Using Risk to Balance Agile and Plan-Driven Methods

A tailorable, risk-based approach lets project developers enjoy the benefits of both agile and plan-driven methods, while mitigating many of their drawbacks.

Barry Boehm
University of Southern California

Richard Turner
George Washington University

Methodologies such as Extreme Programming (XP), Scrum, and agile software development promise increased customer satisfaction, lower defect rates, faster development times, and a solution to rapidly changing requirements. Plan-driven approaches such as Cleanroom, the Personal Software Process, or methods based on the Capability Maturity Model promise predictability, stability, and high assurance.

However, both agile and planned approaches have situation-dependent shortcomings that, if left unaddressed, can lead to project failure. The challenge is to balance the two approaches to take advantage of their strengths in a given situation while compensating for their weaknesses.

We present a risk-based approach for structuring projects to incorporate both agile and plan-driven approaches in proportion to a project's needs. We drew this material from our book, *Balancing Agility and Discipline: A Guide for the Perplexed*, to be published in 2003.[1]

METHOD OVERVIEW

Our method uses risk analysis and a unified process framework to tailor risk-based processes into an overall development strategy. This method enhances the ability of key development team members to understand their environment and organizational capabilities and to identify and collaborate with the project's stakeholders.

We use risk analysis to define and address risks particularly associated with agile and plan-driven methods. The Risk-Based Spiral Model Anchor Points[2] provide the framework for this process. Both the Rational Unified Process[3] (RUP) and the Model-Based Architecting and Software Engineering (Mbase) process[4] have adopted these anchor points, which are essentially an integrated set of decision criteria for stakeholder commitment at specific points in the development process. Our method consists of five steps.

Step 1

First, we apply risk analysis to specific risk areas associated with agile and plan-driven methods. We identify three specific risk categories: environmental, agile, and plan-driven.

While not a simple task, Step 1 provides the basis for making decisions about the development strategy later in the process. If we uncover too much uncertainty about some risk categories, spending resources early to buy information about the project aspects that create the uncertainty may prove prudent.

The candidate risks we describe are just that—candidates for consideration. They are neither complete nor always applicable, but serve as guides to stimulate participants' thinking.

Step 2

Next, we evaluate the risk analysis results to determine if the project at hand is appropriate for either purely agile or purely plan-driven methods. In these cases, the project characteristics fall squarely in the home ground of one approach or

0018-9162/03/$17.00 © 2003 IEEE Published by the IEEE Computer Society

Agile and Plan-Driven Home Grounds and Environmental Dimensions

The home grounds for the agile and plan-driven methods encompass the sets of conditions under which they are most likely to succeed. The more a particular project's conditions differ from the home ground conditions, the more risk in using one approach in its pure form and the more valuable blending in some of the opposite method's complementary practices becomes.

Table A describes the home grounds we have observed.

Analysis of these home grounds and the general characteristics of agile and plan-driven methods led us to define the five critical factors that describe a project environment and help determine method balance, as Table B shows.

As shown, these factors can be summarized graphically in the polar chart shown in Figure A. Of the five axes, Size and Criticality represent the factors Alistair Cockburn uses to distinguish between the lighter-weight Crystal methods[1] toward the graph's center and the heavier-weight Crystal methods that appear toward the periphery. The Culture axis reflects that agile methods will succeed better in a culture that "thrives on chaos"[2] than in one that "thrives on order," while the opposite is true for plan-based methods.

The other two axes are asymmetrical in that both agile and plan-driven methods will likely succeed at one end, while only one of them will likely succeed at the other. For dynamism, agile methods work well with both high and low change rates, but plan-driven methods work best with low change rates. The personnel scale refers to the extended Cockburn method skill rat-

Table A. Agile and plan-driven home grounds.

Project characteristics	Agile home ground	Plan-driven home ground
Application		
Primary goals	Rapid value, responding to change	Predictability, stability, high assurance
Size	Smaller teams and projects	Larger teams and projects
Environment	Turbulent, high change, project focused	Stable, low change, project and organization focused
Management		
Customer relations	Dedicated onsite customers, focused on prioritized increments	As-needed customer interactions, focused on contract provisions
Planning and control	Internalized plans, qualitative control	Documented plans, quantitative control
Communications	Tacit interpersonal knowledge	Explicit documented knowledge
Technical		
Requirements	Prioritized informal stories and test cases, undergoing unforeseeable change	Formalized project, capability, interface, quality, foreseeable evolution requirements
Development	Simple design, short increments, refactoring assumed inexpensive	Extensive design, longer increments, refactoring assumed expensive
Test	Executable test cases define requirements, testing	Documented test plans and procedures
Personnel		
Customers	Dedicated, colocated Crack* performers	Crack* performers, not always colocated
Developers	At least 30% full-time Cockburn Level 2 and 3 experts; no Level 1B or Level –1 personnel**	50% Cockburn Level 3s early; 10% throughout; 30% Level 1B's workable; no Level –1s**
Culture	Comfort and empowerment via many degrees of freedom (thriving on chaos)	Comfort and empowerment via framework of policies and procedures (thriving on order)

* Collaborative, representative, authorized, committed, and knowledgeable.

** See the "Cockburn's Three Levels of Software Understanding, Slightly Revised" sidebar. These numbers will vary with the application's complexity.

the other, as described in the "Agile and Plan-Driven Home Grounds and Environmental Dimensions" sidebar, so we proceed to Step 4.

Step 3

We move to this step if our risk analysis shows that the project has no clear agile or plan-driven home ground. It also applies to cases in which parts of the system have such different risks that they fall into different home grounds.

If possible, the project team develops an architecture that supports using agile methods where their strengths can be best applied and their risks minimized. Plan-driven methods perform the remainder of the work and are considered the default when no suitable architecture can be created. This step can sometimes uncover a new risk or opportunity that requires backtracking to an earlier step.

Step 4

In this step, we focus on developing an overall project strategy that addresses the identified risks. This involves identifying risk resolution strategies for each risk and integrating them.

The specifics of the process will depend primarily on the developer organization's capabilities and experience in the general application area. A successful and experienced development team will have highly capable people responsible for defining, designing, developing, and deploying the application. Such a team could also take advantage of reusable process assets and product patterns to establish the strategy. Less experienced in-house or

Table B. The five critical agility and plan-driven factors.

Factor	Agility discriminators	Plan-driven discriminators
Size	Well matched to small products and teams; reliance on tacit knowledge limits scalability.	Methods evolved to handle large products and teams; hard to tailor down to small projects.
Criticality	Untested on safety-critical products; potential difficulties with simple design and lack of documentation.	Methods evolved to handle highly critical products; hard to tailor down efficiently to low-criticality products.
Dynamism	Simple design and continuous refactoring are excellent for highly dynamic environments, but present a source of potentially expensive rework for highly stable environments.	Detailed plans and "big design up front" excellent for highly stable environment, but a source of expensive rework for highly dynamic environments.
Personnel	Require continuous presence of a critical mass of scarce Cockburn Level 2 or 3 experts; risky to use nonagile Level 1B people.	Need a critical mass of scarce Cockburn Level 2 and 3 experts during project definition, but can work with fewer later in the project—unless the environment is highly dynamic. Can usually accommodate some Level 1B people.
Culture	Thrive in a culture where people feel comfortable and empowered by having many degrees of freedom; thrive on chaos.	Thrive in a culture where people feel comfortable and empowered by having their roles defined by clear policies and procedures; thrive on order.

ing scale described in the "Cockburn's Three Levels of Software Understanding, Slightly Revised" sidebar and places it in a framework relative to the application's complexity. This captures the situation in which a developer might be a Level 2 in an organization developing simple applications, but a Level 1A in an organization developing highly complex applications. Here, the asymmetry is that while plan-driven methods can work well with both high and low skill levels, agile methods require a richer mix of higher-level skills.

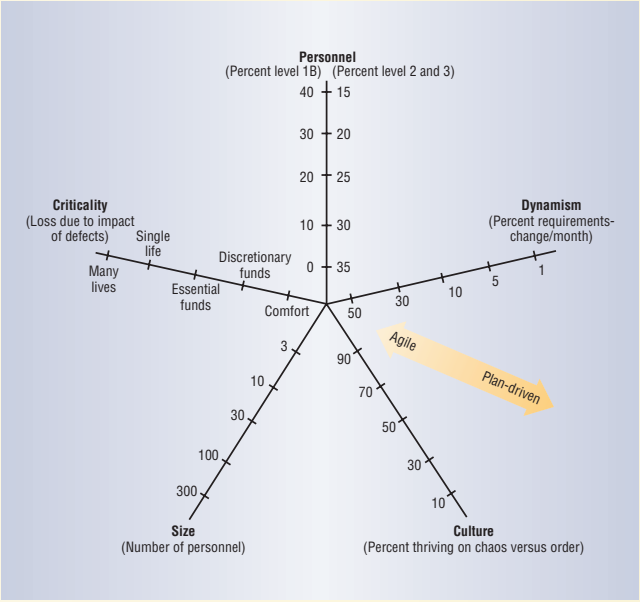

Figure A. Polar chart. The five axes represent the factors we use to distinguish between the lighter-weight agile methods toward the graph's center and the heavier-weight plan-driven methods that appear toward the periphery.

References

1. A. Cockburn, *Agile Software Development*, Addison-Wesley, 2002.
2. T. Peters, *Thriving on Chaos*, HarperCollins, 1991.

external developers must perform additional learning-curve and asset-buildup activities to ensure success.

We advocate using the Life Cycle Architecture anchor point milestone criterion[2] to exit from Step 4.

Step 5

No decision is ideal for all time and, as this step indicates, the management team must constantly monitor and evaluate the performance of its selected processes while keeping an eye on the environment.

This step resembles the agile practice of *reflection*. If a process indicates some strain, developers must backtrack, revalidate, and perhaps adjust the levels of the agile or plan-driven methods established initially.

Adjustments should be made as soon as strain arises. On a more positive note, monitoring can also identify opportunities to improve value to the customer, shorten time to delivery, and improve stakeholder involvement.

The flowchart in Figure 1 summarizes these five steps.

A SAMPLE APPLICATION FAMILY

When illustrating the practical application of our risk-based method, we first establish a realistic context by introducing a family of representative current and future software applications. For each of these three representative systems, the project risks suggest using a different mix of agile and plan-driven process components.

Cockburn's Three Levels of Software Understanding, Slightly Revised

Drawing on the three levels of understanding in the martial art Aikido—Shu-Ha-Ri—Alistair Cockburn identified three levels of software method understanding that can help sort out what people with various skill levels can be expected to do within a given method framework.[1]

Table C. Levels of software method understanding and use.

Level	Characteristics
3	Able to revise a method, breaking its rules to fit an unprecedented new situation.
2	Able to tailor a method to fit a precedented new situation.
1A	With training, able to perform discretionary method steps such as sizing stories to fit increments, composing patterns, compound refactoring, or complex COTS integration. With experience, can become Level 2.
1B	With training, able to perform procedural method steps such as coding a simple method, simple refactoring, following coding standards and CM procedures, or running tests. With experience, can master some Level 1A skills.
–1	May have technical skills, but unable or unwilling to collaborate or follow shared methods.

We have taken the liberty of splitting Cockburn's Level 1 into Levels 1A, 1B, and –1 to address distinctions between agile and plan-driven methods and deal with the problem of method disrupters, as Table C shows. The Level –1 people should be rapidly identified and transferred to other work.

Level 1B people function well in performing straightforward software development in a stable situation, but will likely slow an agile team trying to cope with rapid change, particularly if they form a majority of the team. The 1Bs can form a well-performing majority in a stable, well-structured, plan-driven team.

Level 1A people can function well on agile or plan-driven teams that have enough Level 2 people available to guide them. When agilists refer to being able to succeed on agile teams with ratios of five Level 1 people per Level 2 person, they are generally referring to Level 1A people.

Level 2 people can function well in managing a small, precedented agile or plan-driven project but need the guidance of Level 3 people on a large or unprecedented project. Some Level 2s can become Level 3s with experience, others cannot.

Reference

1. A. Cockburn, *Agile Software Development*, Addison-Wesley, 2002.

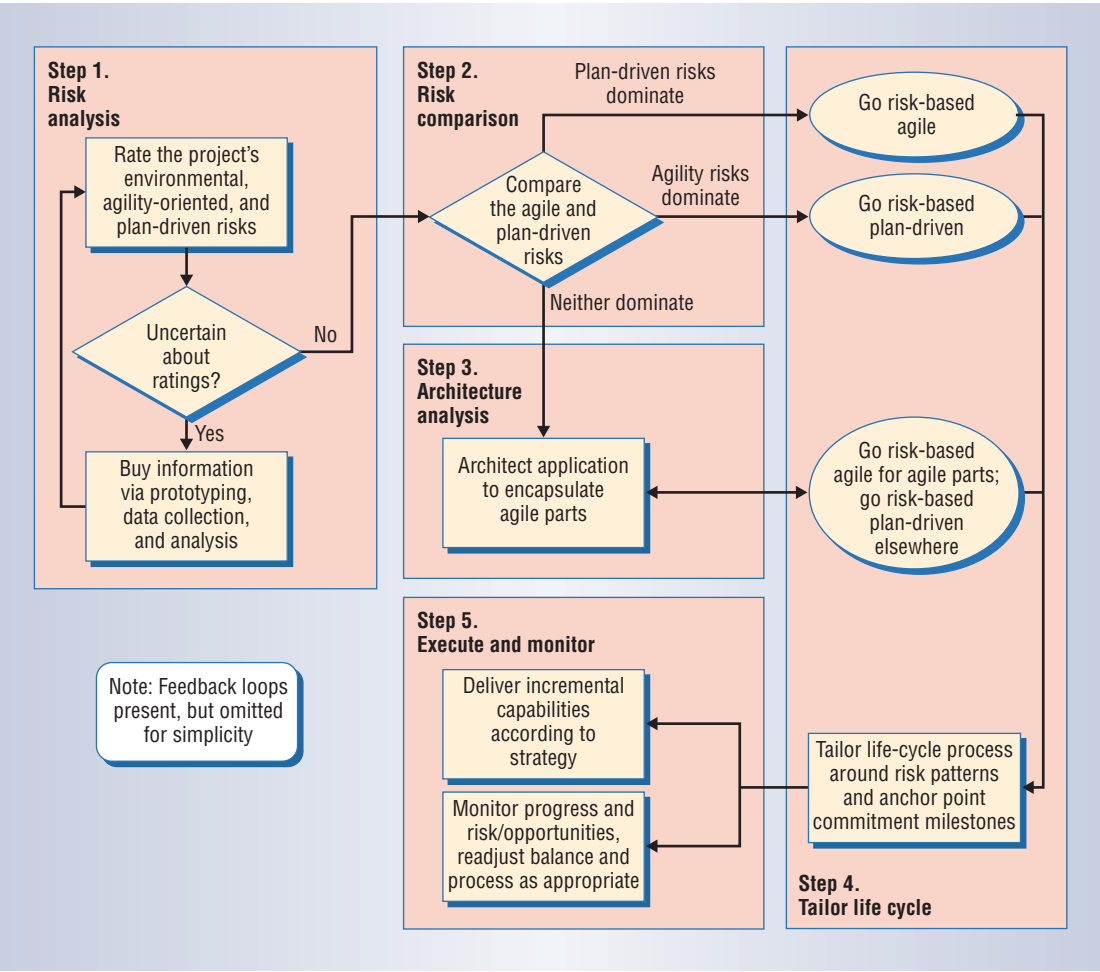

Figure 1. Risk-based method summary. Developers can use a five-step process to determine if agile methods, plan-driven methods, or a combination of the two will work best for their project.

Table 1. Summary of three sample agent-based planning system applications.

Application	Team size	Team type	Failure risks	Clients	Requirements	Architecture	Refactoring	Primary objective
Event planning	5	In-house venture startup, colocated	Venture capital, manual effort	Single, colocated, representative	Goals generally known, details emergent	Provided by single COTS package	Inexpensive with skilled people	Rapid value
Supply-chain management	50	Distributed, often multiorganization	Major business losses	Multiple success-critical stakeholders	Some parts relatively stable, others volatile, emergent	Provided by small number of COTS packages	More expensive, with mix of people skills	Rapid value increase, dependability, adaptability
National crisis management	500	Highly distributed, multiorganization	Large loss of life	Many success-critical stakeholders	Some parts relatively stable, others volatile, emergent	System of systems, many COTS packages	Feasible only within some subsystems	Rapid response, safety, security, scalability, adaptability

Agent-based planning systems[5] are emerging software applications that involve using software agents to

- search for and locate desired information across a network;
- analyze the information and determine choices such as best-buy recommendations;
- develop plans for implementing a course of action involving the chosen elements, including functions such as dependency analysis and constraint satisfaction; and
- monitor implementation of the plans to identify potential difficulties in realizing them.

Agent-based planning systems can provide significant improvements in operational efficiency, human-error reduction, speed of execution, adaptability to changing situations, and support of complex collaborative enterprises.

On the other hand, research and early applications are still addressing several agent-based systems risks. These risks include verification and validation of agent behavior, scalability of multiagent behavior, commonsense reasoning about bad data or unexpected events, and the ability to degrade gracefully versus failing catastrophically.

Given these characteristics, balancing risks and opportunities in agent-based systems is challenging, with the difficulty varying considerably depending on the application's scale and criticality. To explore the full range of this challenge, we chose the following three representative applications:

- *Small, relatively noncritical.* This agent-based planning system for managing events such as conferences or conventions is based on risk patterns observed in small Web-services applications.
- *Intermediate.* An agent-based planning system for supply-chain management across a network of producers and consumers, this application is based on risk patterns derived from the ThoughtWorks experience with scaling up XP techniques to a 50-person project in a lease-management application.[6]
- *Very large, highly critical.* This agent-based planning system for national crisis management is based on risk patterns observed in the US Defense Advanced Research Project Agency and the US Army Future Combat Systems program—an agent-oriented, network-centric system of systems being developed by more than 2,000 people.

Table 1 summarizes each application with respect to the major agile and plan-driven homeground characteristics.

We focus on the intermediate application and use it to examine how the smaller and larger applications' risk patterns determine different balances of agile and plan-driven methods.

INTERMEDIATE APPLICATION

SupplyChain.com is a commercial software house that specializes in developing turnkey agent-based supply-chain management systems for, and in collaboration with, large manufacturing companies. These client companies generally work with complex networks of suppliers feeding their manufacturing processes and distributors dispensing their manufactured products. SupplyChain.com's experience has made it a leader in this area, but its market sector is too dynamic and driven by customer-specific considerations to allow much use of prebuilt plug-and-play application components.

As Table 1 shows, SupplyChain.com's applications typically involve distributed, multiorganization teams of about 50 people. Its primary objective is to provide a rapid increase in value to the manufacturing company through increases in supply-chain speed, dependability, and adaptability. Parts of SupplyChain.com's applications are relatively stable; others are highly volatile. A few key COTS packages drive their architectures, but they also evolve continually. The risk of system failure involves major business losses.

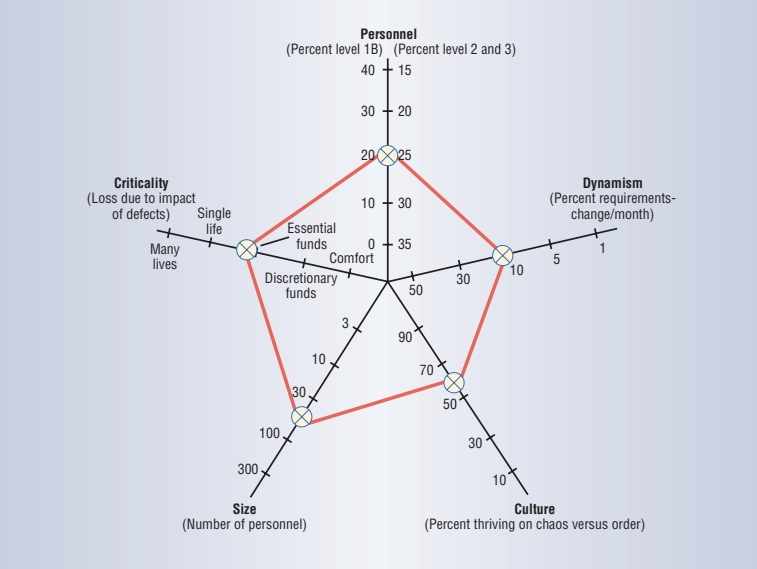

Figure 2. Home ground chart for SupplyChain.com, an intermediate-sized, agent-based application.

Clearly, the program has agile aspects such as rapid value and volatility, as well as plan-driven aspects such as scalability and criticality. Thus, we can apply our five-step process to determine an appropriate strategy for this project.

Project risk ratings

The project team begins Step 1 of its balancing analysis by assessing the major sources of SupplyChain.com's environmental, agile, and plan-driven risks.

Environmental risks. The project faces significant technical risks, including uncertainties about agent-based systems and the existence of many separately evolving supplier and distributor networks that must be coordinated.

Agile risks. Reconciling a 50-person project's inherent diseconomies of scale with the desire to maintain system dependability across constant-increment delivery intervals is difficult. Reconciling the desire of agile developers to follow the "you aren't going to need it" (Yagni) principle and simple design directives with the knowledge that parts of the application will be stable and would thus benefit from anticipatory architectures will also be difficult. SupplyChain.com's relatively stable workforce makes it relatively unlikely that key personnel turnover will disrupt the project's reliance on tacit knowledge.

Plan-driven risks. These risks largely involve incurring extensive delays through rework of elaborate plans and specifications in a rapidly evolving, time-critical marketplace. Adapting elaborate plans and specifications to rapid changes in technology, organizations, and market conditions would probably be slow and expensive. The inability to deliver new capabilities rapidly enough to keep pace with the competition can lead to loss of market share. Overreliance on prespecified requirements in areas where requirements can emerge through user familiarization and experience can affect the project architecture and plans unpredictably.

Compare agile and plan-driven risks

In Step 2, the team first determines whether agile or plan-driven risks dominate the project. Figure 2 shows how SupplyChain.com looks on a home-ground critical-factor polar chart.

The selective application of plan-driven planning and architecting techniques can address the primary agile risks of scalability, criticality, and simple design. The selective application of agile methods within an overall plan-driven framework could address the combined plan-driven risks of rapid change, need for rapid response, and emerging requirements. Either an experienced agile team or a suitably agile RUP or Team Software Process team would be most likely to succeed in implementing this strategy.

Given SupplyChain.com's culture and environment, the team decides to apply a risk-based agile approach. If, however, the company's environment included a more stable, well-understood, financially critical marketplace and a high rate of agile personnel turnover, the risk-based plan-driven approach would be preferable.

Individual risk-resolution strategies

Because Step 2 resulted in a risk-based agile decision, the project team can bypass Step 3 and begin Step 4 by identifying a resolution strategy for each risk.

Many separately evolving networks. Proceeding too rapidly, without the involvement of success-critical stakeholders, presents a major risk. Concentrating on supply-chain logistics management without considering financial stakeholders and concerns is an example of such a risk. Techniques such as the DMR Consulting Group's Results Chain[7] help identify missing success-critical initiatives and stakeholders.

Accepting unqualified stakeholder representatives as team members presents another major risk. Stakeholder representatives should be *Crack* performers: *collaborative*, *representative*, *authorized*, *committed*, and *knowledgeable*. Shortfalls in any of these capabilities can lead to frustration, delay, and wasted project effort, not to mention an unacceptable product. Generally, project teams benefit more from a part-time Crack performer than from a full-time non-Crack performer.

A third major risk involves the difficulty of coordinating interface protocols among dynamically evolving supplier and distributor networks with different internal-evolution strategies. Establishing

mutual commitments to relatively stable interface protocols and continuing to monitor and evolve mutually satisfactory changes requires considerable up-front effort.

Technical uncertainties. The set of technical risks associated with agent-based systems provides a major source of uncertainty. Before committing to a specific agent-coordination approach or package, the project team must determine critical agent and agent-coordination objectives, constraints, and priorities, then use them in evaluating the agent technology candidates via appropriate combinations of reference checking, analysis, benchmarking, and prototyping.

The set of technical risks associated with the performance and interoperability of COTS packages used across the supply chain's stakeholder organizations presents a similar source of uncertainty. The team needs similar risk-assessment strategies to establish satisfactory COTS package combinations. For both these technical uncertainty areas, the team must maintain a continuous watch to assess emerging technology risks and opportunities.

Diseconomies of scale. Most agilists with whom we have discussed this issue blanch at the idea of lengthening the release interval as the application gets larger and the effort required to develop and integrate new *stories*—scenarios describing desired operational capabilities—increases. But something must be done to avoid the problems with integration schedules and the strain on shared responsibility and tacit knowledge that come with the increasing number of stories implemented and the lengthening refactoring times.

One approach that may help conserve the release interval is to make later stories more granular and amenable to timeboxing. However, in larger, multiteam applications, this requires further coordination to ensure that one team's application component does not need features dropped by another team.

Foreseeable change versus simple design. The supply-chain application will have some foreseeable sources of change, such as new data and operations involved in supply-chain transactions. Developers can use factory patterns to create a framework for accommodating such sources much more readily than with simple design and continuous refactoring.

More generally, developers can handle foreseeable sources of change architecturally by using the information-hiding technique of encapsulating the change sources within individual modules,[8] thus eliminating the ripple effects that such changes cause.

Personnel turnover and tacit knowledge loss. Within the development team, rotating people across programming pairs or across feature teams can reduce tacit knowledge losses. The loss of key supply-chain stakeholder representatives can be equally critical, but keeping alternate representatives involved can reduce the impact of such losses.

SupplyChain.com and its operational stakeholders reduced the probability of loss as well as the size of loss due to personnel turnover by establishing significant bonuses for successful project completion.

Prespecified plans and specifications. Delays and reduced competitiveness can result from relying on elaborate prespecified plans and specifications. One effective approach to dealing with the risks of overspecification uses a risk-based approach to specifications:[9]

- *Don't* write specifications when the risk of not using them is low and the risk of using them is high. A good example is a written graphical user interface specification. With a good GUI-builder tool, the risk of not writing specifications is low. With extensive GUI iterations, the risk of expensive specification rework is high.
- *Do* write specifications when the risk of using them is low and the risk of not using them is high. A good example is a written supply-chain interface protocol specification achieved through extensive stakeholder negotiations.

A similar approach can apply to risk-based process plans. For the various organizations making commitments to the timing of introducing new operational interfaces and business workflows, not developing top-level milestone plans and critical-path dependencies would be risky. On the other hand, with an experienced team and rapidly changing circumstances, using a heavyweight earned-value management system would add more risk of delay than it would subtract.

Risk-based testing is another area in which planning can save downstream time and effort. Most projects spend as much time and effort testing the low-risk parts of the code as they spend testing the high-risk parts. Focusing the test effort on the high-risk parts—agent coordination, potentially critical failure modes, and highest-value business workflows—can generate project time and effort savings and avoid high-cost operational losses.

Focusing test effort on the high-risk parts—agent coordination, potentially critical failure modes, and highest-value business workflows—can generate project time and effort savings.

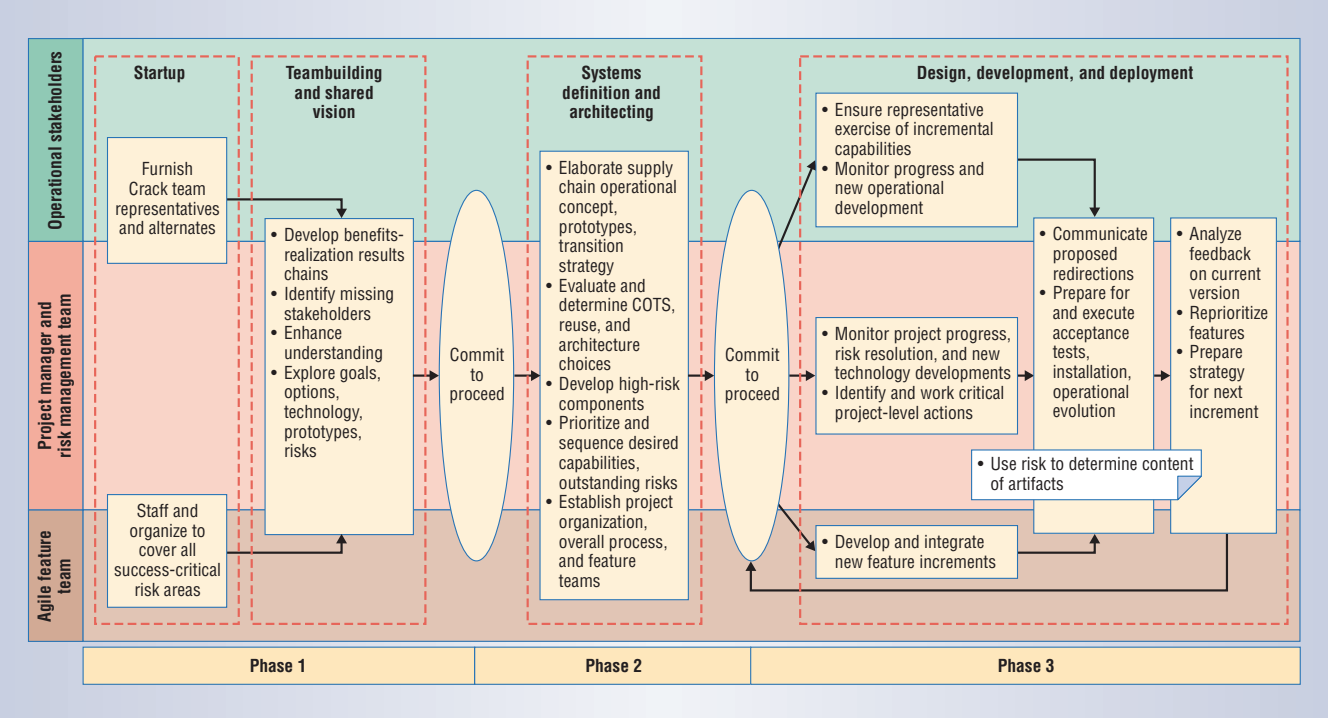

Figure 3. Overall SupplyChain.com project strategy. Horizontal swim lanes describe participant groups, rectangles overlaying lanes show the project activities of the participant groups, shaded areas identify phases, and ovals show decision points.

Risk-based strategy for system development

Continuing with Step 4, the project team implements the overall project strategy summarized in Figure 3. This strategy integrates the individual risk resolution approaches. Figure 3 uses horizontal swim lanes to describe participant groups, rectangles overlaying lanes to show the project activities of the participant groups involved, shaded areas to identify phases, and ovals to show decision points—the spiral anchor point milestones.[2] In this case, there are three primary participant groups:

- The operational stakeholders include a largely dedicated manufacturing-company representative, a part-time supplier-company representative, and a part-time distributor-company representative. Each is a Crack performer who has an alternate to ensure representation continuity. These stakeholders can also call on as-needed specialists from other parts of the organizations they represent.
- The SupplyChain.com project manager and three Cockburn Level 2 or 3 SupplyChain.com staff members form a risk-and-opportunity management team that maintains a continuous watch for emerging project risks or opportunities. When such risks or opportunities arise, the risk management team initiates actions such as prototyping, COTS evaluation, change-proposal evaluation, or complex refactoring to explore options and develop an appropriate response to the risk or opportunity. When no major risks or opportunities are active, these staff members contribute to one of the development teams—often in a mentoring role.
- A group of agile feature teams operate concurrently to develop the features involved in each increment of system capability. The teams primarily focus on particular application areas such as supplier or distributor transactions and manufacturing support. Their team leaders are Cockburn Level 2 personnel. The team members are mixes of Level 2 and Level 1A personnel, with no Level 1B personnel. Personnel rotate across teams. There will be a ramp-up of team members from the manufacturing company when it eventually takes over continuing software development.

The risk patterns for projects of this scale and complexity make it best to organize them into three primary phases, with team leaders representing their teams in the first two phases.

Phase 1, roughly corresponding to a RUP or Mbase inception phase, involves rapid teambuilding and development of a shared system vision among the stakeholders. Techniques used in this phase include prototyping key user features, COTS evaluation, brainstorming, results-chain development, and negotiation of a mutually satisfactory set of strategic project objectives and priorities.

Phase 2, roughly corresponding to a RUP or Mbase elaboration phase, establishes

- the overall operational concept and life-cycle strategy;
- a set of key COTS, reuse, and architecture decisions, including development of high-risk components;
- a full project organization; and

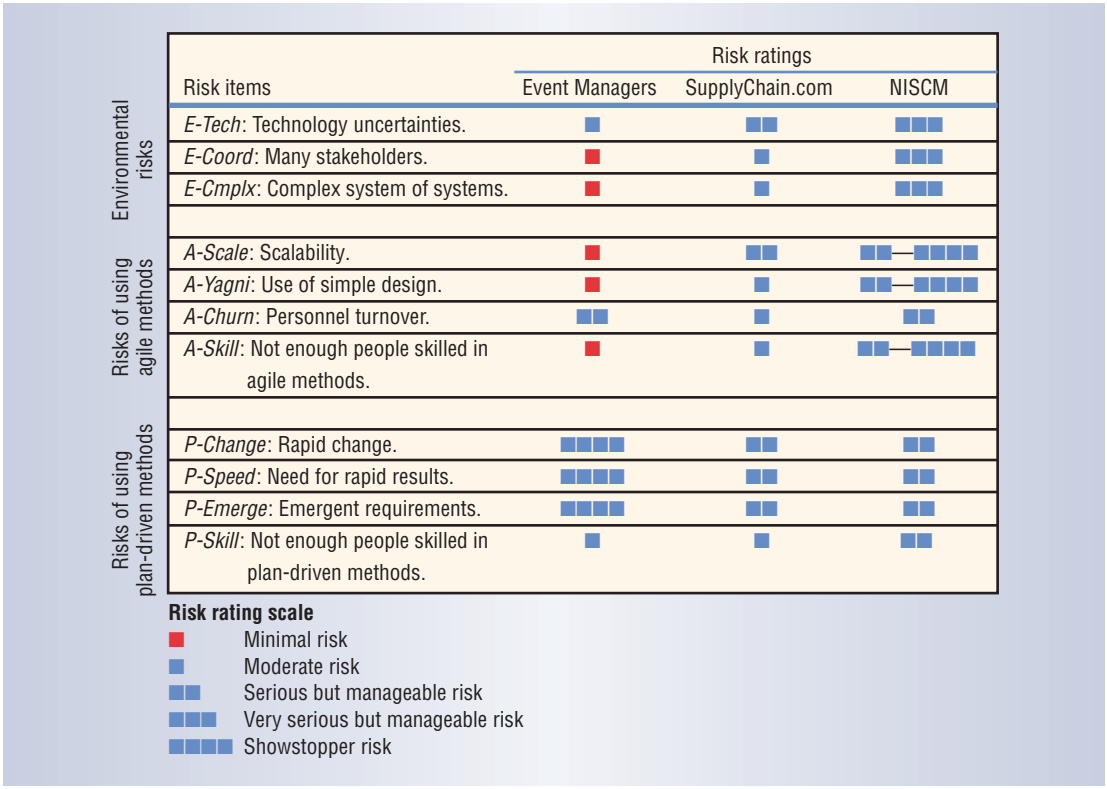

Figure 4. Pertinent risk items and risk ratings for a range of agent-based systems.

- an initial set of top-priority features to develop in Increment 1.

Both Phases 1 and 2 conclude with a review by senior experts and stakeholder representatives, securing full stakeholder commitment to support the next phase. If not, the project is terminated or backtracked to Phase 1 or earlier Phase 2 activities.

In situations involving cohesive, domain-experienced development teams with more than 40 percent Cockburn Level 2 and 3 people, Crack customers with a clear set of business objectives, and mature domain architectures and COTS products, the activities in Phases 1 and 2 can be combined and completed in a week. If most of these conditions do not hold, a better estimate would be three months each for Phases 1 and 2.

In Phase 3, the feature teams develop successive increments of prioritized system capability in parallel. The operational stakeholders ensure that the teams exercise and iterate each increment based on feedback from representative users. The entire team collaborates to discover and deal with emerging risks and opportunities, make and evaluate change proposals, and support each increment's transition into operational use.

A final shared task is to prepare the strategy for the next increment, consolidating lessons learned, planning for process improvement, and perhaps backtracking to earlier phases.

APPLYING THE METHOD FAMILY-WIDE

Figure 4 summarizes the results of applying Step 1 to each project in the agent-based family.

For the small, event-management application, the plan-driven risks clearly dominate the agile risks, with several plan-driven showstoppers. Step 2 thus leads this application to adopt an agile approach. Its process is a simple two-phase version of Figure 3, with two swim lanes representing a single agile team and its stakeholders.

Some aspects of the very large national crisis management application—its size and complexity—clearly represent showstoppers for pure agile methods such as simple design. But other aspects—rapid change and emergent requirements—are higher risks for plan-driven methods than for agile. Step 3 then applies, leading to a strategy that encapsulates the agile parts of this application, using risk-based agile methods in the agile parts, and applying risk-based plan-driven methods elsewhere. Its resulting risk-driven process is a more complex three-phase version of Figure 3, with the development swim lane broken into separate swim lanes for agile and plan-driven feature teams. Further details can be found in our book.[1]

Developers can use our tailorable process to balance agile and plan-driven methods in a customized software development strategy. This process can help organizations and projects take advantage of both the agile and plan-driven methods' benefits, while mitigating many of their drawbacks.

Versions of this process are currently being used

on several small projects and for the planning and risk management of a very large project. A related approach, called Code Science or AgilePlus, has been used successfully on more than a dozen projects of up to 400,000 source lines of code.[10,11] This approach uses most of the XP practices along with a componentized architecture, risk-based situation audits, business analyses, and on-demand automatic document generation.

References

1. B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison-Wesley, 2003.
2. B. Boehm, "Anchoring the Software Process," *IEEE Software*, July 1996, pp. 73-82.
3. I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.
4. B. Boehm and D. Port, "Balancing Discipline and Flexibility with the Spiral Model and MBASE," *Crosstalk*, Dec. 2001, pp. 23-28.
5. G. Anthes, "Agents of Change," *Computerworld*, 27 Jan. 2003, pp. 26-27.
6. A. Elssamadisy and G. Schalliol, "Recognizing and Responding to 'Bad Smells' in Extreme Programming," *Proc. Int'l Conf. Software Eng.*, IEEE CS Press, 2002, pp. 617-622.
7. J. Thorp, *The Information Paradox*, McGraw-Hill, 1998.
8. D. Parnas, "Designing Software for Ease of Extension and Contraction," *IEEE Trans. Software Eng.*, Mar. 1979, pp. 128-137.
9. B. Boehm and W. Hansen, "The Spiral Model as a Tool for Evolutionary Acquisition," *Crosstalk*, May 2001, pp. 4-11.
10. J. Manzo, "Odyssey and Other Code Science Success Stories," *Crosstalk*, Oct. 2002, pp. 19-21, 30.
11. J. Manzo, "Agile Development Methods, the Myths, and the Reality: A User Perspective," *Proc. USC-CSE Agile Methods Workshop*, USC Center for Software Engineering, 2003; http://sunset.usc.edu/events/past.

Barry Boehm *is director of the University of Southern California Center for Software Engineering. Contact him at boehm@sunset.usc.edu.*

Richard Turner *is a member of the Engineering Management and Systems Engineering faculty at the George Washington University in Washington, D.C. Contact him at rich.turner@osd.mil.*