A Collaboration Environment to Support Distributed eXtreme Programming

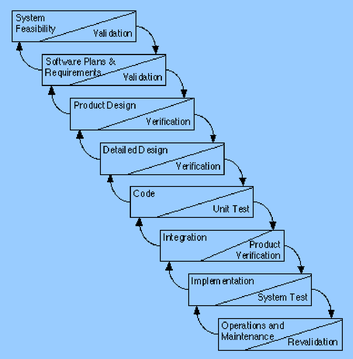
Paul Adams

UNIVERSITY OF LINCOLN

In the beginning... >> What is agility? >> Taking it to the eXtreme... >> ...then calming things down again >> What needs to be done? >> Research goals

In the beginning...

There was a waterfall



And the waterfall brought order to the software engineering process. But there was trouble on the horizon...

- Projects were still late
- Projects were still over budget
- Projects were still not delivering suitable software

So what was wrong?

Today, the waterfall model is so widely recognised that it is often referred to as the “classic” model. But with hind-sight we are able to evaluate its weaknesses and how it contributed to the troubles of software engineering...

- Customer collaboration
  All of the customer collaboration was taking place early in the development cycle, potentially years before the software is developed
- Documentation
  Too much emphasis was given to the production of supporting documentation at each stage, rather than the production of software
- An unresponsive approach
  The process was so strict that there was no maneuvering room for responding to changes in the customer's needs

But now we have learned our lesson. Now we are agile...

What is agility?

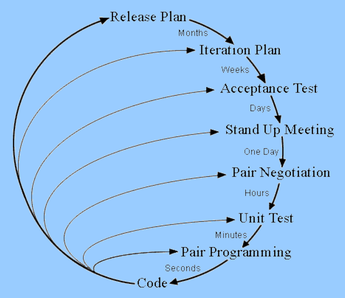
Over time new approaches to the software engineering process developed as practitioners started to address the issues in the waterfall model.
In 2001 representatives from each of these new approaches came together for a skiing holiday in Utah. Motivated by their mutual appreciation of their new processes, they produced...

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

... the agile manifesto

Taking it to the eXtreme...

One of these new approaches is
eXtreme Programming (XP)



- Small development iterations
  Useful software is produced in small increments
- Lightweight method
  XP is based on key principles not on strict processes and rules
- Responsive to change
  XP is particularly useful in an environment of volatile requirements
- Empowered project team
  The eXtreme Programmers have control over the project schedule, not the project managers

...then calming things down again

If eXtreme Programming is assessed against the “individuals and interactions” requirement of the agile manifesto, certain key elements are apparent...

- Daily meetings
  All parties (managers, programmers, customer) meet on a daily basis to ensure that everyone fully comprehends the current status of the project
- Continuous integration
  As new features are developed frequently there is a requirement for the constant integration of new components into the existing system. This is of particular important as often, in XP, there is no set architecture for the system. Instead the architecture evolves as the system does.
- Pair programming
  Programmers work in pairs, one typing, the other watching. The purpose of the second programmer is to assist the typer and to learn about the evolving system. These pairs are reassessed on a regular basis.

All of these interactions, plus the lightweight nature of XP, imply one major requirement for the success of eXtreme Programming...

... the team **MUST** be collocated.

So, what's been done about all this?

- The “Minimally Intrusive Longterm Organisational Support” (MILOS) System
  http://sern.ucalgary.ca/~milos/ papers/2002/MaurerMartel2002a.pdf

MILOS was designed to offer support for task creation, pair programming and task management and awareness. It was implemented as a collection of tools, with pair programming supported by NetMeeting. Similar systems exist based on VNC.

- Adaptive workflow
  Ricardo Jota and Antônio Rito-Silva, *"Supporting Distributed Extreme Programming with Adaptive Workflow"*

The adaptive workflow model is based on using adaptable features within a traditional workflow environment in order to allow for the distribution of team members. This approach offers no specific support for pair programming; a key feature of XP.

What needs to be done?

Why distribute XP?

Enforcing the collocation requirement on an XP project can be very restricting. Here are some reasons why the team may wish to distribute:

- The customer may not be based close to the project team
- Project team members may be involved with many projects and need to move about
- A support tool for distributed eXtreme programming would enable open source developers to use XP.

To allow eXtreme Programmers to distribute their group in an effective manner, a system is required that supports all features of the XP process and that is specifically designed for the purpose of supporting distributed XP.

Existing solutions for distributing eXtreme Programming appear to fall short in two major ways:

- They do not support *ALL* features of XP (such as pair programming)
- They are composed as an ad hoc integration of existing tools

Research goals

I propose that it is desirable and possible to develop a system to enable the distribution of XP so that an existing team of eXtreme Programmers, once distributed, will not suffer a degradation of project velocity (that is, the rate of conversion of required features to implemented features).

To this end there are some intermediate goals:

- To gain a fuller understanding of the XP process
- To gain an understanding of the requirements of distributed software engineers in XP
- To design, implement, test and deploy an environment to support the activities of distributed eXtreme Programmers
- To adapt and improve the the environment through continuous evaluation of its use