Agile Project Management

Vol. 6, No. 6

CUTTER CONSORTIUM

Organizational Patterns:

Building on the Agile Pattern Foundations

by James O. Coplien, Neil B. Harrison, and Gertrud Bjørnvig

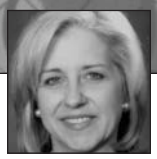
What *really* works in software development management? Practices and deep organizational structures — what the authors refer to as organizational patterns — that result in customer satisfaction and lay the foundation for organizational adaptability and agility. The following *Executive Report* discusses the top 10 patterns that successful organizations and teams have used and provides a framework within which you can customize these practices for your own enterprise.

Executive Report

Cutter Business Technology Council

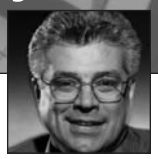

Rob Austin

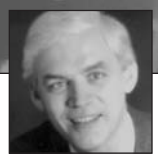

Tom DeMarco


Christine Davis


Lynne Ellyn


Jim Highsmith


Tim Lister


Ken Orr


Lou Mazzucchelli


Ed Yourdon

About Cutter Consortium

Cutter Consortium is a truly unique IT advisory firm, comprising a group of more than 100 internationally recognized experts who have come together to offer content, consulting, and training to our clients. These experts are committed to delivering top-level, critical, and objective advice. They have done, and are doing, groundbreaking work in organizations worldwide, helping companies deal with issues in the core areas of software development and agile project management, enterprise architecture, business technology trends and strategies, enterprise risk management, metrics, and sourcing.

Cutter offers a different value proposition than other IT research firms: We give you Access to the Experts. You get practitioners' points of view, derived from hands-on experience with the same critical issues you are facing, not the perspective of a desk-bound analyst who can only make predictions and observations on what's happening in the marketplace. With Cutter Consortium, you get the best practices and lessons learned from the world's leading experts; experts who are implementing these techniques at companies like yours right now.

Cutter's clients are able to tap into its expertise in a variety of formats including content via online advisory services and journals, mentoring, workshops, training, and consulting. And by customizing our information products and training/consulting services, you get the solutions you need, while staying within your budget.

Cutter Consortium's philosophy is that there is no single right solution for all enterprises, or all departments within one enterprise, or even all projects within a department. Cutter believes that the complexity of the business-technology issues confronting corporations today demands multiple detailed perspectives from which a company can view its opportunities and risks in order to make the right strategic and tactical decisions. The simplistic pronouncements other analyst firms make do not take into account the unique situation of each organization. This is another reason to present the several sides to each issue: to enable clients to determine the course of action that best fits their unique situation.

For more information, contact Cutter Consortium at +1 781 648 8700 or sales@cutter.com.

Organizational Patterns:

Building on the Agile Pattern Foundations

AGILE PROJECT MANAGEMENT ADVISORY SERVICE

Executive Report, Vol. 6, No. 6

by James O. Coplien, Neil B. Harrison, and Gertrud Bjørnvig

INTRODUCTION

You're a project manager, an executive, a line manager, or maybe even a developer, and you have more than a hunch that you could deliver high-quality software faster. You believe that you really *could* meet those schedules and that you really *could* meet requirements regularly, if only …

But it's difficult to put your finger on the "if only." Once in a while, you get a glimpse of an improvement opportunity: after a chance meeting with a customer or a particularly good or bad group meeting, you get a glimpse of a breakthrough. Yes, you're already following this or that management book, methodology, or fad. And it sounds good — or at least it sounded good when the group got excited about it and brought the solution in the door. But now you're not sure whether you're following the technique correctly, or even if you did, whether the technique would save you. You don't know whether it might be the technique itself that is strangling you.

The answers aren't trivial, but there *are* answers. It's impossible to manage a healthy, successful software project without managing the organization itself. And unless you shape the patterns of relationship, communication, and software assembly in an organization, you aren't really managing the organization. It is at these foundations of organization and management where the systemic, long-term problems of software productivity and quality can be redressed.

Organization and management are about people. We know Conway's Law, a heuristic that suggests a close tie between the technical issues of architecture and the human issues of organizational structure. Conway's Law holds that the structure of any IT system reflects the structure of the organization that builds it. This principle works at human scale: it addresses relationships, software structures, organizational structures, and processes that people deal with at a deep level throughout their workday. There are other celebrated principles of organization and management at the enterprise level. While they have their place in an overall corporate framework, they are rarely of

direct value to software managers and developers and, in fact, are often detrimental. The principles of agile and the patterns of software development organizations operate at human scale, at the center of the concerns of the project manager, product manager, and the other roles that touch the bits and bytes that delight your customers and sustain your business.

Over a period of more than 10 years, we have studied hundreds of software development teams to answer the question "What *really* works in software development management?" As time went on, it became clear which practices and organizational structures result in customer satisfaction and lay the foundation for organizational adaptability and agility. We found, researched, validated, and documented about 100 of these configurations. We call them "organizational patterns," or deep structures that define the culture of a software development organization. Our results are collected in the work *Organizational Patterns of Agile Software Development* [10]. This work captures the structural foundations of agile software development practice, and as such, it is one of the best practical starting points for agile process improvement. Mike Beedle, coauthor of *Agile Software Development with SCRUM* [23], writes:

> Although "agile development" perhaps was first practiced by LISP programmers in the 1960's, *Organizational Patterns* is perhaps the first documentation that ever existed on true agile development. No one, to my knowledge, had done so before. (Not Scrum, which started in 1993, nor XP which started much later, etc.). [3]

While all 100 patterns have a role in making organizations strong, a remarkably small number of patterns form the successful core structures of agile organizations. In this *Executive Report*, we present the top 10 patterns from our extensive research over the past 12 years. These top 10 patterns might be all you need to turn your organization around. They certainly lay a broad foundation for powerful software process improvement. This report is a primer and get-started-quick package on organizational patterns.

Deeper Than Processes

Patterns are a crucial tool of organizational design because they go deeper than processes or an organizational chart. They are not just project management for the moment but a form of project management that shapes your organization for the future while honoring short-term project goals. Each pattern individually achieves a small improvement. The patterns are designed to amplify one another so that, over time, you reap the benefits of major restructuring with the low risk of local incremental change. Patterns are unlike many process improvement programs that touch only the symptoms of software development failure or are limited to simple cause-and-effect analyses that fail to redress systemic problems. This *Executive Report* helps to repair, grow, and revitalize your organization by shaping its foundations in small steps. Furthermore, it puts structures in place that help *keep* your organization agile.

A New Twist on Old Ideas

The patterns of successful organizations are well established. Many of them have long been known as hallmarks of great organizations, and you will recognize many from the literature or your own experience. Our work on patterns of software agility adds two components to this mature body of literature. First, it defines and fills in the broad spectrum that ranges from managerial practices all the way down to software implementation

The ***Agile Project Management Advisory Service Executive Report*** is published by the Cutter Consortium, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA. Client Services: Tel: +1 781 641 9876 or, within North America, +1 800 492 1650; Fax: +1 781 648 1950 or, within North America, +1 800 888 1816; E-mail: service@cutter.com; Web site: www.cutter.com. Managing Editor: Cindy Swain, E-mail: cswain@cutter.com. Group Publisher: Kara Letourneau, E-mail: kletourneau@cutter.com. Production Editor: Lauren S. Horwitz, E-mail: lhorwitz@cutter.com. ISSN: 1536-2981.

and the practices of those who create it. These two ends of the project management spectrum are much more closely connected than either a manager or software engineer might realize. Without tying together these two perspectives — and many others — it's difficult to attack the system concerns that are the most serious threats to a software enterprise.

Second, it ties together these patterns according to how they build on one another. Instead of being a list or partitioning of individual rules, these patterns are linked to one another so that you can assemble them into a structure that repairs and strengthens your organization incrementally through a process guided by both experience and feedback.

The Patterns and Agile

All major agile disciplines have their roots in the work that underlies these patterns. These patterns have their origins in a project at Bell Laboratories that was affectionately called the Pasteur project — a pun on the "cultural" nature of Petri dishes one might associate with the pioneer Louis Pasteur. The goal of the project was to analyze the culture of software development organizations using techniques from the social sciences. At the time, most software improvement work was grounded either in ISO 9000 efforts or the SEI's Capability Maturity Model (CMM®). Each had its shortcomings in addressing system problems, guiding architectural structure, and solving the problems germane to software development. The Pasteur project was an applied research project that explored how to extend existing approaches to reach the more fundamental, recurring problems of software development. Unlike many process improvement efforts, it was based on empirical studies and, thus, an understanding of what works in the real world. The empirical groundings are a perfect match for a discipline grounded in empiricism: software patterns.

In August 1993, the software pattern discipline was born at a workshop in Colorado, USA. It was at this meeting that some of the early structural findings of the Pasteur project found their way into pattern form. A collection of organizational patterns grew; and by the fall of 1994, a collection of about 40 organizational patterns existed. Coauthor James Coplien presented these patterns at the first pattern conference in Allerton Park, Illinois, where Cutter Consortium Senior Consultant Kent Beck helped to shepherd the work, providing feedback on the conference paper. Beck would later cite Coplien as one of three influences in his work on Extreme Programming (XP) [2].

Other organizational pattern works soon followed, including papers presented at the same conference by Norman Kerth [18] and Bruce Whitenack [25] that went beyond software architecture to explore the realm of human behavior. In the following year, Cutter Consortium Senior Consultant Alistair Cockburn published "Prioritizing Forces in Software Design" [7], and Ward Cunningham published his EPISODES pattern language [13]. The former would become one of the foundations of the AgileAlliance, and the latter lent major structure to XP. All these works had a pattern focus and emanated from the pattern community. In the meantime, Scrum also drew on the roots of the Pasteur project and in particular on one of the Pasteur case studies published in *Dr. Dobb's Journal*, a study on the Quattro Pro for Windows development project at Borland [9].

Most branches of the agile movement trace back to these original organizational patterns. It took several years to refine these patterns, understand how they worked together in practice, and combine them into sequences that would generate successful software development organizations. The branches were brought back together in the 2004 book *Organizational Patterns*. The collection was edited by two of the coauthors of this report, James Coplien and Neil Harrison, into a cohesive and broad work built on the original core patterns but that drew heavily on patterns — whether in "pattern form" or not — from the branches of the agile movement.

Agility and Adaptiveness

To be agile means to have the ability to ride out the unexpected and to capitalize on flexibility. An organization is a living thing: it grows and shrinks, reorganizes itself according to its environment, and responds to its environment. It never stands still. A good organization isn't rigid but can bend with the winds of change. Taken to an extreme, however, flexibility becomes a weakness. Strength comes from structure. With structure, an organization can coordinate its members to do things that no single member can do.

Extreme Programming suggests a structure based on programmers, a coach, a team of a certain size, and — in its foundations — the Smalltalk programming language. Also in its foundations are strong admonitions against free combinations of the principles and practices; the structure is relatively fixed. This structure gives XP projects a degree of agility in a certain context, most notably in small, single-thread Smalltalk programming projects.

However, the foundations of XP cannot easily *adapt* to projects outside this context; rather than adapting to you, it asks that you adapt to it. Such adaptation too often discards the gems of insight that are the core competencies of an enterprise. Good organizational improvement builds on the strengths of the existing organization, which include its organizational structures, tools, principles, and processes.

As envisioned by Cutter Consortium Agile Project Management Practice Director Jim Highsmith or by Alistair Cockburn, the agile discipline — or organizational patterns — isn't only about agility in the development organization; it's also about agility in the principles themselves. One doesn't follow a method to be agile; rather, agile encompasses a management style that is responsive to its environment. Agile patterns build not only on the responsiveness that comes from feedback but also on the empirical evidence that provides managers with the assurance that an individual improvement has low risk.

A Kit, Not a Religion

Organizations have their own culture. The earth sustains innumerable human cultures, each one of which is "successful" in the sense that it has evolved over decades, centuries, or even millennia to adapt to the climate, history, and makeup of the area. A book on organizations can no more describe how to build an ideal organization than a book on anthropology can describe how to form an ideal culture. Just as there is no single ideal culture, there is no single ideal organization. What makes an organization ideal in its context is that it meets the business's needs that define that context. Each organization has its own business climate, history, and staff that make it unique. Each pattern must be fit to the context where it applies. Different organizations require different combinations of patterns for healing and growth.

Organizational patterns provide a kit of organizational structures that you can assemble according to the needs of your business, adapting each pattern to your organization as you go. You and your organization, rather than the patterns themselves, are in charge. Each organization follows its own path to evolve with these patterns and is unique in the world at each stage of development.

PRINCIPLES OF ORGANIZATIONAL CHANGE

Systems Thinking

To move beyond individual Band-Aids that attempt cause-and-effect attacks on problems, and to combine small patterns to bring about emergent change, is a powerful form of systems thinking. Most process-based approaches are locked in cause-and-effect models, which lead to the belief that you can be in control. To change a system requires subtle changes at a level deeper than process; you must change its structure. One reason that agile approaches have promise is that they are grounded in the deepest foundations of systems thinking: those of *principles* and *values*. Principles and values give rise to the *structure* of an organization. We can communicate the structure of a system as

patterns of relationship between its parts, and the parts themselves can reflect patterns of recurring structure. The structure of a system generates its processes as we consider in process improvement approaches such as ISO 9001. Figure 1 shows how patterns fit in systems thinking in the domain of software development.

For example, consider the process of pair programming or code inspections in your organization. Why do you have these processes? They come about because of a *relationship* between one set of individuals (the coders or drivers) and another set of individuals (the reviewers or observers), each of which bring different perspectives to a design problem. These relationships come about from the structure of the organization. Why does the structure exist as it does? The structure owes much to the organizational principles and values by which the organization came together. We have reviewers and observers because we value the diverse skills necessary to build a quality product. We have managers and developers both because we want a designated position that supports the coders as they produce the artifact for which customers will pay money and because we don't want developers to be sidetracked by common management issues.

One mistake is to effect organizational change at too shallow a level. If you try to change process without changing the structure, the change is transitory, because the structure will eventually win out. On the other hand, if you try to go too deeply too quickly, you increase risk. Changing the value system or principles of an organization takes time, often cannot be done directly, and in any case cannot be done without making structural changes. As structural changes, patterns provide a good balance between the process level and the level of values and principles.

Patterns also provide a unifying formalism with wide applicability. A pattern can change the organizational structure by adding a role, adding a relationship between existing roles, constraining size, changing communications paths, or through numerous other restructurings. All of these structural changes combine to improve the organization. All the patterns we discuss in this report have an empirical basis, and we have taken great care to find the proper ordering that allows them to amplify one another.

Why Use Patterns?

Patterns are about structure, and lasting organizational change can happen at no more shallow a level than that of the organizational structure. Perhaps more important, the pattern approach is a systems thinking approach that embodies many key elements of a philosophy of organizational growth. Among these principles are systems thinking, piecemeal growth and local repair of an organization, feedback, and human comfort.

The best way to understand a pattern is as a structure-preserving transformation of an organization structure. A pattern adds local structure to a system while retaining the overall major rhythms of

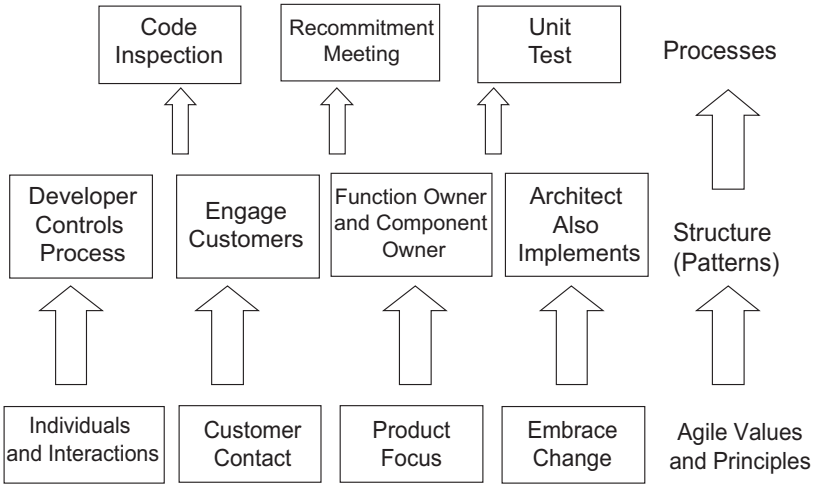


Figure 1 — Systems thinking in software development.

the system to which it is applied. A pattern makes it possible to effect change while maintaining continuity. Pattern-based change is incremental change, and incremental change is low-risk.

Complex Things Are Complicated[1]

One reason that organizational change is so difficult is that organizations are complex. A single simple change to an organization can influence only one thread of interactions or one facet of organizational structure, and that single fix is likely to go unnoticed in light of the myriad other interactions and other facets of organizational structure. A local change can make things better locally, but unless there is an ongoing process of change — through the application of other patterns — to make room for that pattern to grow throughout the organization, the global structure will eventually silence the individual local change.

One can't bring about large organizational impact by making a large organizational change; instead, one has to combine multiple small changes that are applied in a process of local adaptation and piecemeal growth. These changes eventually make room for one another to have impact, and things start to fall into place. Don't think of patterns as changing your organization, but as making change possible. After you apply a few patterns — perhaps with very little noticeable improvement from individual steps — the patterns start to work together to attack system problems.

For example, your problem may be that developer productivity is low. You encourage developers, even buy them tools, and it doesn't seem to help. You might try spreading their work more evenly so that underutilized people get more work and saturated people have time to recharge. Now morale is a bit better, but productivity doesn't change. They have good contact with customers: you may have been flying the customers to your site once a week to meet with developers for two days. You might read the FIREWALLS pattern and decide that these developers' exposure to customers is too much of a good thing, so as a manager, you start filtering some of the interactions between customers and developers. You convey key notions generated by customers to developers but also free up developers' time spent interacting with customers, which gives developers 20 more hours a week to produce code. This also gives you more time with developers, and you try to guide them so that their work best meets project needs. Still, the efforts don't quite seem to get off the ground. The two patterns DISTRIBUTE WORK EVENLY and FIREWALLS remove some obstacles, but it's not until the developers combine their knowledge of their code with the knowledge of customer needs that you convey to them that they are able to use the pattern DEVELOPER CONTROLS PROCESS. Now you notice that productivity starts taking off. It couldn't happen if developers were overloaded; it couldn't happen if their flow was distracted by outsiders. All three patterns work closely together.

The order of pattern introduction is important. Later in this report, we discuss a *pattern language*, which is a grammar of sorts that guides you through a sequence of patterns to build and repair your organization.

THE TOP 10 PATTERNS

The book *Organizational Patterns of Agile Software Development* contains approximately 100 patterns divided into four pattern languages. Some of these patterns are fundamental because they establish the common broad foundations of almost all software development organizations, while other patterns are either germane to specific kinds of software development or describe the fine structures found only in the most mature organizations.

However, the scale of the pattern isn't always the best indicator of success. As the architect Mies van der Rohe noted, God is in the details. An organization can't really fly until its rough edges become a bit polished. It is likely

[1]This idea is not a tautology but an insight that bears deep consideration — think about it. Many thanks to Dave Vlack for this apparent Yogiism.

that a small number of these patterns provide most of the foundations for success in most software development organizations. This section describes 10 critical organizational patterns. All other patterns not described in detail in this report are discussed at greater length in *Organizational Patterns*.

1. UNITY OF PURPOSE
2. ENGAGE CUSTOMERS
3. DOMAIN EXPERTISE IN ROLES
4. ARCHITECT CONTROLS PRODUCT
5. DISTRIBUTE WORK EVENLY
6. FUNCTION OWNER AND COMPONENT OWNER
7. MERCENARY ANALYST
8. ARCHITECT ALSO IMPLEMENTS
9. FIREWALLS
10. DEVELOPER CONTROLS PROCESS

UNITY OF PURPOSE[2]

> … The team is beginning to come together. Team members may come from different backgrounds and may bring with them many different experiences.

✤ ✤ ✤

Many projects have rocky beginnings as people struggle to work together. Often, people have different ideas about what the final product should be. In fact, the final product may well be a pretty fuzzy concept. Yet people must have a consistent view of the product if there is any hope of completing it. Each person is different and has different views and opinions. They come with different backgrounds and experiences, and they must learn to work together. It is important to get off to a good start; initial impressions, good or bad, tend to be lasting.

Therefore: **The leader of the project must instill a common vision and purpose in all members of the team.** This "leader," whether a manager, a worker installed in the PATRON ROLE, or a customer advocate, should be someone who has the team's respect and influence over the team's thinking. Gaining respect and influence requires overt action; you can't count on it happening automatically. The leader should ensure that everyone agrees on the following questions: What is the product supposed to do? Who are the customers, and how will the product help them? What is the schedule? Does everyone feel committed to the schedule? Who is the competition? An important component of these team unification exercises is to identify the strengths of the team and to use these strengths as rallying points. The team thus identifies the challenges and competition and unites to overcome and surpass them, respectively. As time goes on, UNITY OF PURPOSE continues to emerge from ongoing dialogue within the team and with customers and other stakeholders. While the team leader primes the pump, team dynamics take over and give a project momentum.

✤ ✤ ✤

The obvious result is that the team works together rather than at cross-purposes. A more subtle yet probably more powerful effect is what healthy team dynamics can do for the morale of the team. The best teams tend to believe that they are somehow better than others — and they work to prove it! This pattern relates to some deep-seated principles and values of organizational health. There may be no more important property of an organization than that its members have a shared vision that they are motivated to achieve. Communication — which receives the bulk of the attention in this report — is just a means to achieve that shared vision. Thus, UNITY OF PURPOSE is a deeper principle than even effective communication; communication is just a means to the end — UNITY OF PURPOSE.

…

ENGAGE CUSTOMERS

> … An organization is in place, and its QA function has been generally shaped and chartered. The QA function needs input to drive its

[2]With the exception of FUNCTION OWNER AND COMPONENT OWNER, these patterns are adapted from *Organizational Patterns* and are reproduced with the gracious permission of Prentice Hall publishers. Some section cross-references and passages within the book have been omitted.

work. Many people in the enterprise are concerned about quality issues.

✤ ✤ ✤

It's important that the development organization ensures and maintains customer satisfaction by encouraging communication between customers and key development organization roles. This communication isn't the responsibility of any single customer satisfaction group; instead, the need pervades the entire organizational structure. Most organizations hesitate to allow direct contact between developers and customers, fearing that the developers are loose cannons who will promise to deliver things that exceed the scope of a job.

Yet you can't know all of the requirements up front, so developers constantly need to go back to customers for more information. In turn, customers constantly need to come back to developers with their insights, particularly when developers BUILD PROTOTYPES. After all, requirements changes occur even after design reviews are complete and coding has begun.

Many organizations depend on their marketing units to provide requirements data, but marketing doesn't provide design data [4]. The best that marketing can — or *should* — do is understand what will sell and why people will buy what you want to sell. Designers, in turn, must understand how people will use the product in a way that creates value for them. Good value sometimes leads to good market potential, but marketing usually looks at other factors (e.g., brand-name recognition, product name, and posturing in the market) that designers care little about.

Missing customer requirements is a serious problem; in fact, most problems in software systems can be traced to requirements problems [5, 14]. Yet it seems like so much effort to elicit these requirements, especially when the work does not directly produce a marketable artifact. It seems like make-work.

Customers are traditionally not part of the mainstream development effort, which makes it difficult to discover and incorporate their insights. Yet customer contact correlates with project success [17]. Trust relationships between managers and coders are often strained, so you don't want them to be the sole intermediaries between developers and customers.

Therefore: **Closely couple the customer role with the developer and architect roles, not just with QA or marketing roles. In short, developers and architects must talk freely and often with customers. When possible, engage customers in their own environment rather than bringing them into yours.**

Two elements are necessary to allow this interaction to happen: opportunity and culture. Developers must have the opportunity (and the means) to communicate with customers. They should meet customers personally to establish trust and a free flow of communication.

But if the organizational culture builds walls between customers and developers, these visits will be superficial. In particular, if system requirements must go through a lengthy formal process to be approved, the developer will be hamstrung, unable to respond to customer requests. Therefore, the organization must develop a culture in which developers have some latitude to respond to customers. We're not saying, however, that all control of requirements should be given to the developer. Order is necessary.

As Hugh Beyer and Karen Holtzblatt note, "Many common ways of working with customers remove [designers] from their work" [4]. One way to solve this problem is by "putting designers and engineers directly in the customer's work context" [4], a particularly important activity if you use customer engagement to create wholly new market directions for the enterprise rather than simply to refine existing work. Putting developers in the customer's work environment also trains developers' intuition about good design and human interfaces, and this intuition can fill in when

specific detailed requirements are unavailable [4].

Language is a key element of culture that can ensure a smooth customer engagement if treated properly and can smother it if treated badly. Don't make your customers learn UML or other technical notations; instead, do your best to learn *their* language and to communicate with them in the terms of *their* culture.

The QA team can monitor the relationship to keep the direction within contractual business limits while allowing a free flow of insights back and forth between developers and customers. Such communication can often flow unimpeded; at other times, however, it cannot (see the GATEKEEPER pattern). Note that the GATEKEEPER pattern is all about relationships and culture. It is the culture of respect for and interaction with customers that makes the communication effective, such as during the writing of use cases, which is described in PARTICIPATING AUDIENCE [6].

ENGAGE CUSTOMERS supports requirements discovery from the customer, as dictated by the patterns SCENARIOS DEFINE PROBLEM and BUILD PROTOTYPES. Other patterns like FIREWALLS also build on this pattern. The pattern RECOMMITMENT MEETING is a more formal derivative of this pattern in a different context.

A good understanding of customer needs can help you avoid rework after implementation is done. While it is also important to continuously engage customers through each development episode of iteration, early understanding helps launch the effort in the right direction. For example, a Navision team in Copenhagen, Denmark, believes that improvements in customer engagement helped save time on its development schedule.

Some processes and methods are founded on customer engagement, such as IBM's joint application development (JAD). Other methods are conducive to customer engagement, such as Cunningham and Beck's class responsibility collaborator (CRC) design technique. Other methods, and especially most CASE-based methods, are indifferent or harmful to customer engagement.

Even some of the best customer engagement techniques tend to end once the parties achieve some level of contractual agreement about what is to be delivered. Customer engagement in agile processes, however, goes far beyond this traditional stopping point. Developers need to assimilate the context in which their product will be used; this process is called *contextual design.* Contextual design means gathering data on customers' models of how they do their work rather than on how the program will solve the problem, as in use case modeling (see [4]). The pattern is called ENGAGE CUSTOMERS (note the plural) to support a domain view and to avoid the possibility of being blindsided by a single customer.

The project must be careful to temper interactions between customers and developers, using FIREWALLS, GATEKEEPER, and the QA organizational presence, as in ENGAGE QUALITY ASSURANCE. A major part of interacting with customers involves learning how they want to interact with the project as the unfolding software uncovers problems in requirements and systems engineering (APPLICATION DESIGN IS BOUNDED BY TEST DESIGN).

Note that maintenance of product quality is not the problem being solved here. Product quality is only one component of customer satisfaction. One study shows that 20% of customers will leave a company for another when they believe that they are being ignored and that 50% of customers will defect when the attention they receive is rude or unhelpful [26]. If customers experience software problems that cost more than US $100 to fix, and if the company does not fix these problems, only 9% of the customers would continue to do business with the company; 82% would do business with the company again if the problems were quickly resolved following the company's receipt of complaint [26].

…

DOMAIN EXPERTISE IN ROLES

> … You know the key atomic process roles (FORM FOLLOWS FUNCTION), including a characterization of the developer role.

✤ ✤ ✤

Matching staff with roles is one of the most difficult challenges of a growing and dynamic organization. All roles must be staffed with qualified individuals. Just as in a play, several actors may be assigned to a single role, and any actor may play several roles.

You'd like to use domain-nonspecific qualification criteria like college grades or years of experience to qualify people for jobs. Such an approach gives the project flexibility in staff allocation, and it helps it avoid being overly dependent on individual skill sets and experience. In short, the hope that such criteria might work provides project managers with a basis for keeping the project from becoming overly dependent on certain individuals who may leave or who may hold the organization hostage to gain higher salaries or to see their own policies implemented unilaterally. Nonetheless, successful projects tend to be staffed not with employees who possess textbook qualifications but with those who have already worked on successful projects.

Spreading expertise across roles complicates communication patterns. It makes it difficult for a developer or other project member to know whom to turn to for answers to domain-specific requirements and design questions.

Therefore: **Hire domain experts with proven track records, and staff the project with the expertise embodied in their roles.** Teams and groups tend to form around areas of common domain interest and focus. As mentioned, any actor may fill several roles, and in many cases, multiple actors can fill a given role.

Domain training is more important than process training. Organizations can benefit from having local gurus in all areas from application expertise to expertise in methods and language.

✤ ✤ ✤

DOMAIN EXPERTISE IN ROLES is a tool that helps ensure that roles can be successfully carried out. It also helps make roles autonomous. Empirically, highly productive projects hire deeply specialized experts.

…

If expertise becomes too narrow, the organization is at risk of losing key expertise in the event that a single person leaves, is promoted, and so on. Temper this pattern with MODERATE TRUCK NUMBER, which prevents the project from becoming overly dependent on a small number of individuals.

…

ARCHITECT CONTROLS PRODUCT

> … An organization of developers needs strategic technical direction.

✤ ✤ ✤

Even though a product is designed by many individuals, a project must strive to give the product elegance and cohesiveness. One might achieve this end by centralizing control, but most development teams view such control as a draconian measure. One person can't do everything, and no one has perfect foresight. However, the right information must flow through the right roles, and individual areas of competency and expertise must still be engaged. Furthermore, there needs to be some level of architectural vision. While some domain expertise is distributed through the ranks of the development team (DOMAIN EXPERTISE IN ROLES), the system view — and, in particular, the design principles that create a common culture for dialogue and construction — usually benefits from the conceptual integrity we associate with a single mind or small group.

Therefore: **Create an architect role as an embodiment of the principles that define an architectural style for the project and of the broad domain expertise that legitimizes such a style.**

The architect role should advise, influence, and communicate closely with developer roles. The architect doesn't dictate interfaces (other than in cases necessitating arbitration). Instead, the architect builds consensus with individual developers and developer subteams.

The architect is the principal bridge builder between development team members. The architect should also be in close touch with customers to ensure that the domain expertise is current, detailed, and relevant.

✤ ✤ ✤

This pattern does for the architecture what the PATRON ROLE pattern does for the organization: it provides technical focus and a rallying point for both technical and market-related work. The architect doesn't control the product in any dictatorial sense; instead, the architect provides inspirational guiding and leadership. We could have called this pattern ARCHITECT LEADS PRODUCT or ARCHITECT GUIDES PRODUCT, but these variations have their own problematic connotations.

Resentment can build against a totalitarian architect, so patterns like STAND-UP MEETING should be used to prevent this from occurring.

Intellectually large projects can build an ARCHITECTURE TEAM.

We have no designer role, because design is really the whole task. Managers fill a supporting role; empirically, they are rarely seen as controlling a process other than during crises.

While the architect controls the architectural direction, the DEVELOPER CONTROLS PROCESS, and there is still an OWNER PER DELIVERABLE. The architect is a chief developer (see ARCHITECT ALSO IMPLEMENTS) or, as Christopher Alexander describes himself, a "master builder" [1]. The architect's responsibilities include understanding requirements, framing the major system structure, and guiding the long-term evolution of that structure. The architect controls the product in the visualization process that accompanies the pattern ENGAGE QUALITY ASSURANCE.

Because ORGANIZATION FOLLOWS LOCATION and because of CONWAY'S LAW, there should probably be an architect at each location. Architects can be the focus of local allegiance, which is one of the most powerful cultural forces in geographically distributed development.

A more passive way of implementing this pattern is to have the architect review everything. We have seen this process work on several projects. However, the "truck number" — that is the number of employees with knowledge and expertise — was in danger on most of these projects as a result (MODERATE TRUCK NUMBER). Also, if there is a conscious plan for architects to review everything, they — in their capacity as developers (ARCHITECT ALSO IMPLEMENTS) — may swoop down and fix things that are the responsibility of others (CODE OWNERSHIP), which can be demoralizing for the original code author. The architect can review everything if the role still defers to the implementer for execution as well as the decision about making the change (STAND-UP MEETING).

Architectural control must balance developer authority and the role of keeper of the flame. Architects should not tread on developers' sense of code ownership or ownership of code development processes. Architects intervene in processes largely at the business level and should meddle in implementation processes only in exceptional circumstances.

…

DISTRIBUTE WORK EVENLY

> … An organization is working to organize itself in a way that makes the environment as enjoyable as possible and that makes the most effective use of human resources.

✤ ✤ ✤

It is easy to depend on just a few people to carry most of the organization's burdens. Managers like to rely on a few key people in order to minimize the number of interfaces they

need to manage. And some employees strive to do all they can out of a misplaced sense of monumental responsibility. In fact, we find that PRODUCER ROLES tend to have stronger communication networks than other support roles.

But if this uneven distribution of work continues, it becomes difficult for a heavily loaded role to sustain the communication networks necessary to the healthy functioning of the enterprise as a whole. Resentment may build among employees who don't feel central to the action. And those who are central may easily burn out.

Define the *communication intensity ratio* as the ratio of the number of communication paths of the busiest role to the average number of communication paths per role. Empirically, one finds that the organization has a problem — some underlying unhealthiness — if this ratio becomes too large.

Therefore: **Try to keep the communication intensity ratio to two or less.** (The authors have found that it's difficult to get much below two.) The easiest way to keep this ratio low is to have FEW ROLES. It also helps to identify the PRODUCER ROLES and eliminate any deadbeat roles. You can also identify all the communication paths of the most central role and see which are really necessary. Some of this communication overhead isn't very subtle, and these cases are easy to identify. In many cases, you can eliminate redundant or misdirected communication using simple and direct methods without delving into the deep structure or principles of the organization. Other situations require more finesse and generativity, building on other patterns in this pattern language.

✤ ✤ ✤

As we've discussed, if an organization becomes sufficiently out of balance that the work is concentrated in the hands of only a few people, those people are likely to experience burnout. Such unevenness may also indicate deeper organizational problems. For example, those carrying a lighter load may not have the technical skills or the human interaction skills needed to integrate into the larger team or organization. Personality differences can be addressed with human effectiveness training programs (e.g., appreciating differences or giving effective presentations). Skill mismatches can be dealt with by reassigning people or through skill training.

An imbalance may also point to insecurity in the person or clique that tries to take on all the work. Such insecurity may manifest as lack of trust in others. Encounters between the insecure parties and the rest of the project team polarize the positions of each, and a form of "schismogenesis" may set in: the rise of factions in the organization (THE OPEN/CLOSED PRINCIPLE OF TEAMS). Insecure subgroups may withdraw from the rest of the team or even try to hijack the project by strong-arming people into doing their bidding. Such behaviors may be accompanied by some of the dynamics of burnout (e.g., shutting down communication with "outsiders"). Patterns like GATEKEEPER, WISE FOOL, and PATRON can help avoid the creation of unhealthy factions.

In any of these dysfunctional situations, it is the job of the manager to counsel the parties involved and to forcefully intervene. Correcting the problem is often an intricate and time-consuming process. This pattern follows PRODUCER ROLES and PRODUCERS IN THE MIDDLE, which are prerequisites to SHAPING CIRCULATION REALMS. This pattern itself is a refinement of SHAPING CIRCULATION REALMS. FEW ROLES makes this pattern happen. This pattern can be implemented and elaborated by using RESPONSIBILITIES ENGAGE and THREE TO SEVEN HELPERS PER ROLE.

Figure 2 shows the communication intensity ratio data gathered from some of our early research subjects. We find that successful organizations tend to be near the origin point on the graph.

FUNCTION OWNER AND COMPONENT OWNER

> … The functions in your project tend to cut across the components of your project. You recognize the importance of organizing according to the architecture of the project (CONWAY'S LAW), but things aren't simple.

If you organize teams by components, functions suffer, and vice versa.

You may be organized by function or use case, with no component ownership. On the other hand, you may be organized by class or component, with no function or use case ownership. In either case, you ask the same question: "What happens when two people need to program the same function?"

You want ownership and consistency in these functions as well as in the components. And the components must be shared across the teams. If you just assign function owners, the components become shared and lose their identity and integrity. But if you have only component owners, the functions become orphans. Who ensures that they get done?

Therefore: **Ensure that every function has an owner and that every component has an owner.** Make sure that every component has a responsible owner and that every delivered function has a responsible owner. The component owner answers for the integrity and quality of the component. The function owner ensures that the function gets delivered. If the component owners all refuse to incorporate something needed to deliver end functionality, the function owner sees that the missing code is put in a function-specific place.

There may be friction between component owners and function owners. You may find the need for the ENVY/Developer model of ownership, where there is a common part and an application-specific part to each component

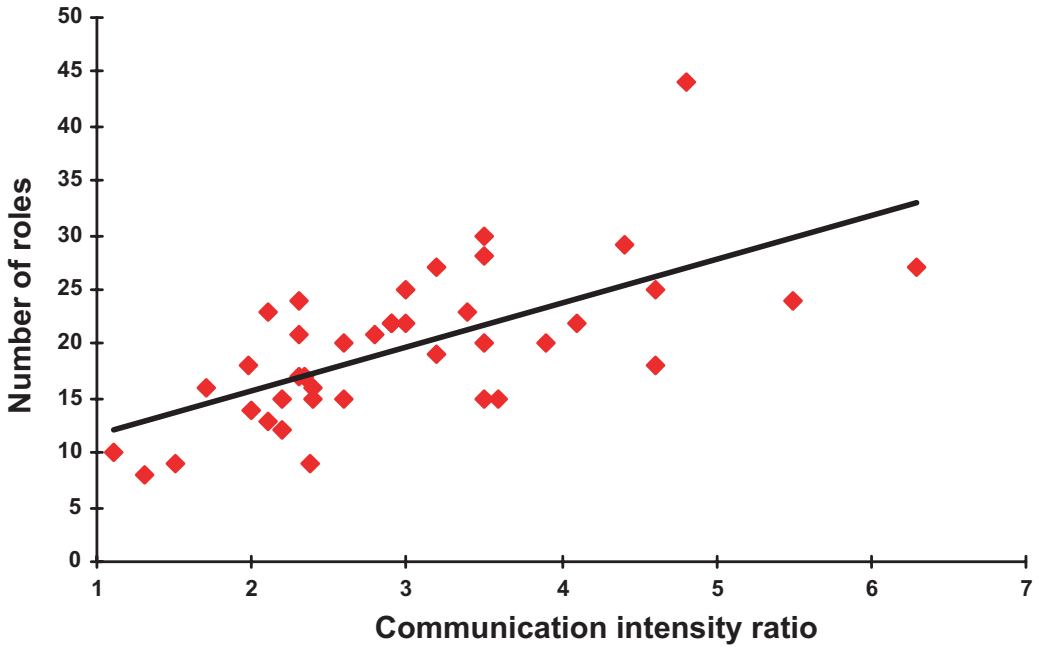


Figure 2 — The relationship between the communication intensity ratio and an organization's number of roles.

or class. No conflict resolution is needed in this strategy since the component owner has right of refusal to any request, and the function owner has recourse by working with others to get the job done.

…

MERCENARY ANALYST[3]

> … You are assembling the roles for the organization. The organization exists in a context where external reviewers, customers, and internal developers expect to use project documentation to understand the system architecture and its internal workings. (User documentation is considered separately.) Supporting both a design notation and the related project documentation is too tedious a job for those who are directly contributing to product artifacts.

✤ ✤ ✤

Technical documentation is the dirty work required in every project. It's important to create — and even more, to maintain — good documentation for the project team's subsequent use. But who writes these documents?

If developers create their own documentation, "real" work is hampered. Meeting software deadlines means money for the organization, and technical documentation is one of those things we tell ourselves that we can defer until there is time to do it. But often, the time never comes, and an organization without good internal technical documentation of its system has a serious handicap. Nonetheless, much internal documentation is often write-only; it is rarely read after it is written. Further, engineers often lack good communication skills.

Many projects use tools like IBM's Rational Rose for design. These tools produce pretty pictures. A good picture, however, is not necessarily a good design, and architects can become victims of the elegance of their own drawings.

Therefore: **Hire a technical writer who is proficient in the necessary domains but who does not have a stake in the design itself.** This person will capture the design using a suitable notation and will format and publish the design for reviews and for use by the organization itself.

✤ ✤ ✤

The documentation itself should be maintained online if possible. It must be kept up to date (therefore, MERCENARY ANALYST is a full-time job), and it should relate to customer scenarios (SCENARIOS DEFINE PROBLEM). Note, however, that all team members need to provide input to keep the documentation current. The AD HOC CORRECTIONS pattern [24] suggests that a master copy of the documentation be kept and that team members write corrections in the margin. One team member is assigned to periodically update the document.

The success of this pattern depends on the ability to find a suitably skilled agent to fill the role of mercenary analyst. If the pattern succeeds, the new context defines a project whose progress can be reviewed (STAND-UP MEETING) and monitored by community experts outside the project.

If the MERCENARY ANALYST really is a "mercenary" who, as Paul Chisholm describes, "rides into town, gets the early stuff documented, kisses his horse, saddles up his girl, and rides off into the sunset" [12], then it's good to retain some of the expertise by combining MERCENARY ANALYST with DEVELOPING IN PAIRS.

This pattern, uncommon though empirically grounded and effective, is found in Borland's Quattro Pro for Windows and in many AT&T projects (e.g., a joint venture based in New Jersey, a formative organization in switching support, and others). It is difficult to find people with the necessary skills to fill this role.

Witold Rybczynski writes the following regarding the difficulties associated with architecture:

> Here is another liability: beautiful drawings can become ends in themselves. Often, if the drawing deceives, it is not only the viewer who is enchanted but also the maker, who is the

[3]A version of this pattern first appeared in [8].

> victim of his own artifice. Alberti understood this danger and pointed out that architects should not try to imitate painters and produce lifelike drawings. The purpose of architectural drawings, according to him, was merely to illustrate the relationship of the various parts. … Alberti understood, as many architects of today do not, that the rules of drawing and the rules of building are not one and the same, and mastery of the former does not ensure success in the latter. [22]

…

In e-mail correspondence with one of the coauthors, Richard Gabriel noted the following important traits that distinguish this role [16]:

- Strong skills as a meeting facilitator
- An inclination for organization
- Attentiveness to detail
- Possesses written instructional material (for software)
- A lack of ego investment in the material being documented
- Intelligence and a high level of education

In exceptional cases, the MERCENARY ANALYST can actually have a stake in the design.

Elizabeth Hanes Perry writes about filling this role:

> When I fill this role, I most definitely have a stake in the design: I want to make sure it's elegant, consistent, and clean. The architect has primary responsibility, of course, but I also suggest places in which the design conflicts with itself or may lead to future misunderstandings. As I see it, a software architecture is an idea. The designer/implementers are responsible for expressing that idea (or those ideas) as code; I express it/them as prose. Both are projections of the idea into a particular plane. When there's a conflict, the code is probably correct. [20]

Many projects put faith in tools and notations such as UML to improve quality. But as Perry points out, tools largely provide the forum and opportunity for a human being to engage in the processes and convey the insights that contribute to quality. For documentation to have added value as a quality tool, the documentation process must proceed in the spirit of this admonition.

…

ARCHITECT ALSO IMPLEMENTS

> … An organization is being built to serve an identified market or markets (ORGANIZATION FOLLOWS MARKET). Going forward, the project needs the necessary architectural breadth to cover its markets and to ensure smooth evolution, but it can't be blindsided by pragmatic engineering and implementation concerns. Furthermore, the project needs to carry through a singular architectural vision from conception to implementation if it is to have conceptual integrity.

A software project must broaden the scope of leadership without sacrificing depth and attention to pragmatics. Though developers are good at making individual design and implementation decisions, a project needs an overall guiding strategic technical direction. This direction usually comes from the architect. However, too many software architects limit their thinking and direction to abstractions, and abstraction is a disciplined form of ignorance. Too many projects fail to capture the "details" of performance, subtleties of APIs, and interworking of components — or at best, they discover such problems too late.

It's possible that this problem could be solved with totalitarian control if one had a perfect plan. But as we've mentioned previously, even if that were possible, most development teams view this level of control as excessive.

The right information must flow through the right roles. In particular, developers must latch onto the strategic vision and carry responsibility for implementation. The architect, and to some degree the developers, must also understand the application needs and be able to portend the long-term structure of the system. But a

more centralized locus of strategic direction should keep the project from floundering, ensure that the necessary details are addressed, and track the emerging fit of all the pieces into a whole. Sometimes, understanding how these pieces fit together requires a corresponding understanding of low-level details of component interaction, protocols, APIs, performance, or reliability concerns.

Therefore: **Beyond advising and communicating with developers, architects should also participate in implementation.** The architect should be organizationally engaged with developers and should write code. The architect may implement along with a developer using DEVELOPING IN PAIRS.

✤ ✤ ✤

If the architect implements, the development organization perceives buy-in from the guiding architects, and that perception can avail itself of architectural expertise directly. The architects also learn by seeing the firsthand results of their decisions and designs, thus providing feedback on the development process.

In a project one of the coauthors worked on recently, the importance of making this pattern explicit became clear. The architecture team was being assembled from dispersed geographic locations with narrow communication bandwidth between them. Though general architectural responsibilities were identified and the roles were staffed, one group expected architects also to implement code, whereas the other did not.

One manager suggests that, on some projects, architects should focus only on the implementation of a common infrastructure; the implementation of noncore code should thus be left solely to the developer role. This division of responsibilities may work on some projects; however, the architect must have a strong feel for recurring application needs in order to build long-term robust frameworks. If architects work only on infrastructural concerns and lack an engaged appreciation of application needs, a disconnect results between the infrastructure (framework and middleware) and the application.

…

Though the architect should be able to understand the minutiae of development, it is not necessarily the architect's business to deal with such details day in and day out. The architect is the keeper of the flame, the owner of the principles that the project follows. These principles in turn shape structure. Much of the structure can emerge from a consensus-driven process guided by the architect; in fact, in practice much of what architects do involves such high-level guidance [11]. A related pattern is GURU DOES ALL from the collection of Don Olson [19], which states the following:

> A newly formed team is given a project with a tight schedule, uncertain requirements, uneven distribution of skills, and new technologies. Let the most skilled and knowledgeable developer drive the design and implement the critical pieces. This can be an antipattern. [21]

The key element of this pattern is to give the critical pieces of the project to the most skilled and knowledgeable practitioners (DOMAIN EXPERTISE IN ROLES).

…

FIREWALLS

> ... An organization of developers has formed in a corporate or social context where they are scrutinized by peers and by funders, customers, and other "outsiders." Project implementers are often distracted by outsiders who feel a need to offer input and criticism.

✤ ✤ ✤

It's important to placate stakeholders who feel a need to "help" by giving them access to low levels of the project, without distracting developers and others who are moving toward project completion.

Isolationism doesn't work because information flow is important. But communication overhead increases nonlinearly

with the number of external collaborators.

Many interruptions are noise.

Maturity and progress are more highly correlated with being in control than with being effectively controlled.

Therefore: **Create a manager role that shields other development personnel from interaction with external roles.** The responsibility of this role is "to keep the pests away."

✤ ✤ ✤

The new organization isolates developers from extraneous external interrupts. To avoid isolationism, this pattern must be tempered with others, such as ENGAGE CUSTOMERS and GATEKEEPER.

This pattern was present in both the Borland Quattro Pro for Windows project [10, Chapter 8] and in a hyperproductive development team we studied [10, Chapter 9]. In addition, see the pattern ENGAGE CUSTOMERS, which complements this pattern.

…

DEVELOPER CONTROLS PROCESS

> ... An organization has come together to build software for a new market in an immature domain or in a domain that is unfamiliar to the development team. Progress will be marked by an INFORMAL LABOR PLAN. The necessary roles have been defined and initially staffed.

✤ ✤ ✤

Like any culture, a development culture can benefit from recognizing a focal point of project direction and communication. Successful organizations work in an organic way with a minimum of centralized control. Yet important points of focus, embodied in roles, tie together ideas, requirements, and constraints into an artifact ready for testing, packaging, marketing, and delivery.

Most development teams view strict control as excessive and heavy-handed. The right information must flow through the right roles. You need to support information flow across analysis, design, and implementation.

Because developers contribute directly to the end-user-visible artifact, they are in the best position to have accountability for the product. Of all roles, they have the largest stake in the largest number of phases of product development, and there should be no accountability without control. A manager has some accountability as well, to the extent that he or she indirectly supports delivery of the user-visible artifacts. These are process issues.

Therefore: **Make the developer the focal point of process information.** In the spirit of ORGANIZATION FOLLOWS MARKET, place the developer role at a hub of the process for a given feature. A feature is a unit of system functionality (implemented largely in software) that can be separately marketed and for which customers are willing to pay. Developer responsibilities include understanding requirements, reviewing the solution structure and algorithm with peers, building the implementation, and performing unit testing.

…

Note that other hubs, such as the manager role, may exist as well, though they are less central than the developer role.

✤ ✤ ✤

The developer who is at the hub of a particular feature may be granted that position according to FEATURE ASSIGNMENT, but more generally developers should be at the communication hub of whatever process engages them in writing code for the customer. This pattern encourages a structure that supports its prime information consumer. The developer can be moved toward the center of the process using the patterns WORK FLOWS INWARD and MOVE RESPONSIBILITIES. Though the developer should be a key role, care must be taken not to overburden the role. This pattern should be balanced with MERCENARY ANALYST, FIREWALLS, and GATEKEEPER and more general load-balancing patterns like

RESPONSIBILITIES ENGAGE, HALLWAY CHATTER, and MOVE RESPONSIBILITIES. The developer should enjoy particularly strong support from the PATRON ROLE, and conflicts can be escalated to the PATRON ROLE when consensus breaks down.

If the developer controls the process, it's possible to implement the pattern WORK FLOWS INWARD. Developers, of course, don't control the process unilaterally but as a collective group, starting with DEVELOPING IN PAIRS.

As we've said previously, we don't have a designer role because design is really the whole task. Managers fill a supporting role; empirically, they rarely control a process except during crises. And again, the developer controls the process, the architect controls the product. This communication is particularly important in domains that are not well understood, so that iteration can take place to explore the domain with the customer.

In a mature domain, consider HUB, SPOKE, AND RIM — a pattern that orchestrates the work of several pipelined workers around a centralized coordinator — as an alternative.

You can still write down your process as part of a process improvement program. But keep the documentation light; many organizations have found that one page per process is good enough. And ensure that each process step meets a need that you can tie to your organization's value proposition. Most often, this value is or should be tied to the product you produce for a paying customer. If it isn't obvious how the process step helps to achieve what you know the customer wants, then do the right thing instead.

APPLYING THE PATTERNS

Patterns as Organizational Building Blocks

Organizational patterns are a tool for organizational change, growth, and repair. An organization is a system; that means changes to one part of the organization affect many other parts. As a system, an organization has an intricate fabric that links it together. In a pattern language, patterns are the fabric of your organization. Any single pattern can be used to patch that fabric, whether to grow the organization or to fix it when you notice something wrong. Even growth should be viewed as an act of repair. Repair is a kind of change that must be compatible with what preceded the fix. So though each pattern should be applied incrementally, with local impact, it refines and complements the patterns that are already there in a way that makes the entire organization more robust *as a system*.

Let's assume that you live in a charming old house. Let's compare the growth and updating of the house with the maintenance of an evolving organization. Like a house, an organization wears and tears over time. You notice where the organization needs work and find the appropriate pattern to strengthen it. However, to repair the organization, you have to understand its basic style, patterns of organization, and principles of construction, just as you would in remodeling your old house. For the new organization to retain the core properties it strives to retain over time, the patterns of the new construction must match the old. The new materials must be malleable to fit the old and to support future growth and adaptation. Similarly, organizational maintenance must mesh with what is already present. You can choose patterns that harmonize with one another and with your organization, since each pattern is flexible enough to adapt to your unique circumstances. Of course, this metaphor doesn't always apply perfectly, and software development organizations have much richer structure than homes. But this notion of mixing compatible patterns is critical to solving your organization's system problems.

The Role of the Pattern Language

We've mentioned several times that no pattern stands alone: patterns are meant to be used alongside one another to be effective. This emphasizes what you probably already know: there is no single miracle cure for an organization, no silver bullet to organizational success. You must do many

things right. And "doing things right" differs from organization to organization: each pattern must be tailored to its context. Each of these customized changes produces a new and slightly better organization that ensuing patterns can build on.

Even though each pattern can be customized, and each organization reflects a slightly different set of patterns, we can still make strong claims about successful software development organizations in general. Most have the pattern ENGAGE CUSTOMERS; without it, you won't know what to build. Most excellent software development organizations share a small number of common patterns. In many cases, the organizations grew in much the same way: the patterns were applied in a certain order.

Think of patterns as words in a dictionary, and think of your organization as a sentence composed from those words. We need a set of rules for how to combine the words in useful ways: that is, we need a language of the words, a language of the patterns. Based on this metaphor, a collection of patterns, together with rules for assembling them in compatible and useful ways, is called a *pattern language*. We can depict the language visually, with the suggested partial ordering of pattern application, with the most basic pattern at the top. Each pattern refines those above it and provides a foundation for those below it. Each pattern helps complete those patterns above and below it in the language.

Figure 3 shows a pattern language for project management, built from organizational patterns that have been collected and published over the past decade. As the figure shows, patterns from other pattern languages overlap with project management and, as such, may appear in several pattern languages.

The language is taken from *Organizational Patterns of Agile Software Development*, as are many other concepts in this report. As mentioned previously, it is one of four pattern languages described in the book. The four languages are Project Management, Organizational Style, Piecemeal Growth, and People and Code. The patterns overlap somewhat across the pattern languages; the "home" pattern language for each pattern is shown in the graph. A pattern language guides you through the evolution of your organization. It is not a prescription but a suggestion based on what has often worked before. Don't be afraid to use your insight, intuition, and domain specialization to fine-tune the pattern language. Think of organizational repair as an almost playful activity of discovery and adventure.

A Language of the Top 10 Patterns

We can make a mini-pattern language from the top 10 patterns discussed in this report. Figure 4 on page 21 shows what this pattern language might look like.

You would start with the pattern UNITY OF PURPOSE as a foundation and then work your way down the figure. You confront a choice right away. Which pattern should come first — DOMAIN EXPERTISE IN ROLES, ARCHITECT CONTROLS PRODUCT, or ENGAGE CUSTOMERS? In many sectors, these three patterns are often independent: domain expertise, as embodied in domain experts and experienced architects, depends more on the core business than on the needs of individual customers. Once you have DOMAIN EXPERTISE IN ROLES and ARCHITECT CONTROLS PRODUCT, your architecture team can begin an implementation: ARCHITECT ALSO IMPLEMENTS. And now you can take the domain experts you hired and divide the work among them, using customer needs to guide project direction. You need FIREWALLS to insulate developers from customer whims, while still allowing for customer insights to reach developers. Now you're ready to start development in full swing with DEVELOPER CONTROLS PROCESS.

The Fundamental Process

All systems grow through a fundamentally iterative process. Christopher Alexander has described such a process for the architecture of buildings and of the fine craftsmanship that "decorates" these buildings [1]. His process draws heavily on the

processes that biologists have come to understand as fundamental to autopoietic systems and life and on the processes that physicists understand as fundamental to the nature of matter and existence. We have found the following process to be an effective way to introduce patterns one at a time, in a way that they fit with one another. Because it is based on local adaptation and piecemeal growth, it minimizes risk [10]:

1. **Consider your organization as a whole and find the weakest part: that is, the one with the highest density of unresolved forces.** If you are working your way through the pattern language, look at the

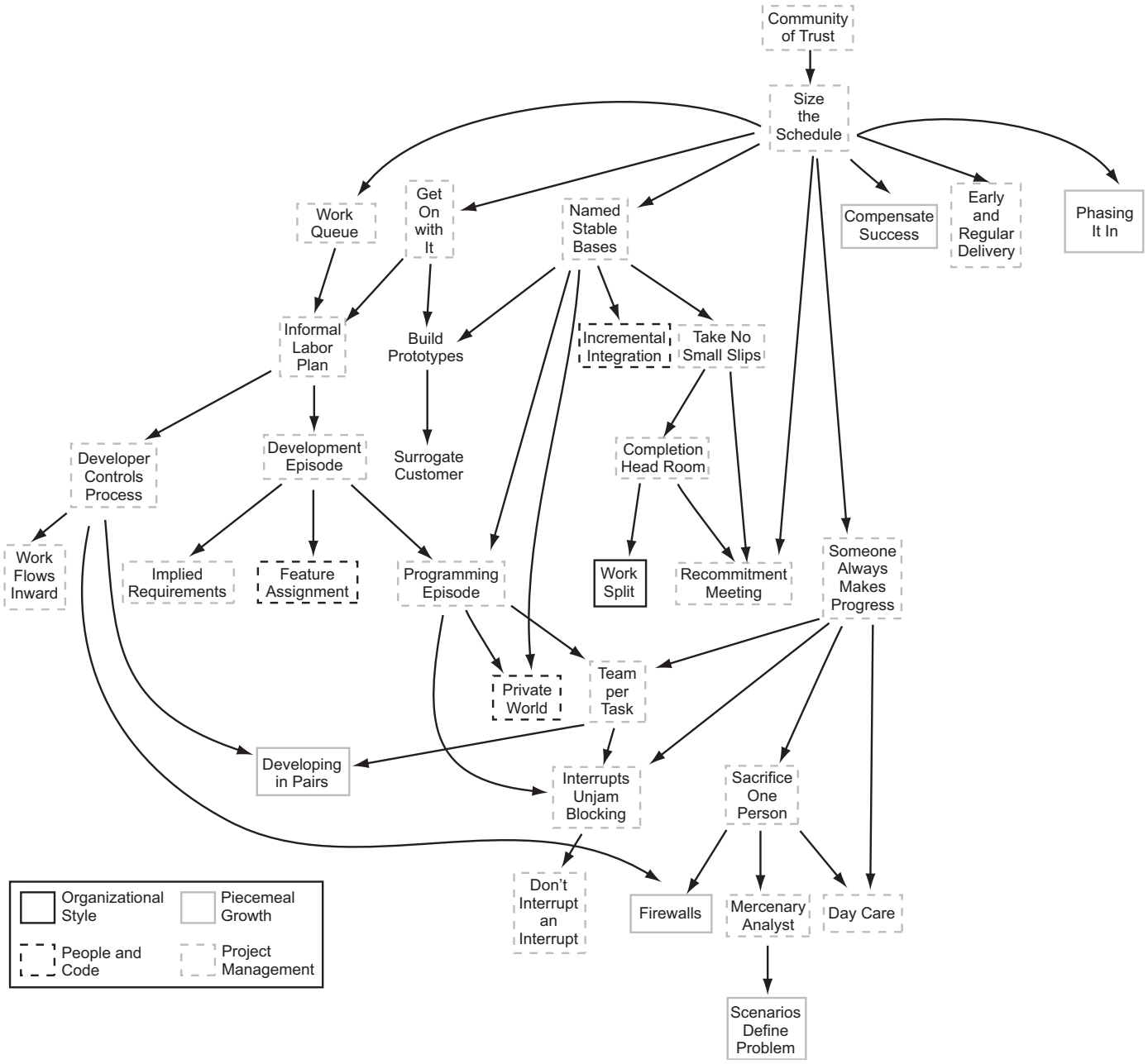


Figure 3 — The Project Management pattern language.

areas touched by the next patterns in the language. If your business environment has changed, look at the areas touched by those changes. If you have just applied another pattern, look at the forces that have been exposed or left unbalanced by the application of that pattern.

2. **Calibrate the forces with respect to your business and organizational goals.** Pay attention to factors that recur in the tradeoffs related to the problem you are solving, treat transients as transients, and then return your focus to the long-term and systems issues. Think about your personal and corporate values and how they amplify some forces and diminish others. Are you striving for profitability? If so, which foundational values in your organization underlie profitability? And is profitability *really* your main concern right now, or is morale affecting productivity, which is in turn affecting profitability? Dig deeper.

3. **Let the patterns inspire you to find a way to balance the forces.** You might find that an existing organizational pattern is ready-made for your situation. It is more often the case that an organizational pattern will inspire your instinct to recognize a customized path to a solution. *Follow your instinct, not the pattern.* In many ways, the main goal of a pattern is to unleash the instinct within you that derives from your experience but that may have been quieted by local management practices or by exposure to a cookie-cutter management technique adopted from a book or corporate training program.

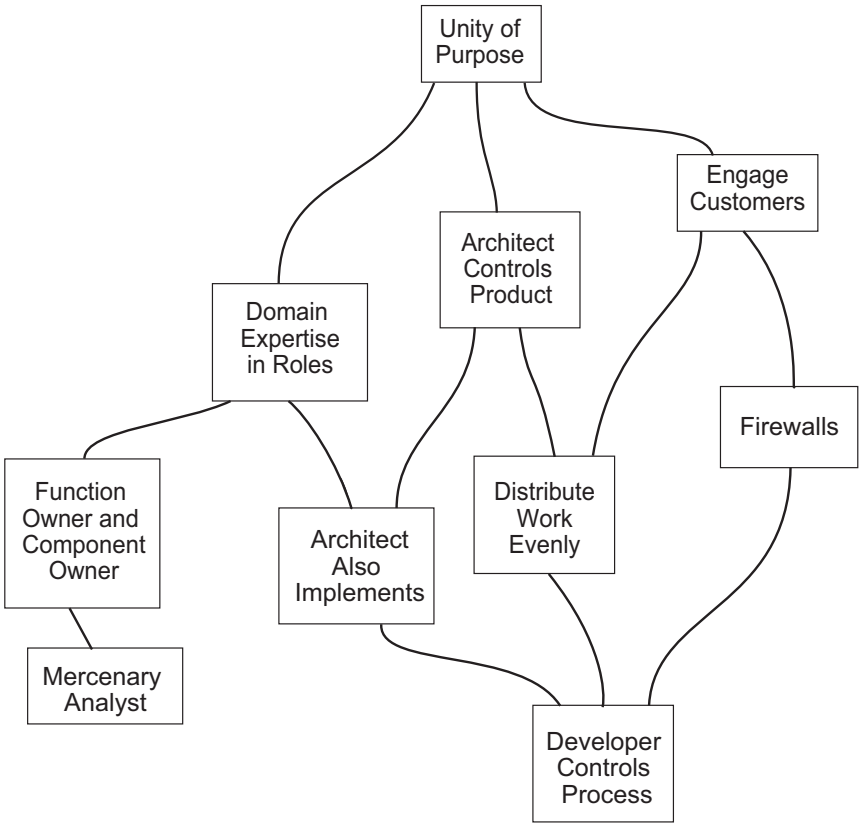


Figure 4 — The top 10 patterns as a pattern language.

4. **Keep the big picture.** Adjust the pattern in a way that increases the organization's health at the next level in the organizational structure or in the next larger context or scope.

5. **Strive for balance.** Most of these patterns are communication patterns, and communication is always a two-way street. Be attentive to both sides of the communication structure or other structure of the pattern.

6. **Be attentive to feedback; if the pattern does not strengthen the organization, then back it out.** Do not try to fix it by adding another pattern. Choosing a pattern is not a science; it is guided, informed guesswork. Occasionally, you will guess wrong. Patterns are small enough that the cost of making a mistake is minor and almost always reversible. Be open with your people that the technique doesn't work, and enlist their support in trying another solution. Keep in mind that people resist change and

that sometimes a bit more time or encouragement is enough to gain acceptance. To succeed, a pattern must be adopted by the culture rather than being forced onto the culture. Eventually, a good pattern will lead you to a new context — and to a new set of forces to balance and problems to address.

RESULTS YOU CAN EXPECT

Of course, we cannot promise miracles. Most important perhaps is to have patience; great change rarely happens overnight and usually comes about from the interaction between several patterns rather than from a single pattern. The most difficult aspect of organizational change is timing: to know when to give up on a pattern, back out, and try something else. Sometimes an organization proceeds step by step toward improved behavior; sometimes organizations persist in bad behavior until, one day, a precipitous change happens. We offer no magic solution to this dilemma but suggest that you should be most attentive to the morale and feelings of your employees as an indicator of whether a pattern is working. Keep in close touch with the people affected by the current pattern and carefully consider their sense of progress during the adoption of a pattern by the organization.

The Success Story

Almost every one of the 100 organizations we have researched has benefited from considering and tailoring these patterns to its organizational needs. Richard Gabriel explains the role of patterns in turning around the ParcPlace Systems software team [15]. One of our organizational interventions probably saved a large insurance company from demise; in another insurance company in Vienna (Alliance Elementar), the patterns helped create connections between the engineering group and management and sparked management-level initiatives that led to renewed organizational health and growth. These patterns have proven themselves in European, American, and Middle Eastern companies. In some cases, the benefits were local and modest; in other cases, the survival of the entire enterprise was at stake.

CONCLUSION

What we have discussed here will help heal the most common problems we have observed in recent software development efforts worldwide. However, organizational development is an ongoing concern of great depth and breadth, and we exhort you to broaden the tools available to you through other resources. Remember that though there is a lot of literature on management theory, and while theory can be useful, it is very difficult to justify any approach to improvement on a theoretical basis alone. The best resources are those that reflect empirical insight. While the organizational patterns literature and the book *Organizational Patterns* in particular are modern examples of empirically grounded organizational literature, they are not without company. The writings of Capers Jones, Gerald Weinberg, and others are also grounded in experience.

Experience is the best teacher, and patterns are designed to provide an incremental, low-risk path to process improvement. One of the best ways to learn more about these patterns is to try implementing them. If you are interested in learning about how to start a pattern-based organizational improvement program inside your group or department, TietoEnator (www.tietoenator/oap.dk) offers a facilitation service under its Organizational Agility Program, or you can contact the authors directly.

REFERENCES

1. Alexander, Christopher, Sara Ishikawa, and Murray Silverstein, with Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, 1977.

2. Beck, Kent. "The Metaphor Metaphor." Web page for *OOPSLA 2002* accessed 20 March 2005 (http://oopsla.acm.org/oopsla2002/fp/files/spe-metahpor.html).

3. Beedle, Mike. Book review of *Organizational Patterns of Agile*

Software Development on Amazon.com, 19 September 2004 (www.amazon.com/exec/obidos/ASIN/0131467409).

4. Beyer, Hugh, and Karen Holtzblatt. *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufmann, 1998.

5. Boehm, Barry W. "Software Engineering." *IEEE Transactions on Computers*, Vol. 25, No. 12, 1976, pp. 1226-1241.

6. Bramble, Paul, Alistair Cockburn, Andy Pols, and Steve Adolph. *Patterns for Effective Use Cases*. Addison-Wesley, 2002.

7. Cockburn, Alistair. "Prioritizing Forces in Software Design." In *Pattern Languages of Program Design 2*, John M. Vlissides, James O. Coplien, and Norman L. Kerth (eds.). Addison-Wesley, 1996.

8. Cockburn, Alistair. *Surviving Object-Oriented Projects: A Manager's Guide.* Addison-Wesley, 1998.

9. Coplien, James O., and Jon Erickson. "Examining the Software Development Process." *Dr. Dobb's Journal,* Vol. 19, No. 11, October 1994, pp. 88-95.

10. Coplien, James O., and Neil B. Harrison. *Organizational Patterns of Agile Software Development*. Prentice Hall, 2005.

11. Coplien, James O., and Martine Devos. "Architecture as Metaphor." Proceedings of the *World Conference on Systemics, Cybernetics and Informatics*, Orlando, Florida, USA, July 2000, pp. 737-742.

12. Chisholm, Paul S.R. Personal communication with James Coplien, 1994.

13. Cunningham, Ward. "EPISODES: A Pattern Language of Competitive Development." In *Pattern Languages of Program Design 2*, John M. Vlissides, James O. Coplien, and Norman L. Kerth (eds.). Addison-Wesley, 1996, pp. 371-388.

14. Daly, Edmund B. "Management of Software Development." *IEEE Transactions on Software Engineering*, Vol. 3, No. 3, May 1997, pp. 229-242.

15. Gabriel, Richard P. "Productivity: Is There a Silver Bullet?" *Journal of Object-Oriented Programming*, Vol. 7, No. 1, March/April 1994, pp. 89-92.

16. Gabriel, Richard. E-mail message to James Coplien, 8 May 1995.

17. Keil, Mark, and Erran Carmel. "Customer-Developer Links in Software Development." *Communications of the ACM*, Vol. 38, No. 5, May 1995, pp. 33-44.

18. Kerth, Norman L. "Caterpillar's Fate: A Pattern Language for Transformation from Analysis to Design." In *Pattern Languages of Program Design,* James O. Coplien and Douglas C. Schmidt (eds.). Addison-Wesley, 1995.

19. Olson, Don. S. "Pattern on the Fly." *The Patterns Handbook,* 1998, pp. 141-170.

20. Perry, Elizabeth Hanes. Personal communication with James Coplien, 1997.

21. Rising, Linda. *The Pattern Almanac*. Addison-Wesley, 2000.

22. Rybczynski, Witold. *The Most Beautiful House in the World.* Penguin, 1989.

23. Schwaber, Ken, and Mike Beedle. *Agile Software Development with SCRUM*. Prentice Hall, 2001.

24. Weir, Charles. "Patterns for Designing in Teams." In *Pattern Languages of Program Design 3*, Robert Martin, Dirk Riehle, and Frank Buschmann (eds.). Addison-Wesley, 1998, pp. 496-499.

25. Whitenack, Bruce. "RAPPeL: A Requirements Analysis Process Pattern Language for Object-Oriented Development." In *Pattern Languages of Program Design*, James O. Coplien and Douglas C. Schmidt (eds.). Addison-Wesley, 1995, pp. 259-291.

26. Zuckerman, Marilyn R., and Lewis J. Hatala. *Incredibly American: Releasing the Heart of Quality.* American Society for Quality Press, 1992, pp. 81-83.

ABOUT THE AUTHORS

James O. Coplien is Object Architect at DAFCA, Inc., an electronic design automation firm in Framingham, Massachusetts, USA. He has been a software professional for more than 30 years. His career spans areas as broad as telecommunications software development (at AT&T), software research (at Bell Laboratories), academia (at Vrije Universiteit Brussel, where he held the 2003-2004 Vloebergh Lehrstuhl), EDA (at DAFCA), and international consulting and lecturing. His book *Advanced C++* shaped a generation of developers, and his two subsequent books have set new standards in software design and development. He is a founding member and member emeritus of the Hillside Group, which founded software patterns. He is a coauthor or editor of three major patterns books, including *Organizational Patterns of Agile Software Development*. He can be reached at JOCoplien@cs.com.

Neil B. Harrison is a distinguished member of technical staff at Avaya Labs, where he develops communications software. He has been involved in software development and research for more than 20 years as both a developer and team leader. He has studied software development organizations for 10 years and is coauthor of the critically acclaimed book *Organizational Patterns of Agile Software Development*.

Mr. Harrison has been a leader in the software pattern community since 1994. He has taught pattern courses and published patterns, including patterns in *Pattern Languages of Program Design*, volumes 2 and 3. He was the lead editor of *Pattern Languages of Program Design*, volume 4. He is acknowledged as the world's leading expert on pattern shepherding and has a shepherding award named after him. He is a member of the board of directors of the Hillside Group. He can be reached at nbharrison@avaya.com.

Gertrud Bjørnvig is a senior consultant in TietoEnator in Copenhagen, Denmark. She does organizational work for clients based on organizational patterns and uses these principles for organizational improvement in the Scandinavian countries, drawing on more than 15 years of experience in the industry. Her past organizational work includes interventions in multiple corporate mergers to align the development processes while retaining the benefits of the highly effective work culture that was in place. She is an accomplished project manager, focusing on on-time delivery using agile approaches. She is an expert on use cases and developed the first patterns for the application of use cases in development projects. She was a founding program committee member for VikingPLoP and is one of the founders of the Danish Agile User Group. She can be reached at Gertrud.Bjornvig@tietoenator.com.

Index of published issues

> Agile Project Management Advisory Service

Upcoming Topics

- ***Agile Estimating and Planing: Embracing Change***
  by Mike Cohn
- ***Integrating Software Development and Project Management Methods***
  by Robert Wysocki

This index includes Agile Project Management *Executive Reports* and *Executive Updates* that have been recently published, plus upcoming *Executive Report* topics. Reports that have already been published are available electronically in the Online Resource Center. The Resource Center includes the entire Agile Project Management Advisory Service archives plus additional articles authored by Cutter Consortium Senior Consultants on the topic of project management.

For information

on beginning a subscription or upgrading your current subscription to include access to the Online Resource Center, contact your account representative directly or call +1 781 648 8700 or send e-mail to sales@cutter.com.

CUTTER CONSORTIUM
ACCESS TO THE EXPERTS

Executive Reports

Vol. 6, No. 6 *Organizational Patterns: Building on the Agile Pattern Foundations* by James O. Coplien, Neil B. Harrison, and Gertrud Bjørnvig

Vol. 6, No. 5 *Principle-Centered Agile Project Portfolio Management* by Donna Fitzgerald

Vol. 6, No. 4 *The Politics of Design and Usability* by Whitney Quesenbery

Vol. 6, No. 3 *The New World of Teams: Ad Hoc and Virtual* by Rob Thomsett

Vol. 6, No. 2 *Industrial XP: Making XP Work in Large Organizations* by Joshua Kerievsky

Vol. 6, No. 1 *Agile for the Enterprise: From Agile Teams to Agile Organizations* by Jim Highsmith

Vol. 5, No. 12 *Extreme Programming Practices: What's on Top?* by Laurie Williams

Vol. 5, No. 11 *Getting It Right, Getting It Done: Improving Team Productivity and Quality* by Pamela Hager

Vol. 5, No. 10 *Usability and the Agile Project Management Process Framework* by Jonathan D. Addelston and Theresa A. O'Connell

Vol. 5, No. 9 *Making Decisions Using Software Product Metrics* by Khaled El Emam

Vol. 5, No. 8 *Leading Extreme Projects to Success* by Doug DeCarlo

Vol. 5, No. 7 *Pushing the Envelope: Managing Very Large Projects* by Ken Orr

Vol. 5, No. 6 *The Usability Challenge* by Larry L. Constantine

Executive Updates

Vol. 6, No. 9 *Why Is Agile Development So Scary?* by Preston G. Smith

Vol. 6, No. 8 *How Do Agile Teams Manage Risk* by Laurent Bossavit

Vol. 6, No. 7 *Measuring Up to Metrics: Part III — Overcoming Project Complexity* by E.M. Bennatan

Vol. 6, No. 6 *Donkeys and Carrots: Customer Collaboration and Helping Others* by Ole Jepsen

Vol. 6, No. 5 *Mitigating Large Software Project Risk Through Organic Growth Organization* by Nick Christenson

Vol. 6, No. 4 *Measuring Up to Metrics: Part II — Are Software Organizations Measuring by Data?* by E.M. Bennatan

Vol. 6, No. 3 *Measuring Up to Metrics: Part I — How I Became a Missourian* by E.M. Bennatan

Vol. 6, No. 2 *Agile: Changing the Organization* by David Spann

Vol. 6, No. 1 *Using the Retrospective for Positive Change* by Diana Larsen

Vol. 5, No. 23 *Absorbing Sarbanes-Oxley Within the Agile Community* by Charles W. Butler and Gary L. Richardson

Vol. 5, No. 22 *A New Era of Software Project Management: Part III — Agile Project Management and the Stealth Fighter* by E.M. Bennatan

Vol. 5, No. 21 *A New Era of Software Project Management: Part II — Are Agile Projects Filling the Void?* by E.M. Bennatan

Vol. 5, No. 20 *Agile Development: Lessons Learned from the First Scrum* by Dr. Jeff Sutherland

Vol. 5, No. 19 *A New Era of Software Project Management: Part I — Are We Adjusting?* by E.M. Bennatan

About the Practice

Agile Project Management Practice

Cutter Consortium's Agile Project Management Practice helps companies succeed under the pressures of this highly turbulent economy. The practice is unique in that its Senior Consultants — who write the reports and analyses for the information service component of this practice and do the consulting and mentoring — are the people who've developed the groundbreaking practices of the Agile Methodology movement. The Agile Project Management Practice also considers the more traditional processes and methodologies to help companies decide what will work best for specific projects or teams.

Through the subscription-based publications and the consulting, mentoring, and training the Agile Project Management Practice offers, clients get insight into Agile methodologies, including Adaptive Software Development, Extreme Programming, Dynamic Systems Development Method, and Lean Development; the peopleware issues of managing high-profile projects; advice on how to elicit adequate requirements and managing changing requirements; productivity benchmarking; the conflict that inevitably arises within high-visibility initiatives; issues associated with globally disbursed software teams; and more.

Products and Services Available from the Agile Project Management Practice

- The Agile Software Development and Project Management Advisory Service
- Consulting
- Inhouse Workshops
- Mentoring
- Research Reports

Other Cutter Consortium Practices

Cutter Consortium aligns its products and services into the nine practice areas below. Each of these practices includes a subscription-based periodical service, plus consulting and training services.

- Agile Software Development and Project Management
- Business Intelligence
- Business-IT Strategies
- Business Technology Trends and Impacts
- Enterprise Architecture
- IT Management
- Measurement and Benchmarking Strategies
- Enterprise Risk Management and Governance
- Sourcing and Vendor Relationships

Senior Consultant Team

The Cutter Consortium Agile Project Management Senior Consultant Team includes many of the trailblazers in the project management/peopleware field, from those who've written the textbooks that continue to crystallize the issues of hiring, retaining, and motivating software professionals, to those who've developed today's hottest Agile methodologies. You'll get sound advice and cutting-edge tips, as well as case studies and data analysis from best-in-class experts. This brain trust includes:

- Jim Highsmith, Director
- Scott W. Ambler
- Christopher M. Avery
- James Bach
- Paul G. Bassett
- Kent Beck
- E.M. Bennatan
- Tom Bragg
- David R. Caruso
- Robert N. Charette
- Alistair Cockburn
- Mike Cohn
- Ken Collier
- Doug DeCarlo
- Tom DeMarco
- Khaled El Emam
- Donna Fitzgerald
- Kerry F. Gentry
- Michael Hill
- David Hussman
- Ron Jeffries
- Joshua Kerievsky
- Bartosz Kiepuszewski
- Brian Lawrence
- Tim Lister
- Michael C. Mah
- Lynne Nix
- Ken Orr
- Mary Poppendieck
- Roger Pressman
- James Robertson
- Suzanne Robertson
- Rob Thomsett
- Colin Tully
- Bob Wysocki
- Richard Zultner