



# Preventive Risk Management for Software Projects

Sanjay Murthi

**D**eveloping large software systems is risky business. According to a report from The Standish Group, “CHAOS: A Recipe for Success,” only 28 percent of all software projects in 2000 were on time and within budget and had all their planned features—which means the other 76 percent either failed or did not meet original goals. This is scary in an economy where software systems can make or break the organization. Internet auction company eBay lost millions of dollars when its systems were unavailable for

even a few hours. Software product companies like Microsoft and Oracle lose millions of dollars when product releases are late or do not work as expected. Even small to medium-size projects suffer costs from delays. If the cost of each person on a 10-person team is \$100 per hour, a company spends \$40,000 for every week of delay. The cost in terms of lost opportunities, lost sales, and dissatisfied customers could be even greater.

Many companies have adopted detailed and heavily process-oriented methodologies, hoping to reduce delays and the number of failures. Unfortunately, these methods contribute their own overhead and delays, and frequently provide little guarantee of success. They are also generally *prescriptive* in that the team takes action (implements a cure) when they find a problem (illness). The cure is often worse than the disease. Senior management becomes aware of a problem when the team misses certain milestones or customers report software problems. They scramble to fix the problem by drastically reducing project scope, replacing project managers, hiring expensive con-

tractors, or taking resources from other projects. In the end, the company kills the project because the cure has become too costly.

A better approach is *preventive* risk management based on more flexible development practices. The idea is to identify potential steps in development and deployment that can delay the project or cause it to fail and devise strategies that will mitigate the risk should it materialize. Unlike prescriptive risk management, in which the team identifies risks only when they occur and attempts to mitigate them only after the fact, the team identifies and evaluates risks and devises mitigation strategies throughout development. And unlike prescriptive risk management, all project stakeholders are intimately involved in identification, evaluation, and mitigation.

## IDENTIFYING AND EVALUATING RISKS

The first step in managing risks is to identify them: What could possibly cause the project to be late or to fail? Risk taxonomies can guide the project team in identification, but although much work has been done to develop such taxonomies, they still tend to overlook risks that typically plague actual projects. The Software Engineering Institute’s taxonomy, for example, focuses on internal project risks, such as design and integration, almost to the exclusion of the external events


*A preventive approach to risk management makes it part of development and emphasizes flexible processes.*

## Inside

### Resources

**Risk Categories in Real Projects**

**Principles Behind the Agile Manifesto**



## Resources

- **The Agile Manifesto**, <http://www.agilemanifesto.org>: The results of a 2001 meeting of major practitioners of agile methods brought these different methods under a common umbrella.
- **Agile Modeling**, <http://www.agilemodeling.com>: Site devoted to applying agile methods to modeling.
- **“CHAOS: A Recipe for Success,”** [http://www.pm2go.com/sample\\_research/chaos1998.pdf](http://www.pm2go.com/sample_research/chaos1998.pdf): A report by The Standish Group; it’s an old version, but it’s free.
- **Lean Programming**, Mary Poppendieck; <http://www.sdmagazine.com>: Series of articles in *Software Development* on an agile method that arose from experience in applying total quality management to manufacturing.
- **“The XP Paradigm Shift,”** Ed Yourdon; <http://www.cutter.com/summit/read03.html>: Valuable insight into Extreme Programming from the Cutter Consortium.

that can derail a project—politics, changing business requirements, platform deficiencies, and so on. The “Risk Categories in Real Projects” sidebar describes how these external events can also influence projects.

Risk identification must also acknowledge that risks change with time. New risks arise that the team has not planned for. The Microsoft solutions framework, which suggests that managers continuously assess risks, is one of few methodologies that acknowledge changing risks.

The second step in managing risks is to evaluate them. Risk evaluation should start soon after the team has finalized project goals, and it should continue through the entire project life cycle. Once the team identifies the risks, they can take the following steps:

- **Establish potential impact.** If the risk materializes, what items would fail or have problems? Suppose the risk is that the infrastructure to test an application will be two weeks late. The impact will be large if testing must start earlier (before the delay), but minor if testing falls later in the cycle. Some risk may remain if it takes time to set up the infrastructure after delivery.
- **Rank the risks according to potential impact.** Ranking can be as broad as high, medium, and low.
- **Calculate the probability that the risk will occur.** Again, broad categories of high, medium, and low will suffice. You do not need an exact percentage.
- **Rank the risks by their combined impact and probability of occurrence.** The combinations of high impact-high probability, high impact-medium probability, medium impact-high probability, and medium impact-medium probability are the ones to watch for.
- **Develop contingency plans for major risks.** Aim to have

more than one plan.

- **Determine the resource requirements for the contingency plans.** This will give you the cost impact of each contingency plan.
- **Put this risk information in the review plan.** You can then communicate it to project sponsors and development managers. Also include information about likely contingency plans and costs so that everyone will be on the same page if the risks occur.
- **Track risks as the project progresses.** Some risks may become moot. Other risks will become less likely or their potential impact will decrease. New risks can arise after the previous review. Track risks in weekly reports so that all players know what is happening.
- **Periodically evaluate and modify the risk evaluation approach.** Use post-project meetings to review the effectiveness of risk evaluation and management. Use feedback to improve the process.

## PROACTIVE PREVENTION

Preventive risk management is the proactive management of three important areas: people, process, and control systems.

### People

Of the three, people are probably the most important. Unless they are committed and eager to work with each other, projects frequently fail. In many projects I have observed, teaching people better estimation, scheduling, and risk management skills really helped. These skills produced better estimates and timelines, and people were less likely to underestimate or miss work items. Consequently, there were fewer 80-hour weeks, and stress levels dropped. Meeting deadlines on target produced a solid team spirit, which in turn produced a *virtuous* cycle of change—people became more patient with each other and were willing to help each other solve problems. Continuous firefighting and struggling to meet deadlines, on the other hand, led to a *vicious* cycle. Stress levels rose, people became irritable and impatient, and the desire to help someone else disappeared, as each person totally focused on meeting his dates and solving his personal mountain of problems. Things went steadily downhill until management had to resort to drastic surgery or terminate the project.

### Processes

Processes also influence risk management. Flexible, adaptable processes let the team respond quickly to change. Without them, it is hard to keep a project on track. If a change control board must approve every requirement change, and the board meets only once a month, how can the team try

## Risk Categories in Real Projects

Many risk taxonomies fail to consider the external risks that affect real projects. This list reflects the risk categories most projects are likely to encounter. A flexible development process will help mitigate risks in the first three categories. The team is better able to stay loose in defining requirements, modify system designs or platforms late in the process, or work around failed partnerships that introduce business risk.



structure isn't in place or the support team isn't ready for training or is already stretched too thin. Sometimes this risk occurs because the deployment and support teams have no idea what the project team is doing, let alone what it needs. In this case, communication will help mitigate the risk.

- **Requirements.** Unclear or uncertain requirements introduce large risks. This is the most common type of risk and is probably responsible for most failed or delayed projects. Competitive forces and business agreements with new partners force the organization and its software systems to change. Users find it hard to visualize software until they use it, which makes requirements fuzzy and subject to change.
- **Technology.** At some point in development (usually late in the cycle), the team finds that the technology can't satisfy system requirements. For example, team members could assume that the database they use is not easily corrupted, but when they actually build the system, they find that it has bugs that can cause it to become corrupted frequently.
- **Business.** Business decisions introduce many risks. A deal with a vendor does not get signed in time to use the desired platform. A conflict with a partner who supplies part of the solution stalls the project or the partner goes out of business.
- **Political.** These are the most difficult risks to overcome. Large organizations tend to behave like large families. Members are busy jockeying for power and influence over each other. Some groups might find the project threatening, which means that people do not cooperate, budgets get cut, or the project gets cancelled. Contingency plans for this risk are hard to define, because these plans can cause embarrassment later, and if project opponents find out about the contingency plans, they can sabotage them.
- **Resources.** When a project doesn't get the required people, money, facilities, or equipment, the shortfalls degrade both schedule and morale. Identifying an alternative resource can help. For example, another team might be willing to share some of its servers until yours come in.
- **Skills.** These risks arise if, for example, the team is unfamiliar with the technology or business process. Providing training and bringing in consultants with the missing skills, who can mentor the team, will help mitigate this risk.
- **Deployment and support.** The team can't deploy the software on schedule because the required infrastructure isn't in place or the support team isn't ready for training or is already stretched too thin. Sometimes this risk occurs because the deployment and support teams have no idea what the project team is doing, let alone what it needs. In this case, communication will help mitigate the risk.
- **Integration.** Most applications must integrate with other applications. Miscommunication and misunderstandings cause systems to miss sharing accepted interfaces, so they don't work together as expected. Communication is key to reducing this risk. Try to do integration in parallel with development (using stubs that satisfy agreed-on interfaces). Communicate variations and unexpected behavior to all interested parties as soon as possible. A flexible development process will let the team try different designs if it encounters intractable integration issues.
- **Schedule.** These issues include components that aren't available when needed, delivery at an extremely busy time, and so on. Communication can help people see that timing a product upgrade towards the end of a quarter is not a good idea because the sales staff are trying to reach their quotas. They don't want to reprice products and reeducate customers just when they're about to close deals. Good project planning can help prevent schedule risk.
- **Maintenance and enhancement.** The company can't maintain and enhance the software properly because the documentation is inadequate, the support team isn't properly trained, or the platform has become obsolete. Planning and setting aside time for adequate training and documentation can help reduce these risks. A flexible development process can make things worse if managers don't allow enough time, money, and people for training, documentation, and support.
- **Design.** Bad design decisions can degrade the software's usability and system performance. A flexible development process that can accommodate user input and changes late in the process can reduce this risk.
- **Miscellaneous.** This is a catch-all category for risks that are hard to foresee—a hurricane or fire shuts down your offices for a week, a development server crashes, a virus attacks, and so on. Most of these risks involve a loss of some resource, so a mitigation strategy is to arrange for back up. Set people up to work at home, store project data in multiple locations, or plan to use a spare server from another team.

out new ideas? In some projects I have observed, the team would work on certain requirements only to find the effort wasted because the requirements changed at the next monthly board meeting. In projects involving subcontracted work, these delays caused major budget overruns.

Agile methods—such as Extreme Programming, feature-driven development, and Lean Programming—can help avoid this inefficiency because their mission is to cut items that add no real value. As such, they provide low-cost ways to work with requirements changes: Instead of detailed requirements documents, they advise close interaction between programmers and customers to flesh out requirements. They also recommend frequent iterative releases so that usable code is available as soon as possible. Users can then provide the team with valuable feedback for further development. Finally, agile methods concentrate heavily on automated test cases so that the team can continuously check for bugs introduced from one release to the next. Clearly, agile methods work extremely well in situations where needs change quickly.

Software projects using agile methods tend to be extremely flexible and have lower overhead. The focus on continuous user input and the repeated demonstrations of working software helps catch issues early and reduces wasted time and effort. The emphasis on close stakeholder interaction means that process decisions involve all stakeholders. On one project I was responsible for, our releases came at the same time as another project's. This caused problems for the support team, which was short staffed. We solved this conflict by modifying our release cycle and appointing some team members to provide second-level support.

## Control systems

Management control systems are important because the team needs some mechanism for measuring and monitoring all aspects of development, including risk management. Poor measurement and monitoring can doom a project, just as a good system can save it. I have sometimes seen project teams try to downplay certain risks and ignore them, but the control system forced them to address these issues before it was too late. In one case, stakeholders from the software company's professional-services team questioned some of the project team's security assumptions. This caused the team to revisit the items of concern and change the product. In another case, senior management became quickly aware of problems during a project to bring the company's products into an international market. Each project team was reporting different dates and risk issues. This triggered warning bells that made the senior management look more deeply into what the company was attempting. Each team was using different techniques, and many were incorrect. If management had missed this discrepancy, the whole effort would have been a disaster and a costly waste of time.

## Deployment issues can cause a project to fail.

## ADDING FLEXIBILITY TO DEVELOPMENT

Traditional project management says that you must manage three dimensions of a project: scope (requirements), resources (people, equipment, and money), and time. The dimensions are interdependent; hence, a broader project scope requires more resources or time. Less time requires more resources or a smaller scope.

Project managers don't often consider dealing with risk on a par with these three dimensions, but it should be. If a platform does not work as the team expects, both time and resource needs could balloon if the team is locked into maintaining the scope. The extreme and sudden effect of risk argues for making risk management another leg of the stool supporting successful project management—equal in importance to resources, scope, and time. Thus, each leg represents a dimension that can greatly influence the project, and effective project management must consider all four legs simultaneously for the life of the project.

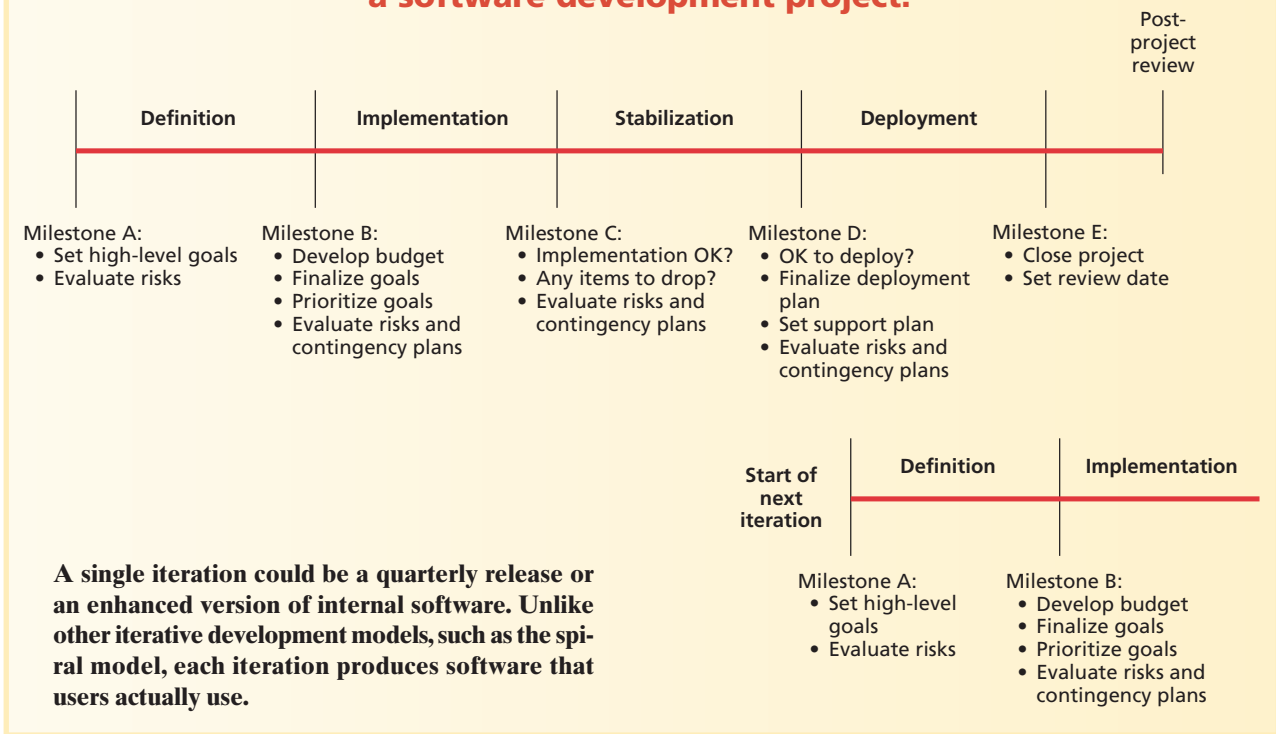
If risk management is to be an integral part of managing a software development project, then traditional development must become more flexible so that the team can deal with risks if they arise. One way to achieve flexibility is to break the project into a series of subprojects. Any project that lasts more than a few months is a candidate. These subprojects effectively form a series of small-project iterations, each of which adds functionality to deliverables from the previous iteration. Each iteration should last at most two to three months.

I propose using an iterative model that follows these ideas. Figure 1 shows the phases of a single iteration, which could be from eight to 12 weeks. The concept of iterations with increasing functionality borrows from the spiral development model that Barry Boehm and others proposed in the late 1980s. A similar focus on iterative development is in the Rational Unified Process (RUP) and the more recent agile methods.

The spiral model and my iterative model differ slightly, however. In the spiral model, iterations build on functionality but do not necessarily end with usable software after each subproject in the iteration. In my model, a single iteration is itself a project, so the result is software that users will actually use. Thus, the concept is more like agile methods, where the focus is on the early release of usable software. If the team produces commercial software, a single iteration would be a quarterly release. For a large in-house project, a single iteration will release a new and enhanced version of the software to internal users. Real feedback is essential to improving the next release.

In my experience, most projects, even small ones, go through four phases: definition, implementation, stabilization, and deployment. The iterative development models in agile methods recognize the first three phases, but most

**Figure 1. Four stages of a single iteration within a software development project.**



ignore deployment. This is shortsighted because deployment issues, such as an inadequate infrastructure or unprepared support team, can cause a project to fail. Moreover, killing a project at this stage is perhaps the most costly and demoralizing: The team has seen the software through all its implementation issues, but cannot see the fruit of all that work. The model I propose includes a deployment phase in each iteration of each subproject, as Figure 1 shows.

My model is suitable for both software development and maintenance. It might be overkill for smaller maintenance tasks, but it will work for any maintenance activity that takes more than a few days. In some cases, such as a release to fix a major bug, the team may have to collect a series of unrelated tasks to form the project's activities. For example, some bugs may require fixes to the installer; others will require fixes to the documentation; and still others to the configuration. All these might take more than a few weeks to accomplish. Treating these maintenance activities as a project with deliverables, risk management, and reporting will help reduce problems and delays. Of course, the lengths of each phase could vary with an organization's and project's needs. For example, the stabilization phase could correspond to the beta-test phase for a software company, which could last a few weeks to a month or more. For an internal IT project, the stabilization phase might last only a few days.

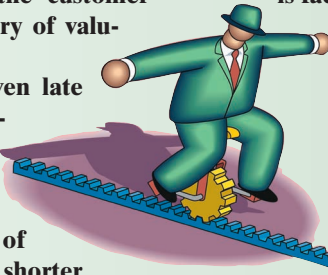
The team can start the next iteration as soon as they complete the stabilization phase for the previous one. In such cases, managers must ensure that this previous iteration has enough time and resources to support deployment issues. The team can use feedback from deployment issues to improve the product in the next iteration's implementation phase.

### Phase 1: Definition

Each iteration starts with an initial meeting of major players to decide on project goals or high-level requirements for the iteration and to define initial priorities and possible risks. Project team members then investigate what they require to accomplish these goals. They get rough estimates of work required, collect more information on risks, plan the next iteration, and evaluate any new technologies for potential problems. For the first iteration, the definition phase can be more elaborate because the team must still define major goals and decide on the sequence and length of subsequent iterations.

Remember that the iteration must provide usable software, not a demo. Prototypes can help you better understand project needs, but they should be as simple as possible and created mainly within this phase. Elaborate prototypes and demos eat up time and resources that the team could use during the implementation phase; they often create unrealistic expectations or requirements. Unless properly targeted, these elaborate prototypes fail to uncover potential problems with the technologies to be used, interfaces, and so on. More than once I have seen customers seduced by a prototype and then focus on defining must-have look-and-feel items like complex data entry forms and cool widgets. When the realities of implementing the application become more obvious, the team must frequently drop these items, thus dashing the customer's expectations and wasting time. The phase ends with a meeting to finalize the iteration's goals, decide on the goal priorities, evaluate risks, and accept contingency plans. The team usually has more goals than appear feasible for the iteration. Flag goals that look unlikely in light of time and

## Principles Behind the Agile Manifesto

- 
- ▶ Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
  - ▶ Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
  - ▶ Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
  - ▶ Business people and developers must work together daily throughout the project.
  - ▶ Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
  - ▶ The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
  - ▶ Working software is the primary measure of progress.
  - ▶ Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
  - ▶ Continuous attention to technical excellence and good design enhances agility.
  - ▶ Simplicity—the art of maximizing the amount of work not done—is essential.
  - ▶ The best architectures, requirements, and designs emerge from self-organizing teams.
  - ▶ At regular intervals, the team reflects on how to become more effective, and then tunes and adjusts its behavior accordingly.

resources and treat them as something to attempt if time permits; otherwise, they should carry over into the next iteration. Make sure that everyone reviews and accepts contingency plans, because they represent an alternative path for the project team if the expected risk occurs.

### Phase 2: Implementation

Team members work closely with each other and with user representatives to flesh out, implement, and test the requirements defined in the previous phase. This is a good time to use one of the agile methods such as Extreme Programming, Scrum, and feature-driven development, customizing practices as needed. When customizing, remember the principles of agility described in the “Principles Behind the Agile Manifesto” sidebar.

The team works on the requirements in order of priority. A weekly demo of the evolving software to interested parties helps catch potential problems during this phase. I have often seen stakeholders surprised by what they see. Sometimes they have not recognized a problem until they see it, or a miscommunication becomes clear in the demo. On one of my projects, stakeholders realized during a product demo that major pieces required to administer and configure the application were missing. No one had ever clearly defined them, and the developers never realized this problem, because they always used the database to make the changes when they needed to test the application. These demos also help build trust because stakeholders see the team's timely response to their requests. Demos also build a sense of urgency, as groups see the project evolve toward its goals. Stakeholders start taking care

of items they are responsible for without waiting until the last minute. In one case, stakeholders realized they had to finalize the product name soon as they kept seeing an unsuitable working name on the screens and in documentation.

Regular testing should also occur during this phase. Because my model is limited by time, rather than requirements, the team must be able to switch to prior software builds if some functionality will not be usable by the cut-off date. This means that the team should do regular builds and have effective version control. Best practices from agile methods can be very useful. Doing daily builds can catch problems early. Daily builds, in turn, require the team to have a working configuration management system as soon as possible. Project teams should report their goals and risk status to stakeholders weekly. Stakeholders should also have a time and place to raise issues they see as risks. Stakeholders are a valuable source of identifying risk because their universe is typically larger than that of the team members. The stakeholder in the professional-services group who had raised important security-related questions, for example, was passing on what he had learned from customer visits.

This phase ends with a review meeting to see if the implementation is adequate and complete.

### Phase 3: Stabilization

In this phase, the software undergoes extensive quality assurance tests, and the team fixes major bugs. Many more users use the software. Documentation and support plans become final. This phase is crucial because the team will

have a truly representative sample of users. On a project to create a new product, we found that the install program did not work properly on dual-processor machines from a well-known manufacturer. We had not discovered this during implementation, despite extensive tests. These machines were fairly new, and we soon found that much of our customer base was planning to upgrade to them. The stabilization phase gave us the opportunity to make fixes before we went into highly expensive disk duplication and documentation printing.

This phase ends with a review meeting to see if the software is stable and ready for deployment. When stabilization ends, a new iteration can begin in parallel with the next phase of the old one.

#### Phase 4: Deployment

In this phase the team deploys, or releases, the software to its users. Although it is hard to pinpoint the end of this phase, since a company can continue to deploy the software long after its first release, this phase is generally over once all stakeholders get together for a final project acceptance meeting. When the project is to build subcontracted applications for an external customer, the acceptance meeting is a crucial milestone for the release of payments. It may also represent the end of a support period.

As part of the meeting, it is a good idea to set a date in the immediate future for the post-project review, in which the team will go over everything, good and bad, about the project with an eye toward improving the next one. The two meetings address different needs, so it is best to keep them separate. The acceptance meeting marks the acceptance of the project deliverables and closes the project. The post-project review is primarily to look at lessons learned and future improvements in project management.

Unexpected risks can cause major delays and escalating costs. Preventive risk management can help reduce unpleasant surprises, and I have seen it implemented successfully in many instances. Flexible development practices using proactive risk management make it easier to plan for and handle risks as they occur. To implement flexible development practices, companies must improve team skills, and review and improve their processes for software definition, development, and deployment. They must also put in place the management systems to effectively plan, measure, and share information about software definition, development, and delivery. With these practices, they can deliver better applications with fewer delays in spite of unexpected problems and the many requirements and strategy changes that can happen along the way. ■

*Sanjay Murthi is president of SMGlobal Inc., a firm that helps companies review and improve their software definition, development, and delivery. Contact him at [smurthi@smglobal.com](mailto:smurthi@smglobal.com).*

## Innovative New IT Titles

### The Elements of UML Style

Scott W. Ambler  
0-521-52547-0, Paperback, \$15.00

### Anytime, Anywhere

Entrepreneurship and the  
Creation of a Wireless World  
Louis Galambos and  
Eric John Abrahamson  
0-521-81616-5, Hardback, \$29.00

### The Simplicity Shift

Innovative Design Tactics in a  
Corporate World  
Scott Jensen  
0-521-52749-X, Paperback, \$28.00

### UML Xtra-Light

How to Specify Your Software  
Requirements  
Milan Kratochvil and  
Barry McGibbon  
0-521-89242-2, Paperback, \$21.00

### Numerical Recipes in C++

The Art of Scientific Computing  
Second Edition  
William H. Press, Saul A. Teukolsky, William T. Vetterling, and  
Brian P. Flannery  
0-521-75033-4, Hardback, \$70.00

### Numerical Recipes in C and C++ Source Code CDROM with Windows, DOS, or Macintosh Single Screen License

The Art of Scientific Computing  
Second Edition  
0-521-75037-7, CD-ROM, \$50.00

### Numerical Recipes Example Book [C++]

0-521-75034-2, Paperback, \$35.00

### Numerical Recipes Multi-Language Code CDROM with LINUX or UNIX Single Screen License

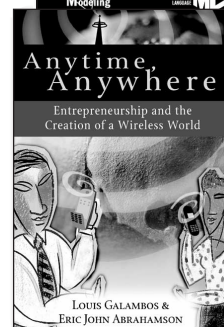
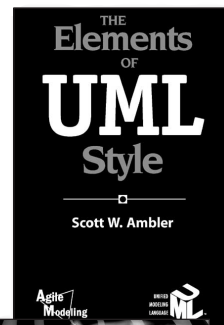
Source Code for Numerical Recipes in C, C++, Fortran 77,  
Fortran 90, Pascal, BASIC, Lisp and Modula 2 plus many extras  
0-521-75036-9, CD-ROM, \$150.00

### Numerical Recipes Multi-Language Code CDROM with Windows, DOS, or Macintosh Single Screen License

Source Code for Numerical Recipes in C, C++, Fortran 77,  
Fortran 90, Pascal, BASIC, Lisp and Modula 2 plus many extras  
0-521-75035-0, CD-ROM, \$90.00

### Business Services Orchestration

The Hyper-Tier of Information Technology  
Waqar Sadiq and Felix Racca  
0-521-81981-4, Hardback, \$45.00



Available in bookstores or from



CAMBRIDGE  
UNIVERSITY PRESS

800-872-7423

[us.cambridge.org/computerscience](http://us.cambridge.org/computerscience)