Wednesday June 8, 2016

Agile Software Engineering Process

**Agility** – *“a software engineering characteristic that acknowledges change as a necessary part of software engineering and promotes efficient, appropriate response to change.”*

* Based on iterative software engineering process model (iteration = sprint)
* Deliver working software faster

Agile Manifesto

*“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value”*

* Individuals and interactions (over processes and tools)
* Working software (over comprehensive documentation)
* Customer collaborating
* Responding to change

Agile Principles

* Customer satisfaction by rapid, continuous delivery of useful software
* **Working software** is delivered frequently (weeks rather than months)
* Working software is the principal measure of progress
* Even late **changes in requirements** are welcomed
* Close, daily cooperation between business people and developers
* Face-to-face conversation is the best form of **communication** (co-location)
* Projects are built around well rounded individuals, who can be **trusted**
* Continuous attention to technical excellence and good design
* **Simplicity**
* **Self-organizing** teams (team itself, the process, and sprint schedule)
* Regular **adaptation** to changing circumstances

Extreme Programming (XP)

* Defines 5 values (fundamental principles that drive the process)

1. **Communication** – close and frequent collaboration between the developers and other stake holders
2. **Simplicity** – you design only for the immediate need (and accept the fact that you may have to change that later)
3. **Feedback** – feedback from testing (unit, acceptance), scheduling (timeline), etc
4. **Courage** – committing to the extreme programming process as a whole and sticking with it.
5. **Respect** – trust the developers and give them the authority to take on the responsibility they decided to take on (don’t micromanage the process). Trust the customers and their feedback, managers, etc.

* Facets

1. Accept customer as a team member
2. User story: user perspective of functional requirement
3. Test-driven development (writing tests before you write any code and the code you write only focuses on passing that test)
4. Pair programming
5. Minimal design
6. Refactoring
7. Spikes (prototypes)

Industrial Extreme Programming (IXP)

* Enhancements
  + Readiness assessment – is the team ready for extreme programming?
  + Project community – training of all stakeholders in the extreme programming process as well as establishing the communication?
  + Project chartering
  + Test-driven management
  + Retrospectives – looking at what has happened in past iterations and using that information on future iterations
  + Continuous learning – being encouraging and supportive of the team learning new techniques to better support the process
* Issues
  + All-or-nothing commitment to XP (partial is bad)
  + Customer as team member makes requirements volatile
  + Work products are limited (because they focus on whatever is being produced in that single iteration)
  + Minimal design

Adaptive Software Development (ASD)

* Software releases in increments
* Process activities
  + Speculation – gathering the requirements, planning your tasks and schedules, and adapting to change
  + Collaboration – analyzing the requirements
  + Learning – coding, testing, gathering feedback, learning from feedback, and adjusting to the next cycle (iteration)

Scrum

*“An iterative and incremental*[*agile software development*](https://en.wikipedia.org/wiki/Agile_software_development)*framework for managing product development”*

* Agile method delivering highest business value first
* Stakeholders (i.e., clients) assign value to requirements
* Scrum team selects subset of work tasks for next iteration (sprint)
  + Selected based on business value, dependency, and finish time
* Sprint length: 2 to 4 weeks
* Lots of meetings for task assignment, progress updates, problem resolution, and brainstorming (planning, standups, reviews)

Scrum Backlogs

* **Product backlog** is a master list of things to do
  + Items have 2 extra descriptors: business value and time to finish
* **Sprint backlog** is a subset of product backlog items for that sprint
  + Items selected based on business value, how long items will take, and how quickly the team feels it can do that sprint (a **velocity** calculated from previous finished sprint items)
  + Items not finished return to Product backlog
* Product backlog can change during a sprint but the Sprint backlog **SHOULD** **NOT** be changed.

Scrum-specific Roles

* **Product Owner (PO)** – voice of the customer, manages product backlog and user stories
* **User Story** – a free text, story-like, step-by-step description of a functional requirement from an end-user perspective (less formal than a use case detail)
* **Scrum Master (SM)** – enforcer of scrum rules (meeting goals and time limits, no distractions during sprint)

Building a Product Backlog

* Make a set of all items that need to be done to deliver the software (maintained by Product Owner)
  + I.e., features, bugs, tool research, requirements, meetings
  + Items can be user stories, use cased, free text, …anything that is understandable and useful
* All items have a score for business value, dependencies, time to finish (can also have risk)
  + Product Owner assigns business value; team assigns time/risk

Building a Sprint Backlog

* Set of items for the next sprint
* Selected highest business value items first (and their dependencies)
  + Keep picking until the team feels it can’t do any more
* Items can be broken into smaller tasks (i.e., UI task, DB task, test case creation task, testing task, etc)
* Team members picks tasks to work on (self-organizing)

Dynamic Systems Development Method (DSDM)

* Starts with feasibility study
* Prototype each iteration’s functionality
* Determine if prototypes add value
  + If yes, “operationalize it” (prototype it)
* Focus on delivering 80% of product quickly (20% time)
  + Plan to finish it once it’s being used

Agile Unified Process

* Agile version of the Rational Unified Process (a heavier, object-oriented, iterative process hybrid)
* Same linear activities:
  + **Inception** – identify scope, architecture, budget, and signoff (figuring out what you need to do)
  + **Elaboration** – prove the architecture works (make sure what comes out of inception activity is going to work)
  + **Construction** – build working software in highest-priority first iterations (one piece at a time)
  + **Transition** – validate and deploy system to production
* Iteration in each activity listed above
  + Model – analyze and understand the goals of the activity
  + Implement – design and build according to models
  + Test – make sure what you built is correct and not defective
  + Deploy
  + Manage

Summary

* All deal with the same basic actions (requirements, design, etc), but arrange actions differently
* **All accept change as part of the process**
* None of these techniques work for every piece of software
* Remember that agile methods are tools, not a silver bullet
* Be an engineer, not a zealot