Software Development and Professional Practice

Extreme Programming (XP)
Unless otherwise expressly stated, all original material of whatever nature created by John F. Dooley and included in this web site and any related pages, including the site's archives, is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.
References

• most of this material is drawn from
Lightweight Models

• Lightweight means less documentation and process controls
  - it doesn’t mean NO documentation or process.
• Useful for small to mid-sized projects with relatively small teams (up to about 20 developers)
• An antidote to the CMM and rigid models
• can be hard to manage
• Depend heavily on excellent communications
• agile methodologies are examples of lightweight processes
Extreme Programming (XP)

- circa 1995
  - Kent Beck & Ward Cunningham
- Useful for small teams (10-20 developers)
- Relies on
  - Heavy customer involvement (on site customer required)
  - Continuous Unit Testing and reviews
  - Pair programming
  - Short release cycles and frequent releases
• **Risk** is the most basic problem in software development
  - Schedule slips
  - Project canceled
  - System goes sour (software entropy)
  - Defect rate
  - Business problem misunderstood
  - False feature rich (but we never use that!)
  - Staff turnover (burned out - gotta go)
XP Solution(?)

• Extreme programming seeks to minimize and manage these risks using
  - Short release cycles
  - Involving the customer on-site
  - Continuous testing
  - Only implementing the highest priority features
  - Team responsibility and ownership

• Maintenance is the “normal” state of a project; initial development is not.
Four Variables in Development

• **Cost**
  - Probably the most constrained.
  - You can’t spend your way to quality or production.

• **Time**
  - Usually imposed from the outside.

• **Quality**
  - You can make short-term gains by sacrificing quality, but the cost is enormous

• **Scope**
  - Focus on this. It’s the most important and the one you (the programmer) have the most control over. Controlling scope allows you to provide managers and customers control over cost, quality and time.
Cost of Change

• One motivation for XP
• Traditional models work under the assumption that the later in the process you find defects the more expensive they are to fix them.
• XP claims this assumption is no longer valid.
Cost of Change

![Graph showing the cost of change over the software development lifecycle stages: Requirements, Analysis, Design, Implementation, Testing, and Production. The cost increases significantly as we move from Requirements to Production.]
Cost of Change

• XP claims you can develop a process where the cost of change rises only slowly over time and can eventually reach an asymptote.

• “If the cost of change rose slowly over time, you would act completely differently from how you do under the assumption that costs rise exponentially. You would make big decisions as late in the process as possible.” (Beck00)
Revised Cost of Change
Raw Materials

• The raw materials of XP are
  - Learning to drive
  - The four values
  - The principles
  - The four basic activities
Learning to Drive

• Learning to drive is not pointing the car in the right direction

• It’s pointing the car, constantly paying attention and making the constant minor corrections necessary to keep on the road.

• In programming, the only constant is change.

• If you pay attention and cope with change as it occurs, you can keep the cost of change down.
Four Values

• Communication
  - Keeping the team small helps keep the lines of communication small
  - Pair programming & collective ownership of code help spread knowledge around the team.
  - No silver bullet here, but pair programming and a coach help.
Four Values

• Simplicity
  - XP focuses on developing the simplest piece of software to solve today's task.
  - XP bets that “...it is better to do a simple thing today and pay a little more tomorrow to change it if it needs it, than to do a more complicated thing today that may never be used anyway.
  - All developers are allowed (encouraged) to “refactor” code to make it simpler at any time.
Four Values

• Feedback
  - “Concrete feedback about the current state of the system is absolutely priceless. Optimism is an occupational hazard of programming. Feedback is the treatment.”
  - XP programmers typically write tests before they write the code, so that they always have immediate feedback about their code and its impact on the system.
  - Also, the customer is writing functional tests so those are available to measure how well the system is adhering to the “stories” used to develop it.
Four Values

• **Courage**
  - Be willing to make changes at any time when the design no longer fits.
  - Be prepared to throw code away if it doesn't work.
  - Simplicity supports courage because you’re less likely to break a simple system.
  - Track schedule daily and involve the customer in re-prioritizing features as soon as needed.
15 Basic Principles

• Rapid Feedback
  - Get feedback, interpret it, and put it back into the system as quickly as possible. Automated tests are crucial here.

• Assume Simplicity
  - Focus on today’s task and solve it in the simplest way possible.

• Incremental Change
  - Integrate your new code into the system every day.

• Embracing Change
  - It’s gonna happen, so be prepared for it.

• Quality Work
  - Quality isn’t free; strive for defect-free code.
15 Basic Principles

• Teach Learning
  – Teach how to learn to do testing, refactoring, and coding better

• Small initial investment

• Play to Win
  – As opposed to playing not to lose

• Concrete Experiments
  – Every abstract decision (requirements or design) should be tested.

• Open, honest communication
  – You have to be able to criticize constructively and be able to deliver bad news as well as good.
15 Basic Principles

• Work with people’s instincts, not against them
  - People generally like to win, like working with others, like being part of a team, and especially like seeing their code work.
  - Don’t do things that go against this.

• Accepted responsibility
  - The team as a whole is responsible for the product.

• Local adaptation
  - Change it to fit your local circumstances and project.
15 Basic Principles

• Travel light
  - The team and process artifacts you maintain should be few, simple, and valuable.

• Honest measurement
  - Measure at the right level of detail and only measure what makes sense for your project.
Basic Activities of XP

• **Coding**
  - The code is where the knowledge of the system resides so it’s your main activity.

• **Testing**
  - The tests tell you when you are done coding.

• **Listening**
  - To your partner and to the customer.

• **Designing**
  - Design while you code.
The Practices - how to Implement XP

• The Planning Game
  - develop the scope of the next release by combining business priorities and technical estimates.

• Small Releases
  - Put a simple system into production quickly, then release new versions on a very short cycle.

• Metaphor
  - “a simple shared story of how the whole system works.” The metaphor replaces your architecture.

• Simple Design
  - Keep the design as simple as you can each day. Re-design often to keep it simple.
The Practices - how to Implement XP

• Testing
  - Programmers constantly write unit tests. Tests must all pass before integration.

• Refactoring
  - Restructure the system “without changing its behavior” to make it simpler - remove redundancy, eliminate unnecessary layers, add flexibility.

• Pair Programming
  - All production code written by two programmers at one machine. Any code written alone is thrown away.

• Collective ownership
  - The team owns everything => anyone can change anything at any time.
The Practices - how to Implement XP

• Continuous integration
  - Integrate and build every time a task is finished. Possibly several times a day. (as long as the tests all pass!)

• 40-hour week
  - Work a regular 40-hour week. Never work a second week in a row with overtime.

• On-site customer
  - A customer is part of the team, is on-site, writes and executes functional tests, and helps clarify requirements.

• Coding Standards
  - The team has 'em, follows 'em and uses them to improve communication.
Lifecycle of an XP Project

• Exploration
  - Done when “the customer is confident that there is more than enough material on the story cards to make a good first release and the programmers are confident that they can’t estimate any better without actually implementing the system.”
  - Get as many requirements (story cards) written as you can
  - Explore architecture possibilities by doing a prototype - in a week or so.
  - Estimate all tasks done during exploration to practice.

• Planning Game
  - Prepare during exploration
  - First release should take no more than 2 to 6 months.
Lifecycle of an XP Project

• Plan the iterations till first release
  - Iterations are 1 to 4 weeks each.
  - Each iteration produces functional test cases for each story scheduled for the iteration.
  - 1st iteration puts the architecture in place.
  - Subsequent iterations add new features based on the prioritized list.
  - Re-schedule as necessary.

• Implement
  - Design, code, test
  - Design, code, test
  - Design, code, test
Lifecycle of an XP Project

• Productizing
  - Occurs in the last major iteration
  - Add mini-iterations
  - Add daily “spot” meetings to communicate progress and problems.
  - Implement the acceptance processes now
  - Tune performance if necessary
  - RELEASE
Lifecycle of an XP Project

- **Maintenance/Evolution**
  - “simultaneously produce new functionality, keep the existing system running, incorporate new people into the team, and bid farewell to members who move on.”

- **Death**
  - If the customer can’t come up with new stories, mothball the code.
  - If the system can’t deliver anymore, mothball the code and start over.