Agile and XP Development

Dan Fleck

2008

What is Agile?

- Software development practice aimed at:

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

That is, while there is value in the items on the right, we value the items on the left more.

- Excerpt from The Agile Manifesto

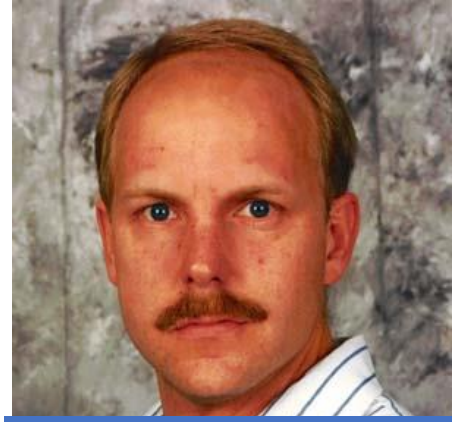
What is Agile?

- A classification of methodologies that adhere to the agile principles
- Developed at a conference in Utah in 2000
- Agile Manifesto (agilemanifesto.org) documentation

An Agile Process

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple ‘software increments’
- Adapts as changes occur

eXtreme Programming (XP)



- Predates Agile
- XP was created by Kent Beck at DaimlerCh[illegible] 1996
- Kent Beck attended the conference in Utah, 2000.
- Is probably the best-known and most complete "agile-method"

XP Fundamentals

- Take the good things we do and turn them up to 10!
  - Simplicity
  - Communication
  - Feedback
  - Courage

XP Practices

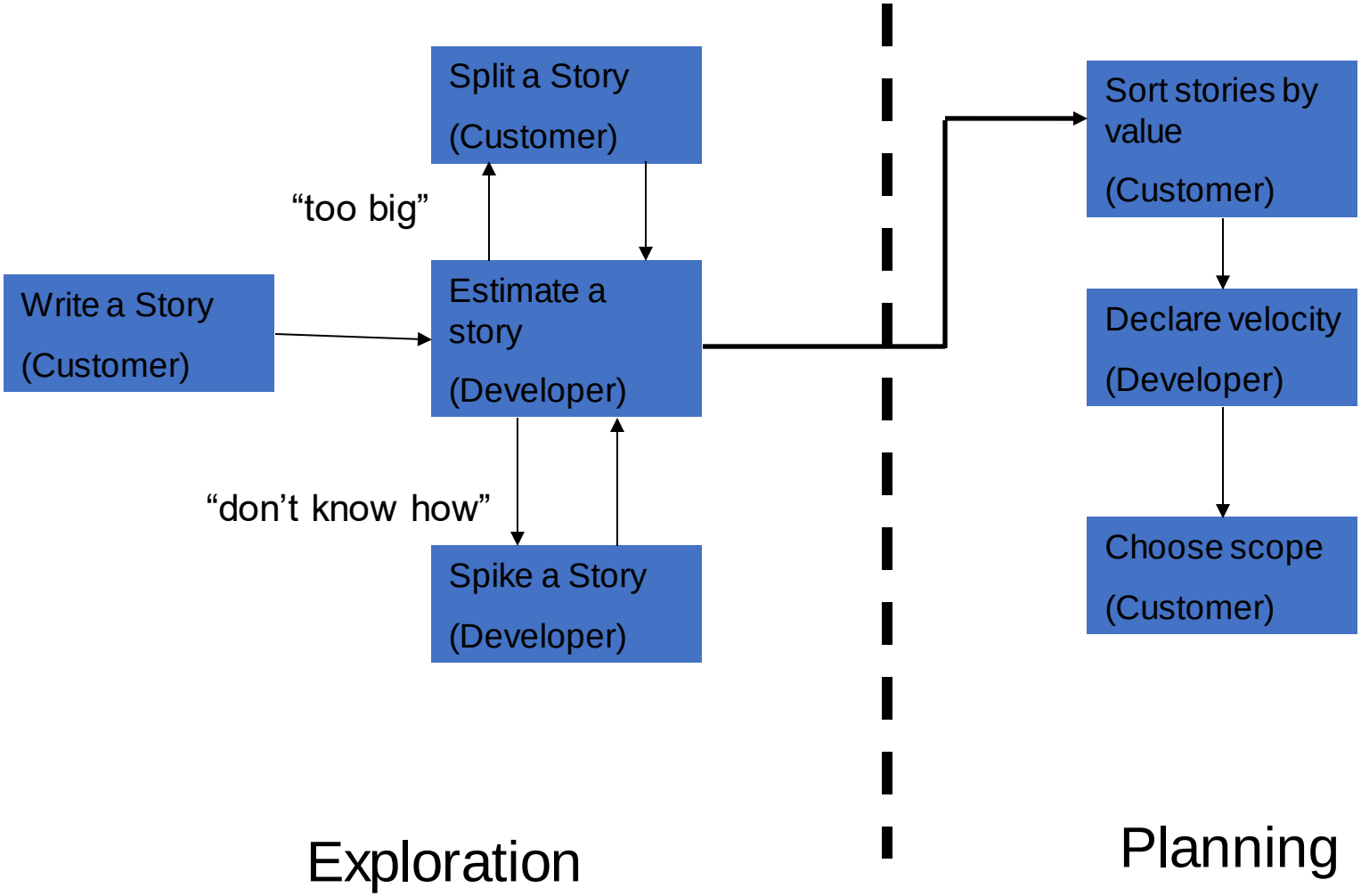
- The Planning Game
- Small Releases
- Metaphor
- On-site Customer
- Simple Design
- Pair Programming
- Test-Driven Development

» Refactoring
» Continuous Integration
» Collective Ownership
» Coding Standards
» Sustainable Pace

The Planning Game

- Distinguish between business people's decisions and developer's decisions
- Short iterations (1-2 weeks)
- Each iteration satisfies a number of user-stories
- Total time for user stories cannot exceed previous iteration's user story time
  - Velocity is a measure of the number of stories finished during an iteration.

Planning Game



XP - User Stories

- Similar purpose as use cases
  - Written by **customers**
  - Estimated by **developers**
  - Replaces large requirements documents
  - Represents anything that is “progress” to the customer
  - Examples:
    - Students can purchase monthly parking passes online.
    - Parking passes can be paid via credit cards.
    - Parking passes can be paid via PayPal
    - Professors can input student marks.
    - Students can obtain their current seminar schedule.
    - Students can order official transcripts.

The Whole Team

- Communication is key!
  - Developers, business analysts, QA, project management, customers, etc… all work in one room
  - Maximizes collaboration

Small Releases

- Systems released to production (or pre-production) very frequently (2-3 months maximum, 1 month is better!)
- Much easier to plan next month than the next 6 months

Continuous Integration

- The whole system is built and tested several times a day
- Automated testing is required (see TDD later)

System Metaphor

- Establishes common vocabulary for the system
- Consistent naming of classes and methods
- Names should be easy to learn and relate to

Example System Metaphor

- Examples: Shared blackboard
  - An Expert puts a Problem on the Board.
  - There are a number of Experts sitting around: when anyone sees a problem they can solve (or know how to break into easier sub-problems), they do so.
  - There's a protocol that defines, "Who gets the chalk next?" and "When are we done?"

- This metaphor suggests a few potential problems:
  - experts have different skills, and they may not necessarily agree on how to solve a particular problem.
  - The chalkboard may become a scarce resource.
  - The most knowledgeable person may find they're doing all the work.
  - We may have "experts" who aren't as good as they think they are.

Sustainable Pace

- Coding is a marathon, not a sprint.
- Team works 40 hours a week - MAXIMUM!
- Tired people aren’t productive

Pair Programming

- All code is written in pairs
- One developer writes code while the other thinks about the code
  - Is the overall system going to work
  - Are there better ways of doing this
  - What test cases still don’t work
- Pairs switch roles frequently (every two hours or so)

Collective Ownership

- No individual owns any piece of the software. All pieces may be worked on by any team member
- Coding Standard - All team members must abide by a coding standard

Test Driven Development (TDD)

- Write automated unit tests FIRST
- Tests must run and fail before code is written
- Code then written until unit tests pass
- Coding must STOP when unit tests pass (no extra features/functions)
- No previously working unit tests can fail

Refactoring

- All code is continuously reviewed and cleaned. Working code is not enough -- must be clean!

- Simple Design - the simplest working design that satisfies the task at hand is used. More complex and general designs may become useful, but not now so we don’t use them!

XP Project People

- Customer
- Developers
- Project Manager
- Tracker
- Coach

Tracker

- Tracker
  - Tracks release plan
  - Tracks iteration plan
  - Tracks acceptance tests (passed/failed)

» Coach

- Watches everything
- Responsible for the process (keep it extreme!)
- Helps with anything else needed… but stay back to let the team be self-reliant!

When not to use XP

- Customer requires a big specification
- Large teams > 100 -- no way! Approx 15 people max.
- If your solution requires you to create complex solutions for future problems (exponential cost curve)
- When you can't get feedback immediately (space shuttle?)
- When you can't get people physically close together (same room)

Summary

- eXtreme Programming is a set of practices that conform to Agile principles
- Xp is one of many Agile methods: DSDM, Crystal, FDD, SCRUM, and others…
- These processes are a logical next step from the older “prescriptive” or “heavyweight” processes

References

- These references were used to create these slides:
  - http://xp123.com/xplor/xp0004/
  - http://www.objectmentor.com/omSolutions/agile_xp_differences.html
  - Beck K., Extreme Programming Explained, 2000
  - Pressman R., Software Engineering: A Practitioner's Approach, 6/e, 2005