, integrations and/or extensions are planned, changes to legacy systems entail significant risk. Legacy transformation projects experience cost and schedule overruns with daunting frequency. Scope creep is endemic on in-house efforts, while COTS-based replacements regularly fail to meet business needs and user expectations. Not even automated re-engineering of a legacy code base ensures project success.

Assessing Current Value

A preliminary step in any legacy system project is to assess what parts of the legacy asset are worth reusing. Is some of the code still usable, such as the business rules? Does the design of the system mesh with the current enterprise architecture? Should the database management system remain intact? Should

Any legacy system can be leveraged as a source of information from which to derive requirements.

the business processes executed by the system be re-engineered and/or extended with new features and functions? And how will any proposed changes affect integration points with other enterprise systems?

The older the system, the more difficult it often is to sort out what parts can be cost-effectively reused. Outdated specifications, undocumented changes and numerous patches and bug fixes can create a tangled web. But even in worst-case scenarios, any legacy system can be effectively leveraged in at least one manner—as a point of comparison and a source of information from which to derive user and business requirements for transforming the existing environment.

Approaches to legacy transformation

Organizations have a wide range of options for leveraging legacy assets including extension, transformation, integration and migration.

Organizations have a wide range of short- and longer-term options for leveraging legacy assets as they move toward more strategic solutions. Outmoded applications can be extended, for example, or wrapped in more user-friendly interfaces. A common interim approach is to add client/server middleware and web GUIs to legacy environments, keeping the business logic and data intact while modernizing access for end users. Or the legacy system may be replaced, often gradually, with a new custom or COTS solution that offers equivalent functionality.

There are also many ways to define and categorize approaches to legacy transformation. For example, IBM differentiates between legacy extension, where a new GUI is developed to access legacy data and business rules; versus legacy transformation, which is the process of modifying the form, design, and/or function of the legacy application. Other sources refer to "migration," where the legacy system is adapted to new hardware, operating system, and/or middleware platforms.

Any or all of these approaches, or a combination of them, may be an appropriate strategy at a given point in time. In general, one or more of the following system elements are likely to be affected when legacy systems are altered:

Hardware:

- *Production platform (mainframe to client/server)*
- *Network topology (token ring to a more modern configuration such as hub-and-spoke)*
- *Storage devices (outmoded disk drives to RAID devices)*
- *Physical/user interface (character-based "green screen" to web-based GUI)*

Software:

- *Programming language (COBOL to Java™)*
- *Database management system (IMS to DB2®)*
- *Middleware (COM to Enterprise JavaBeans)*
- *Software architecture (move from remote procedure calls to Web Services, add new classes, rebuild modules, extend utilities, etc.)*
- *Business logic (time-driven to customer-driven)*
- *User interface (character-based to GUI)*

Establishing a Vision

Having assessed the value of a legacy asset, and determined many organizations find it beneficial to create a "vision document." Vision documents outline a plan for moving the project forward. They typically describe:

- *A big-picture description of the approach for transforming the legacy asset; for example, making key components web-accessible, followed by middleware updates, followed by moving the application to a strategic hardware and operating system platform, etc.*
- *The goal for the next stage in its transformation; e.g., porting the system from VMS/VAX to Linux on an IBM iSeries™ server.*

A vision document includes a description of the approach for transforming the legacy asset, along with goals for the next stage of the project and the delineation of project level requirements.

A vision document is a good place to define requirements at the project level to guide the overall effort. These requirements are mostly non-functional and goal-directed. For example:

- *"The project shall re-engineer the data to reduce redundancy wherever possible."*
- *"The character-based screens of the current system shall be reproduced as exactly as possible within the constraints of the new graphical interface."*
- *"The system shall use DB2 for database management."*
- *"The new system shall operate 20% faster than the existing system."*

The starting point for defining functional requirements is to describe what currently exists.

If the functional capabilities of the legacy environment will change over time, descriptions of these new "feature" requirements should also be added to the vision document.

Discovering What You Have

The basic premise of re-engineering a legacy application is to reproduce in some manner the useful capabilities of the existing system. Therefore, the starting point for defining functional requirements is to describe what currently exists.

But understanding what an existing system actually does can be a real challenge. Supporting artifacts like requirements and design documents, program specifications, or comments in the code, may be nonexistent or so old that they are no longer relevant to the current system.

There are a number of shortcuts that can help development teams "reverse engineer" the functional requirements for a legacy system. Which options are most viable may depend in large part on how familiar team members are with the system or others like it.

Here are some proven ways to "discover" the functional requirements of an existing system:

There are a number of shortcuts that can help you "reverse engineer" project requirements based on the current system.

- *The current system behavior—along with related manual processes—can be "reverse-engineered" to derive functional requirements. Use cases (see below) provide an excellent format for describing and managing these requirements.*
- *It is almost always valuable to examine any available documentation on the legacy application; screen shots, user manuals, release notes, test cases, and existing defect reports and enhancement requests. User documentation in particular can be very helpful in constructing use cases.*
- *User priorities, complaints and needs relative to the legacy system can help*

Understanding how the legacy system works and how it is used is essential for determining project direction.

identify the business uses of the system, as well as ways to improve it.
- *If reverse engineering of the legacy source code is planned, this can also offer an "internal" perspective on the system's functionality, in the form of classes or error flows for instance, which can aid requirements definition.*

A focus on "discovering" functional requirements based on the current system is a key prerequisite for determining where the project needs to go and how to get it there. It is similarly important to derive the high-level architecture of the legacy application. What are its significant subsystems and how do they relate to each other and other systems? What are its critical interfaces? What data does the system store and manage?

One thing that is not likely to be required is a complete re-documentation of the entire legacy environment. The goal is to understand how the system works and how the business uses it currently. A functional description of the components that will be leveraged in the transformation effort is generally adequate.

Use Case Basics

Use cases express the behavior of a system—i.e., the functional requirements—in a way that enables all project stakeholders to understand and verify that behavior. Many teams find use cases to be the ideal framework in which to combine new requirements with discovered requirements and related useful information. This is because use cases naturally extend the requirements format beyond the limited efficacy of "shall-based" functional descriptions like "the system shall prompt users to perform operation xyz."

Use cases express functional requirements in a way that is understandable to all project stakeholders.

A well-written use case clearly expresses a functional requirement by outlining, in text and in a diagram, the sequence of steps that users and the system will perform to accomplish the stated business goal. Use cases have become a de facto format for documenting requirements because of their widely recognized value to a project:

- *Use cases are understandable and usable by both technical and non-technical stakeholders.*
- *Because they are easy to understand, use cases promote the quick resolution of disagreements among stakeholders, thus saving costly re-work later.*
- *Use cases work very well to support modern, object-oriented design and development practices.*

- *Use cases simplify the creation of test cases, because they include a clear description of the order and content of user actions.*

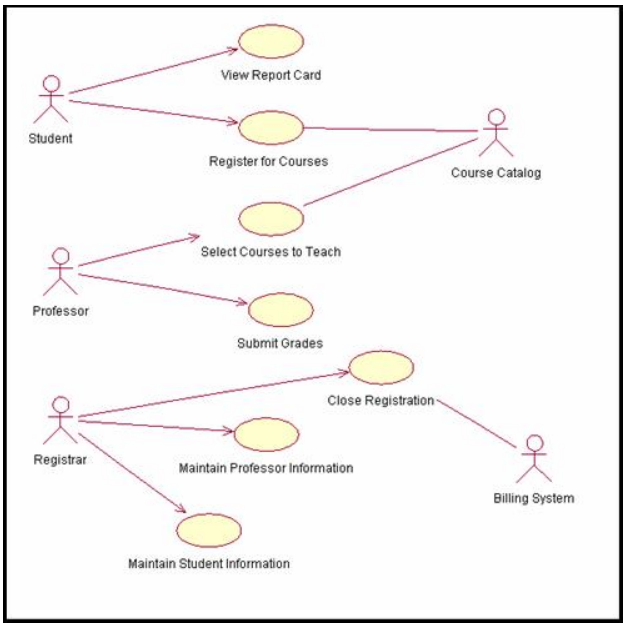

Figure 1. A use case diagram.

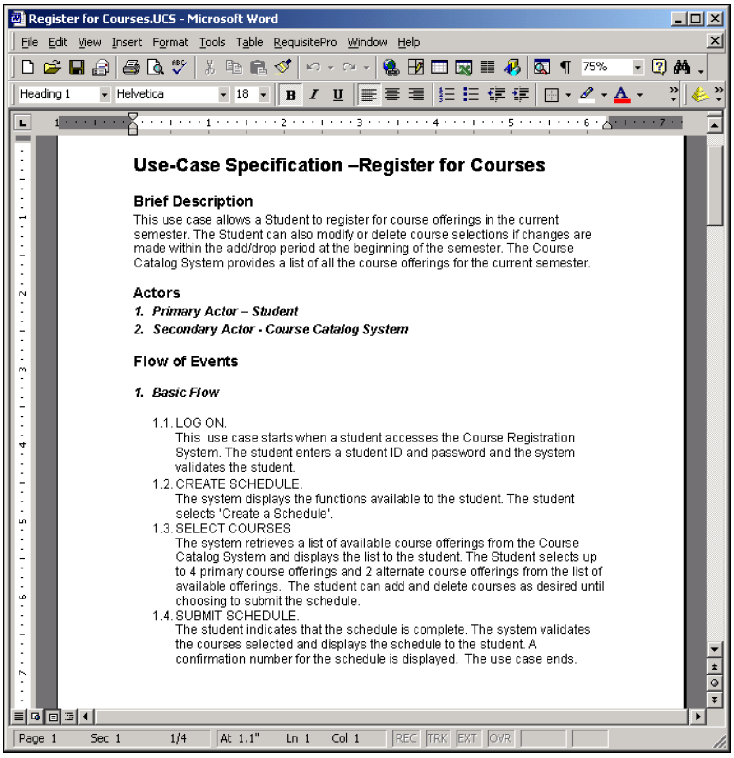

Figure 2. Textual specification of a use case.

Use cases typically include:

- ***The use case name***
- ***A text description***
- ***The actors***
- ***The flow of events***
- ***Implementation notes***

- *Use cases provide ideal input for business modeling efforts, change management, defect tracking, and so forth.*

Figure 1 illustrates a use case diagram.

Figure 2 illustrates a textual specification of a use case.

In general, the contents of a use case include the following elements:

- *The use case name; an action- or goal-orientation is useful here.*
- *A brief description of the use case.*
- *The actors; i.e., the people or systems performing the activity described. For a legacy system the actors will be the users of the system and/or other systems.*
- *The flow of events, including the main flow and any possible alternate flows, such as exceptions and error conditions.*
- *Any relevant special requirements, such as business rules, data definitions, or user needs.*
- *Pre- and/or post-conditions for implementing the use case.*

With legacy systems, the flow of events is especially relevant because it describes how the actor accomplishes the key goal. Alternative flows describe regular variants or error flows. One critical decision to be made when describing flows on legacy projects concerns the level of detail to use. The best course of action usually follows from how familiar the development team is with the legacy environment.

Deriving Use Cases from the Current System

Applying proven, step-by-step techniques make the derivation of use cases based on an existing system more straightforward.

A pragmatic specification of the legacy environment, in the form of use cases, provides an invaluable foundation for ongoing design and development efforts. Equally important, it serves as the basis for gaining agreement with the project sponsor and other stakeholders as to what exactly is to be done. The technique for creating this foundation quickly and cost-effectively is to approach it step-by-step, as described below:

- *Identify the users (actors) of the system's most important processes, and then categorize them according to their roles.*
- *Apply interviews and observation to determine the business value of these*

Identifying the actors

In the use case model, actors—that is, the users of the system—have roles associated with them. Identifying actors and categorizing them according to their roles is critical, because this enables you to clearly identify essential functional capabilities, as well as the business value the functionality provides.

It is often easier to identify users than to categorize them according to roles, but both steps are essential to ensure that all desired functions are implemented.

Often it is easier to identify the users of a legacy application than to categorize them according to their roles; this is especially true with large, distributed systems. However, it is important to identify all classes of users. The consequences of missing a role could be missing functionality in the re-engineered system.

Begin pinpointing actors and roles by interviewing the users of the system. Ask them about their job titles and job descriptions, and find out who else they know who uses the system. Also examine the system's menus, as these can point you to the classes of users it is designed to support. A user's manual can be very helpful here. Any security policies regarding access to various system functions can provide an excellent way to identify classes of users. For example, there may be a group of users who have access to a subset of personal information like address; plus a "manager" class with additional permission to access salary information.

As you gather information, begin sketching use case diagrams illustrating the actors you identify. Initially a whiteboard or flip charts work well for this purpose. As the diagrams become more complex you will probably wish to model your use cases with a visual tool like IBM Rational Rose® or IBM Rational XDE™. Be sure to review the diagrams with users, their managers, domain experts, and the project sponsors. This will further ensure that you have identified all the key roles, and have done so correctly.

You can start building use cases on a whiteboard or flip chart. As the diagrams are fleshed out, most teams switch to a software tool that helps automate use creation and management.

Determining business value

Each use case you create embodies a number of discrete system functions, which in combination deliver business value to the actor/user. Having identified the actors and their roles, the next step is to identify the functions involved in delivering the value. While it might seem feasible to bypass users and simply interrogate the system, this rarely works in practice. Large systems have hundreds, perhaps thousands, of screens. Sorting through them by trial-and-error to identify the value they provide to users is not only frustratingly inefficient, but can easily lead to erroneous scenarios and thus to useless functionality in the new system.

Another good reason to talk to users about how they use the system is that they will frequently call your attention to functions that are no longer used or are faulty.

Identifying how a legacy system is used is best accomplished by interviews and/or by observation. Always ask "why do you use the system?"

There are two complementary methods for identifying functions and their value: interacting in person with the actors, and observing them using the system. Personal action can be in the form of interviews or workshops; the latter being increasingly popular for reasons of efficiency. (For more information on conducting use case workshops, see www.rational.com.)

A good starting point for identifying the uses of the system is to ask the actors, "Why do you use the system?" This usually elicits an answer at the right level. Users are an excellent source of more detailed information, but this can be confusing in the earlier stages of building use cases. Observations are important, too. Many times a user knows his or her job so well that it becomes difficult to describe it to someone else.

As you gather information, begin adding it to the diagrams you've created that illustrate the appropriate actors, and begin drawing associations between the actors and these candidate use cases. In a workshop setting, you can review this process with users on the spot. Look not only for missing use cases, but also for overlap among them. Add any new actors you identify to the diagrams at this time also.

Interviews can be done individually or in a workshop format. Users are an excellent source of details on how the system works.

Capturing the details

Every legacy re-engineering project will benefit simply by identifying actors and use cases for the current system. The next step beyond that, which is even more useful as a basis for requirements, is to add relevant details to each candidate use case.

Use cases work best when they encapsulate a level of detail that enables designers, developers and testers to apply them directly to their respective jobs. On a "green field" project it is appropriate to provide a comparatively low level of detail, because the application doesn't yet exist and team members are still unfamiliar with its details. But when re-engineering a legacy application the knowledge base and level of familiarity of team members may be much greater. The customers for the application likewise are well-understood.

Every legacy project will benefit simply by identifying actors and use cases. The next most beneficial step is to add important functional details to support designers, developers and testers.

Thus, the optimal level of detail for use cases should be determined based on how familiar the development team is with the system. For example, if the team undertaking the legacy transformation has many of the same people who maintain the legacy system, than there is less need to add a plethora of details to every use case.

If the re-engineering project is being outsourced, however, then the need for details is greater. In fact, the level of detail to be provided in use case specifications is increasingly part of contract agreements with third party developers.

Whatever the team's level of expertise, detailed use cases will help ensure they build the correct system. The risk the organization can tolerate if the new system does not work as expected is thus a worthwhile factor to consider when deciding how much effort should go into detailing use cases. If the system is for internal use only, and bugs will not impact mission critical processes, it may make sense to balance the resources required to glean use case details against the cost of fixing or adding functionality down the road. Where functional problems will significantly impact the bottom line, such as by damaging customer relationships or threatening user safety, then the work of creating detailed use cases will be more than worth it.

The level of detail to be gathered is related to the team's level of expertise with the legacy system. Outsourcing development, for example, often means that more details on the must be gathered. With a highly experienced team it may be possible to simply outline some use cases, while adding details where necessary.

On some projects, it may be possible to simply outline some use cases, while adding significant details to others: some may be simple and low-risk, while others are more complex. Another time-saving technique is to refer to screens in the existing system rather than detailing the flow of events.

A good source of further information about use cases can be found at: http://www.rational.com/centers/usecases/index.jsp.

Analyzing "black box" systems

Some systems require little or no user interaction; they may be started simply by a user clicking a button on a screen, for instance. Use cases are helpful in reverse-engineering these systems, too. The actors in such systems will be the people or other systems that use the output of the system under consideration. Use cases will show the significant workflow(s) the black box system provides. In such cases, it is likely that non-functional requirements (tables, algorithms, business rules, etc.) will play a larger part in the new system. Reverse-engineering requirements may involve reverse-engineering code, and/or database internals. The details uncovered can be placed in a "special requirements" section of a use case, or in a separate document, which the IBM Rational Unified Process® or RUP® calls the Supplementary Specification.

The role of business modeling

Business modeling helps teams understand how the legacy system relates to the wider enterprise architecture, and how it delivers business value.

Business modeling can be an effective means of understanding not only legacy system behavior, but also the legacy system's relationship to the wider enterprise architecture. Particularly in situations where a legacy system or a major component of it cannot be efficiently reverse-engineered, a business model can provide a solid understanding of how the system's functionality relates to business value, and can serve as a basis for deriving requirements and communicating changes.

System design models are a good way to describe new system functions, and new interfaces to other systems.

System design models, ideally created in UML, are another excellent way to describe new system functions and new interfaces to other systems. In particular, system models can help define Web Services interfaces, specify data access privileges among multiple systems, or describe communication flows among subsystems or between applications. Tools like IBM Rational XDE are available to help reverse-engineer application code as well as RDBMS functionality; IBM Rational XDE in particular can also generate UML models automatically.

Identifying New Requirements

In addition to establishing a goal-directed vision for a legacy transformation project, and understanding the behavior of the legacy system, it is also crucial to delineate requirements at the component level for those areas of the legacy system that will be significantly modified. If the goal is to port the application to a new operating system, what are the key requirements for accomplishing this? If the goal is to re-write the user interface and make the business logic web-accessible, what changes to the software architecture are required? If the goal is to integrate the legacy system with other applications, what changes must be made to its architecture?

Defining requirements at the component level is crucial for those parts of the legacy application that will undergo substantial modification.

As new requirements develop, proposed changes to one requirement invariably impact other requirements. For example, a requirement to change the programming language of a legacy system from COBOL to Java may require changes to the hardware and/or software infrastructure, or the DBMS. This is another area where requirements management provides the traceability essential to understanding the impact of changes on cost and schedule.

Consider a hypothetical scenario in which a global organization has decided to re-engineer a legacy application that manages parts inventory. This is likely to be a complex, multi-year effort that entails changes to many other data repositories and business processes across the enterprise. Whatever the scope of the project, the first question to ask is: what applications, business processes and people access this legacy data? The second question follows from the first: is this legacy data copied to other repositories and transformed in some way? And thirdly, is it appropriate to unify these variants as part of the current project?

Changes to one requirement invariably impact other requirements. Whatever the scope of the project, a key question to ask is: "What processes access this legacy data?"

This line of inquiry leads to more specific questions, like how many different data definitions exist for "part number" and what will it take to rationalize them? Security information in the legacy database can be a good starting point for tracking down these answers. Another technique for reverse-engineering legacy data is to obtain a list of all recent transactions against the database. Reverse-engineering the application code can sometimes be helpful, but may be of little value where dynamic SQL is used, for instance.

A related concern with re-engineering legacy databases is its business rules. Business rules are considered a type of requirement on some projects. They may be in the database, in the application code, or both. It is usually worthwhile to identify as many business rules as possible, as a starting point for deciding how to implement legacy business rules in the new database architecture.

In addition to the above, key non-functional database requirements that will need to be defined include availability, backup/recovery, performance, and data distribution. These types of requirements can be described in a separate document, such as the Supplementary Specification in RUP.

Summary

The identification and management of requirements is just as important for risk mitigation on legacy transformation projects as for "green field" development. These are often complex, long-term efforts whose scope and focus continually shift and evolve. Requirements provide an intermediate representation of the legacy functionality that is essential for describing application behavior and ensuring that all desired capabilities are implemented in the new system.

The goal of requirements definition for a legacy system is not to describe the existing system in its entirety, but rather to gain an understanding of how it works, how it is currently used, and what components of it can be cost-effectively reused.

Key elements for requirements definition on legacy re-engineering projects include:

- *A vision document, which defines non-functional requirements at the project level to guide the overall effort.*
- *A functional requirements specification, based on the behavior and business rules of the current system*
- *A supplementary specification, which describes component level requirements for those parts of the legacy environment that will be significantly modified.*

Whether you plan to integrate a legacy system with other applications, extend it with new code, or re-engineer it entirely, understanding the requirements for where you need to go is a critical part of ensuring your success.

IBM software integrated solutions

IBM Rational supports a wealth of other offerings from IBM software. IBM software solutions can give you the power to achieve your priority business and IT goals.

- *DB2® software helps you leverage information with solutions for data enablement, data management, and data distribution.*
- *Lotus® software helps your staff be productive with solutions for authoring, managing, communicating, and sharing knowledge.*
- *Tivoli® software helps you manage the technology that runs your e-business infrastructure.*
- *WebSphere® software helps you extend your existing business-critical processes to the Web.*
- *Rational® software helps you improve your software development capability with tools, services, and best practices.*

Rational software from IBM

Rational software from IBM helps organizations create business value by improving their software development capability. The Rational software development platform integrates software engineering best practices, tools, and services. With it, organizations thrive in an on demand world by being more responsive, resilient, and focused. Rational's standards-based, cross-platform solution helps software development teams create and extend business applications, embedded systems and software products. Ninety-eight of the Fortune 100 rely on Rational tools to build better software, faster. Additional information is available at www.rational.com and www.therationaledge.com, the monthly e-zine for the Rational community.

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

Printed in the United States of America
01-03 All Rights Reserved. Made in the U.S.A.

The IBM home page on the Internet can be found at **ibm.com**

TPXXX