Software Development Processes

January 27, 2020

Overview

Context

- "Embrace Change"
  - Changing requirements
  - Social/cultural change
  - Change in responsibility and roles

Defined vs. Empirical Processes

Defined processes

- do the same thing every time and end up with the same result
- assembly line

Empirical processes

- expect the unexpected
- exercise control through frequent inspections
- adapts the process as necessary

XP is

- A philosophy based on some core values
- A set of practices
- A set of principles that help us translate the values into practices when we need to
- A community that shares these values and practices

Components of XP

- Values - what we do/don't like about a situation
- Practices - the techniques we can use to accomplish the task
- Principles - "domain-specific guidelines for life"

Principles connect values to practices

Value 1: Communication

- Most software problems can be traced back to a lack of communication
- lack of notification about a change to design/requirements
- user doesn't explain requirements thoroughly
- progress mis-reported
- etc!

Types of Communication

- pair programming
- short term planning
- regular meetings with customer
- automated testing and embedded documentation
- regular status meetings

Value 2: Simplicity

- the goal of XP is to develop the "simplest thing that could possible work"!
- do NOT design for features to be developed later (YAGNI)
- gamble by doing the simple thing today and change it tomorrow if necessary
- more complexity might not be needed because future requirements often change
- we REALLY need the cost of change to stay low!

Note that communication supports simplicity

- the more you know, the easier it is to be comfortable with the simple solution
- the simpler the system, the less we need to communicate!
- We look for simplicity in the process as well as the system

Value 3: Feedback

- The system gives feedback on its functionality and design
- programmers by nature are optimistic and feedback about the actual state of the system is the treatment for that condition.
- feedback on customer requests
  - customer writes new "story" (feature description)
  - programmers immediately estimate them so customer knows the cost of the feature
  - Stories are prioritized and progress is tracked by stories completed.

Value 3: Feedback - cont.

- system is put into production as soon as possible so customer can see the system in action
- Valuing customer feedback means that we have to release early and often Feedback is one of the types of communication we require
- Non-customer-related feedback:
  - Automated tests
  - Pair programming
  - Regular status meetings
  - Continuous integration

Value 4: Courage

- when you find a design flaw
- making big changes may cause many of the tests to fail
- fixing the code to make the tests pass takes concerted effort
- XP values the better design - the time you spend fixing things now will be regained in the improved quality (less fixing things later)
- courage is supported by making the cost of change stay as low as possible.
- Courage when the process needs a change
- XPers are always looking for ways to improve the process
- Courage when something you tried fails (Not all changes lead to improvement)

Value 5: Respect

(Added in second edition)

- Balances courage in how we approach problems
- In general, people don't want to be bad or lazy engineers. The conversation should always start with the assumption that the process is the problem - not the person.

Primary Practice - Pair Programming

Pilot

- Owns the keyboard and mouse
- Is writing the code
- Responsible for narrating what s/he is doing and why

Co-pilot (navigator)

- Responsible for keeping an eye on the big picture
- Can do research on details like API syntax

- Should change roles at least every five minutes
- TDD game . . .

Other Primary Practices We Will Use

- Sit together
- Whole Team
- Informative Workspace
- Energized Work (sustainable pace)
- Stories
- Weekly Cycle (Iteration)
- Continuous Integration
- Test-First Programming - TDD
- Incremental Design

Other Primary Practices

- Quarterly Cycle
- Slack (Golden Cards)

Principles (Edition 1)

- rapid feedback
  - psychology experiments have shown that the time between an action and its feedback is critical to learning
  - we want feedback (from tests, developers, management and customers) ASAP
- assume simplicity
  - coding the simplest thing will work 98% of the time
  - the savings will let you spend a lot of time re-doing things for those 2% of the times that more complexity is required
- incremental change
  - big changes are very difficult to complete with quality

Principles (Edition 1) - cont.

- embracing change
  - having to change something doesn't mean it was incorrect the first time (i.e. the need for re-write is not the developer's failure)
  - having to make changes something means we have learned something new and the simplest solution is no longer sufficient (i.e. the developer correctly implemented the simplest thing possible)
- quality work
- teach learning
- small initial investment
- play to win
- the goal is not preventing losing

Principles (Edition 1) - cont.

- concrete experiments
- open, honest communication
- work with people's instincts - not against them
- accepted responsibility
- local adaptation
- honest measurement

Principles (Edition 2)

- Humanity
- Economics
- Mutual Benefit (in what we do and when we do it)
- Self-similarity
- Improvement
- Diversity
- Reflection
- Flow (in development rate)
- Opportunity
- Redundancy (in how we approach process and technical issues)
- Fail (it imparts knowledge)
- Quality
- Baby steps

Corollary Practices

- Real Customer Involvement
- Incremental Deployment
- Team Continuity
- Shrinking Teams
- Root-Cause Analysis
- Shared Code
- Everything is in the code and tests
- Single Code Base
- Daily Deployment
- Negotiated Scope

What do we control?

Development is controlled by

- resources
- time
- scope
- quality

If we try to control all 4, what happens?