Foundations of Software Engineering

Process: Agile Practices

Claire Le Goues

institute for SOFTWARE RESEARCH

Learning goals

- Define agile as both a set of iterative process practices and a business approach for aligning customer needs with development.
- Explain the motivation behind and reason about the tradeoffs presented by several common agile practices.
- Summarize both scrum and extreme programming, and provide motivation and tradeoffs behind their practices.
- Identify and justify the process practices from the agile tradition that are most appropriate in a given modern development process.

What problems are there in software development?

Agile Software Development Is …

Both:

- a set of software engineering best practices (allowing for rapid delivery of high quality software)
- a business approach (aligning development with customer needs and goals)

Brief History of Agile

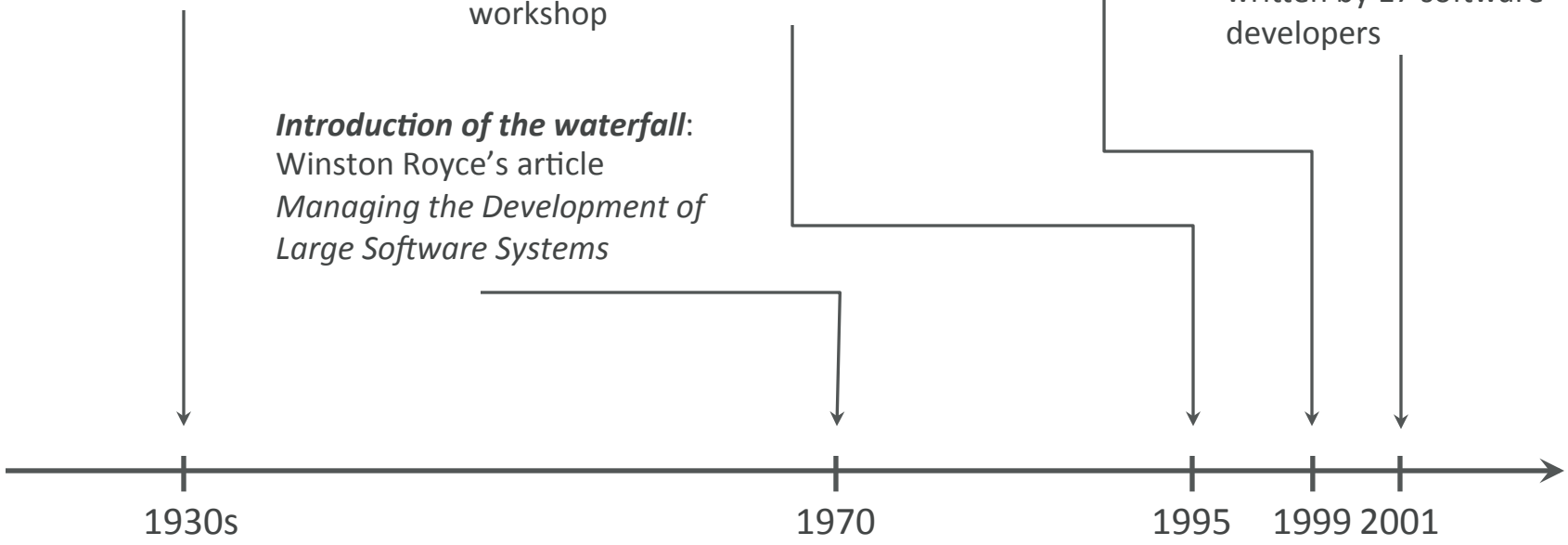
Inception of Iterative and Incremental Development (IID): Walter Shewhart (Bell Labs, signal transmission) proposed a series of "plan-do-study-act" (PDSA) cycles

Introduction of Scrum: Jeff Sutherland and Ken Schwaber presented a paper describing the Scrum methodology at a conference workshop

XP reified: Kent Beck released *Extreme Programming Explained: Embrace Change*

Introduction of "Agile": The Agile Manifesto written by 17 software developers

Introduction of the waterfall: Winston Royce's article *Managing the Development of Large Software Systems*



Agile in a nutshell

- A project management approach that seeks to respond to change and unpredictability, primarily using incremental, iterative work sequences (often called “sprints”).
- Also: a collection of practices to facility that approach.
- All predicated on the principles outlined in “The Manifesto for Agile Software Development.”

The Manifesto for Agile Software Development (2001)

Value

Individuals and interactions	*over*	Processes and tools
Working software	*over*	Comprehensive documentation
Customer collaboration	*over*	Contract negotiation
Responding to change	*over*	Following a plan

The Twelve Principles of Agile Software Development

Individuals and interactions

1. Projects are built around motivated individuals, who should be trusted
2. Face-to-face conversation is the best form of communication (co-location)
3. Self-organizing teams

Working software

4. Working software is delivered frequently (weeks rather than months)
5. Working software is the principal measure of progress
6. Sustainable development, able to maintain a constant pace
7. Continuous attention to technical excellence and good design
8. Simplicity—the art of maximizing the amount of work not done—is essential

Customer collaboration

9. Customer satisfaction by rapid delivery of useful software
10. Close, daily cooperation between business people and developers

Responding to change

11. Welcome changing requirements, even late in development
12. Regular adaptation to changing circumstances

Agile Practices

- Backlogs (Product and Sprint)
- Behavior-driven development (BDD)
- Cross-functional team
- Continuous integration (CI)
- Domain-driven design (DDD)
- Information radiators (Kanban board, Task board, Burndown chart)
- Acceptance test-driven development (ATDD)
- Iterative and incremental development (IID)
- Pair programming
- Planning poker
- Refactoring
- Scrum meetings (Sprint planning, Daily scrum, Sprint review and retrospective)
- Small releases
- Simple design
- Test-driven development (TDD)
- Agile testing
- Timeboxing
- Use case
- User story
- Story-driven modeling
- Retrospective
- On-site customer
- Agile Modeling
- 40-hour weeks
- Short development cycles
- Collective ownership
- Open workspace
- Velocity tracking
- Etc.

40-hour Weeks

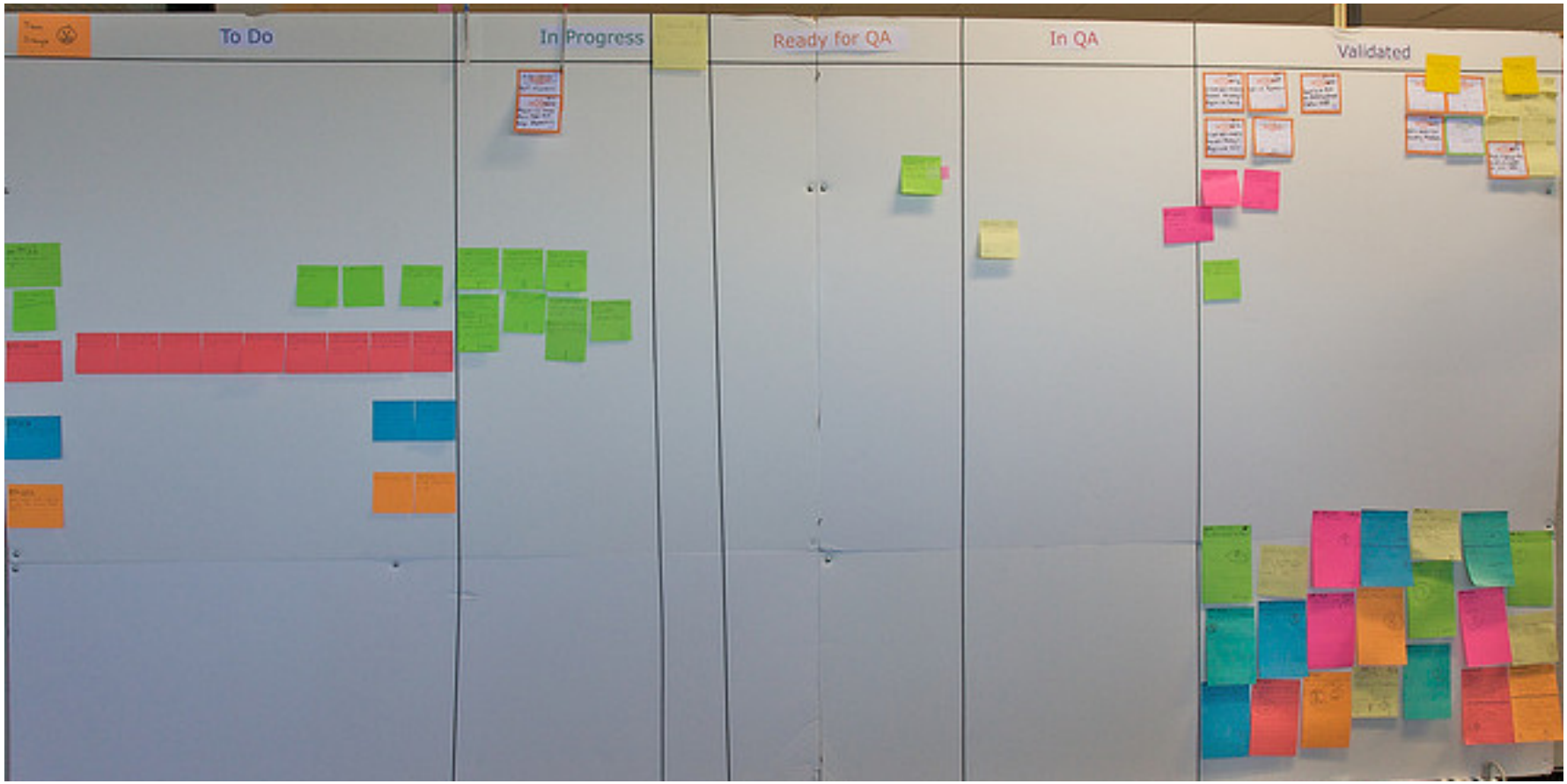
No one can work a second consecutive week of overtime. Even isolated overtime used too frequently is a sign of deeper problems that must be addressed.

Planning Poker

Collective Ownership

Every programmer improves any code anywhere in the system at any time if they see the opportunity.

Kanban Board



Simple Design

"Say everything once and only once":

At every moment, the design runs all the tests, communicates everything the programmers want to communicate, contains no duplicate code, and has the fewest possible classes and methods.

On-site Customer

A customer sits with the team full-time.

Pair Programming

Short development cycle

The software development process is organized in a way in which the full software development cycle—from design phase to implementation phase to test and deployment phase—is performed within a short timespan, usually several months or even weeks.

Small Releases

The system is put into production in a few months, before solving the whole problem. New releases are made often—anywhere from daily to monthly.

Refactoring vs. Design

The design of the system is evolved through transformations of the existing design that keep all the tests running.

Continuous Integration (CI)

New code is integrated with the current system after no more than a few hours. When integrating, the system is built from scratch and all tests must pass or the changes are discarded.

Test-driven development

Programmers write unit tests minute by minute. These tests are collected and they must all run correctly. Customers write functional tests for the stories in an iteration.

Open workspace

Solving Software Development Problems with Agile Practices

	Problem in Software Development	Agile Methods That Mitigate It
1.	Requirement changes during the development process	Close relation with customer, short development cycle, small releases, planning poker, Kanban board
2.	Scope creep	Short development cycle, small releases, planning poker
3.	Architecture erosion	Collective ownership, pair programming
4.	Under- or overestimation (time and budget), sticking to the plan	Close relation with customer, planning poker, short development cycle, small releases
5.	Bringing in new developers (time and effort for their training), steep learning curve	Collective ownership (pros & cons), planning poker
6.	Change of management during the development process	Close relationship with customer
7.	Introducing new bugs as you develop software	40-hour week, collective ownership, short development cycle, small releases, tests, CI, pair programming

Contd.

Solving Software Development Problems with Agile Practices* (contd.)

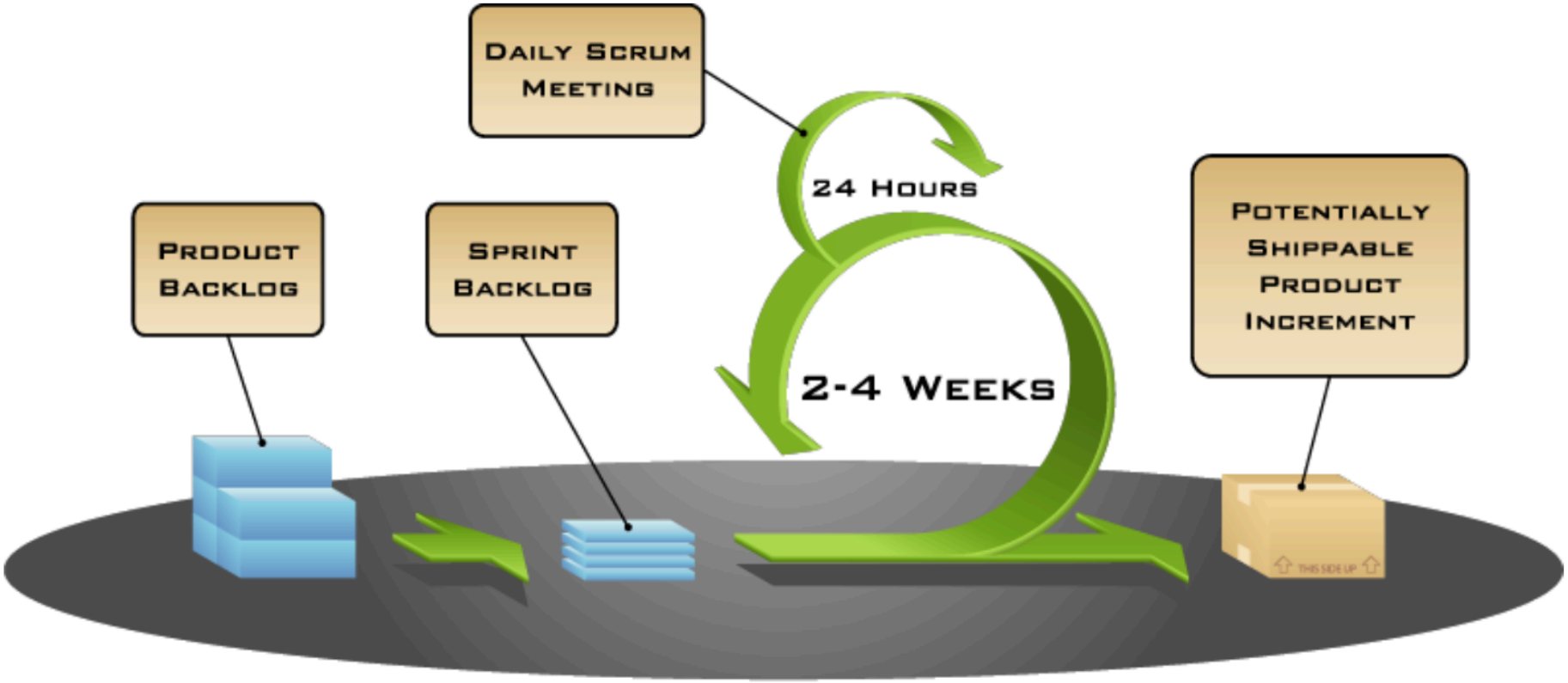
	Problem in Software Development	Agile Methods That Mitigate It
8.	Challenge of communication	Close relation with customer
9.	Developer turnover	Collective ownership (pros & cons), 40-hour week
10.	Integration issues	Collective ownership
11.	Difficulty of tracking bugs	Collective ownership, short development cycle, small releases, CI, tests
12.	Disagreement between developers	Close relation with customer
13.	Scheduling problems (global team)	Close relation with customer
14.	"Groupthink" (tendency of developers to agree with one another, common thinking among them), fear of hurting the feelings of other developers	Planning poker, pair programming
15.	Challenges with integrating with legacy code	Collective ownership

* This is an expanded, but still not comprehensive list.

Scrum

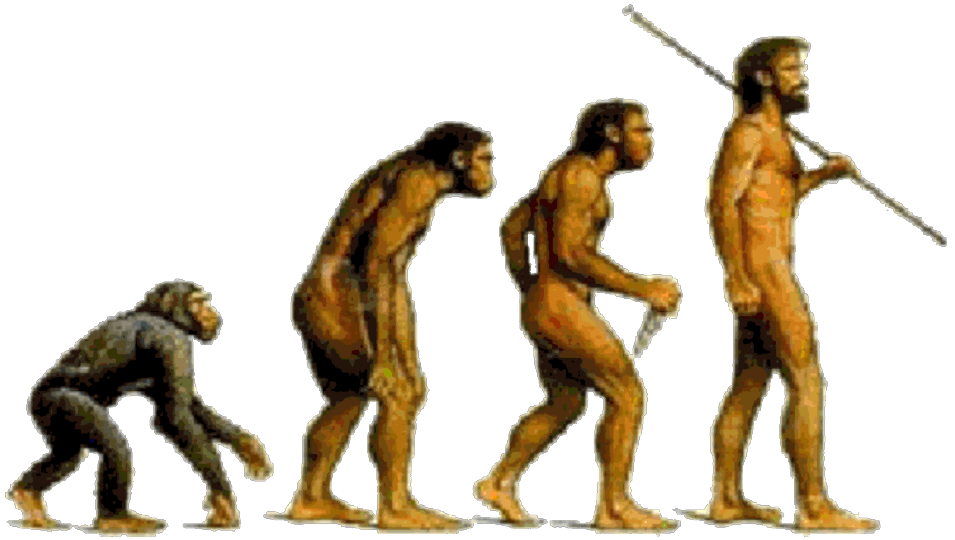
Customer, team, scrum master (?)

Scrum Process

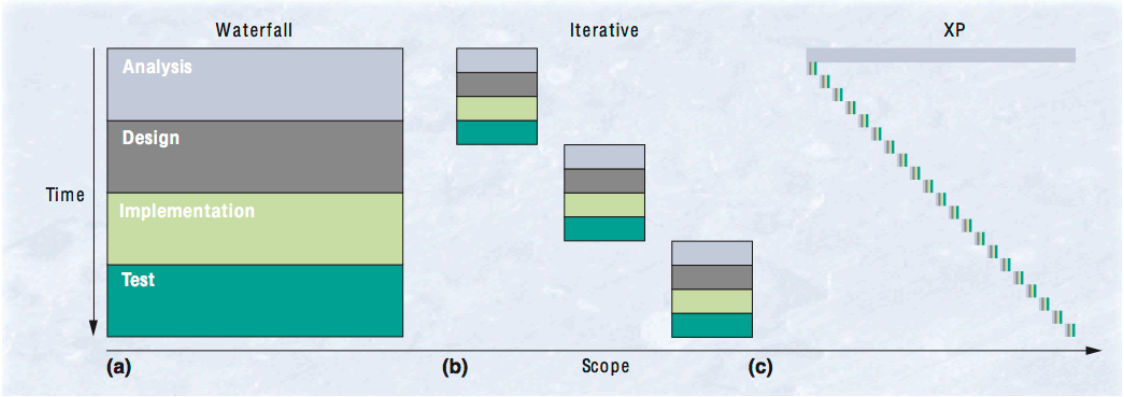


Extreme Programming (XP)

Human evolution



XP evolution

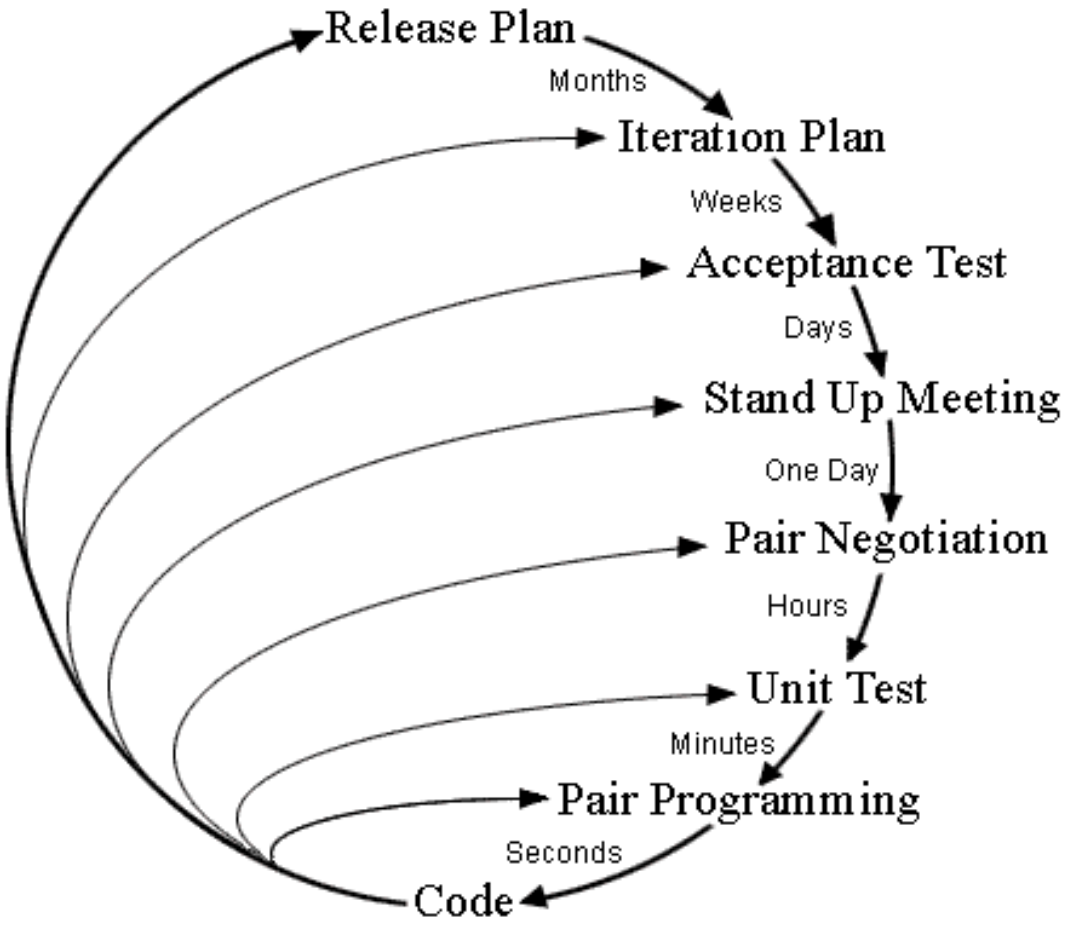


Programming is 4 activities

"Listening, Testing, Coding, Designing. That's all there is to software. Anyone who tells you different is selling something."

–Kent Beck (Extreme Programming Explained)

Extreme Programming (XP)



XP Values

- Communication: Verbal communication is better than written communication.
- Simplicity: Do the simplest thing that could possibly work.
- Feedback: Get lots of feedback, esp from customer ("first-effort" prototype).
- Courage: (somewhat underspecified)

XP Practices (subset of Agile!)

- TDD (test-first approach).
- Planning game: 1-3 week iterations, one iteration at a time, customer decides which user stories to use
- Whole team/on-site customer: “customer speaks with one voice.” Customer may be a whole team.
- Small releases, with valuable functionality, to guard against unhappy customers.
- System metaphor is a single shared story of how it works. (Sort of like architecture)
- Simplest thing that possibly works (coding for today)
- Refactor all the time, because you don’t have up-front design before programming.
- Collective ownership. Everyone is responsible for everything. If a programmer sees something she doesn’t like, she can go change it. Task ownership is individual.
- Pair programming. can code alone for nonproduction code like prototypes
- Continuous Integration. A day of development at most.
- Sustainable pace. 40 hour work weeks.
- Coding standards, Especially since all code can change at all times.

Evolution, exploration

- Evolutionary: code grows/evolves rather than being planned)
  - contrast with RUP (iterative and incremental)
- No requirements documents: programmers and the customer assemble and discuss the customer's needs.
  - Compile stories, remove ambiguity from the stories by making sure that they are testable and estimable.
  - Order by business value.

Questions/Conversation

- Case study: What happened with C3?
- Tradeoffs of practices: on-site customer/ co-located team, TDD, user stories/ planning game, small releases, system metaphor, code for today, refactor, collective ownership, pair programming, continuous integration, sustainable pace, coding standards.