◆ The Neglected Management Activity: Software Risk Management

F. Michael Dedolph

Software risk management is a crucial part of successful project management, yet it is often neglected. Why is it important? Why is it neglected? This letter discusses barriers to implementing risk management and provides examples of good (i.e., successful) risk management.

What is Software Risk Management?

Risk management terminology is inconsistent, so it is worthwhile to establish working definitions of risk, risk management, software risk management, and risk exposure (or risk magnitude).

Risk can be defined as "the possibility of harm or loss." Typically, risks are described as some kind of event that may or may not occur, coupled with a consequence that follows if the event occurs. This definition leads naturally to considering the probability of the risk event and the cost(s) associated with the consequence.

Risk management is the set of activities used to manage risks. Ideally, the activities are both systematic and effective (i.e., the overall risk is reduced to a level that is acceptable). Risks are acceptable if the project is willing to accept the worst-case consequences.

Risk management activities have three primary goals: identifying, analyzing, and mitigating risks. These activities are typically done using some variant of a Shewart (plan-do-check-act) cycle.

Identification and analysis can be formal (e.g., the Bell Labs RiskIQ^{SM} methodology, which is based on extensive checklists and supporting financial models) or informal (e.g., a risk list derived from a brainstorming session followed up by working on the top five or so items on the list).

Mitigation, also called control or abatement, can involve taking actions to reduce the probability of the risk event, taking actions to reduce the impact of the risk event, or both. Many mitigation strategies depend upon the characteristics of the specific project for which they are developed, and their design requires considerable creativity. There are also generic mitigation strategies, including:

- Acceptance (i.e., doing nothing),
- Buying information (i.e., hiring a consultant, performance modeling, and prototyping user interfaces),
- Transfer (i.e., outsourcing),
- Prevention (i.e., canceling the project), and
- Contingency reserves (i.e., padding the schedule, using budget reserves).

The most common risk mitigation strategy is acceptance (or assumption). Use of this strategy is often the unintentional consequence of not managing risks.

Risk management practices at Lucent Technologies vary. Typically, projects create a risk list early in the development cycle. Some projects update the list proactively and manage risks closely; others do not. The risks considered are usually technical; organizational and environmental risks are not normally

Bell Labs Technical Journal 8(3), 91–95 (2003) © 2003 Lucent Technologies Inc. Published by Wiley Periodicals, Inc. Published online in Wiley InterScience (www.interscience.wiley.com). • DOI: 10.1002/bltj.10077

considered. Team risk management (i.e., risk management involving customers and suppliers) is not generally practiced.

Software risk management is risk management applied to the development and/or deployment of software-intensive systems.

Risk exposure is the product of the probability of the risk event and the impact of the risk event, measured in financial terms. For example, a .001 probability of an automobile accident with a weighted financial impact of $450,000 has an exposure of $450. (The weighted impact may seem high, but the value includes multimillion-dollar lawsuits as well as fender-benders).

Risk magnitude [1] is based on estimates of probability and impact, like risk exposure. However, risk magnitude, rather than being a calculated estimate of financial impacts, is typically mapped to values of low, medium, high, and (sometimes) catastrophic. The use of magnitude values is appropriate when financial impacts are unknown or quantitative risk data are lacking.

If the goal of a project is to pick the top five to ten risks, a simple prioritization of the known risks using any convenient, agreed-upon measure of magnitude is sufficient; but if the project's goal is to motivate investment in risk-management activities or to decide how much investment is appropriate, a more rigorous approach using financial measures of exposure is needed.

Why is Software Risk Management Important?

The short answer is that unmanaged risks cost money.

It is difficult to motivate software risk management during development, because many of the costs are intangible or deferred. While hardware drives many of the most visible development costs (e.g., new ASICs, models, and manufacturing ramp-up), software often drives the schedule, even in hardware-intensive projects. Software schedule slips affect overall product financials indirectly.

Slips in software schedules are notorious and widespread. Industry-wide, software development projects are only successful (i.e., on time and on budget) 16.2% of the time. Of the remaining projects, 52.7% are late—or delivered with a subset of planned functionality—and 31.1% are cancelled before completion [2].

Almost every software development risk event affects schedule, quality, or both. The risk *event* may be any of a number of things (e.g., overloaded test equipment, requirements churn, performance problems, staff turnover, and integration of third-party software); however, the *consequence* of the risk event will usually affect the schedule or quality, indirectly affecting the product's financials.

The cost of a schedule slip can easily exceed the development cost of a product. The obvious financial impact of a schedule slip is increased staff costs, but an extended schedule slip can cause other, more serious effects, such as delayed receipt of revenue and lost market share.

To understand why this is so, consider a hypothetical project that has a 2-year development plan and staff costs of $16M (i.e., $670K per month). The net income is expected to be 10% of $500M in sales over 5 years, or $50M. Sales are expected to be front loaded: $50M per quarter in the first year, dropping to $25M per quarter in years two and three, and $12.5M per quarter in years four and five. A lead customer has imposed a penalty clause of $0.5M if the product is more than 90 days late; further delay will also cause loss of market share. Based on these assumptions, **Table I** shows the possible financial impacts resulting from different schedule slips.

Delivering a product with reduced software functionality—a common event—can be viewed as one type of quality problem. The effects of doing so, like those of late delivery, are subtle, but they can include increased overall development costs, lost sales, erosion of market share, and expensive retrofits for the installed software base when the deferred functionality is deployed in a later release.

If it seems that the message of these data is "make the schedule at all costs," it must be remembered that many schedules are unrealistic and cannot be met, because they are established without an understanding of the size and complexity of the software effort. Furthermore, there are significant risks associated

Table I. Possible financial impacts of different schedule slips.

Risk magnitude	Description of effects	Financial impact
Low	Schedule slip < 1 month	Minimal
Medium	Schedule slip = 2 months Staff costs: ~$1.3M Net profit lost: ~$3.3M	$4.6M
High	Schedule slip = 4 months Staff costs: $2.7M Net profit lost: $6.7M Penalties: $0.5M Lose 20% of remaining market, causing net profit loss of ~$7.3M	$17.2M
Catastrophic	Schedule slip = 6 months, resulting in project cancellation Development investment loss: $20M Penalties: $0.5M 100% net profit lost: $50M Unconsidered costs: loss of future sales to lead customer, time value of money, supply chain costs	At least $70.5M

with rushing a product to market. Products delivered on time with major quality problems incur a different set of financial liabilities, which show up during deployment or post-deployment. For example, if defects cause network outages and/or loss of revenue for customers, the supplier of the product may share liability and incur the cost of litigation, which can easily exceed the costs shown in the example above.

Why is Software Risk Management Neglected?

There are good reasons to manage risks. If we apply industry averages to the hypothetical project discussed above, the product would be late 52.7% of the time, and 31.1% of the time the project would be cancelled. Using the financial impact estimates listed in Table I, a back-of-the-envelope exposure estimate is approximately $27M. If this overall exposure is reduced by only one third, the potential savings are around $9M.

Why, then, is software risk management neglected? There are many reasons, but organizational inertia is the most important. Organizations by their nature resist change. There are many things that contribute to this organizational resistance, including:

- Visible (and tangible) development costs get more attention than intangibles like loss of net profit and downstream liability.
- The value of risk management cannot easily be proved; the savings do not seem real.
- Project teams are too busy fire-fighting; there are no resources available for any extra work.
- Risk management—particularly formal methods with a high initial overhead—seems difficult.
- Project teams (and managers) are rewarded for problem-solving, not prevention.
- Mitigation actions may require organizational or process changes.

Attitudes can also stand in the way of doing risk management. Some of the common barriers resulting from attitudes are:

- Discussing risks goes against cultural norms (e.g., bringing up potential issues is viewed as negative thinking or as causing conflict within the group),
- Overconfidence (e.g., Lucent only hires really good engineers), and
- Fatalism (e.g., software is always late anyway; there is no way to change that).

The author's consulting experience indicates that groups that do not understand and apply the basics of project management do not manage risks effectively, because they are too busy fire-fighting.

Overcoming all these barriers can be done top-down by insistent leadership or bottom-up by

enlightened project teams. In either case, it requires substantial effort to keep communication open and honest.

Examples of Good and Bad Software Risk Management.

Examples of bad software risk management abound, as shown by the poor track record for on-time software delivery. The following are some common software risks that will exacerbate almost all others:

- Unanticipated requirements growth,
- Setting schedules before the scope of the software effort is known,
- Not using code inspections for software that must be highly reliable,
- Shortening test cycles to make ship dates,
- Not reviewing lessons learned from previous projects,
- Rubber-stamping (or not having) architecture and gate reviews.

The following subsections contain a few examples of good risk management observed by the author.

Team Risk Management

A large Air Force project had experienced three consecutive late point releases with unacceptable quality. A risk team was formed comprising the developer, customer, systems architecture group, and contract management. An outsider was brought in to do an initial risk evaluation, facilitate monthly team risk meetings, and act as a devil's advocate. The results were twofold: issues were addressed proactively (resulting in three on-time releases with improved quality) and bonuses improved the contractor's net profit. Here, risk management was driven top-down. The notable innovation was using a customer/supplier team to manage risks and creating a forum for open discussion of risks that crossed organizational lines.

Creating a Marketing Function

A small project was charged with making new software development tools work in the Navy environment and with propagating their use. It piloted the tools on a real project. Requests for information and tool assistance began to interfere with meeting the schedule commitments. The project established a marketing function and staffed it full-time with a senior engineer. The results were no schedule slip and widespread adoption of the tool set. These actions were taken at the project level (i.e., risk management was driven bottom-up). The key insight was noticing that the risks were environmental, so a buffer was needed to shield the project.

Building a Platform

A platform team was building a platform concurrently with the development of platform applications. Anticipating that necessary changes to the platform would adversely affect the application development schedules, the platform team streamlined its problem resolution process and allocated senior engineers to work directly with the application development teams. These engineers were assigned to support each application throughout development and test. The results were fast problem resolution, minimal schedule slips, and acceptance of the platform. This project was a good example of teaming with customers to reduce the overall risk to both sides.

Conclusions

Software development is often the determining factor of both the overall schedule and the quality of the system. Therefore, managing software development risks is essential to minimizing both schedule risks and the cost of quality.

Unmanaged risks cost money. Schedule slips incur significant financial risk, while software quality problems incur significant post-release liability.

Risk management can be viewed as proactive as opposed to reactive management. It can be formal or informal and can be driven top-down or bottom-up, but it requires sustained effort and good communication to be effective.

Acknowledgments

I owe my risk management colleagues a large debt for most of the ideas presented here. Thanks to Audrey Dorofee, Dick Murphy, Frank Sisti, and Ray Williams of the Software Engineering Institute; Elaine Hall of Level 6 Software; Anwar Kurshid, David Laurance, David Mongeau, Kiron Rao, John Schaefer, and Pat Sciacca of Lucent Technologies.

References

[1] R. C. Williams, G. J. Pandelios, and S. G. Behrens, "Software Risk Evaluation Method V2.0," CMU/SEI-99-TR-029, Software Engineering Institute, Pittsburgh, 1994, <http://www.sei.cmu.edu/pub/documents/99.reports/pdf/99tr029-body.pdf>.

[2] Standish Group, The Chaos Chronicles: Version 1.0, The Standish Group, West Yarmouth, MA, 1994, <http://www.standishgroup.com/sample_research/chaos_1994_1.php>.

Further Reading

— D. P. Gluch and A. J. Dorofee, "A Collaboration in Implementing Team Risk Management," CMU/SEI-95-TR-016, Software Engineering Institute, Pittsburgh, 1995, <http://www.sei.cmu.edu/pub/documents/95.reports/pdf/tr016.95.pdf>.

— E. M. Hall, Managing Risk: Methods for Software Systems Development, Addison Wesley Longman Inc., Reading, MA, 1998.

(Manuscript approved May 2003)

F. MICHAEL DEDOLPH is a technical manager in the

Software Technology Center (STC) of Bell Labs at Lucent Technologies in Pittsburgh, Pennsylvania, and a member of the System Architecture Review Board (SARB). His primary responsibility is to lead product architecture reviews. He is also the lead instructor for a Lucent class in telecommunications systems architecture and for a new Design for Profitability class. His professional interests include creative problem-solving and the application of team-based strategies for product development, process improvement, and organizational improvement. He has a B.S. degree in mathematics from Texas Tech University at Lubbock and an M.S. degree in computer science from the University of Colorado at Boulder. ♦