SOFTWARE DEVELOPMENT

Process, Models, Methods, Diagrams
Software Development Life Cycles

Part - V
Extreme Programming (XP) was conceived and developed by Kent Beck to address the specific needs of software development conducted by small teams in the face of vague and changing requirements.

Extreme Programming nominates coding as the key activity.

The programmer is the heart of XP.
Why Extreme?

XP takes commonsense principles and practices to extreme levels.

- If code reviews are good, we’ll review code all the time (pair programming).
- If testing is good, everybody will test all the time (unit testing).
- If design is good, we’ll make it part of everybody’s daily business (refactoring).
- If integration testing is important, then we’ll integrate and test several times a day.
- If short iterations are good, we will make the iterations really, really short – seconds, minutes and hours, not weeks, months and years.
This new lightweight methodology challenges many conventional tenets, including the long-held assumption that the cost of changing a piece of software rises dramatically over the course of time.

The cost of change curve for XP is a flat curve, which is achieved by simple design, tests, and an attitude of constant refinement of the design.
**OVERVIEW**

**Historical Cost of Change Curve** - The cost of change rising exponentially over time
**OVERVIEW**

**XP Cost of Change Curve** - The cost of change may NOT rise over time
OVERVIEW

Fundamentals of XP include:

- Writing unit tests before programming and keeping all of the tests running at all times.
- Integrating and testing the whole system—several times a day.
- Producing all software in pairs, two programmers at one screen.
- Starting projects with a simple design that constantly evolves to add needed flexibility and remove unneeded complexity.
- Putting a minimal system into production quickly and growing it in whatever directions prove most valuable.
Why is XP so different?

- XP doesn't force team members to specialize and become analysts, architects, programmers, testers, and integrators—every XP programmer participates in all of these critical activities every day.
- XP doesn't conduct complete up-front analysis and design—an XP project starts with a quick analysis of the entire system, and XP programmers continue to make analysis and design decisions throughout development.
- Develop infrastructure and frameworks as you develop your application, not up-front—delivering business value is the heartbeat that drives XP projects.
- Don't write and maintain implementation documentation—communication in XP projects occurs face-to-face, or through efficient tests and carefully written code.
OVERVIEW

XP - What is involved: The four basic activities of Extreme Programming are coding, testing, listening, and designing.

- **Coding:** You code because if you don't code, at the end of the day you haven't done anything.
- **Testing:** You test because if you don't test, you don't know when you are done coding.
- **Listening:** You listen because if you don't listen you don't know what to code or what to test.
- **Designing:** And you design so you can keep coding and testing and listening indefinitely (good design allows extension of the system with changes in only one place).
XP VALUES/PRINCIPLES

There are four basic values in XP:
Communication, Simplicity, Feedback, Courage

Principles
- Rapid feedback
- Assume Simplicity
- Incremental Changes
- Embrace Change
- Quality Work
KEY PRACTICES OF XP

- Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring

- Pair Programming
- Collective ownership
- Continuous integration
- 40-hour week
- On-site Customer
- Coding Standards
PLANNING GAME

Business People
- Scope
- Priority
- Composition
- Release Date

Technical People
- Estimates
- Consequences
- Process
- Detailed Scheduling
Every release must be as small as possible

Contain the most valuable business requirements

Release has to make sense
Helps understand
• Basic elements of the project
• Relationships
The system must communicate everything you want to communicate

No duplicate code

The system should have the fewest possible classes

The system should have the fewest possible methods
TESTING

Sources
- Programmers
- Customers

Types of Testing
- Unit Testing
- Functional Testing
REFACTORING

- After getting something to work we refactor
- Revise and edit
- Followed by running all the tests

**We cannot check in our code until:**

- All tests are green.
- All duplication has been removed
- The code is as expressive as we can make it
PAIR PROGRAMMING

- Two programmers collaborate on the same design, algorithm, code or test case
- Pairing is dynamic
- Encourages communication, productivity and enhances code quality
- Pairing is useful for cross-training established employees and for training new employees
- Pair programming research reveals that:
  - Pairs use no more man-hours than singles
  - Pairs create fewer defects
  - Pairs create fewer lines of code
  - Pairs enjoy their work more
University of Utah Experiment: Pairs spent 15% more time on the program than individuals.
UNIVERSITY OF UTAH EXPERIMENT: CODE WRITTEN BY PAIRS PASSED MORE TEST CASES THAN CODE WRITTEN BY INDIVIDUALS
University of Utah Experiment: Pairs consistently implemented the same functionality produced by individuals in fewer lines of code.
COLLECTIVE OWNERSHIP

- Any team member may add to the code at any time
- Everybody takes responsibility for the whole system
- Encourages simplicity:
  - Prevents complex code from entering the system
- Increases individual responsibility and personal power
- Reduces project risk:
  - Spreads knowledge of the system around the team
CONTINUOUS INTEGRATION

- Code is integrated and tested after a few hours
- Daily builds are for wimps
  - Build, end to end, at every check in
  - Check in frequently
  - Put resources on speeding build time
  - Put resources on speeding test time
- Reduces project risk:
  - You’ll never spend days chasing a bug that was created some time in the last few weeks
- Provides valuable human benefit during development
Overtime is a symptom of a serious problem on a project

Occasionally, programmers may work one week of moderate overtime. Two weeks in a row is out of the question

Programmers need to be well rested to work efficiently
ON-SITE CUSTOMER

- A real customer must sit with the team full time
- The on-site customer enables an XP team to explore business requirements as it needs to and gives direct access to someone who can make key decisions quickly
- Provides value to the company by contributing to the project, thus reducing project risk.
- In XP, if the system isn’t worth the time of one customer, maybe it’s not worth building.
CODING STANDARDS

- Programmers write all code in accordance with rules adopted voluntarily by the team
- Make it impossible to tell who wrote what
- Having no standards slows pair programming and refactoring

Constraints
- No duplicate code
- System should have the fewest possible classes
- System should have the fewest possible methods
- Comments should be minimized
ADVANTAGES/DISADVANTAGES

ADVANTAGES

- Customer focus increase the chance that the software produced will actually meet the needs of the users
- The focus on small, incremental release decreases the risk on your project:
  - by showing that your approach works and
  - by putting functionality in the hands of your users, enabling them to provide timely feedback regarding your work.
- Continuous testing and integration helps to increase the quality of your work
- XP is attractive to programmers who normally are unwilling to adopt a software process, enabling your organization to manage its software efforts better.
DISADVANTAGES

- XP is geared toward a single project, developed and maintained by a single team.

- XP is particularly vulnerable to "bad apple" developers who:
  - don't work well with others
  - who think they know it all, and/or
  - who are not willing to share their "superior" code

- XP will not work in an environment where a customer or manager insists on a complete specification or design before they begin programming.

- XP will not work in an environment where programmers are separated geographically.

- XP has not been proven to work with systems that have scalability issues (new applications must integrate into existing systems).
CONCLUSION

- XP focuses on people
- Values team work over power
- XP works well when there are uncertain or volatile requirements
- **XP is a process not a miracle cure** for all software development problems