Agile Project Management

Vol. 6, No. 2

CUTTER CONSORTIUM

Industrial XP: Making XP Work in Large Organizations

by Joshua Kerievsky, Senior Consultant, Cutter Consortium

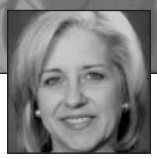
Industrial XP (IXP) is a brand of Extreme Programming (XP) tailored for large organizations. IXP builds on the principles of XP to address the challenges that companies with greater than 500 employees often face when implementing XP. This *Executive Report* explains the history of IXP; its values, practices, roles, and responsibilities; and offers advice on successfully transitioning to IXP.

Executive Report

Cutter Business Technology Council

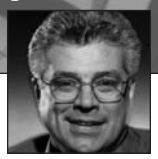

Rob Austin

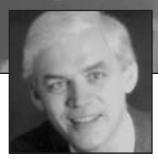

Tom DeMarco


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.

Industrial XP: Making XP Work in Large Organizations

AGILE PROJECT MANAGEMENT ADVISORY SERVICE

Executive Report, Vol. 6, No. 2

by Joshua Kerievsky, Senior Consultant, Cutter Consortium

Industrial XP (IXP) evolved in response to challenges often associated with implementing Extreme Programming (XP) in organizations with 500 or more employees.[1] IXP is a brand of XP that is specialized for use by large organizations. Since 2001, IXP has helped biotech, finance, engineering, and entertainment organizations deliver outstanding results on dozens of diverse projects.

How does IXP differ from XP? Before we can answer that question, we must clarify which XP we are talking about. XP was first introduced to the world in Cutter Consortium Senior Consultant Kent Beck's 1999 groundbreaking book *Extreme Programming Explained* [1]. In 2004, Beck completed the second edition, which substantially changed and improved the process. Since IXP evolved while the second edition was being developed, IXP is much closer in spirit and practice to the second edition of XP than to the original version. Therefore, in this report, my comments will, for the most part, refer to the first edition of XP.

Now back to our original question. IXP differs most from XP in that it:

- Improves on XP's original 12 practices
- Extends XP with important management and learning practices
- Defines new practices for important XP ideas that were only implicitly stated in the original process

This *Executive Report* recounts the story that led to the evolution of IXP; explains IXP's values, practices, roles, and responsibilities; and concludes with a look at IXP's approach to transitioning organizations to the process.

A FAILED XP PROJECT AND WHAT IT TAUGHT US

IXP's beginnings can be traced to a large, unsuccessful XP project in 2001 that occurred within a well-known financial conglomerate. Technical and project managers, as well as most analysts, developers, and testers, worked out of a San Francisco office, while executives and core members of

[1]This report's author pioneered IXP. His organization, Industrial Logic, is a dedicated team of IXP coaches and programmers who've worked on small, large, and distributed XP and IXP projects in North America since the early days of XP.

the customer community (including domain experts) worked out of a New York City office. In total, about 180 people were involved in the project.

The individual leading the charge was new to the organization and was determined to pioneer the use of XP within his department. Unfortunately, this man, whom I'll call "Bill," lacked two factors crucial for achieving success: strong connections within the organization and enough power to effect changes across departments.

Prior to starting work on the project, I conducted what I now consider to be my most naive readiness assessment. I had been asked to dinner by a woman I'll call "Sue," a high-powered executive VP with the financial conglomerate. During dinner, Sue described her project: a 25-person effort to work on a new system, which she referred to as "Phase 3." Phases 1 and 2 were apparently part of the same large-scale, 180-person project, which had been underway for nearly 15 months.

Sue explained that the folks in San Francisco were doing XP. I asked which XP practices were being implemented, but she didn't know. All she knew was that people were having trouble, which was why they were now seeking expert help with XP.

The San Francisco Phase 3 team consisted of one project manager, numerous programmers, a few analysts, a technical writer, numerous testers, and one database administrator. That seemed reasonable to me. So I had the following four questions to ask Sue to gauge my level of comfort with taking on this work:

1. Could we establish an open workspace for the team?
2. Could we practice XP fully (i.e., all the values and practices)?
3. Could we train the entire team for 10 days before commencing our work?
4. Could she pay us $X/month for our coaching services?

Imagine my delight when Sue said "yes, yes, yes, and yes" to all four of my questions!

The project began a few weeks later. I chose to pair-coach with my friend Eric Evans. He and I met the Phase 3 team in a cramped meeting room in a high-rise office building in downtown San Francisco. The room was filled with people and lined with tables and computers. It was a far cry from an ideal open space, yet it would do.

Before commencing our 10-day XP workshop, Sue asked if we could accommodate five more people who had recently been assigned to her. We accepted. Now there were 32 of us in a room probably not meant to hold more than 15.

Mid-morning of the second day of training, Sue barged into the room to tell us that all training had to stop. The customers in New York were not happy with the progress of Phase 1 and now wanted all hands to work on critical defects that were preventing Phase 1 from going live. This was not a major problem for us. We would simply make the defect fixing a part of our workshop by demonstrating what it is like to fix defects using test-driven development (TDD) and merciless refactoring.

An hour or so later, we had three defects fixed. It was time to integrate — or so we thought. At this point we learned that there was a source code version control department and that people in that department had locked down the Phase 1 code. We could not integrate! A few phone calls got us nowhere. The code was locked, and no one was going to integrate anytime soon. What? Weren't we supposed to be fixing *critical* defects?

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. Group Publisher: Kara Letourneau, E-mail: kletourneau@cutter.com. Production Editor: Lauren S. Horwitz, E-mail: lhorwitz@cutter.com. ISSN: 1536-2981.

This was our first rude awakening. It would not be our last.

After the workshop, the Phase 3 technical team assembled in our cramped room to do our first iteration planning session for the Phase 3 project. We awaited our analysts, who were really proxy customers for the real customers in New York. These analysts would play a critical role in providing the project community with user stories and helping to prioritize them.

Thirty minutes ticked by with no sign of these critical folks. Several phone calls were made, but they didn't help the situation. Sue finally asked Eric and me to let three of her most knowledgeable programmers play the role of customer, since it seemed the customer proxies didn't have time to join us. Eric and I reluctantly agreed, even though we knew that developers rarely understand the priorities of real customers, let alone proxy customers.

In hindsight, this change didn't really matter because the first iteration never got off the ground. On the second day of the first iteration, Sue once again barged into the room to announce that all work on Phase 3 had to stop. Apparently, Sue had been battling forces in New York that wanted everyone to help get Phase 1 live before any work commenced on Phase 3. Sue had temporarily lost her battle.

A particularly awful task our team was given was to help straighten out the Phase 1 code involving a particular Web page. A printout of this Web page revealed that it was 40 pages long, with 28 of the 40 pages containing some of the most poorly written JavaScript I'd ever seen. To help fix this Web page, five programmers were recruited to spend a marathon weekend of testing with the 40-person QA team (yes, that's 40 people, out of a total of 180). The programmers were to manually work through countless scenarios described in Excel spreadsheets. For each scenario, they would punch values into a form on the Web page, press a button, and then manually check if these values matched ones in the spreadsheets.

At that point, it did not matter that several JavaScript unit-testing tools had been available for more than a year or that every single calculation on the Web page could have been automatically tested at the touch of a button had the Web page's programmers simply written automated tests for their code. Phase 1 was too far gone for any such information to be of use.

Within a week, Sue got our team back on Phase 3. We created a new iteration plan and soon found that our new plan involved some database changes. Apparently we would need to add a few new columns to some existing tables. No problem. Our database administrator could surely make those changes — or so we thought.

It turned out that our database administrator had zero rights to make changes in the databases. All he could do was write e-mails to another fellow, who would pass on those e-mails to the "enterprise" database group. Once the e-mail made it to that group, we might get a response in two weeks, if we were lucky. Two weeks! Totally un-agile!

So we decided to investigate this situation. We spoke with Sue's boss and arranged for several meetings with the enterprise database group. I'll spare you the details. The end result was that we made no progress, despite plenty of meetings. The enterprise database group simply didn't trust our group and didn't understand why we couldn't give them all of our requirements up front. We explained XP to them and even suggested ways to work together harmoniously, but it had no effect. There was simply no trust or willingness to work together. The group would not even agree to give us a sandbox database environment!

This was where Bill's lack of connections and lack of power within the organization really hurt us. He simply could not get the necessary level of cooperation from other departments within the organization. And so, after two months of making very little

progress, Eric and I decided to call it quits. We simply could not help this client. At the time, we were disappointed with this experience. However, within a few months, we came to see that this work had taught us many valuable lessons about implementing XP within large organizations. Here are eight important lessons we learned:

1. **Do a thorough readiness assessment.** Before committing to help any large organization transition to XP, thoroughly assess how prepared the organization is to make the transition.
2. **Obtain high-level executive support.** Executives with the most power in the organization need to be recruited from day one to help make an XP transition run smoothly.
3. **Ensure interdepartmental cooperation.** Managers and employees in different departments must cooperate. If they don't, someone with power (such as an executive) must step in to establish the necessary level of cooperation.
4. **Increase communication with management.** Executive management must communicate regularly with everyone who is implementing XP within the organization.
5. **Define success.** We must define what success on a project means and report our findings, on a regular basis, to management. This is particularly important when trying to show how a new process is better than an older process.
6. **Make collaboration explicit.** We must be explicit in how we expect customers to collaborate with developers before we begin the transition.
7. **Ensure committed resources.** Before the project begins, everyone who is critical to the success of the project must commit to spending sufficient time on it. This includes ensuring that customers and analysts have enough time to write user stories, help prioritize work, and assess whether features have been programmed acceptably.
8. **Ensure sufficient control.** Programmers must have the ability to continuously integrate and evolve whatever is essential to the system (such as an enterprise database).

A NEW BRAND OF XP

By mid-2003, I'd been actively implementing XP for more than four years. Following the 2001 San Francisco project, I spent the next two years experimenting to learn how to successfully practice XP in large organizations. I do not have adequate space in this report to explain these experiments; however, the result of my work was an expanded and modified version of XP that I named Industrial XP.[2]

[2]The name "Industrial XP" was chosen for two reasons: to reflect that this brand of XP is "industrial strength" and to relate the brand to my company's name, Industrial Logic, Inc.

IXP is an organic evolution of XP. It is imbued with XP's minimalist, customer-centric, test-driven spirit. IXP differs most from the original XP in its greater inclusion of management, its expanded role for customers, and its upgraded technical practices.

During its initial two-year evolution, IXP came to address what large organizations needed in order to be comfortable with XP as well as what was required of XP in order to work well within these large companies. While the original XP consisted of four values and 12 practices, IXP takes a different approach to values and includes 23 practices as seen in Figure 1.

The four values in XP are communication, simplicity, feedback, and courage. If members of a project community actively seek to apply these values to their practice of IXP, they will influence their project in many useful and unexpected ways. In the second edition of *Extreme Programming Explained*, Beck observes that "practices are barren. Unless given purpose by values, they become rote." Beck gives a good example: "Pair programming to please your boss is just frustrating. Pair programming to communicate, get feedback, simplify the system, catch errors, and bolster your courage makes a lot of sense" [1].

Since values are so vital to how people practice XP, in IXP we ask each project community to

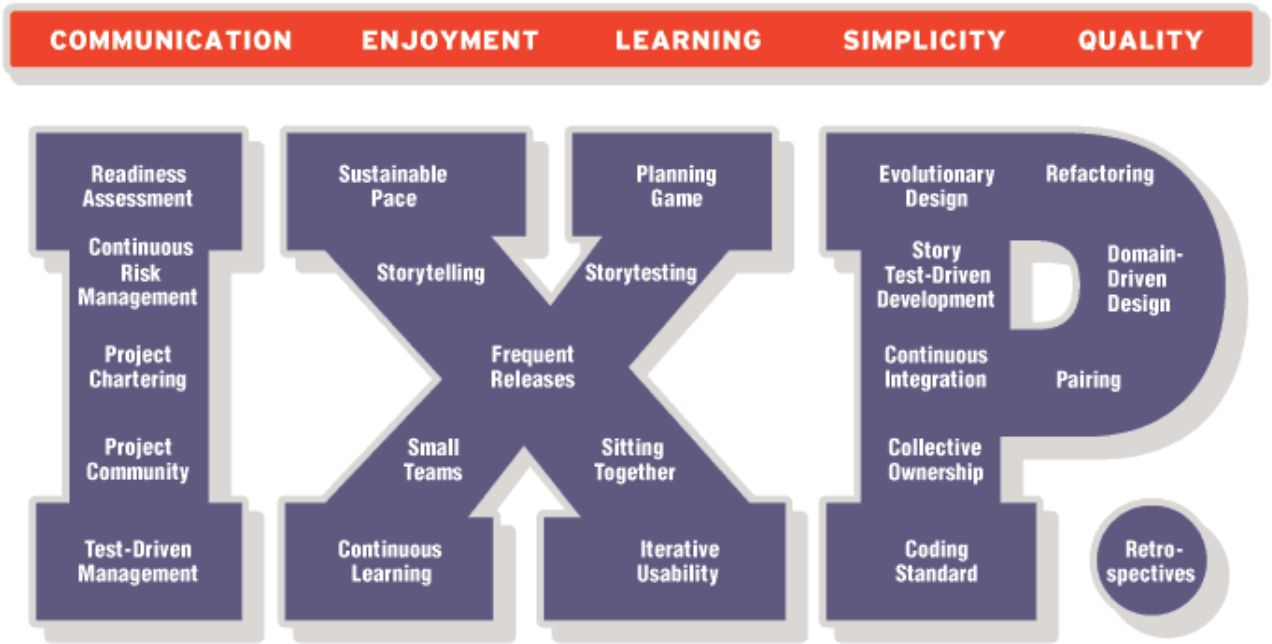


Figure 1 — IXP's 23 practices.

discover its own values. The process is as follows:

1. Break up a project community into subgroups.
2. Let each subgroup come up with its list of values — however many it likes.
3. Consolidate the lists of the subgroups by finding which groups of values are similar.
4. Choose the top-five values that most appeal to the project community.

The five values listed at the top of Figure 1 — communication, enjoyment, learning, simplicity, and quality — resulted from following the above process with people from my company. Since doing that exercise, we have found that our values thoroughly influence our decisions and actions.

Once a project community's shared values have been discovered, the community can assess whether the IXP practices it will implement align well with its values. This work occurs as part of a readiness assessment (an IXP practice described below).

Given a good match, a project community's values will be an invaluable aid in helping the group stay on track. It is useful to keep values posted prominently in the workspace, so that people can refer to them continually and use them to help make decisions and find direction.

The next section discusses the practices. Six of these 23 practices are new to XP; five are improvements to existing XP practices; five are explicit practices that make implicit XP ideas more tangible; and seven practices come from the original XP. The next few sections present a description of all 23 IXP practices, categorized by whether they are new, improved, explicit, or original.

New Practices

IXP's new practices resulted primarily from needing to properly include management on projects and ensure that project communities live up to XP's spirit of continuous improvement. Like many of the ideas in XP, most of these practices aren't new, though some have names that may be new to the reader. The six new practices in IXP are:

1. **Readiness assessment.** Continuously evaluate whether your organization and project community are ready to do IXP.

2. **Project community.** Identify and include people who effect a project or are affected by the project. A project community is *always* bigger than you think.
3. **Project chartering.** Ensure that what a project community will build is in harmony with the organization's needs.
4. **Test-driven management.** Inspire, compel, enlighten, and align a project community by defining SMART (specific, measurable, achievable, relevant, and time-based) objectives.
5. **Retrospectives.** Include a rigorous, future-focused process for capturing lessons learned, best practices in a context, and multiple perspectives on how to improve a software development environment.
6. **Continuous learning.** Continuously improve your skills to deliver greater value and enjoy what you do.

Readiness Assessment

Readiness assessments typically last one to two days and are conducted by experts in IXP. An expert begins an assessment by talking to a project or process sponsor: someone who can describe a project, the people involved in the project (or those likely to be involved), and how those people fit into the organization. Following this meeting, the expert meets with individuals in small groups of three to five people. During each meeting, the expert explains how IXP works and both asks and answers many questions.

Below are questions an expert commonly asks during an assessment:

- Can we establish the kind of programming environment IXP requires (staging machines, pair-programming workstations, a database we can actually evolve, version control that is under our control, etc.)?
- Will subject matter experts (SMEs) be available for the project? Can we expect to get ongoing feedback from end users of the evolving software?
- Is there a dedication to continuous improvement? Is there a commitment to doing retrospectives both during and at the end of the project?
- What is the organizational culture and structure like? Will other departments within the organization support change or construct barriers to change? What is the history of other changes that have taken place within the organization?
- Will the project community for this project have the right people?

During a typical assessment, an expert will meet with project sponsors, programmers, analysts, testers, database administrators (DBAs), project and product managers, domain experts, version control (or source control management) managers, software security people, architects, facilities workers (who will be in charge of setting up an open workspace), and process people. When necessary, an expert will also meet with auditors (or folks involved in compliance endeavors) and members of human resources (HR) and legal departments.

Project Community

A few years after publishing *Extreme Programming Explained*, Beck introduced the term "whole team" into the XP vocabulary. The notion is that software teams need the right people in order to be successful. While I wholeheartedly agree with that sentiment, I've found it useful to replace the term "whole team" with "project community."

While we often think of a "team" as being a constrained set of people, the term "community" implies something broader. Communities have active members who may be at the center of an effort and less active members who may be on the periphery of the effort. Lawyers, auditors, and facilities folks are often on the periphery of an IXP project community, yet they may play important roles on a project. If a project community fails to include important people inside or outside the organization, it will often face numerous problems that can delay or even stall a project.

David Schmaltz, an expert in project management and the individual who taught me the term "project community," says, "The primary issue facing every project is the lack of awareness of its own 'community-ness.' That's why the first steps are well focused upon increasing this awareness within the community."

Awareness of a project community begins during project chartering. An IXP coach leads people through a session to identify everyone within the project community. This exercise nearly always leads to a list of names that fills up several whiteboards. Schmaltz points out that a project community is "always bigger than you think." During or after a project, most project communities realize that they failed to include someone important. So the practice of defining the project community is an ongoing endeavor that people improve at over time.

Project Chartering

IXP's chartering process is based on what I've learned from a senior consultant named III (pronounced "three").

After seeing XP work successfully on small projects and unsuccessfully on the large and infamous San Francisco project, I started to see the need for chartering in XP. So in late 2001, I began collaborating with III to practice an agile approach to chartering on XP projects.

Project chartering helps people answer questions, such as:

- Is the idea for the project worthwhile?
- How does the project further the organization's vision/mission?
- How would we know if the project is a success?
- Who is part of the project community?

Like many XP technical practices, project chartering is an ongoing endeavor. Writing and revising a charter helps establish the following project characteristics:

- **Vision** — a desired future
- **Mission** — the strategy for obtaining the vision
- **Project community** — the people involved in the endeavor
- **Values** — concepts to guide decision making and conduct
- **Management tests** — measures of success or failure that align and inspire a project community
- **Context diagram** — a depiction of key events flowing into or out of a software system or community
- **Community agreements** — agreements shared and practiced by a project community

Test-Driven Management

How does a project community learn whether its project work is successful? The same way programmers learn whether their code works: tests.

Test-driven management directs the specification of management tests, which are statements that indicate a measurable, time-limited goal, framed in a binary manner. We either achieve the management test or we fail. Good management tests are SMART.

Management tests are statements about the world external to the project, treating the project as a boundary of responsibility and authority. They avoid specifying ways in which external effects (i.e., things that occur outside the boundary of the project and the software) should be achieved. In other words, good management tests set a destination, but don't specify how to get there.

Management tests provide an excellent way for a project community to understand what unites it. This echoes Tom DeMarco and Timothy Lister's observation: "The purpose of a team is not goal attainment, but goal alignment" [2]. Management tests create goal alignment by delineating how and when success will be measured, enabling individuals to understand the effects of their own actions.

Retrospectives

Unlike a more typical review session, a retrospective does not yield a list of items but continues on to establish next steps, accountabilities, and measures

for making improvements happen. IXP's approach to retrospectives is based on the work of Norman Kerth, author of *Project Retrospectives: A Handbook for Team Reviews* [7].

IXP retrospectives are conducted at the end of every iteration and release. An iteration retrospective is a study of what is working well, what needs improvement, and who will take ownership of an issue to help find its resolution. If a project community's iterations are one or two weeks long, iteration retrospectives will last between a half hour and two hours. Members of a project community, including the coach and project manager, help facilitate iteration retrospectives.

Release retrospectives focus on issues, events, and lessons learned across an entire release. Typical release retrospectives look at either one release (generally three months long) or two releases (generally six months long). Release retrospectives tend to last anywhere from a half day to two days. It is best to have someone who is not part of the project community facilitate the retrospective.

One of the most popular ways for a project community to review and learn from its collective experience of a release is to create a timeline. Timelines are made by listing the months of a release on posters (which span the length of a large wall) and then letting participants write up their experiences on cards and attach those cards to times on the timeline when their experiences occurred. Timelines provide a wealth of information that can uncover new insights and ideas for improvement.

Continuous Learning

Continuous learning means actively and regularly learning new techniques and skills. While pairing helps people acquire new skills from others within the project community, continuous learning focuses individuals and groups on important technical and nontechnical subjects that can help them improve at their jobs.

IXP project communities usually hold weekly or biweekly study sessions. Typical sessions often involve studying an important piece of literature or some significant items in a code base.

In environments where people are not encouraged to learn new skills or improve on existing skills, résumé-based development (RBD) often takes hold. A learning-deprived employee wants to get some technology or experience on his or her résumé so badly that he or she finds some way to convince the project community to use that technology, even if it isn't actually a good fit. This is bad, as it often leads to overly complicated and costly solutions that developers are afraid to change. Continuous learning provides an effective way to manage this risk and also helps a project community find enjoyment in personal growth.

Improvements to Existing Practices

IXP's improvements to XP's practices resulted from feedback and learning. In most cases, improvements took a good thing and simply made it better or expanded its scope. Many of the improved practices resulted in a greater inclusion of customers and managers, since many of the original XP practices tended to be rather programmer-centric. The following four practices offer improvements on some of XP's original practices:

1. **Storytest-driven development (SDD).** Never write a line of code before writing a failing storytest (aka acceptance test). *Improvement on:* TDD and acceptance testing. SDD applies the principles of TDD to storytests.
2. **Domain-driven design (DDD).** Evolve a domain model that is insulated from domain-independent code and that accurately represents how domain experts think about their subject. Speak, code, and write using a ubiquitous language. *Improvement on:* system metaphor.
3. **Pairing.** Pool knowledge by working in pairs when programming, coaching, managing, storytelling, etc. *Improvement on:* pair programming. Pairing simply

widens this practice to include everyone within a project community.

4. **Iterative usability.** Practice test-driven usability with real users and iteratively evolve a system's usability rather than doing big, costly, up-front usability. *Improvement on:* on-site customer.

Storytest-Driven Development

A *story* roughly describes an action invoked by a user on a piece of software. Stories are written by the customer community and are composed of a heading, a brief description, and usually a time- or unit-based estimate provided by a development community.

The detail found on a story or obtained through dialogue about a story usually isn't sufficient to ensure that a development community produces just what a customer community needs. To obtain such precision, more details are required.

Storytests provide the missing detail. In the original XP, storytests were written either during an iteration or sometime after code was produced. In IXP, storytests are written before any work occurs on a story. This ensures that developers program only what is needed and nothing more. Storytests get written by domain experts and testers within the customer community.

SDD is the process of making storytests execute against a system to demonstrate whether the system is functioning correctly. SDD may or may not involve TDD, which involves writing low-level unit tests. Developers usually combine SDD and TDD when stories are complex.

Domain-Driven Design

IXP's practice of DDD is based on the work of Eric Evans, author of *Domain-Driven Design* [4]. We've found this work to be a better guide for project communities than the literature on system metaphor.

A domain model is a vital part of a software system, since it reflects how experts and users think about their work. Evolving a good domain model involves much collaboration between experts, users, and the system's developers. A great aid to that collaboration is a ubiquitous language: a common language for the domain that everyone within a community uses when speaking, writing, or programming.

IXP's practice of SDD typically leads to the creation of domain objects. As knowledge of a domain model deepens, domain objects must be refactored to better reflect how community members think and talk about their domain.

Good domain models make it easier to add new domain-specific behavior to a system, while poor domain models often create confusion and slow down development.

Pairing

When two people collaborate on something, they are pairing. In IXP, pairing happens on code, stories, storytests, technical documents, and more. Pairing helps people:

- Share knowledge and improve skills
- Stay focused and involved
- Improve existing work
- Reduce errors
- Enjoy feeling part of a closely knit community

On the other hand, numerous risks relate to solo work, including:

- Tunnel vision
- Fatigue
- Higher defects
- Less knowledge transfer
- More distractions
- Less refactoring
- Weaker problem solving

Pair programming is a controversial practice in XP because it is sometimes nonintuitive how two people programming together can be more productive than people who work independently. In addition, some programmers simply don't want to collaborate with

others. XP says that programmers *must* collaborate to reduce risks, produce quality work, share knowledge, and generally help their organization succeed.

IXP extends the practice of pairing to include everyone in a project community, not just programmers. Pairing is recommended for important work, such as writing stories, creating storytests, test driving and refactoring code, or integrating work into a repository.

In some cases, it is better to work with more than two people on a task. In other cases, solo work actually makes more sense then pairing. The key is to be aware of the risks for solo work so that pairing and group work can help to effectively manage the risks.

Iterative Usability

Beck's ideas on usability were inspired by Kristen Nygaard's work on participatory design. An on-site customer participates in the design of software by providing continuous feedback on the usability of a system. While this is important, it does not sufficiently articulate the practices that such an on-site customer needs to do.

Iterative usability is a form of usability that closely follows the evolution of a system. As a system's features and tests evolve, so does its usability and usability tests.

Practitioners of iterative usability work closely with domain experts and real users of an evolving system to define look-and-feel standards and to measure — with the help of tests — a system's usability. One practitioner I know uses software and video to record not only what a user clicks on but also the user's facial expression as he or she is clicking. Such feedback provides invaluable information that often influences how to redesign a user interface.

Most usability experts who have never done iterative usability are more than qualified to figure out how to do it. The trick is for them to first renounce doing big, up-front usability. The problem with such work is that it anticipates the behavior and look of a system, which is entirely contrary to the evolutionary nature of XP.

Implicit Practices Made Explicit

Extremely important ideas in the XP literature were often lost to new practitioners because many of these ideas were implicit in the literature (i.e., they were not sufficiently spelled out or explained as specific practices). Based on experience, IXP produced explicit practices for the most important, and most easily forgotten, implicit XP ideas. While this increased the number of practices in IXP, it also helped new practitioners understand the art of XP, which enabled them to progress more quickly to mastery.

The following six practices make implicit XP ideas explicit:

1. **Continuous risk management.** Continuously identify your project's risks and determine which risks to take, mitigate, or remove.
2. **Evolutionary design (ED).** Evolve a system by beginning with the most important, yet primitive, functionality and steadily make features more robust based on importance, ROI, risk, etc.
3. **Storytelling.** Roughly communicate the functionality of a system by telling stories from a user's perspective.
4. **Storytesting.** Communicate detailed story specifications using system input and expected output data.
5. **Sitting together.** Enable efficient and effective collaboration by letting folks sit and work in a shared physical space.
6. **Small teams.** Break large project communities into smaller teams that can collaborate effectively.

Continuous Risk Management

Risk management is built into many of the practices in XP and IXP, for example:

- The planning game helps people manage the risks of what can or cannot be released by a given date.
- Sustainable pace helps effectively manage fatigue and, in some environments, burnout.

- ED helps manage the risks of not getting enough early experience with all aspects of a system. It also helps designers and developers avoid overengineering.
- Frequent releases help manage the risk of not delivering value early and not having enough experience getting into a production environment.

By making risk management a practice in IXP, the process is effectively saying that everyone within a project community needs to be aware of risks, so that risks can be intelligently managed. Risk lists are often created during iteration-planning meetings and reviewed during iteration retrospectives. Project managers often review risk lists with higher-level managers and discuss how the risks are being managed.

For more sophisticated techniques in risk management, IXP recommends Tom DeMarco and Tim Lister's *Waltzing with Bears* [3].

Evolutionary Design

Design in XP is evolutionary. Practices such as TDD and refactoring lead to the evolution of a system. And yet evolutionary design is bigger than just code; it is a way of thinking. IXP elevated ED to its own practice because everyone within a project community — from managers to customers to developers — must understand how to implement this important practice.

ED impacts how stories are written and how they are chosen for release and iteration plans. ED is critical in finding the vertical slice or end-to-end implementation of a system. ED influences such diverse work as usability, technical documentation, and domain modeling.

In addition, ED directs people to always seek a rapid ROI and to find ways to reduce the risk of not delivering value early. It directs folks to backtrack when they aren't happy with their work and to constantly review their work to find areas for improvement. ED suggests it is OK to do something several times and pick the best implementation from numerous choices. It also implies that better designs occur through teamwork rather than solo work. ED suggests that it is wise to consider what to automate and what not to automate when choosing which features must be implemented for a release. Finally, ED suggests paying attention to "dead reckoning" — an awareness of where you want to go and whether or not you are getting there.

Storytelling

A story describes something a user would do with a software system. Stories contain just enough detail to help developers estimate how long it would take to complete a story. Story details are captured in spoken language and through the process of storytesting.

There are two kinds of stories in IXP: release and iteration stories. Release stories tend to be bigger than iteration stories, meaning they typically take more time to implement. For example, a release story that describes a fully featured install program could take a while to evolve, while an iteration story may involve creating a bare-bones installation program. Iteration stories often start out representing embryonic versions of release stories. Over time, the stories in iterations come to address all parts of a release story, until the release story is fully implemented.

Stories are often modified, split, or deferred to other releases or iterations when project communities engage in planning sessions. The ability to alter the stories in release and iteration plans allows customer communities to respond to changes to priorities.

To make it easier to change the contents of a story, its heading is written on an index card while its details are written on a Post-it Note, as shown in Figure 2.

When it is time to estimate a story, another, smaller Post-it Note will be affixed to the index card to indicate the estimate for the story.

Story headings tend to contain a noun and a verb and are usually less than five words in length. Here are some examples:

- Open an Account
- View Open Invoices

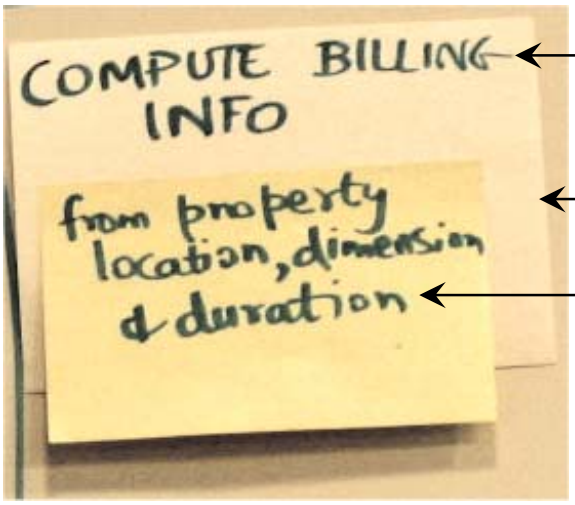


Figure 2 — A story heading and its details.

- Book Nonstop Flight
- Validate Credit Application

Storytesting

Storytesting is how customers communicate details about stories. The details are provided using example data that specifies inputs and expected outputs for system features. Storytesting is not a technical practice: storytests may be specified on whiteboards, paper, or in electronic documents.

Domain experts and testers in a customer community write storytests. When completed or partially completed, developers work with customers to understand the storytests and automate them using the practice of SDD.

Good storytests are part of what IXP calls "executable documentation." A typical executable document contains the heading and description of a story, some graphics or diagrams to illustrate details of a story, and storytest data to validate whether a story is working as expected. Making these documents executable is what happens during SDD.

Sitting Together

The highest-bandwidth communication known to man is face-to-face communication. Project communities that sit and work together tend to collaborate better and faster than teams that work in individual cubes.

Yet how is it possible to concentrate in a shared, open environment? It's easy. As individuals become used to pairing, everyone is focused on a task. The noise in the room becomes background noise, unless one decides to tune in to help others. Many folks who have never done XP can't imagine how this could possibly work, yet time and time again we see that those who try it learn to love it. There is simply no faster way to collaborate than sitting together.

The best arrangement for sitting together allows diverse members of a project community to collaborate easily. Organizations that adopt XP usually knock down cubicle walls to build an effective open workspace. In the best environments, people can move between public and private spaces with ease.

Small Teams

Project communities come in all shapes and sizes. The larger ones tend to have smaller teams that specialize in certain areas.

For example, the technical side of a 40-plus person project community I once worked with had small teams for mainframe, data warehouse, and Java programmers. While this project community worked on a single product, it broke up its work into three iteration plans that were implemented by the three small teams. Work completed in each of these iteration plans contributed to the functioning of a feature in the new system.

Another project community I worked with consisted of seven small teams. These small teams were working together on a new release of an existing product. Every other week, each of the seven small teams would conduct its own iteration-planning

meetings. Leads from each team would collaborate to ensure that features for the end product were being completed. One of the seven teams was known as the refactoring and architecture team. Its job was to simply help the other six teams reuse existing software solutions and improve the design of the product's code.

In *Extreme Programming Explained*, second edition, Beck points out that some large teams would be better off as small teams. If a small team can solve the problem at hand, there is no need to scale to a large team. Beck recommends beginning with small teams and growing only as necessary. When a small team becomes too large, it can be split into two teams, which are still part of one project community.

Unchanged Practices

The following seven practices were all in Beck's original formulation of XP, therefore I will not provide in-depth explanations of these practices:

1. **Refactoring.** Continuously revise code to make it succinct, simple, and straightforward. Do this using the safety of automated tests to ensure that changes don't break existing functionality.
2. **Planning game.** Creatively plan releases (typically three months long) and iterations (typically one week long) to achieve the greatest bang for the buck.
3. **Continuous integration.** Continuously integrate tests and code to make integration painless and rapidly educate others about new changes.
4. **Collective ownership.** Anyone can change anything at any time so long as all the automated tests still run successfully.
5. **Coding standard.** Programmers evolve a coding standard and follow it to make programming efficient and have code reflect a single formatting style.
6. **Sustainable pace** (originally called 40-hour week). Ensure that a project community is energetic at work by going at a sustainable pace.
7. **Frequent releases** (originally called small releases). Release systems early and often to put value into customer hands as quickly as possible.

IXP ROLES AND RESPONSIBILITIES

To gain a deeper understanding of IXP, it helps to study the roles and responsibilities of the people within a project community. A typical project community consists of a project manager, one or more coaches, and the following subcommunities:

- **Management community —** includes executives, board members, product/product line managers, sales and marketing management, project/functional management, and QA management
- **Customer community —** includes product and/or project manager; SMEs (aka domain experts, researchers); analysts (market, technical, etc.), testers/QA; end users/beta sites; sales, marketing, and service support; usability experts; and technical support
- **Technical community —** includes programmers, technical writers, database administrators, and architects
- **Related community —** includes software services, facilities, HR, legal, auditors, and third-party vendors

There is no clear delineation of roles in mature IXP project communities. Beck explains this nicely in his second edition:

> Roles on a mature XP team aren't fixed and rigid. The goal is to have everyone contribute the best they have to offer to the team's success. At first, fixed roles can help in learning new habits. … After new, mutually respectful relationships are established among the team members, fixed roles interfere with the goal of having everyone do their best. [1, p. 82]

Everyone within a project community shares the following responsibilities:

- **Helps deliver value to the organization.** Every member

of a project community is there to deliver the greatest value to the organization. In the best project communities, people get great enjoyment out of doing their jobs well and delivering exceptional value.

- **Lives the values.** A project community identifies its values at the start of a project. These values guide decision making and influence how people collaborate.
- **Evolves/adheres to community agreements.** Every project community evolves a set of agreements and consents to abide by them. Such agreements can relate to the coding standard, usability guidelines, pairing hours, etc.
- **Learns and shares knowledge/responsibilities.** The best project communities are continuously learning new skills and sharing knowledge with others.
- **Identifies and helps mitigate risks.** Risks to the success of a project are often rampant in large organizations. Therefore, everyone should work together to continuously identify and help remove or mitigate risks.
- **Reflects and improves.** To really improve, members of a project community often need to thoroughly reflect on what needs improvement and then take action to make the improvements.

The rest of this section lists the responsibilities of the different groups within the project community described above. To begin, a project manager's primary responsibilities are to:

- **Build the community.** The "community-ness" of the project community is vital to its smooth functioning. Project managers help a project community efficiently collaborate with all of its diverse members. Community building includes ensuring:
  - — End users provide sufficient feedback on the evolving software.
  - — Communication software is running to aid collaboration between on-site and distributed members of the community.
  - — The customer community effectively collaborates on decisions about how to evolve the system or product.
- **Ensure sufficient resources.** A project community that lacks the right people, hardware, or software will struggle. A project manager is instrumental in working within the organization to ensure that sufficient resources are available and remain available during a project.
- **Manage risks.** Project managers actively manage the diverse risks associated with not delivering a valuable system or product by a given date. Such risks could include failing to decide what is in or out of scope for a release, failing to meet auditing or regulatory requirements, failing to fix enough defects, failing to obtain an architectural group's blessing on the design (thereby causing delays in going to production), and so forth.
- **Report progress.** Executives usually need to know how a project is doing so that they can feel confident about their investment decisions and make good decisions about staffing other projects. Project managers report on the milestones of a project, what risks are present and how they are being addressed, and relevant metrics — including how the project community is doing with respect to its management tests.

A coach's primary responsibilities are to:

- **Mentor everyone.** This means helping all of the diverse people within a project community understand their roles and how they contribute to the community's success. Great coaches quickly render themselves unnecessary.
- **Customize the process.** Coaches understand how to effectively customize a process to fit a given context. Much of the customization work

often occurs during readiness assessments.

- **Keep everyone on process.** If people within the project community veer off the process for no good reason, like not refactoring enough, not getting enough feedback from users, or not doing sufficient planning, coaches help them get back on track.
- **Facilitate meetings.** Coaches facilitate release/iteration planning meetings and 10-minute status meetings until others within the community can successfully take over this responsibility.
- **Review code.** Coaches regularly review code and an evolving architecture. They work with developers to improve code quality without giving all of the answers away.
- **Promote productive collaboration.** Coaches help people learn to collaborate in a productive way. This could mean teaching people to effectively negotiate the scope of a story or helping them learn to be respectful when they disagree with each other.

A management community's primary responsibilities are to:

- **Shape and approve a charter.** Managers attend project chartering sessions and play an important role in crafting and approving the language and spirit of a charter.
- **Promote harmony.** Given friction between people or departments, a management community must help establish harmony. This is especially true of people who are important in an organization but only peripherally involved in a project community.
- **Approve plans.** Release plans, budgets, and risk-mitigation strategies are the kinds of plans a management community regularly reviews and approves.
- **Review progress.** A management community may review a project community's progress by attending daily 10-minute status meetings as well as iteration- and release-planning sessions.
- **Help deliver value.** When an organization's existing people or processes impede a project community's ability to deliver value quickly, a management community must help the organization change to support the quick delivery of value to end users.

A customer community's primary responsibilities are to:

- **Roughly and thoroughly define features.** A customer community writes and communicates stories and helps define storytests (small, structured, testable specifications).
- **Intelligently evolve software.** Evolutionary design is a core practice in IXP. A customer community is constantly considering how best to evolve a software system or product.
- **Plan and negotiate.** A customer community is constantly planning releases and iterations. During release and iteration planning, customers are actively engaged in negotiating scope and inventing creative and cost-effective plans.
- **Review and accept work.** Completed work is approved by a customer community throughout iterations and at the end of iterations and releases.
- **Track defects and incomplete stories.** A customer community keeps track of existing defects and stories that are incomplete.
- **Obtain feedback from users.** Feedback is a core value in XP. Getting regular feedback on various evolutions of a product/system is a vital responsibility of a customer community.
- **Collaborate with developers.** Members of a customer community work closely with members of the development community to help them develop faster.
- **Conduct exploratory testing.** Testers frequently engage in exploring a system/product for defects.
- **Implement iterative usability.** The iterative, test-driven nature of XP applies equally well

to usability. A customer community evolves the usability of a system and regularly conducts usability tests with real end users.

A technical community's primary responsibilities are to:

- **Estimate stories.** A technical community provides customers with time- or unit-based estimates on release and iteration stories. These estimates help customers decide what to schedule in releases/iterations and where to change scope.
- **Automate testing.** Storytests and unit tests are constantly being written and executed by a technical community. SDD involves close collaboration with customers.
- **Produce simple, clean, intention-revealing code.** A technical community is responsible for producing high-quality code. Code that is easy to read, extend, and maintain results from intelligent and continuous refactoring.
- **Continuously integrate.** Integrating code continuously makes integration quick, helps developers stay in sync, and provides an excellent way for developers to learn what work has been started or completed. Good continuous integration requires fast build times, which includes the running of all automated tests.
- **Suggest alternatives/cheaper solutions.** The secret of good planning is to not simply agree to implement every story without first exploring how to deliver greater value with less effort. Developers collaborate with customers during release and iteration planning to creatively find alternatives or cheaper solutions for expensive stories (i.e., stories that have high estimates).
- **Collaborate closely with customers.** Close collaboration with customers is always essential.
- **Conduct exploratory testing.** Developers routinely conduct explorations for defects.

The roles that make up the "related" portion of a project community have diverse responsibilities. Below is a list of those roles and related responsibilities:

- **Software services.** Ensures that workstations and servers have the necessary software. A poor relationship with software services often leads to lengthy delays that hurt productivity and collaboration.
- **Facilities.** Configures an open workspace and helps modify it when changes are requested.
- **HR.** Aids in helping a project community learn how to assess individual and community performance. While not all HR groups participate in individual performance assessments, the ones that do often do so in a way that is at odds with the collaborative nature of IXP.
- **Legal.** Helps a project community's customers gain the necessary legal protection to allow them to safely show an evolving software product/system to would-be users. This is essential for gathering valuable feedback from potential users of the software. Yet it isn't possible if the legal work hasn't been done.
- **Auditors**. Study how the IXP process maps to whatever standard the organization would like to (or must) abide by. Then help to ensure that the process is followed in such a way to ensure a passing grade.
- **Third-party vendors.** Work closely with project community members who rely on the products or services.

TRANSITIONING TO IXP

A transition to IXP can be done in small pieces over time or by beginning with the complete process. In general, if a project community passes our readiness assessment, my colleagues and I prefer to begin transitions by helping everyone practice the complete IXP process. While this is no easy feat, we find that it works well for several reasons, including:

- Total immersion leads to mastery faster than piecemeal exposure.

- Expert coaches can efficiently mentor people in the complete process.
- The complete process delivers far greater value to organizations than partial implementations of the process.

While reviewing final drafts of Beck's *Extreme Programming Explained*, second edition, I was surprised to find that Beck recommends a piecemeal transition to XP. He writes:

> It's easy to start by changing one thing at a time. I think it's hard to jump in and do all the practices, embrace all the values, and apply all the principles in novel circumstances by reading this book and deciding to do it. The technical skills in XP and the attitudes behind them take a while to learn. XP works best when it is done all together, but you need a starting place. [1, p. 55]

I agree with Beck that it is hard for people to read his book and then effectively implement his advice, since XP impacts the behavior of a broad spectrum of people. However, I disagree that piecemeal transitions are an effective solution since I've consistently found that such transitions:

- **Are more painful than complete transitions.** If you are asked to change your behavior two or three times on a project, you may go along with it. If you are asked to change your behavior a dozen or more times on a project, you will likely not want to do it, as you are sick and tired of all the darned changes! Such was the case with a team I worked with in 1998-1999. My colleague and I initially got the team to transition to working in small iterations. A month later, we suggested that the team members sit and work together in a conference room. They reluctantly agreed. A month later we suggested that they do continuous integration. It took them two months to obtain a hardware and software configuration to support continuous integration, and, once they had it, few wanted to follow this practice. By the time we suggested the practice of pair programming, no one wanted to implement any more changes to the process.
- **Fail to address root problems.** Technical practices, like continuous integration, TDD, or refactoring, are popular early targets of many piecemeal transitions because they are *easy* for developers to introduce. Meanwhile, more significant problems, such as consistently poor requirements or a significant lack of collaboration between developers, customers, and managers, often don't get addressed because the practices to address these issues are considered too difficult to transition to.
- **Rarely lead to complete transitions.** None of the project communities that I know that fully (or very close to fully) practice XP or IXP got there by means of piecemeal transitions. Meanwhile, all of the piecemeal-transition project communities I've worked with or learned about have consistently stopped well short of a full XP or IXP implementation. As a result, they've seen improvements that don't match the substantial improvements obtained by complete-transition project communities.
- **Produce changes too slowly.** The pace of most piecemeal transitions is too slow for organizations that are in a competitive business climate. Complete transitions achieve higher ROI faster.
- **Tend to be done without expert help.** Why call in an expert when you are only adopting small pieces of a process at any given time? Piecemeal transitions are often done in the absence of experts. As a result, many of them wind up with odd process concoctions that don't deliver substantial results. Worse yet, I've seen some piecemeal transitions lead to processes that make a substantial number of people very unhappy. On the other hand, experts who live and breathe a process for many years are far better equipped to help project communities efficiently learn what process they need, how to tailor that process, and how to effectively transition to it.

So if partial transitions are risky or even fundamentally flawed, and, as Beck says, "it's hard to jump in and do all the practices, embrace all the values, and apply all the principles," then what are organizations to do?

The natural solution is to hire expert help. Since that happens to be the business I am in, such a suggestion could be perceived as utterly self-serving. It is not. Instead, it is a recognition of the fact that large ROIs may be obtained from transforming mediocre software development groups into outstanding performers, and such transformations are far more likely to be successful when guided by experts.

Implementing a Successful IXP Transition

IXP transitions tend to last three to six months and are led by expert coaches. We do not recommend transitioning to IXP without expert help, since there is simply too much to know to effectively help project communities and organizations make the transition successful. Once a few employees within a company are mentored as coaches, it is possible for them to continue helping others within their organization transition to the process.

The best way to begin a transition to IXP is to start with a pilot project. The project need not be too small or too large; a project community of 15 people is typical. In addition to programmers, most projects need dedicated help from domain experts, QA staff, managers, sales and marketing folks, technical writers, and so forth.

The first step involves creating a physical working environment that will facilitate plenty of collaboration within the project community. We tend to use flat-screen monitors, fast computers with multiple mice and keyboards, lots of whiteboards, plus plenty of comfortable chairs and even a sofa or two. If we are doing distributed work, we like to use certain software, like Virtual Network Computing (VNC), to help us collaborate.

Our work begins with an intensive workshop. We find that training helps get core members of a project community on the same page about how the process works and how everyone will work together. Typical workshops introduce the crucial ideas underlying evolutionary design, frequent releases (and what that means to a business), being test-driven, as well as other important practices. We use simulations and games in our training to help folks get an impressive and memorable education. A typical workshop lasts three to five days.

Work on the pilot project typically begins with a two-day meeting to conduct chartering. Next, we spend roughly two days doing release planning. Now we begin coaching a team through several iterations of the project. Our approach to coaching involves three players: two coaches who pair coach (alternating weeks and occasionally overlapping) and a third coach who manages the pair coaches during the project. All of the coaches stay in touch by writing in a daily diary and participating in regular phone discussions.

During the weeks of coaching, each coach helps the project community master release and iteration planning, technical practices (e.g., SDD, DDD, and refactoring), as well as key management practices, like test-driven management. The key to success of a pilot project is that coaches work with the right people on the project. If we suddenly lose all access to the domain experts who define stories or who select stories for iterations, the success of the project will be threatened. Therefore, support from management is needed in order to make the transition a success.

During the transition, coaches mentor developers, customers, and management in the art of IXP. As always, the coach's objective is to become unnecessary as quickly as possible. In addition, coaches set up and facilitate study groups to help folks (both on the pilot project and not on it) get better at what they do. Popular books we study together include Martin Fowler's *Refactoring* [5] and Jim Highsmith's *Agile Project Management* [6].

At the end of the project, we conduct a release retrospective, so that the project community can learn from its project and decide how to improve. We complete the transition by conducting exit interviews, which give us a chance to offer some final thoughts and guidance to the folks we've mentored.

CONCLUSION

IXP evolved out of many experiences and experiments with XP in large organizations. Over the past three years, it has helped numerous companies in diverse industries repeatedly achieve significant successes in software development. IXP owes much of its wisdom to XP and remains an organic evolution of XP that is tailored to meet the needs of large organizations.

If you'd like to track the continued evolution of this process, please visit the IXP home page at http://industrialxp.org.

ABOUT THE AUTHOR

Joshua Kerievsky, a Senior Consultant with Cutter Consortium's Agile Project Management practice, began his career as a professional programmer at a Wall Street bank, where he programmed numerous financial systems for credit, market, and global risk departments. After a decade at the bank, he founded Industrial Logic to help companies practice successful software development. Mr. Kerievsky has programmed and coached on small, large, and distributed XP projects since XP's emergence. He has pioneered Industrial XP, a brand of XP tailored for large organizations. Mr. Kerievsky is the author of the acclaimed book *Refactoring to Patterns*. He can be reached at jkerievsky@cutter.com.

REFERENCES

1. Beck, Kent. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1st edition, 1999; 2nd edition, 2004.

2. DeMarco, Tom, and Timothy Lister. *Peopleware: Productive Projects and Teams*. Dorset House, 2nd edition, 1999.

3. DeMarco, Tom, and Timothy Lister. *Waltzing with Bears: Managing Risk on Software Projects*. Dorset House, 2003.

4. Evans, Eric. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2003.

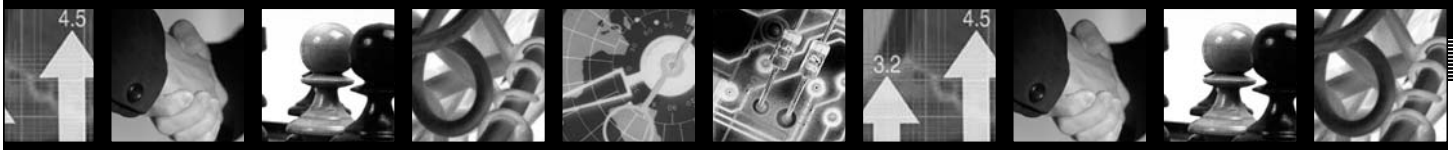
5. Fowler, Martin et al. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1st edition, 1999.

6. Highsmith, Jim. *Agile Project Management: Creating Innovative Projects*. Addison-Wesley, paperback edition, 2004.

7. Kerth, Norman L. *Project Retrospectives: A Handbook for Team Reviews*. Dorset House, 2001.

SUMMIT 2005

2-4 May 2005
Royal Sonesta Hotel
Cambridge, MA, USA



>>> Interact with the experts in IT

Debate and discuss what can — and will — make IT a hyper-differentiator for your enterprise at Cutter Consortium's *Summit 2005*

This three-day event is held in an intimate venue, where you can interact with other high-level delegates and top-name experts from around the world.

Each day focuses on two key IT topics, with a 90-minute keynote and interactive panel discussion for each. The conference includes plenty of opportunities for you to have one-on-one discussions with the experts and other participants — including long breaks, lunches, and evening events, where you can continue the day's debate.

Join us as the top thinkers in the industry come together to debate and discuss these critical, senior-level IT issues:

Add-On Workshops

Agile Business Intelligence
LEADER: KEN COLLIER
1 MAY 2005; 8:30 am – 12:30 pm

Enterprise Project Portfolio Management
LEADER: DONNA FITZGERALD
1 MAY 2005; 8:30 am – 12:30 pm

Enterprise Agile: From Agile Teams to the Executive Suite
LEADER: JIM HIGHSMITH
1 MAY 2005; 1:30 pm – 5:30 pm

Business Enterprise Architecture Modeling
LEADER: KEN ORR
1 MAY 2005; 1:30 pm – 5:30 pm

Keynotes & Debates

Embracing Continuous Change for IT Success
Keynoter: Warren McFarlan, Professor, Harvard Business School

Survivability and Security of Information Infrastructures: Computers Under Attack? What Should We Do?
Keynoter: Rich Pethia, Codirector, CyLab

The Service-Oriented Enterprise: The Business Value of SOA
Keynoter: Mike Rosen, Senior Consultant, Cutter Consortium

Breaking the Glass Ceiling: Beyond the Role of IT Director
Keynoter: Robina Chatham, Senior Consultant, Cutter Consortium; Visiting Fellow, Cranfield School of Management

Innovation: Experimentation Matters
Keynoter: Stefan Thomke, Professor, Harvard Business School

The IT Landscape: Radical Changes, Radical Responses
Keynoter: Steve Andriole, Senior Consultant, Cutter Consortium; Professor, Villanova University

Summit 2005: Themes and Implications
Keynoter: Tom DeMarco, Fellow, Cutter Business Technology Council

Panelists include:
Rob Austin, Christopher Avery, Tom Davenport, Dan Dixon, John Halamka, Tim Lister, Helen Pukszta, Borys Stokalski, Berit Svendsen, and others

To Register simply visit www.cutter.com/summit/ or fill out the form on the back of this page.

SUMMIT 2005

Register Today

Two ways to register for *Summit 2005:*

1. Online at http://www.cutter.com/summit/register.html.
2. Complete the form below and return to Cutter Consortium:

FAX: +1 781 648 1950
PHONE: +1 781 648 8700
E-MAIL: service@cutter.com
MAIL: Cutter Consortium
37 Broadway, Suite 1
Arlington, MA 02474-5552,
USA

Registration Fee:
Seats are $1,995 each and include conference attendance for all three days, roundtable discussions, breakfast, lunch, a Monday night cocktail reception and a Tuesday night book giveaway party.

Multiple-Attendee Discount:
Seats are $1,795 each for the fourth, and each additional, registrant from one company.

"I am amazed that this venue exists, that I can listen and interact with these individuals. I am more amazed that I have not done this before."

— MARK RUBIN,
DEVELOPMENT MANAGER/ARCHITECT,
FIDELITY INVESTMENTS

Event Payment	#Registrants	Subtotal
Summit 2005 Conference (2-4 MAY 2005)		
☐ US $1,995 each (one to three people)	_____	_____
☐ Multiple-Attendee Discount: US $1,795 each (Fourth and each additional person)	_____	_____
Please photocopy this order form and fill out the registration information below for each registrant.		
Agile Business Intelligence (8:30 am – 12:30 pm, 1 MAY 2005)		
☐ US $697 workshop only	_____	_____
☐ US $597 when registered for *Summit 2005*	_____	_____
Enterprise Project Portfolio Management (8:30 am – 12:30 pm, 1 MAY 2005)		
☐ US $697 workshop only	_____	_____
☐ US $597 when registered for *Summit 2005*	_____	_____
Enterprise Agile (1:30 pm – 5:30 pm, 1 MAY 2005)		
☐ US $697 workshop only	_____	_____
☐ US $597 when registered for *Summit 2005*	_____	_____
Business Enterprise Architecture Modeling (1:30 pm – 5:30 pm, 1 MAY 2005)		
☐ US $697 workshop only	_____	_____
☐ US $597 when registered for *Summit 2005*	_____	_____

Total $_____

☐ Check enclosed (payable to Cutter Consortium)
☐ Invoice my company
☐ Please charge my
☐ Mastercard ☐ Visa ☐ AmEx ☐ Diners Club
(Charge will appear as Cutter Consortium.)

Name on Credit Card

Card #

Exp. Date

Signature

To pay by wire transfer: Please have your bank transfer funds to Bank of America, account no. 0086181574, routing no. 011000138. Ask your bank to include "Summit 2005" and your company's name and address, in the advice of credit.

APCON 2005
60R*001H2

Registration Information

Name | Title

Organization | Department

Address/P.O. Box | City | State/Province | ZIP/Postal Code | Country

Telephone | Fax | E-Mail

Cancellation/Substitution Policy

If you have registered for the conference and find that you cannot attend, you will be able to cancel or transfer your registration. If you cancel prior to 1 March 2005, you will be responsible for a $200 processing fee.

If you cancel after 1 March 2005, you will be responsible for the entire registration fee. All cancellations must be in writing and should be sent via fax or mail to: Summit 2005, Cutter Consortium, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA. Our fax number is +1 781 648 1950.

If you have registered for the conference and find that you cannot attend and would like to send someone in your place, you may do so at any time until the first day of the conference. In the unlikely event that Cutter Consortium cancels the conference, Cutter Consortium is not responsible for any airfare, hotel or other costs incurred by registrants. Speakers subject to change without notice.

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 Caruso
- Robert N. Charette
- Alistair Cockburn
- Mike Cohn
- Ken Collier
- Doug DeCarlo
- Tom DeMarco
- Khaled El Emam
- Donna Fitzgerald
- Kerry Gentry
- Michael Hill
- 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
- Ken Schwaber
- Rob Thomsett
- Colin Tully
- Bob Wysocki
- Richard Zultner