Selecting a Project's Methodology

This article describes a framework for methodology differentiation, principles for methodology selection, and project experiences using these ideas.

Alistair Cockburn, *Humans and Technology*

How do we determine the need for various processes or methodologies, and what helps us choose the appropriate one for our project? To answer these questions, we need to get to the bottom of the controversy over methodologies and discover the dimensions along which they vary.

The punch line is that having multiple methodologies is appropriate and necessary. We can differentiate them according to staff size and system criticality (more dimensions exist, but these two serve well initially). For any point in the size–criticality space, methodology designers select a scope of concerns, prioritizing some quality of the project. Based on these choices, the project team selects how light or heavy a methodology they want for their project. (See the "Vocabulary" sidebar for a discussion of methodology weight and other pertinent terms.)

To help software development teams handle the selection, I describe several principles that underlie methodology design. I have successfully used these principles on several software development projects, one of which I describe in some detail in this article.

Methodology

A *Big-M methodology* includes at least the elements shown in Figure 1: people, roles, skills, teams, tools, techniques, processes, activities, milestones, work products, standards, quality measures, and team values. Under standards, we find notations, such as the drawing and programming languages selected; policies, such as incremental-development use; and conventions—the standards the project team determines. Team values, the least obvious element in the above list, are what the team strives for in terms of how it communicates and works. Different team values promote different methodologies.

Some companies have methodologies that cover from the initial sales call through maintenance and all roles for which project funds pay (see Figure 2). Most books containing what people call methodologies have mainly just the designer or programmer in mind, showing a few techniques and drawing standards. Fitting these techniques and standards into a methodology framework, we understand why software developers get frustrated when they see authors call these things methodologies. The individual design

Vocabulary

In 1995, Sam Adams coined two terms to describe the difference between the concept of *methodology* as found in large consulting houses and as described in then-current books by Grady Booch, James Rumbaugh, and others. He said that those books discussed *little-m methodologies*, describing a few techniques and drawing notations for a few roles. The large consulting houses use *Big-M methodologies*, encoding as much as possible about their way of working—processes, techniques, and standards being only part of the overall picture.

Because I needed a term to cover *all* aspects of working together, I started using his phrase, *Big-M methodology*. In this and other articles, I use this term to denote everything about how a group repeatedly produces and delivers systems: whom they hire and why, what people expect from coworkers, the processes they follow, their conventions, work products, and even seating arrangements. When a group places a job advertisement in the newspaper, the ad is an artifact of their Big-M methodology. We need this broad a view to get practical results about methodology and process diversity.

The American *Merriam-Webster Dictionary* gives as its first definition of methodology, "a series of related methods or techniques." The *Oxford English Dictionary* defines it only as "the study of methods." I use the American version to reflect the breadth of topics involved.

Some writers like to talk about a team's *process*. However, a process targets a series of steps; it does not have the necessary breadth of meaning we need to carry out the discussion. For example, it will not cover cultural values, seating arrangements, and other things. This article addresses overall methodology diversity, which naturally includes process diversity as a subset.

A methodology's *size* is its number of control elements, including deliverables, standards, activities, milestones, quality measures, and so on (see Figure 1 in the main text). Its *density* is the detail and consistency required in the elements. Greater density corresponds to tighter controls. A methodology's *weight* is its size times its density (conceptually only, because I do not attach numbers to size and density).

Project size is the number of people the organization allocates for the project. You might expect project size to match problem size, but it is not that simple. Problem size has no absolute measure, because a new person might see a simplifying pattern in the problem. I therefore carefully separate project size from problem size (see Principle 4 of "Principles Involved" in the main text).

techniques or drawing standards rarely are critical to the project's final success.

Principles Involved

To understand the need for various processes and methodologies, and to design and select them, we need to know the underlying principles. After several dozen project interviews and half a dozen methodology designs, I have developed confidence in four principles.

Principle 1

A larger group needs a larger methodology.

A methodology is larger when it contains more elements (roles, work products, reviews, standards, and so on). Because the methodology exists primarily to coordinate people, it appropriately will be larger on a larger project. Fortunately, the methodology grows with the number of roles rather than the number of individual people.[1] Principle 1 tells us that we should not expect a small-team methodology to work properly for a big team, and vice versa.

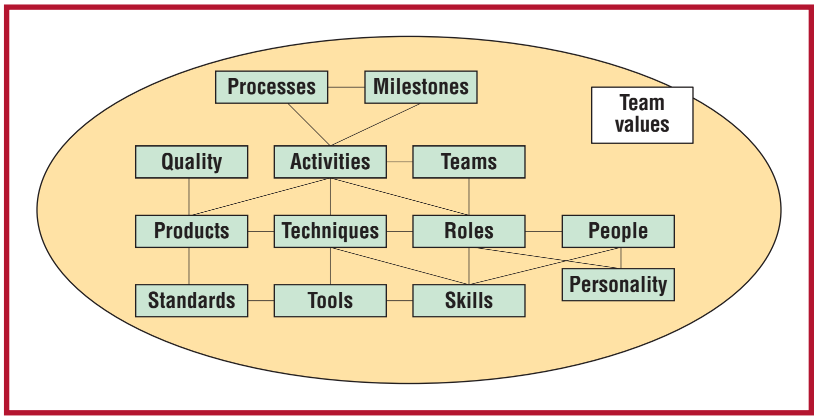

Figure 1. Elements of a Big-M methodology. People with particular skills and personalities fill project roles, working in various types of teams. They use techniques to construct work products that follow certain standards and meet selected quality criteria. The techniques require certain skills and tools; the tools help enforce the standards. The teams engage in activities that fall within the project's overall process; each activity passes key milestones that indicate how the process is moving forward. These elements operate within the team's value system, which needs to align with both the people's values and the process being used.

Principle 2

A more critical system—one whose undetected defects will produce more damage—needs more publicly visible correctness (greater density) in its construction.

I separate system criticality into these loss zones (obviously, others are possible):

- *Loss of comfort* means that with a system failure, people will only be less comfortable, have to do more work by hand, or call each other to repair a mis-

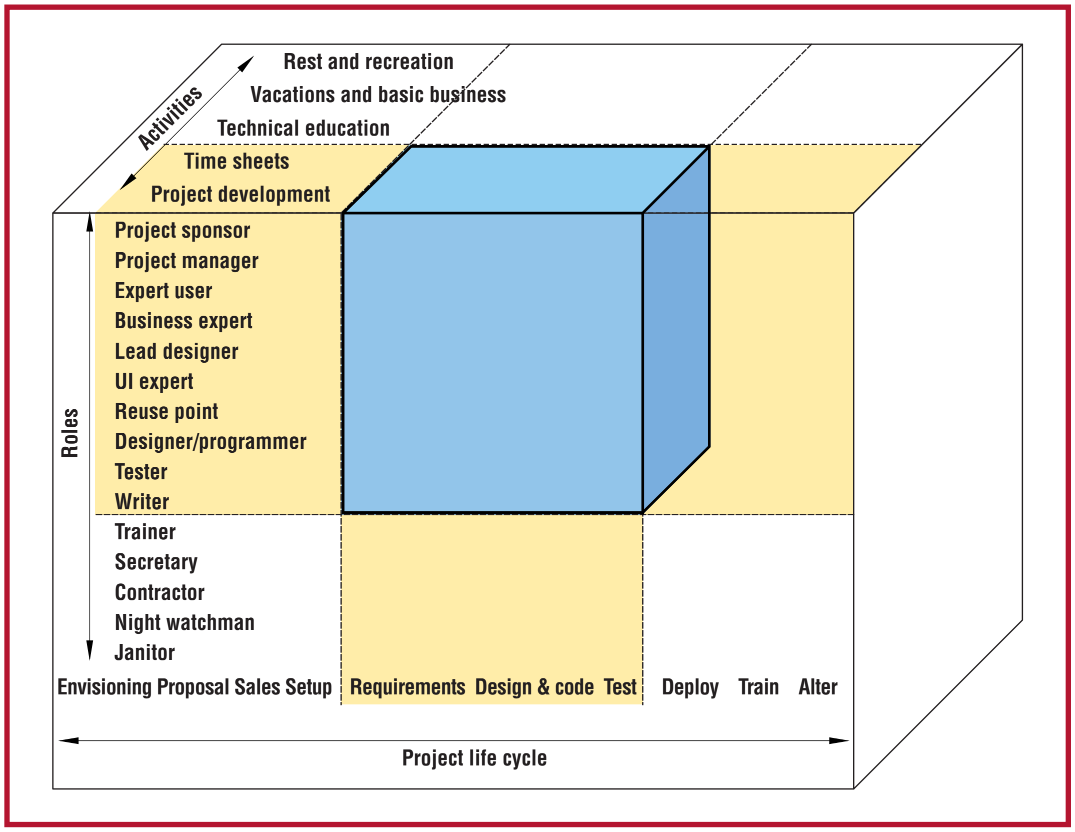

Figure 2. Identifying a defined methodology's boundaries. Every methodology has a boundary, either in the portion of the project lifecycle it covers, the people, or the subset of their activities it addresses. Understanding the boundaries will help you examine a methodology or compare any two.

communication. Purchase support systems and corporate infrastructure programs lie in this zone.

- *Loss of discretionary moneys* means that system failure produces loss of money or related valuables, but only in the range of discomfort. Typically, invoicing systems lie in this zone.
- *Loss of irreplaceable moneys* means that the loss of money or related valuables has an effect similar to that of bankruptcy. A run on the national banks would lie in this zone.
- *Loss of life* means people will likely die from a system malfunction. Atomic power plants, space shuttles, and airplane control systems fit here.

This principle says that the team can justify greater development expense for protecting against mistakes as the system moves from zone to zone.

For example, failure in an atomic power plant is more serious than failure in my bowling-match tracking software. Accordingly, the methodology the developers use in building the power plant software can afford to be more laborious and expensive. It will contain more elements, and the elements will have greater density.

Suppose both projects incorporate use cases. The bowling league might write them in a few sentences on the board, on a scrap of paper, or in a word processing document. The power plant team will insist on writing them using a particular tool and filling in particular fields. They will call for version control, reviews, and sign-offs at several stages in the life cycle. Developing the use cases for the power plant will cost more. The benefit is that more writers and readers will be able to collaborate and fewer mistakes will be made, which is supposed to justify the extra cost. Principle 2 says when the additional methodology cost is worthwhile—which is a good thing to know, considering Principle 3.

Principle 3

A relatively small increase in methodology size or density adds a relatively large amount to the project cost.

Pausing development to coordinate with other people costs not only time but concentration.[2,3] Updating requirements documents, design documents, and test documentation is also time consuming. Principle 3 does not question whether the coordination activities and deliverables are beneficial or hazardous. It addresses the cost of adding elements and control to the methodology.

At this point we can examine the relationship between methodology size, project size, and problem size. This discussion can be tricky, because of the tendency to assume that larger problems require more people to solve them.

A positive feedback loop connects project size and methodology. With relatively few people, relatively little methodology is needed. With less weight, they work more productively. With greater productivity, they can address a larger problem with their smaller team and lighter methodology.

On the other hand, a project with more people requires more coordination—that is, more methodology. The heavier methodology lowers their productivity, so accomplishing the same work requires more people. Methodology grows more slowly than project size, so eventually they get to a point where they can solve the problem and manage the coordination activities (assuming sensible management).

So, for a given problem (see Figure 3), you need fewer people if you use a lighter methodology and more people if you use a heavier methodology. But there is a limit to the size of a problem that a given number of people can solve. That limit is higher for a large team using a heavier methodology than for a small team using a lighter methodology. In other words, as the problem changes in size, different combinations of methodology and project size become optimal.

The difficulty is that no reliable way exists to determine the problem size at a project's start or the minimum number of people needed to solve it. Even worse, that number varies based on exactly who is on the team.

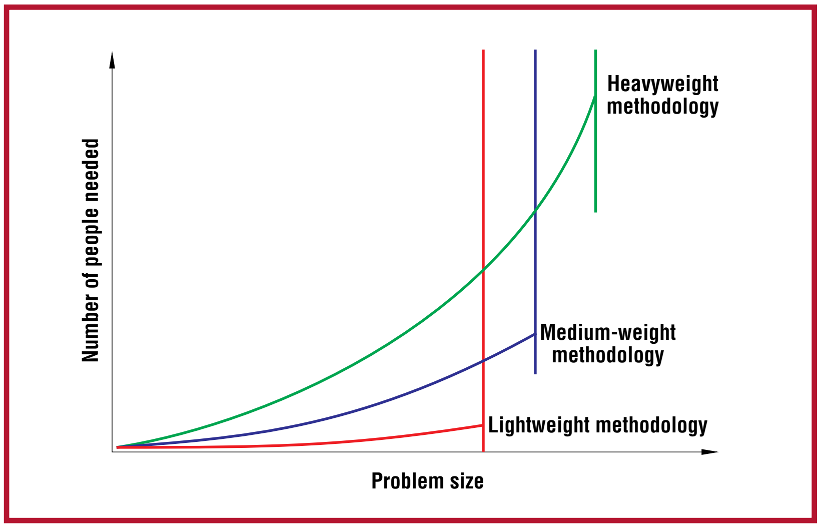

Figure 3. How problem size and methodology affect staff numbers. As long as the smaller team can deliver the system, fewer people and a lighter (well-founded) methodology are needed. However, as the problem gets larger, eventually, the smaller team simply cannot deliver the system in time. At that point, a heavier methodology, coordinating many more people, becomes necessary.

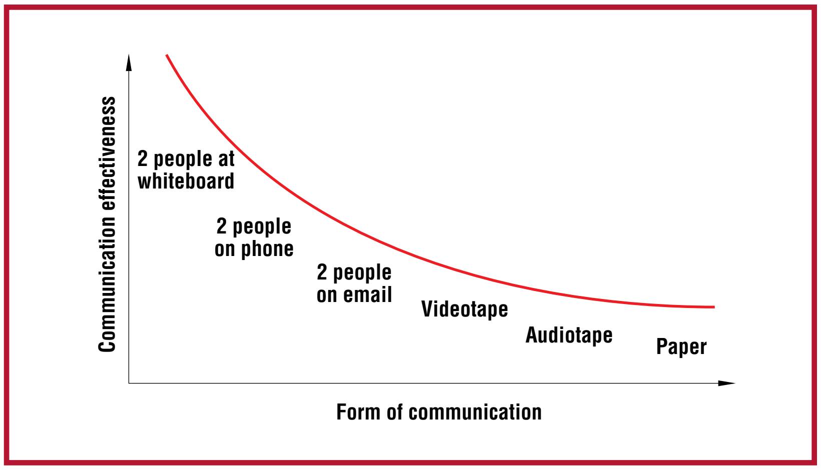

Figure 4. Communication efficiency decreases as personal contact decreases.

Principle 4

The most effective form of communication (for transmitting ideas) is interactive and face-to-face, as at a whiteboard.

Principle 4 implies that people sitting near each other, with frequent, easy contact, will develop software more easily; that is, the software will be less expensive to develop. It implies that as the project size increases and interactive, face-to-face communication becomes hard to arrange, communication effectiveness goes down, and the associated cost goes up.

Figure 4 shows effectiveness declining as the participants move from standing at a whiteboard to talking on the phone, to an e-mail chat session, to a videotape, and to a written document. As the curve progresses, the people lose close personal contact, multimodal communication, voice inflection, and real-time questioning capabilities.

The principle does not imply that a few people sitting in a room can develop all software. It does imply that a methodology designer should emphasize small groups and lots of personal contact if productivity and cost are key issues.[4] A discussion of different aspects of intragroup communication can be found elsewhere.[5]

The recently completed Chrysler Comprehensive Compensation (C3) experience exemplifies the above discussion. After 26 people failed to deliver what was considered a large system, an eight-person subset of the team restarted, using eXtreme Programming (XP),[6] an extremely light and rigorous methodology. (For more on XP, visit www.extremeprogramming.com, www.xprogramming.com, or www.c2.com/ppr/wiki/ExtremeProgrammingRoadmap.) The eight people successfully delivered in a year what the larger team with heavier methodology had failed to deliver.[7,8] Part of the success was its adherence to Principle 4.

Principles 1 through 4 reveal a useful spectrum along which methodology design lies: methodology weight can be traded off against personal communication. With better communication, we can shed bureaucracy. Conversely, to the extent that the group cannot get frequent and rich personal communication, they must add compensation elements to the mehthodology.

Two other factors affect what methodology is appropriate: the project priorities and the methodology designer's peculiarities.

Two Last Factors

Two other factors affect what methodology is appropriate: the project priorities and the methodology designer's peculiarities.

Project priorities

It matters greatly whether the project sponsors want to have the software soon, want it defect free, or want to have the process visible. Each prioritization produces a different recommendation.

Understanding the priorities is only sometimes easy. James Martin and James Odell's object-oriented methodology[9] is general and can be tailored, but what the methodology optimizes, or whether different optimizations are possible on different projects, is not clear. The OPEN (*o*bject-oriented *p*rocess, *e*nvironment, and *n*otation) process family appears to prioritize for program correctness, progress visibility, and repeatability.[10] Watts Humphreys' Personal Software Process fairly clearly optimizes for predictability.[11]

Three recent methodologies announce their priorities. The methodology family Crystal[3,12] optimizes for productivity and tolerance. XP optimizes even more for productivity by reducing tolerance. Adaptive Software Development[13] targets highly unstable project situations, where requirements, design, and incredibly short timelines shift as a function of each other (for example, Web development).

The methodology designer

"All methodology is based on fears," quipped Kent Beck in a methodology discussion. Although this sentence appeared merely dismissive at first, I have found it to be largely true. Each element in the process or methodology can be considered a preventative against a bad experience some project has had. Afraid that programmers make coding mistakes? Hold code reviews. Afraid that designers will leave in the middle of the project? Have them write extensive design documentation as they proceed. If methodology designers would or could state their fears and wishes, much of the methodology's design would become immediately apparent.

We should distinguish, however, between the actual risks the individual project faces and the methodology designer's background. The methodology designer casts his or her experiences and biases into the methodology in an attempt to capture the invariants across projects. However, the risks the team faces vary with each individual project. Therefore, the final methodology design fits a project as well as the designer's assumptions fit the team members' attitudes and the project's true risk profile.

The Selection Framework

Figure 5 illustrates my framework for methodology selection, separating seven project sizes, four criticality zones, and several possible project priorities. These are arbitrary but plausible divisions, set approximately where the nature of the methodology will shift. This framework's advantage is that it is relatively objective. We can count the people on the project and assess its criticality and priorities.

The methodologies should correspondingly get bigger (more communication elements) toward the right, and denser (tighter controls) going up. According to Principle 3, moving to the right or up adds a large cost to the project development, so the team should find economic incentive to place a project as far to the left and down as possible. Other incentives, such as the manager's prestige and career safety, might drive a project to be considered bigger and more critical, even though that increases the cost.

Each cell allows several methodologies, depending on whether the project sponsors are searching for productivity, visibility, repeatability, or correctness. Using the four principles, we can make basic decisions about which methodology to use. After that, personal preferences will drive the details.

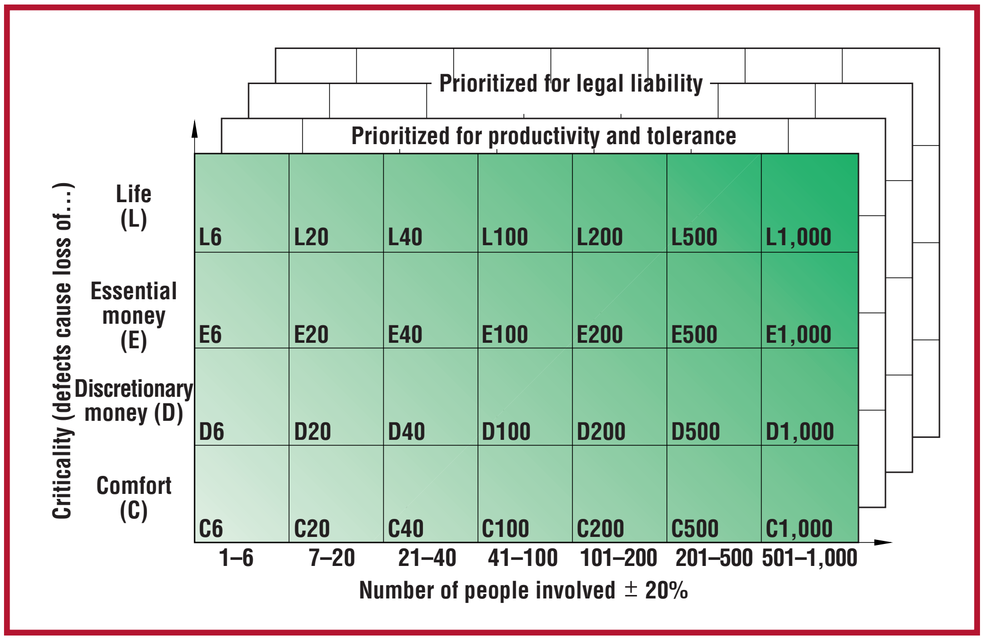

Figure 5. A methodology grid, organized as people × criticality × priority. The letter–number combination in a cell indicates the maximum criticality and project size for that cell. For example, C6 indicates a loss-of-comfort project with up to 6 people. D40 indicates a loss-of-discretionary-money project using 21 to 40 people.

Applying the Principles and Framework

To illustrate the use of these ideas across a series of projects, let us look first at a diverse range of projects, and then more closely at one of the projects specifically, whose changing characteristics moved it across the selection framework grid.

A range of projects

The programming staff at the Central Bank of Norway plays a critical role in coordinating that nation's banks, investing national funds, enforcing national banking policy, tracking the movement of physical currency around the country, and providing IT services for the internal staff. The bank employs approximately 40 employees, plus a fluctuating number of contractors, to work on the astonishing number of contractors that accomplishing these objectives requires. I hope it is obvious that no one, right methodology fits the following range of projects found in just this one organization.

- A 35-person Y2K project. Its objective was to see that the Norwegian banking system did not collapse on 1 January 2000. The primary technology was a traditional mainframe. The project criticality ranked in the essential-moneys category (see Figure 5); timeliness and correctness were the top project priorities. This was clearly an E35-category project.
- A project to collect and check all of Norway's bank-to-bank transactions, in concert with another company. It also used mainframe technologies. This project, which I will discuss in detail in the next section, was the first on which I explicitly used the grid in Figure 5 to shift the project methodology.
- A five-person project to produce a prototype of the mission-critical banking system that would eventually replace the mainframe system. The team was developing an intranet-based, object-oriented, Java–Corba component architecture, running as a C5-category project, working in one room, with minimal distraction. Both the manager and team agreed on the key to success: "Get good people; keep the team size low; put them in one room; give them adequate training; keep distractions away from them." (How different from the Y2K project!)
- An internal, three-person project to let people track their purchase requests. It was to be built in Java, using Web browsers and the e-mail system, and run as a C4 project prioritzing low-cost development. The team worked quite informally, writing only a sketch of the use cases and a draft project plan—until they decided to buy rather than build the software (true to the low-cost priority).
- Two internal, one-person projects. One was to let people order special dinners from the cafeteria when they worked

late at night. It was developed in Delphi, using Web browsers. The other was to let selected bank staff produce summary reports of various investment and spending activities. It was programmed in SQL and ran on a mainframe. In both of these projects, the single programmer simply worked with representative users and had no other development controls.

This project was particularly interesting because it shifted cells in the methodology framework twice.

Outside the Central Bank of Norway, I visited or worked on the following projects, all of which were internal business systems programmed primarily in Smalltalk. I present them from smallest to largest.

- The C3 project,[6–8] described earlier, grew from 10 to 14 people. The team replaced all the usual written work-product documents with face-to-face discussions, whiteboard notes, index cards, and extensive regression tests. They made some innovations, including rotating partners in a continual pair-programming environment and delivering every three weeks. In terms of this article, they used the principles to stretch a D6 methodology to fit a D14 project, thus keeping costs down and productivity up.
- Project Winifred was a D40 Smalltalk project with time to completion as its top priority, a colocated team, and good internal communications.[12] As process consultant, I examined the team size and physical proximity, system criticality, and the project priorities, and designed a customized lightweight methodology that maximized the use of interactive, person-to-person communication to reduce cost.
- Project Rishi was a D90 Smalltalk project. I was part of the management group that introduced interteam coordination, including special meetings and documents. Even working to let this project run as lightly as possible, we could not use the same simple mechanisms as in Project Winifred, but had to instead create multiple design teams with cross-team mentors and standards.

One specific project

At the Central Bank of Norway, I worked as technical coordinator and pinch-hitting architect on the bank-to-bank transactions project. This project was particularly interesting because it shifted cells in the methodology framework twice.

When the bank introduced this project to me in November 1997, they described it as a 15-workweek, three-person project, with the design already mostly complete and the people already experienced from the previous system version. Based on the methodology framework and my experience, I suggested they view it as a D4 project—a loss-of-discretionary-money project using four people—and simply work together casually ("Get it done and go home" was the phrase), with minimal bureaucratic interference.

As part of picking up the project, I reviewed the project plan with the team and discovered that the developers had only estimated part of the task. The complete estimate came to 130 workweeks. Additionally, the designers were adding new technology, with new real-time response and failure sensitivity requirements, and faced a mutual-exclusion problem inside the main database.

Two of the developers were in Lillehammer, and the other developer, the project manager, and I were in Oslo. The developers needed to interface with a sister organization in a different part of Oslo, working on a separate computer system. Our lead architect was taking paternity leave in two months, the project manager was new to both IT and project management, and the project would report to a national project board.

At this point, I shifted our side of the project to the E5 category (see the difference in Figure 5). This meant instituting a risk-reduction milestone plan, incremental delivery, weekly teleconferenced group meetings, monthly status reports, and other measures.

The team delivered the first increment on time, at the start of February 1998. But then the lead designer went on paternity leave, the Y2K project preempted the other senior developer, and we uncovered a flaw in the failure recovery and mutual-exclusion designs. We grew to 10 people—most of whom were novices in the domain—spread across multiple floors of multiple locations. Daily face-to-face communication was impossible.

In mid-February, I shifted our project to the E15 category. We designed smaller milestone markers for each individual, set up a testing simulator, and increased communications between team members. Because of time pressures, we did not institute more paperwork for the team members (see Principles 3 and 4) but stepped up personal communications, including phone calls, teleconferences, and train trips.

The story ended relatively happily. We were able to stabilize both staffing and the project plan in March 1998. The project sponsors accepted the March 1998 plan, even though it was considerably different from the original, November 1997 version. The novice project manager learned how to read the people on the project and tracked the milestones well. The project delivered on time (according to the March 1998 plan) in December 1998 and was installed the following February. The project pleased management on all sides, both technically and because the cost and schedule remained stable over the project's last 10 months.

It is quite easy to say, "One methodology can't fit all projects," but that notion seems to escape many process and methodology designers. Actually, it is quite hard to see how to proceed beyond making that statement. A start is to use the four principles I described for methodology design and apply the project grid.

The task facing us next is to find ways to tailor a methodology to the idiosyncracies of any particular project fast enough to get the benefits of the tailoring before the project is over.

References

1. N.B. Harrison and J.O. Coplien, "Patterns of Productive Software Organizations," *Bell Labs Technical J.*, Vol. 1, No. 1, Summer 1996, pp. 138–145.
2. T. DeMarco and T. Lister, *Peopleware: Productive Projects and Teams*, 2nd ed., Dorset House, N.Y., 1999.
3. A. Cockburn, *Surviving Object-Oriented Projects*, Addison-Wesley, Reading, Mass., 1998.
4. L. Plowman, "The Interfunctionality of Talk and Text," *Computer Supported Cooperative Work*, Vol. 3, Nos. 3–4, 1995, pp. 229–246.
5. J.A. Sillince, "A Model of Social, Emotional and Symbolic Aspects of Computer-Mediated Communication within Organizations," *Computer Supported Cooperative Work*, Vol. 4, No. 1, 1996, pp. 1–31.
6. K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Reading, Mass., 1999.
7. The C3 Team, "Chrysler Goes to 'Extremes,'" *Distributed Object Computing*, Oct. 1998, pp. 24–28.
8. R. Jeffries, "Extreme Testing," *Software Testing and Quality Eng.*, Mar./Apr. 1999, pp. 23–26.
9. J. Martin and J. Odell, *Object-Oriented Methods, Pragmatic Considerations*, Prentice Hall, Upper Saddle River, N.J., 1996.
10. I. Graham, B. Henderson-Sellers, and H. Younessi, *The OPEN Process Specification*, Addison-Wesley, Reading, Mass., 1997.
11. W. Humphreys, *Introduction to the Personal Software Process*, Addison-Wesley, Reading, Mass., 1997.
12. A. Cockburn, *Crystal/Clear: A Human-Powered Methodology for Small Teams*, to be published by Addison-Wesley, Reading, Mass., 2000; access our early version at members.aol.com/humansandt/crystal/clear (current June 2000).
13. J. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House, N.Y., 2000.

About the Author

Alistair Cockburn is consulting fellow at Humans and Technology. He works to understand and improve human factors in software development. He speaks six languages, has forgotten two others, and enjoys dancing, and sitting underwater. Contact him at Humans and Technology, 7691 Dell Rd., Salt Lake City, UT 84121; arc@acm. org.