3

Risk-Managed Modernization

First ponder, then dare.
—Helmuth von Moltke (1800–1891)

We are prepared to take risks, but intelligent risks. The policy of being too cautious is the greatest risk of all.
—Jawaharlal Nehru,
Speech to Parliament
New Delhi, 18 February 1953

The modernization approach described in this book integrates software engineering concepts with an organized understanding of the information systems technologies that both constrain and define the solution space. The objective of this approach is effective risk management and mitigation leading to the development of a modernization plan that minimizes the risk of the modernization effort.

This chapter introduces the *risk-managed modernization (RMM)* approach. This chapter briefly reviews the topic of risk management, recommends the use of portfolio analysis to select candidate systems for modernization, and introduces the activities involved in risk-managed modernization. The rest of the book expands on these activities and uses the case study introduced in Chapter 2 to illustrate this approach.

3.1 Risk Management

Risk management is a software engineering practice that continuously assesses what can go wrong (risks) determines what risks are important, and implements strategies to deal with those risks. Risk management is not a new idea; it is an integral element of the spiral development process developed by Barry Boehm

[Boehm 88] and is documented in a large body of work [Boehm 91, Karolak 96, Higuera 96, Hall 98].

Figure 3-1 illustrates a UML activity diagram of the RMM approach. Ovals in the diagram represent activities. Arrows represent transition between activities. The horizontal *synchronization* bars require completion of the previous activities before starting new ones. The process starts with a modernization project that has been selected using portfolio analysis. The end state for the process is an integrated modernization plan.

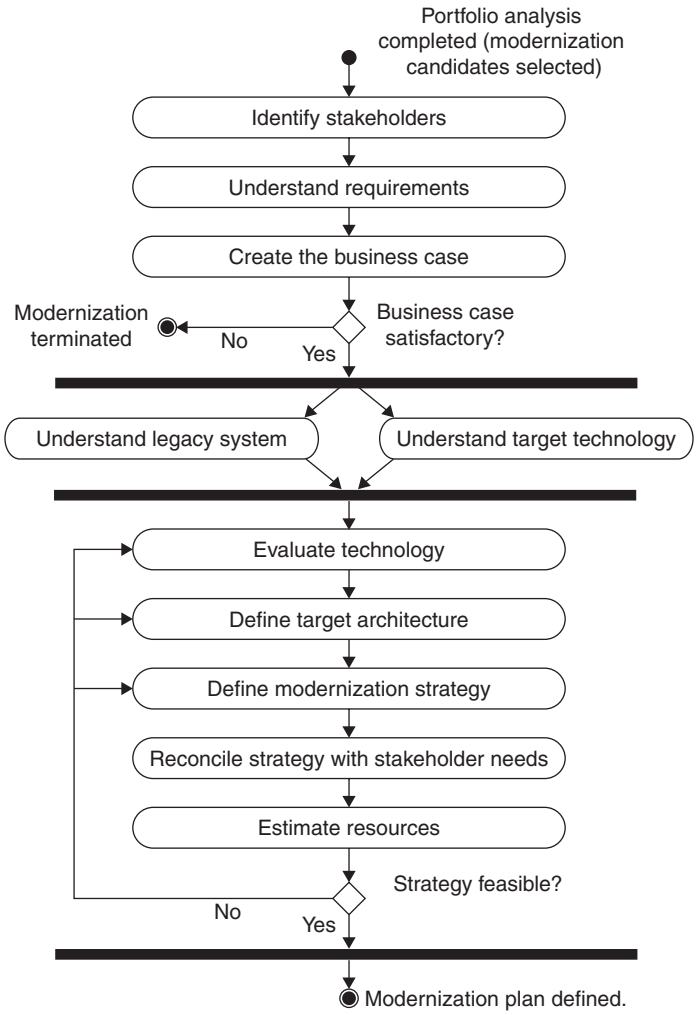

Figure 3-1 Risk-managed modernization approach

PORTFOLIO ANALYSIS

The portfolio analysis establishes measures of technical quality and business value for a set of systems and evaluates this set against the measures [Warren 99].

Technical quality is the measure of goodness of an application or system against a defined set of technical criteria. Example criteria for technical quality include frequency of new releases, ease of making changes, hardware and software reliability, organizational infrastructure, system performance, accuracy, ease of operation, availability of training, and number of vendor related tools and hardware.

Business value is a measure of importance of the system or application to the organization. Example criteria for business value include contribution to profit, level of usage, number of business goals satisfied, system value, user satisfaction and the value of the information that the system or application stores.

Legacy systems are evaluated against these measures and positioned on a portfolio analysis graph, such as the one shown in Figure 3-2. Positioning a system in one of these quadrants requires the establishment of criteria to measure technical quality as well as business value. The quadrant each system appears in can suggest an appropriate evolution strategy.

- **Quadrant 1:** Systems having low business value and poor technical quality are logical candidates for replacement with commercial packages, for two reasons. First, because they have low technical quality, they need to be improved or replaced. Second, because they have low business value, they do not provide any critical services or support any core competencies. These systems may be used for payroll, human resources, or similar services that are not specific to the company's core business.

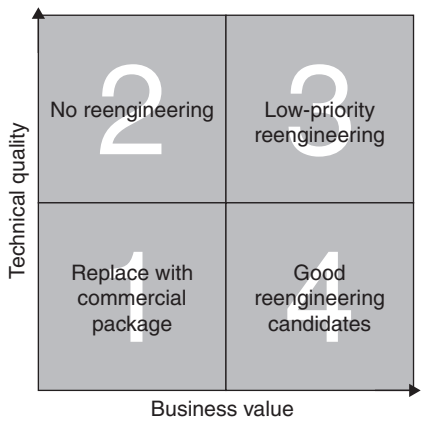

Figure 3-2 Portfolio analysis graph

- **Quadrant 2:** Systems with high technical quality and low business value should not require reengineering, modernization, or replacement efforts.
- **Quadrant 3:** High-quality systems with high business value should be actively evolved, using the evolutionary development practices discussed in Chapter 1.
- **Quadrant 4:** Systems with high business value and low technical quality are the best candidates for modernization or replacement.

After completing the preliminary *portfolio analysis*, you must take into account any organizational or resource issues. These issues can significantly increase risk to a modernization effort. Typical issues include politics, cost, schedule, available staff, technical skills required to perform the modernization effort, development staff, end user training, training personnel to use the modernized system, and user acceptance of the modernized system. Once you have performed the *portfolio analysis* and identified modernization issues, you can select and prioritize candidate systems and develop your overall modernization strategy.

IDENTIFY STAKEHOLDERS

Developers, testers, maintainers, system administrator, customers, vendors, sponsors, end users, architects, and representatives of interacting systems are stakeholders: people who have a vested interest in a project. These people will ultimately judge the outcome and impact of the modernization project. As a result, it is important to obtain their agreement and support. This can often be a challenge because stakeholders from different groups have different interests and view risks in different ways.

UNDERSTAND REQUIREMENTS

Requirements definition may be the most difficult task in software development and modernization. In general, requirements can be divided into the following four categories:

1. **User.** User requirements are capabilities that must be provided by the system. These requirements are often expressed as tasks or activities that must be supported by the system.
2. **System.** System requirements describe the capabilities of the system and the system itself.
3. **Constraints.** Constraints include decisions that have already been made, such as interactions with other systems, development standards, and cost.
4. **Nonfunctional.** Nonfunctional requirements include behavioral properties, such as performance, usability, and security, that the system must have.

Accurate software requirements clarify what is expected of the system, therefore reducing risk. Many software projects fail because of lack of rigor in the requirements process or an inadequate definition of requirements. As in any software project, requirements must be complete, consistent, understood, and valid. Using stakeholders to verify and validate requirements helps ensure that the system will ultimately perform as expected.

CREATE THE BUSINESS CASE

A business case is a document that supports decision making and planning. Because software modernization projects require substantial labor and financial resources, a business case can help convince management that the project is financially viable. Therefore, it is often key in obtaining project funding and approval.

As an essential part of the management decision process, a business case should not be viewed simply as a way to justify a particular effort. Rather, it should help the decision maker decide which approaches, if any, should be evaluated further. Therefore, a good business case considers several modernization approaches and contingencies, along with the technical and economical justification for selecting a particular approach. In general, a good business case should provide information about the project purpose and objectives, a description of the current system and current business process, a description of the future system and future business process, a cost estimate, a cost-benefit analysis, a risk assessment, a change analysis, and measures of performance. Change analysis includes changes to personnel, equipment, software, hardware, and support. Measures of performance are used to assess achievements, effectiveness, and efficiency.

It is crucial to understand stakeholder requirements to build an accurate business case. Chapter 4 provides more details on identifying stakeholders, understanding requirements, and creating the business case.

UNDERSTAND THE LEGACY SYSTEM

Understanding both the legacy system and its context are essential to the success of any modernization effort. The challenge is developing this competency in an efficient, cost-effective, and timely manner. Techniques available to meet this challenge include reverse engineering, program understanding, and architecture reconstruction. Chapter 5 describes these techniques in detail.

Reverse engineering and program understanding were briefly introduced in Chapter 1. Architecture reconstruction is the process of determining the as-built architecture of an implemented legacy system [Kazman 01]. This is done through a detailed analysis of the system, often using tool support. The tools extract information about the system and help build and aggregate successive levels of abstraction. If successful, the end result is an architectural representation that helps reason about the system.

When the Best and the Brightest Fail

Several years ago, we were involved in a legacy system modernization effort at a trucking company. The project was initially attempted by the internal IT department, which failed because of the team's lack of knowledge of the target J2EE technologies and because of organizational problems. Wishing to avoid repeating this failure, the trucking company hired professional services from one of the companies instrumental in developing the J2EE technologies being used. In fact, nothing negative could be said about this organization's understanding of J2EE technology and its ability to apply the technology in the construction of software systems.

Surprising many, this second effort also failed. The root cause of this failure was a lack of understanding by the professional services organization of the legacy system. Of course, what we mean by "legacy system" here goes well beyond simply understanding the source code and the hardware platforms. In this case, the professional services organizations had failed to

- Understand the organization of the trucking company
- Properly identify the stakeholders and quantify stakeholder needs
- Understand the operating policies and procedures of the trucking company
- Understand the modernization goals
- Properly elicit, understand, and negotiate system requirements

Despite the fact that this professional service organization consisted of intelligent, highly qualified software engineers, it was not able to succeed in this modernization effort. The cause was a failure to understand the legacy system.

—rCs

Architecture reconstruction introduces a second major theme of our modernization approach: "architecture driven." Architecture is necessary both to identify a desired end state and to guide you there.

Many excellent texts have been written on software architecture, in particular *Software Architecture in Practice* [Bass 98]. It is unnecessary to reproduce these efforts here. We have, however, included a chapter on architectural representation (Chapter 6) written in collaboration with Len Bass, Felix Bachmann, Paul Clements, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. This material is derived from their book *Documenting Software Architectures: Views and Beyond* [Clements 02]. Because the practice in this area is ad hoc, we felt it useful to include a summary of that work in this book.

UNDERSTAND EXISTING SOFTWARE TECHNOLOGIES

Software practitioners trying to modernize a legacy system often complain that the available software engineering techniques, methods, and processes are disconnected

from reality. The reality of these practitioners is like reality anywhere: It is messy and lacks identifiable features that can be abstracted into a repeatable process. Instead of evaluating similar software development efforts and looking for common features, many software engineers simply create idealized processes that maximize one variable, such as quality, while ignoring numerous other variables, such as cost, schedule, and technology.

Of these variables, only technology is fixed; cost, quality, and schedule can be traded off. As a result, it is important to understand both the technologies used in the legacy system development and those that can be used in the modernization effort. The reason for this is simple: It is necessary to understand the fixed constraints in the problem space before considering how to bind values to the variables.[1]

In general, three classes of information system technology are of interest in legacy system modernization:

1. Technologies used to construct the legacy systems, including the languages and database systems.
2. Modern technologies, which often represent nirvana to those mired in decades-old technology and which hold (the often unfulfilled) promise of powerful, effective, easily maintained enterprise information systems.
3. Technologies offered by the legacy system vendors. These technologies provide an upgrade path for those too timid or wise to jump head-first into the latest wave of IT offerings. Legacy system vendors offer these technologies for one simple reason: to provide an upgrade path for system modernization that does not necessitate leaving the comfort of the "mainframe womb." Although these technologies can provide a smoother road toward a modern system, they often result in an *acceptable* solution that falls short of the ideal.

This book discusses all three classes of information system technologies. In Chapter 7, we describe the COBOL and Java programming languages, which figure prominently in the case study. We also discuss various forms of data repositories, including database management systems (DBMS) and data warehouses. Finally, Chapter 7 includes a discussion of data representations for information exchange, including electronic data interchange (EDI) and the eXtensible Markup Language (XML).

We further explore information system technologies relevant to our case study in Chapters 8 and 9. Chapter 8 discusses distributed transactions, providing background information on both distributed communication and transaction technologies. Chapter 9 describes middleware technologies and standards that may be used to develop a modernized enterprise information system (EIS) including Enterprise JavaBeans (EJB), message-oriented middleware (MOM), Java 2 Enterprise Edition (J2EE), and XML Messaging. We also identify products that implement these technologies, particularly in the Unisys ClearPath 2200 and Sun Solaris

[1] The same rationale applies to understanding the legacy system, as it is another fixed constraint in the equation.

operating system environments, because these two environments primarily frame the case study.

EVALUATE TECHNOLOGY

Once we understand available technologies and their capabilities, we can compare and contrast them. If their capabilities overlap, we must see whether these technologies solve the same problem with a different quality of service (QoS). We might include multiple technologies that serve the same purpose but provide different QoS in a modernization effort, as long as boundary conditions for each of them are clearly understood and communicated.

Technology evaluation is one of the first steps of building a *component ensemble*. An ensemble defines the collection of components that evolve into the architecture for the system. The book *Building Systems from Commercial Components* [Wallnau 01] describes the use of component ensembles, model problems, and other techniques for generating just-in-time competencies in components and component integration. For now, we will simply state that this evaluation is necessary to formulate the eventual architecture and design of the system.

In Chapter 10, we evaluate the feasibility of transactions that span over legacy and modernized components, on two different platforms, by constructing a model problem. In Chapter 11, we evaluate, contrast, and compare two approaches for component integration: a synchronous approach based on J2EE technologies and an asynchronous approach based on a business-to-business integration model. Each approach offers advantages over the other in particular areas. An understanding of the advantages and disadvantages of each approach is crucial in defining the target architecture.

DEFINE TARGET ARCHITECTURE

A target architecture represents the as-desired architecture of the system, providing the technical vision for a modernization effort. Thus, the target architecture must be described in a manner that supports adequate communication among the stakeholders. This usually requires descriptions using different views with different levels of granularity and specificity.

In an incremental modernization effort, the target architecture will likely evolve as the boundaries, constraints, and functionality of the legacy system become better understood and the underlying technology used to build the modernized system matures. Therefore, it is important to reevaluate and update the target architecture throughout the modernization effort.

In Chapter 12, we describe a generic enterprise information system architecture for a data-driven system as a collection of architectural patterns. Each of these architectural patterns illustrates common operations and how they are implemented in a compliant system.

DEFINE MODERNIZATION STRATEGY

Legacy system modernization is often a large, multiyear project. Because these legacy systems are often critical in the operations of most enterprises, deploying the modernized system all at once introduces an unacceptable level of operational risk. As a result, legacy systems are typically modernized incrementally. Initially, the system consists completely of legacy code. As each increment is completed, the percentage of legacy code decreases. Eventually, the system is completely modernized. A migration strategy must ensure that the system remains fully functional during the modernization effort.

An effective strategy defines the transformation from the legacy system architecture to the modernized system architecture. During the modernization effort, technologies may change, additional knowledge about the existing system may be acquired, and user requirements may change. A modernization strategy must accommodate these changes. In addition to meeting these requirements, a modernization strategy should minimize development and deployment costs, support an aggressive yet predictable schedule, maintain quality of interim and final products, minimize risk, meet system performance expectations, and maintain complexity at a manageable level.

Development of a modernization strategy is described in Chapters 13, 14, and 15. Chapter 13 describes getting from the as-built architecture to the as-desired architecture. This architectural transformation strategy includes code migration, database migration, and deployment approach. In our case study, this architectural transformation process is referred to as componentization because we are moving from a largely unstructured legacy system to a modern, component-based architecture. Chapter 13 also describes the use of data and logic adapters to support incremental development and deployment.

System preparation is an optional, potentially beneficial but often risky step that is implemented before architectural transformation. In system preparation, we evolve the legacy system to where it will be easier to perform the desired architectural transformation. The benefit is a reduction in overall modernization costs. The risk is that the system preparation does not go as planned and that the development team gets mired in the legacy code. Chapter 14 describes the analysis of alternatives that was performed as part of the system preparation work for RSS.

Chapter 15 describes the refinement of the selected modernization strategy. Refinement includes the development of a code migration plan and a data migration plan, so that the cost of the effort can be estimated and the strategy implemented effectively.

RECONCILE MODERNIZATION STRATEGY WITH STAKEHOLDERS

In modernizing any legacy system, stakeholders will have varying opinions on what is important and what is the best way to proceed. It is necessary to develop consensus before implementing a modernization plan.

In our modernization approach, the development team is responsible for producing the detailed modernization plan, primarily because that team will be responsible for implementing it. Also, the modernization plan is designed to minimize development costs and schedules while remaining technically feasible. This planning is best accomplished by the development team. Once this baseline modernization plan is in place, other business drivers can be reconciled with it.

Because the baseline plan seeks to minimize development costs, all other plans, in theory, will be more expensive to implement. However, the additional costs of implementing an alternative plan may be offset by other benefits to the business. Chapter 16 discusses a process for reconciling the modernization strategy with stakeholder needs.

ESTIMATE RESOURCES FOR MODERNIZATION STRATEGY

Estimating the costs of executing the modernization strategy is the final RMM step. Once this step is completed, you should have an understanding of the legacy system and modernization technologies, a target architecture, modernization strategy, cost estimate, and notional schedule. Based on this information, management must now determine whether the modernization strategy is feasible, given the available resources and constraints. If the strategy is adopted, the modernization plan is finalized and executed. If the strategy is not feasible, the question, Why not? must be answered.

Depending on the response to this answer, it may be necessary to repeat some RMM steps. For example, if the code migration plan does not result in functionality being deployed early enough in the schedule, you may need to revise the plan before the overall strategy is considered feasible. It is also possible that the target architecture was too ambitious and must be reconsidered or that the target technologies evaluated are too complex, expensive, or otherwise fail to satisfy the constraints of the modernization effort. In any of these cases, it is necessary to go back and revise the modernization strategy until a feasible approach can be identified or the modernization effort is determined to be infeasible, given current constraints, and terminated.

In many ways, application of the RMM approach described above can be thought of as a first-fit evaluation. We are developing and evaluating a modernization strategy to determine whether it is adequate and implementing the first plan that meets our minimum criteria. The RSS case study described in this book follows this approach. It is, however, also possible to apply RMM as a best-fit model. In this case, multiple modernization strategies—or contingencies in *Building Systems from Commercial Components* terminology—are evaluated simultaneously. Management can decide among these plans or, possibly, reject them all. In Chapter 17, we describe cost estimation approaches and how these can be applied in estimating costs for a legacy modernization effort.

3.2 Summary

The risk-managed modernization approach introduced in this chapter requires the application of a wide range of software engineering methods and techniques, as well as detailed understanding of legacy and modern technologies. Through the remainder of the book, we take you through this process in detail, using the RSS case study to illustrate each step. We also provide practical how-to guidance along the way and pointers to other processes and skills you may also need to accomplish your goal of system modernization.