Extreme Programming

A humanistic discipline of software development

Ron Jeffries
www.XProgramming.com
RonJeffries@acm.org
CAREER DAY

Class, today Dilbert will tell us what a career in engineering is all about.

MY JOB INVOLVES EXPLAINING THINGS TO IDIOTS.

THEN THE IDIOTS MAKE DECISIONS BASED ON MISINTERPRETING WHAT I SAID.

THEN IT IS MY JOB TO TRY TO FIX THE MASSIVE PROBLEMS CAUSED BY THE BAD DECISIONS.

EVENTUALLY, RUMORS OVERWHELM FACTS, AND I GIVE UP.

IN THE FINAL PHASE, I ASSIGN BLAME TO AN UNPOPULAR COLLEAGUE.

SO WHATEVER YOU DO IN LIFE, DON'T BE UNPOPULAR.

DON'T LISTEN TO HIM! SAID THE UNPOPULAR TEACHER.
Programmer Rights

- You have the right to know what is needed, with clear declarations of priority.
- You have the right to produce quality work at all times.
- You have the right to ask for and receive help from peers, superiors, and customers.
- You have the right to make and update your own estimates.
- You have the right to accept your responsibilities instead of having them assigned to you.
Manager and Customer Rights

- You have the right to an overall plan, to know what can be accomplished, when, and at what cost.
- You have the right to get the most possible value out of each programming week.
- You have the right to see progress in a running system, proven to work by passing repeatable tests that you specify.
- You have the right to change your mind, to substitute functionality and to change priorities, without paying exorbitant costs.
- You have the right to be informed of schedule changes, in time to choose how to reduce scope to restore the original date. You can even cancel at any time and be left with a useful working system reflecting investment to date.
Could it be this easy?

- If you want
  - more software
  - higher quality
  - in less time
  - for less money
  - with more certainty

- Then just do it with
  - fewer people
  - less management
  - less complexity
  - fewer tools
  - developer commitment

“Extreme Programming”: Act like all you have to do is “write the program”!!
How we do it

**Extreme Programming**

- Focus: simplicity
- Build for NOW
- Build Clean
- Customers specify,
  Developers estimate
- Steering to Success

**Our Rules**

- Do the simplest thing that could possibly work
- You’re not gonna need it
- Merciless Refactoring
- Iteration Planning
- Release Planning, Project

Values
But what about ...

- Unmaintainable code
- Might never get done
- Might not work
- People might not know what to do

- Simple, clear code, relentlessly tested, mercilessly refactored
- Simple, clear schedule and progress report
- Worst things first
  - CRC design
  - Functional tests
- User Stories, Engineering Tasks
Extreme Values

- Simplicity
- Communication
- Feedback
- Courage
Feedback through Power Sharing

Customer

Developed
Customer responsibilities

- Need
- Stories
- Resources
- Priorities
- Acceptance
Developer responsibilities

- Time estimates
- Design
- Code
- Quality
Customer needs to know

- How long
- What's done
- How good
Developer needs to know

- What to do
- When to do it
- When done
Programmer Rights

- You have the right to know what is needed, with clear declarations of priority.
- You have the right to produce quality work at all times.
- You have the right to ask for and receive help from peers, superiors, and customers.
- You have the right to make and update your own estimates.
- You have the right to accept your responsibilities instead of having them assigned to you.
Manager and Customer Rights

- You have the right to an overall plan, to know what can be accomplished, when, and at what cost.
- You have the right to get the most possible value out of each programming week.
- You have the right to see progress in a running system, proven to work by passing repeatable tests that you specify.
- You have the right to change your mind, to substitute functionality and to change priorities, without paying exorbitant costs.
- You have the right to be informed of schedule changes, in time to choose how to reduce scope to restore the original date. You can even cancel at any time and be left with a useful working system reflecting investment to date.
XP Practices

- The 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 decides ...
  - Scope
  - Priority
  - Release Composition
  - Release Dates

- Technical decides ...
  - Estimates
  - Consequences
  - Process
  - Detailed Scheduling (Risk)
Small Releases

- Deliver real business value
- On a very short cycle
  - Business value sooner
  - Rapid feedback
  - Sense of accomplishment
  - Reduced risk
  - Customer confidence
  - Adjustments to requirements
Metaphor

The Internet is like air-traffic control, with all-powerful master controllers observing everything, telling each packet where to go.

The Internet is like spiders crawling on a web - when one way is blocked, they just go another way, always trying to get to their goal.
Simple Design

The right design for the software is one that

- Runs all the tests
- Has no duplicated logic.
- States every intention important to the programmers.
- Has the fewest possible classes and methods

Not ...

- Most "hooks"
- Most abstract
- Designed for the ages
Testing

- Any program feature without an automatic test simply does not exist.
- Unit tests make confidence in the operation of the program part of the program.
- Functional tests do the same for customers.
Refactoring

- Before adding - is there a way to change the program to make the addition simple?
- After adding - is there a way to make the program simpler while still running all the tests?
- Don’t refactor on spec; refactor when the system and process ask you to.
Pair Programming

- All production code is written with two people looking at one machine, with one keyboard and one mouse.

Tactics

Strategy
- Will this work?
- What other tests might not work?
- How can we make this simpler?
Collective Code Ownership

- Natural development of code with objects leads all over the map - you must go where it leads.
- Coordinating class changes takes time, builds up resistance.
- Don’t coordinate, just do it; integrate frequently. *(Continuous Integration)*
Collective Ownership

- Anybody who sees an opportunity to add value to any portion of the code is required to do so at any time.
- Subject to the current requirements,
- Subject to simple design.

- Everyone is responsible for the whole of the system.
Collective Code Ownership

- Whoever finds a snake, kills it
- Unit tests protect functionality
- Collective Code Ownership, plus Refactoring Mercilessly drives reuse
  - Need it? Enhance it, subclass it, use it.
  - Lots of programmers, no waiting!
Continuous Integration

- Integrate and test after a few hours - a day at most.
- If a test fails - you broke it!
40-hour week

I want to

- Be fresh and eager every morning
- Be tired and satisfied every night.
- On Friday, be tired and satisfied enough to feel good about two days doing something other than work.
- On Monday, come in full of fire and ideas.

Overtime is a symptom of serious problems.
On-Site Customer

- A real customer must sit with the team, available to answer questions, resolve disputes, and set small-scale priorities.
- Can’t afford a customer? If the system isn’t worth the time of one customer, maybe it’s not worth building.
Coding Standards

- Collective code ownership, partnering, refactoring all require common coding standards.
- Make it impossible to tell who wrote what.
- Standard requires least work possible.
- Emphasize communication.
- Adopted voluntarily by entire team.
Extreme Values

- Simplicity
- Communication
- Feedback
- Courage
Communication

Team integration through effective efficient communication
Communication comes from

- User Stories
- Commitment Schedule
- Iteration Plan
- CRC Card design
- Pair Programming
- Continuous Integration
- Listen to what the objects tell you
Communication comes from ...

User Stories

- Get the customers to tell stories about how the system will work.
- Write them on cards
- Organize topics
- Classify High, Medium, Low priority
- Associated Functional Test
Why are user stories on cards?

- Tangible Unit of
  - Discussion
  - Scheduling
  - Estimation
  - Functional Testing
  - Completion
101 Union Dues

Bargaining Unit EEs have union dues withheld from the first pay period of the month. The amount varies by union, local, and in some cases varies by the individual.

If dues cannot be taken in the first pay period, they should not be taken until a UD30 transaction is received.

Priority High       Risk Low       Estimate 1       Jan
Communication comes from ...

**Release Plan**

- One to two days
- The team, in concert, estimates all stories, in terms of Engineering Weeks
- Estimate resources, estimate velocity
- Arrange into two- or three-week iterations
- Count 'em
Story Estimation

- Finally based on experience
- Discuss task breakdown
- Estimate task difficulty
- Rate stories 1, 2, 3 “Perfect Engineering Weeks”
Story Estimation

- Excellent accuracy estimating relative difficulty.
  - More experience -> more accuracy
  - Existing code -> easier development

- Very poor accuracy estimating elapsed time.
  - Interruptions, meetings, etc
  - Individual differences
  - ????
Story Estimation

- Time estimation is scary and difficult
- Therefore move as quickly as possible to estimating by comparison.
  - Gummi Bears
  - Story Estimation Units
  - Bucks
  - Cost
  - Points
Communication comes from ...

Release Plan

- One to two days
- The team, in concert, estimates all stories, in terms of Engineering Weeks
- Estimate resources, estimate velocity
- Arrange into two- or three-week iterations
- Count 'em
Communication comes from …

**Iteration Planning**

- About one-half day per 2 or 3-week iteration
- Customer selects stories to implement
- Developers
  - ask questions to clarify understanding
  - create Engineering Tasks
  - sign up for tasks
  - estimate tasks
- Balance load, get to work
Engineering Task

The tasks necessary to implement some User Story:

Union Dues Refund Transaction

- define a new input transaction
- build transaction editing rules
- build station to read transaction
- add bins to receive results
- put bins into entitlement composite bin
- export reconciliation record
101 Union Dues

Bargaining Unit EEs have union dues withheld from the first pay period of the month. The amount varies by union, local, and in some cases varies by the individual.

If dues cannot be taken in the first pay period, they should not be taken until a UD30 transaction is received.

Priority High    Risk Low    Estimate 1
Iteration Plan with Cards

101
H 1

167
H 2

231
M 1

151
H 2

134
H 1

165
M 1

234
H 1

132
H 1

173
M 2
Iteration Plan with Cards
(after First Iteration)
Iteration Plan with Cards

101
√
H 1

151
√
H 2

234
√
H 3

167
H 2

134
M 1

132
H 1

231
H 1

165
M 1

173
M 2

Copyright © 2000, Ronald E Jeffries
Iteration Plan with Cards
Communication comes from ...

**Iteration Tracking**

- Twice weekly
- Ask each developer, for each task:
  - how much work did you get in on it
  - how much work is there to go
- Note increasing estimates
  - possibly too complex a solution
- Note inability to get much time in
  - possibly too much overhead
Communication comes from reporting using ...

Project Variables

- **Resources**
  - People, computers, software

- **Time**
  - How long till you’re done?

- **Scope**
  - What do you have to do?

- **Quality**
  - How good must it be?
Communication comes from reporting using ...

**Resources**

- Report on whether you have the number of people and other key resources upon which you based your estimate
- If not, describe the impact
Communication comes from reporting using ...

**Time**

- Report on how long the most recent Commitment Schedule says the project will take.
- Are people working significant overtime?
- Report clearly, honestly, frequently
Communication comes from reporting using ...

Scope

- Report on how many user stories exist, and how many are done. Show whether the number done is proportional to how far you are through the schedule.

- Worst Things First means that schedule risk reduces as you go forward. Fewer big surprises!
Communication comes from reporting using ...

Quality

Display the same graphs you use to report Functional Test results to the team. They should show:

- increase in number of tests,
- increase in number correct,
- curve predicts 100% by release date
Functional Test Scores

![Functional Test Scores chart](chart-image)
Communication comes from reporting using ...

**Common Truth**

- Report the same information you use daily
- Don’t sugar-coat it. Honesty now will help when you get in trouble later.
- Invite managers to drop by any time. Require them to drop by regularly.
- No dog and pony shows. Let everyone see what is really going on.
Communication comes from ...

**CRC Card Design**

- **Class - Responsibility - Collaborator**
  - Wirfs-Brock: Responsibility-Driven Design
- **Customers and Managers can understand**
- **Easy to get everyone involved**
- **Drawback? No permanent record**
What about UML?

- It's OK to use UML but ...
- Draw manually, on cards or whiteboard
- Throw away all but 2 or 3 pictures
- Wash hands after each use
Communication comes from ...

Pair Programming

- All production code is written with two people sitting side by side at the computer.
  - Better communication
  - Cross-training
  - Better adherence to standards
  - Faster development
Pair Programming: Faster Development???

- One watches strategy; one watches tactics
- Keep going longer
- Less susceptible to interruptions
  - People less likely to interrupt
  - One can carry on if interrupted
- Much faster discovery of problems
Extreme Values

- Simplicity
- Communication
- Feedback
- Courage
Simplicity

Do the simplest thing that could possibly work
Simplicity comes from:

- Do the simplest thing that could possibly work
- Refactor mercilessly
- You’re not gonna need it
- Spike solution
- Supported by: Worst things first
Do the simplest thing that could possibly work

- Find the simplest code that can add the new functionality you need.
- The quickest implementation isn’t the simplest: use refactoring, use inheritance.
- Well-crafted, simplest objects supporting the requirement.
Do the simplest thing that could possibly work?

* Doesn’t this focus on simplicity lead to cut-and-paste programming, redundant or unused methods, obsolete classes?
  * No! Because we Refactor Mercilessly
Simplicity comes from …

**Refactor Mercilessly**

- Two methods look the same?
  - Refactor and combine them
  - No “Cut and Paste” programming

- Two objects have similar function?
  - Refactor till there’s just one

- Replace all references to use the new methods or objects
Why Refactor Mercilessly?

- Simpler system
  - more reliable
  - easier to change
- Single implementation
  - more reliable
  - easier to change
- Go faster!!
Simplicity comes from ... You're not gonna need it!

What would have to be true for it to be better to add unused foreseen capability now, rather than when it is actually needed?

- Must actually be needed in the future.
- Cost more later compared to now.
- Nothing more urgent to do now.
Maybe we do need it?

"How do I know whether I really need something?"

- If your tests will run without it, you don’t need it.
- Note: the code must be good code; it just shouldn’t have functionality that isn’t used.
Maybe we do need it?

"We should be building for the long term."

- Yes, but in the context of the deliverables we’re being paid to implement.
- Build solid, well-factored code, implementing only the functionality you need

See also: Worst Things First
Maybe we do need it?

"But objects are supposed to have integrity, cohesion, completeness."

Yes. Of course objects must be built well. But they need not have function that isn’t required.
Maybe we do need it?

“What about reuse?”

Use comes before reuse. Your current application and its tests will exercise the part of your object that is actually useful.

If your estimate of what is needed is accurate, the need will arise, you’ll implement it, and it will be used at that time.
Maybe we do need it?

“What if someone else needs the feature and winds up building it incorrectly?”

See also: Communication
See also: Pair Programming
See also: Collective Code Ownership
See also: Refactor Mercilessly
Simplicity comes from …

Spike Solution

We’re facing a hard problem that may take a long time to solve. What should we do?

- Solve it quickly. Two days is about right.
- Smallest possible core solution; shortest possible time. Use to learn about your technology, domain, design and process.
- See also: Do the simplest thing that could possibly work.
Spike Solution

- Throw it away - unconditionally
- Don’t let the sun set on bad code
Worst Things First

In what order should stories be addressed?
- Start with highest risks first

What if a high risk story depends on one of lower risk?
- Maybe it really doesn’t - make it so
- See also: Spike Solution
Extreme Values

- Simplicity
- Communication
- Feedback
- Courage
Feedback

Rapid progress and team confidence through feedback, especially testing
Testing gives

- Speed of development
- Confidence
- Support from your constituency
Testing leads to

- Unit tests
- Functional tests
- Continuous integration
Unit Tests

- Owned by developers
- For all classes
- Test a single class
  - (collaborators are involved of course)
- Everything that could possibly break
- Run all unit tests before any release
- Must always score 100% !!!
Unit Test GUI (failure!)

- OOPS!
- Fix this before releasing code!
Unit Test GUI (success!!)

- OK to release code!
Functional Tests

- Owned by customer
- Test every story
- End to end
  - Input through output
  - Check your export files
- Score increases show progress
- Catch regressions
Functional Test GUI

Test Script Interface

Status  Desc  Time  Step
OK      Run  00:007  Start Step group: #fiter02
OK      Run  00:520  ReadInitialEventFileStep
OK      Run  14808  Process events for employee ID 926943
OK      Run  00:590  ReadFmTransStep
OK      Run  00:276  ProcessFmTransStep
OK      Run  00:001  File processing setup for month: 3
OK      Run  00:270  Processing change events for file: chev3
OK      Run  00:090  Processing transfers for file: fntmfr3
OK      Run  00:045  Processing effFile for file: eff3
OK      Run  00:330  Processing file maintenance for file: fmedit3
OK      Run  00:221  Processing CHEC transfer for file: chtmfr3
OK      Run  00:594  Processing overtime for file: ot3
OK      Run  00:135  Processing relocation for file: reloc3
OK      Run  00:021  Post file processing
OK      Run  00:142  Prepare to pay for month 3
OK      Run  00:469  Gross for month 3 year: 1996
OK      Run  00:308  Net for month 3 year: 1996

100%
Time (min): 1.10
Run: 33
Remaining: 0
Errors: 0
Failures: 0
Stubs: 0
Functional Test Scores

Number of Tests

Functional Test Scores

0
50
100
150
200
250

incorrect
unvalidated
correct
Specialized Tests

**C3 Compare Tool**
- parallel old system with new
- compared every value of every check
- classified size of difference
- classified *CAUSE* of difference!
Specialized Test Results

- Within $5: 87%
- State/Local: 3%
- Other: 0%
- Savings: 8%
- Overtime: 2%
What if there's trouble?

- There will be trouble:
  - Resource changes
  - Requirement changes
  - Technical mistakes
  - Process mistakes

- Recognize
- Fix
- Report
Recognize trouble …

- Recognize
- Load factor increases
- Engineering tasks not completing
- Tests not approaching 100%
- Feeling of impending doom
Fix trouble ...

- Go back to your basic processes
  - 90% of the time you are off process

- Tune the process
  - carefully
  - locally
  - inexpensively

- Test the process change
Report trouble ...

- You can't make up the time, therefore
- Report to management
  - admit the mistake
  - describe:
    - effect of mistake
    - action taken
    - impact on project success including schedule
- They won't kill you, it's illegal (at least in the USA)
Dealing with projects under stress

- Projects under stress
  - You’re late
  - They want you to do it sooner
  - They need to cut your budget

- Assess in terms of project variables
  - resources, scope, time, quality
Project stress: using project variables to ...

**Shorten time**

- add resources (maybe)
- reduce quality (if you're not careful)
- reduce scope

This one works
Project stress: using project variables to …

Reduce Resources

- reduce quality (if you’re not careful)
- reduce scope
- increase time

Normal case
Project stress: using project variables to ...

**Increase Quality**

- usually increases time
- may require new resources
- may need to reduce scope
- but there IS a way!

Relentless Testing
Making Rapid Progress

A few additional items contributing to going very fast!
Listen to what the objects are telling you

- Good object development goes quickly: if you're bogged down, you're doing something wrong
- If it hurts, don't do it
- Don't let the sun set on bad code
Continuous Integration

- Integration Hell
  - The longer you wait to integrate, the longer it takes, and the more painful it is
  - THEREFORE: don’t wait at all!

- Microsoft integrates daily to weekly

- We integrate multiple times PER DAY
Roles

- Customer
- Developers
- Tracker
- Architect of the moment
- Mentor
Special Deliverables

- Schedules
  - Commitment Schedule
  - Iteration Plan

- Tracking
  - Project Variables
Tool Set

- Support true rapid development
- Support complex domain model
- Support continuous integration
- Support refactoring
- Language must communicate
Communication

Customer

Product

Variables

Developer
What’s Hard about XP?

- Always doing the simplest thing
- Admitting you don’t know
- Collaborating
- Breaking down emotional walls
XP trouble zones

- Pointing culture rather than driving
- Big Specification
- Death March
- Too Smart
- Size Matters
- Exponential Cost Curve
- Slow Feedback
- No Testing Allowed
Do consider XP if

- You are in a world of rapid change
- You have uncertain requirements
- You value success over ceremony
- You value teamwork over power
eXtreme Programming Resources

- Beck: *eXtreme Programming explained*, 1999
- www.XProgramming.com
Extreme Programming

A humanistic discipline of software development

Ron Jeffries
www.XProgramming.com
SD 2000