

Chapter 3

What Are XP Projects Scared Of?

A popular, successful software development methodology will upset a lot of people.

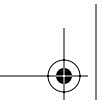
Extreme Programming is an interesting approach because it evolved out of successful software development practices in the Smalltalk world. As such, it pays much more attention than any of the other approaches to what developers do on a project. It also is mainly focused on fears that developers have about projects as opposed to fears that the organization has about projects.

One way of characterizing XP would be to say that it was created by developers to allow them to do what they want to do, while reassuring the organization that the project will deliver successfully. Although this could be seen as an uncharitable interpretation of XP, it turns out that the fears that developers have and the fears an organization has about a project are reasonably aligned.

XP Was Created to Address Project Risks

As Kent Beck states in the opening sentence of *Extreme Programming Explained*, “The basic problem of software development is risk.” [Beck, 2000, p. 3] The examples of risk are instructive:

- ❖ Schedule slips
- ❖ Project canceled
- ❖ System goes sour
- ❖ Defect rate



- ✧ Business misunderstood
- ✧ Business changes
- ✧ False feature rich
- ✧ Staff turnover

Each of these risks is grounded in developer fears.

Developers really fear schedule slips because they are high profile and are never forgotten by the organization. Plus, developers feel really stupid having to explain last-minute slips. Canceled projects are a real drag because they, too, are never forgotten by the organization. Developers also fear cancelation because it creates a gap in the resume, and because developers never want to be part of the conversation that starts, “I see you worked on the project that flushed \$10 million.”

Similarly, developers are scared of projects when the system goes sour and/or has a high defect rate. Both mean lots of stress and long hours, marathon “debugging” sessions stepping through incomprehensible code, and having to explain to the organization why major portions of the new system are going to have to be rewritten.

Misunderstanding the business is a real fear for developers because it is a great way to get into a really awkward situation. Not only do the developers have to explain why the mistake occurred, but they also have to go back and fix it.

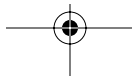
Business changes are terrifying because there is nothing worse for a development team than to be told that the software they have lavished most of their waking hours on is irrelevant.

False feature rich is something that developers get scared of only as they gain experience. Early on in their careers, developers just love to add cool stuff into applications, but eventually every developer ends up in a situation where he has to explain to a project lead why he wasted time on the cool feature when there was more important stuff to work on.

Staff turnover itself is not really something that developers fear, but they are scared of being on a really great project team and then finding that conditions deteriorate to the point that they start looking for work elsewhere.

But Project Risks Are Symptoms, Not the Disease

In reviewing a draft of this book, Andy Hunt (personal communication, February 2002) pointed out that the risks associated with various



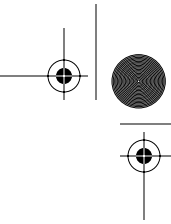
adverse outcomes are just the symptoms. Project teams could easily go crazy trying to catalog, quantify, and mitigate all of the myriad risks that even a small project faces. Instead, what teams need to do is pay attention to the disease that is causing all the symptoms. Looking at this pragmatically, teams have a much simpler challenge. The disease affecting projects is two headed: ignorance and haste.

Ignorance is difficult to treat because it is hard to admit to our own ignorance. Reframing the problem as “how to be successful with only partial knowledge” makes the conversation more palatable. By talking about partial knowledge environments, we enable a team to talk about the research, investigation of prior art, and learning involved in successfully delivering an application. This is in marked contrast to most teams, which seem to specialize in reinventing wheels and ignoring previous work.

Haste is an endemic problem in the software industry. Project teams are nearly always pressed for time and hence end up ignoring prior work because the team does not take the time to do the necessary research and learning. The problem with haste has been described by Tom DeMarco as a consequence of what he calls *Lister's Law*: “People under time pressure don't *think* faster.” [DeMarco, 2001, p. 50] When teams are under time pressure, the team members make sure that they look busy, even though they know that what they should really be doing is taking the time to research and think about what they are doing.

Unfortunately, haste makes the effects of ignorance even worse. When faced with partial knowledge, developers can either make assumptions based on their own experience or they can ask questions and do research. In all too many organizations, the developers have been trained to make assumptions. True, the training department does not put on a course called “Assumptions 101,” but by word and deed developers are encouraged to keep on working and to ask only really important questions.

Interestingly, Extreme Programming addresses this two-headed disease very effectively. Requiring the customer to work as an integral part of the development team makes it easier for developers to ask questions, and many of the practices are aimed at discouraging developers from making assumptions. By colocating the team, XP encourages the entire team to ask questions rather than make assumptions. This goes a long way toward addressing the problem of delivering quality applications in



the face of partial knowledge, because by involving the entire team there is less chance that important issues will be overlooked.

The problem of haste is addressed by talking about the concept of a sustainable pace and small releases. Indeed, the entire planning process in Extreme Programming addresses the issue of haste by the way that it divides up the responsibility for the planning between the developers and the customer [Beck and Fowler, 2001]. Overall, XP addresses haste through a predictable, sustained, and sustainable pace.

Summary

Although Kent Beck [Beck, 2000, p. 3] claims that the basic problem is risk, the underlying problem is that many projects are expected to make haste in the face of partial knowledge.

- ✧ One way of characterizing XP would be to say that it was created by developers to allow them to do what they want to do, while reassuring the organization that the project will deliver successfully.
- ✧ Developers really fear schedule slips because they are high profile and are never forgotten by the organization.
- ✧ The disease affecting projects is two headed: ignorance and haste.
- ✧ Ignorance is difficult to treat because it is hard to admit to our own ignorance.
- ✧ Haste is an endemic problem in the software industry.
- ✧ Unfortunately, haste makes the effects of ignorance even worse.
- ✧ XP addresses these underlying problems by enabling developers to take the time to learn what they need to know while reassuring the organization about the eventual success of the project through predictable, sustained, and sustainable delivery.

