Risk Management Fundamentals in Software Development

By George Holt
Materials, Communication and Computers Inc.

Software development and engineering is a rewarding endeavor, but not without risks and challenges. Efficiently managing these risks is critical and requires that all parties, including researchers, engineers, developers, programmers, and project managers keep the fundamentals of sound application development top-of-mind. Staying focused on the basics is an essential part of minimizing risks and ensuring the success of even the most challenging and complex development projects. Periodic review of these principles and practices is valuable for even the most experienced developer.

The Challenge

Developing software is seldom an easy task. Each project entails unique demands, challenges, and problems. Failure to predict and prevent risks can lead to costly delays, revenue loss, increased stress on team members, a lesser product—even project failure.

The Solution

Although each project has its own requirements, the characteristics of effective risk management remain the same. By identifying risks and developing solutions before and during the development process, you maximize the team's efficiency and the quality of the finished product.

First Line of Defense: Strong Fundamentals

Careful evaluation of potential risks, and solutions to address them, is only the first step. Recognizing that sound application development principles are central to effective risk management and that the risk-management process is ongoing is the key to maximizing the team's efficiency and ability to create excellent software.

Flexible Planning

First, remain flexible. Giving team members the freedom to make decisions as new information becomes available enables them to react quickly; inflexible plans and schedules impede the ability to deal with new challenges.

Inflexible schedules also present team members with a tainted view of the project's progress. While plans and schedules are necessary to measure progress and meet deadlines, they cannot be too rigid. Promoting flexibility in the initial stages of the project encourages proactivity, positions the team to meet the challenges that undoubtedly arise, and also makes them better prepared to maintain realistic schedules as the project progresses.

The importance of flexible planning is particularly important during the creation of the software development plan—the road map that guides the team's efforts.

Sound Development Principles

During the development phase, important unknowns critical to success will unfold as the project proceeds. These unknowns are often risks. The key to accomodating these unknown factors is to recognize that you cannot know everything that may happen. It is important to breed a healthy respect for lucid ignorance, particularly at the beginning of a new project and to guard against individuals who pretend they know all of the answers.

It is often tempting to place too much emphasis on pseudo-order to balance this uncertainty. Reports are completed, meetings are attended, and schedules are met, but few realize how much progress is being made or what risks are increasing in scope. Recognize that the final goal of software development and risk management is excellent software. Do not get so wrapped up in reports and schedules that they become the key area of concern.

One key sound development principle is to reduce the complexity of your code and to modularize common functions. Segregate code modules that will not be shared with all target platforms or operating systems. Make sure the remaining code is common. This optimizes your efforts and lays the groundwork for easy fixes down the road as well as efficient reuse of software on future products.

Integrated Product Teams (IPTs)

Forming IPTs is another valuable approach to containing costs and reducing risks, especially those that might effect scheduling. The IPT facilitates problem solving, enables the team to rapidly respond to changing requirements, and prompts everyone to work on schedule.

IPTs are also an excellent way to keep the customer, in most cases the government, up-to-date on how changing requirements will effect the cost and scheduling of a project, or present the team with additional risks to consider. This is particularly true if the customer plans to add additional features. Most importantly it helps you, the developer, appraise risks and provide acceptable solutions as the customer raises questions.

Prototyping

Exploratory prototyping is the first step toward avoiding the costs of full research and development. It also is an excellent strategy if project requirements are ill defined or likely to change before project completion. In addition, exploratory prototyping is an excellent way to clarify requirements, identify desirable features of the target system, and promote the discussion of alternative solutions.

Prototyping should answer two questions that are fundamental to software development and risk management—"Is the concept sound?" and "Is it worth proceeding further?" If the answer is not a clear "yes," you may be setting yourself up for failure. More importantly, without this insight, you will give the customer a false sense of what can be accomplished. Better to know this up front. Sometimes the most important risk management action you will take is to ask these fundamental questions.

If the answers and the risks are satisfactory, you can move on to evolutionary prototyping, which offers several benefits. It enables your team to quickly and efficiently build on proven aspects of the software. In addition, it enables end users to better define the remaining requirements. As a result, the core of the software's foundation is tested and proven early in the project, significantly reducing exposure to unknowns.

Process Improvement

Improving processes should be ongoing throughout the project. It is important to continually ask, "Is there a better way to get the job done?" Improving the way you do things cannot be done in a vacuum—communication at all levels is critical. Participate with your customers in IPTs and system management teams. In addition, be sure to meet with the teams' engineers on a regular basis for focused, but informal, discussions. While these meetings are exceptionally valuable, guard against extended meetings that cut into your teams' work time.

One alternative to lengthy meetings is to develop and distribute weekly status reports. These give each member insight into the progress of the entire project and a clear view of the big picture. Remember that you can have the best processes in place and still fail miserably in software development. A motivated, goal-oriented, and knowledgeable workforce will succeed even when the process is lacking.

Quality Management

The best quality management approaches use empowerment, ownership, and consensus to minimize risks and maximize productivity of the project's workforce.

Empowering your team members enables them to be their best. All development teams are concerned with giving their engineers the tools they need to succeed, but many fail to provide them with the freedom to use them. Even the most prepared project will suffer if the team is bound by extensive rules and regulations. Do not stifle initiative and creativity.

Always allow the team to propose solutions or actions without the fear of undue punishment and ridicule. Empowering the members of your team improves efficiency, encourages exchange of ideas, and increases the intellectual capacity of the group. In the end, the customer reaps the benefits of this increased expertise.

As in all things, ownership is a key component of success. People care more about the things they own, and software development projects are no exception. Fostering a team-wide sense of ownership makes each member accountable for the success of the project. In addition, members who feel they own their work are much more receptive to constructive critique of their decisions.

Developing consensus at the beginning of a project is also a key aspect of quality management. From the beginning, allocate sufficient time for everyone to agree on the functionality of the end product and the development of the theme. The theme should describe the purpose of the product rather than technical details.

Teams that lack this consensus and a clear-cut theme for the software often place too much emphasis on additional features and technical aspects. These features, although nice to have, usually do not contribute to the basic functionality of the software and tend to lengthen the development process. A good theme guides the order of development and provides the team with a focus that is supported by all involved.

Third Parties: A Mixed Blessing

If a product does fail, it is common for many developers to blame the project's failure on third parties. In some cases they are correct. At times you will have no choice but to elicit their help. The key is to minimize how much you depend on them.

Anytime you rely on a product or service from someone outside of your group your risk of failure or delay increases. Your team may do everything right, but if a crucial third party does not, your work may be in vain. To illustrate this, consider the risks you assume by depending on three crucial components of your project from start to finish. Assume each product has an 80 percent chance of arriving on time and fully functional. The probability of success for all three combined is not 80 percent, it is .80 x .80, x .80 or 51 percent. In other words, your project now has a 50-50 chance of failing.

Implementing Effective Risk Management

Now that we have reviewed the first line of defense against risks in the development process, let us look closer at risk management techniques you can use in your own projects.

An effective risk-management program is dynamic and ongoing throughout the development process and requires the participation of everyone involved. First, remember it is an idealized process. Do not follow it lockstep for each and every development undertaking. Modify the process depending on the type of work to be performed and the members of your team. For example, software rehosting or software block updates may not require the same degree of discipline as a new development effort.

To implement an effective risk-management program requires careful review of risk assessment and risk control.

Be Prepared: Risk Assessment

Carefully assessing challenges inherent in any project is the first step in implementing a successful risk-management program. Risk assessment includes three key processes: risk identification, risk analysis, and risk prioritization. Focusing on each ensures the successful application of risk control tactics.

Risk Identification

Identify the risks most likely to occur in the project and pay particular attention to those created by changes in customer requirements or target systems. Develop a checklist of the risks that may arise and include input from each team member.

Risk Analysis

Once you know what risks you face, it is important to analyze each one in two ways; first, the chance of the risk occurring and second, the consequences if it does. Looking at each risk and answering each question enables you to determine which risks require focused attention. Modeling techniques can be helpful in determining the odds of each risk occurring, but each team member's input is essential in determining potential consequences.

Risk Prioritization

Developing a plan and determining which risks you need to deal with first is critical. Prioritize your risks based on your analysis of the risk's chance of occurring and the potential consequences. Risks that pose the greatest danger to the project must be dealt with first.

Take Action: Risk Control

To implement a successful risk management effort requires continuous assessment. Risk control includes three key processes: risk management planning, risk

Project managers and developers should collect and monitor metrics that give a picture of the project's progress. If the results of these metrics suggest that a change in process is required, the development practices will need to be modified. Some metrics that pay big dividends are:

REVIC/COCOMO II: These models estimate the cost of software projects and have very good predictive capabilities if environmental factors are carefully estimated and applied to the models.

Schedule Adherence Measures: Use this metric to plot the progress of the project against your planned progress for all major and subtasks.

McCabe's Cyclomatic Complexity Metric: This tool measures the complexity of single-code modules.

Fan Out: This model measures system or structural intermodule complexity. Measuring such factors as fan out, or the number of modules called by any given module, will show the direct correlation of overall system complexity and development defect rate.

Software Cleanroom Process: The cleanroom software process stresses proof of correctness in the design and coding phase of development.

Readiness, Maturity, Growth Model: This model provides management with a quick look at the state of the software development effort. It is very useful to determine when software is ready for formal qualification testing.

Curvilinear Relationship between Defect Rate and Module Size: This metric provides the optimum size of modules to minimize overall defects.

Early Defect Removal: Software engineers should concentrate heavily on early defect removal in the design and coding phases of software development. It has a substantial impact on reducing program costs. It is also important during development to conduct informal analyses to eliminate processes that cause errors.

SW Error/Fix Ratio and Defect Removal Efficiency: This measure is used to gain insight into the software team's proficiency in resolving errors, or bugs, in a timely fashion.

Software Lines of Code (SLOC) Tracking by Functional Builds: Compares the actual vs. estimated SLOC per build.

resolution, and risk monitoring.

Risk Management Planning

Risk-management planning is based on the likelihood and consequences of a risk occurring as determined by risk analysis in the assessment phase. Reacting to risk requires resources and time. Always evaluate whether the costs incurred by implementing risk control outweigh the benefits. This enables the team to plan its risk management efforts rather than respond to each risk as it occurs.

Risk Resolution

Once you identify the risks in a development project, the next step is to resolve or reduce them. Employ staffing decisions, cost/schedule estimates, quality monitoring, new technology evaluation, prototyping, scrubbing requirements, benchmarking, and simulation/modeling to react to these risks. Remember, in most projects, 20 percent of identified risks account for 80 percent of potential for project failure.

Focus on the 20 percent with the most likelihood of causing problems. Employ early prototyping and frequent functional system builds to determine if the steps you took to resolve the risks were successful.

Risk Monitoring

It is important to continually monitor your risk control efforts and the appearance of new risks as a result of prior fixes. As the key to your risk monitoring efforts, insist that the program manager, the technical lead, and each developer are able to state their top three risks at any time. These are by their nature dynamic and will change.

Risk Tracking Tool: Risk Radar© is a useful tool to track, manage, highlight, and contain risks, especially those that lay on a critical path. It does not attempt to replace professional judgment, but it is a straightforward, easy-to-use tool for tracking, prioritizing, and communicating project risks. It focuses on ease of use by displaying important, fundamental risk data and provides a graphical display of risks by probability of exposure and has extensive reporting features. It is free and can be found at www.spmn.com/rsktrkr.html

Summary

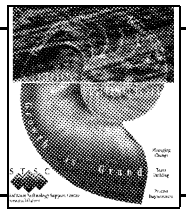
Software development will always include risks, but none are insurmountable if you are prepared to face them at the start. Risk management is an excellent way to prepare for daily challenges.

A viable risk management plan can mean the difference between success and failure. It should, above all else, be flexible and encourage initiative. Remember to always look ahead, use rapid prototyping if necessary, follow a defined program to minimize and manage risks, use a good set of metrics, keep the customer in the loop, and always follow the fundamentals of sound application development. Following this risk management approach will not guarantee excellent software

About the Author

George Holt is a vice president of Systems Development Division of MATCOM Inc. He has developed software for more than 15 years, ranging from military tactical systems to simulators to computer-based tutorials. His division recently rehosted 180,000 lines of Army tactical software to a nonproprietary, open-architecture, Pentium-based system for which it received the Department of Defense Standardization Award. He also managed the Digital Fire Control System prototype for the Lightweight Howitzer, which was successfully fielded in six months. He is the author of many magazine articles, technical publications, and co-author of the book, *Strategy: A Reader.*

1050 Waltham St.
Lexington, Mass. 02421
Voice: 781-862-3390
Fax: 781-402-1515
E-mail: gholt@lexma.meitech.com



Transition is difficult . . . Change is tough . . . Growth is Grand . . . .

With this in mind, CROSSTALK would like to thank Larry W. Smith for designing the back cover ad on this and the previous three issues (in addition to the current and several previous covers).