

Extreme Programming

What is it?

- A new approach to “responsive” software development (Managing change, show graph vs. waterfall).
- Emphasizes:
 - continual customer involvement
 - staged delivery of features
 - pair programming
 - software refactoring (re-organizing)
 - Continual quality assurance.
- Data indicates that it works well in some settings.

Underlying Principles

- Keep communications channels open. This reduces errors.
- Keep things simple. This reduces costs, improves quality, and reduces interval.
- Constant feedback (from testing). Improves quality.
- Be brave (courageous) to refactor code when needed.
- 12 major concepts (next).

Concept 1: Metaphor

- Essentially an architectural style (e.g., stages in a pipeline, client-server, etc).
- All components referred to using names drawn from this metaphor (pipeline stage etc).
- Provides consistent terminology for team members.

Concept 2: Release Planning

- Requirements are collections of “user stories”
 - Customers prioritize
 - Programmers evaluate cost/risk.
 - Customers come up with an initial plan, in chunks of stories.
- Staged development: every 2 weeks or so:
 - Customers draws up the next desired chunk, based on current progress.
 - Developers deliver previous chunk,
 - and then sign up for subtasks of next chunk

Concept 3: Testing

- Tests are key to development
 - Tests are written along with code
 - Test scripts are automated, using jUnit etc.
 - Continually run along development cycles.
- Acceptance tests for “chunks”
 - contracted with customer
 - run before delivery of chunk.

Concept 4: Pair Programming

- Two engineers (*Driver & Navigator*) work at one keyboard.
- One programmer, the other observes, thinks, and kibbitzes
- Roles get switched off periodically.

This seems to really work. Why??

- Strong anecdotal evidence from industry
 - “We can produce near defect-free code in less than half the time.”

- Empirical Study
 - Pairs produced higher quality code
 - 15% less defects (difference statistically significant)
 - Pairs completed their tasks in about half the time
 - 58% of elapsed time (difference not statistically significant)
 - Most programmers reluctantly embark on pair programming
 - Pairs enjoy their work more (92%)
 - Pairs feel more confident in their work products (96%)

- India Technology Company
 - 24% increase in productivity (KLOC/Person-Month)
 - 10-fold reduction in defects

(from *Laurie Williams, North Carolina State University*)

Concept 5: Refactoring

- Be bold: restructure code as you add features, and it evolves:
 - Simplify when possible
 - Find abstractions
 - Eliminate duplicate code
- Always rely on automated testing to avoid regression.

Concept 6: KISS Design

- No big design up front (BDUF)
- Do the simplest thing that could possibly work!
- Be very skeptical about bells & whistles

Concept 7: Collective Code Ownership (*huh?*)

- Entire team responsible for the code.
- Any engineer has authority to change any code.
- Cleaner code: (more eyeballs, nobody need to work around bad code)
- No “critical sections”: waiting around some bozo to show up.

Concept 8: Continuous Integration

(huh?)

- First each pair does code & unit tests completely.
- Then each pair integrates their code with everyone else.
- Then all run all tests! Only then move to next task
- Continuously clean code.

(alternatives?)

Concept 9: On-site Customer

- o Customer always around to clarify needs.
- o No need to wait for clarifications, or misunderstand anything.
- o Builds continuous customer confidence: less chance of project cancellation!

Concept 10: Bite-size release

- Fixed time-chunk, e.g., 2 weeks.
- Pretty small, still useful “business value”
 - No releases to ‘optimize the transaction manager’.
- Customer continuously involved in evaluation, risk management, planning next release.

Concept 11: 40 hour work week.

- Sleepy, tired programmers with grumpy boyfriends and girlfriends work slower, and
- make more mistakes, and
- get into fights with each other, managers & (worst of all) customers.
- “ . . . fresh and eager every morning, and tired and satisfied every night”
--Kent Beck, inventor of XP.

Concept 12: Coding Conventions

- Because everyone must read code, must use standards
- Layout of code (indentation, space)
- Commenting
- Assertions
- Exception handling.

Concept 13

Start every day with a 15 minute, stand-up meeting.

- Keeps it short
- Say what you did, what problems you had, and what on the agenda for the day.
- Good way to make friends, find your partner for the day.

Conclusion-1

- Seems to work
- Seems to work best & and is popular in IT-style applications.
- Probably of questionable value for embedded software and safety-critical applications.

Conclusion 2--criticisms

- Emphasis on code: no real design document
- Can code really be made all that readable ?
For large systems, may not be practical.
- No real inspections with >2 people.
- No gathering of metrics that can be used for process improvement.
- Not usable with very large teams.