Agile Development

Rapid Software Development

- Rapid development and delivery is now often the most important requirement for software systems.
- Businesses operate in a fast-changing requirement and it is practically impossible to produce a set of stable software requirements.
- Software has to evolve quickly to reflect changing business needs.

- Rapid software development
  - Specification, design and implementation are inter-leaved
  - System is developed as a series of versions with stakeholders involved in version evaluation
  - User interfaces are often developed using an IDE and graphical toolset.

Agile Methods

- Not satisfied with the traditional software process (overhead!). Instead, agile methods:
  - Focus on the code rather than the design
  - Iterative
  - Deliver working software quickly and evolve quickly to meet changing requirements.

- The aim of agile methods is
  - to reduce overheads in the software process (e.g. by limiting documentation) and
  - to be able to respond quickly to changing requirements without excessive rework.

Manifesto for Agile Software Development

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 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.
What is “Agility”?

- Effective (rapid and adaptive) response to changes
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed

- Yielding...
- **Rapid, incremental software delivery**

Agility and the Cost of Change

- 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
Agility Principles - I

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Agility Principles - II

- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The stakeholders should be able to maintain a constant pace.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Human Factors

- The process molds to the needs of the people and team, not the other way around
- Key traits must exist among the people on an agile team and the team itself:
  - Competence.
  - Common focus.
  - Collaboration.
  - Decision-making ability.
  - Fuzzy problem-solving ability.
  - Mutual trust and respect.
  - Self-organization.
Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck, 1990s
- Dealing with requirement changes
- Uses OO approaches as preferred paradigm

- Pursuing values
  - Communication (metaphor)
  - Simplicity (design for immediate needs, refactoring later)
  - Feedback (based on unit testing and user stories)
  - Courage (design for today)
  - Respect

Source: J. Hong

Extreme Programming (XP)

12 Practices in XP
- Planning process
- Small release
- Metaphor
- Simple design
- Continuous testing
- Refactoring
- Pair programming
- Collective code ownership
- Continuous integration
- 40 hour week
- On-site customer
- Coding standard
Extreme Programming (XP)

- **XP Planning**
  - Begins with the creation of "user stories"
  - Agile team assesses each story and assigns a value & a cost
  - Stories are grouped to for a deliverable increment
  - A commitment is made on delivery date
  - After the first increment "project velocity" is used to help define subsequent delivery dates for other increments

Extreme Programming (XP)

- **XP Design**
  - Follows the KIS principle
  - Encourage the use of CRC cards (Class-Responsibility-Collaborator cards)
  - For difficult design problems, suggests the creation of "spike solutions" - a design prototype
  - Encourages "refactoring" — an iterative refinement of the internal program design

Extreme Programming (XP)

- **XP Coding**
  - Test-driven: recommends the construction of a unit test for a store before coding commences
  - Encourages "pair programming"
  - Continuous integration

- **XP Testing**
  - All unit tests are executed daily
  - "Acceptance tests" are defined by the customer and executed to assess customer visible functionality — derived from user stories
Extreme Programming (XP) – Again

12 Practices in XP
- Planning process
- Small release
- Metaphor
- Simple design
- Continuous testing
- Refactoring
- Pair programming
- Collective code ownership
- Continuous integration
- 40 hour week
- On-site customer
- Coding standard

XP – Why eXtreme?

In the preface of the book “eXtreme Programming explained”,

To some folks, XP seems like just good common sense. So why the “extreme” in the name? XP takes commonsense principles and practices to extreme levels:
- If code reviews are good, we’ll review code all the time (pair programming)
- If testing is good, everybody will test all the time (unit testing), even the customers (functional testing)
- If design is good, we’ll make it part of everybody’s daily business (refactoring)

XP Debate

- Requirements volatility.
  - Too frequent requirement changes

- Conflicting customer needs.

- Informal requirements description.
  - User stories and acceptance tests are the only explicit manifestation of requirements in XP.
  - Omissions, inconsistencies, and errors are possible

- Lack of formal design.
  - Would it work for complex systems?

Adaptive Software Development

- Originally proposed by Jim Highsmith (2000)

- Focus
  - Human collaboration
  - Team self-organization

- ASD - distinguishing features
  - Mission-focused
  - Feature based
  - Iterative
  - Risk driven
  - Change tolerant
  - Emphasizes collaboration for requirements gathering
  - Emphasizes “learning” throughout the process
Adaptive Software Development

Dynamic Systems Development Method

- Promoted by the DSDM Consortium (www.dsdm.org)
- Driving Strategy, Delivering More
- Framework for agile projects
- DSDM—distinguishing features
  - Incremental prototyping
  - Follow 80%-20% rule (require just enough in each iter.)
- Phases in DSDM life cycle
  - Feasibility (basic requirements and constraints)
  - Foundations (establish requirements)
  - Evolutionary development
    - Iterative Development, timeboxing, and MoScOw prioritization
  - Deployment

Source: www.dsdm.org
Crystal

- Originally proposed by Cockburn and Highsmith
- Distinguishing features
  - A family of process models that allow “maneuverability” based on problem characteristics
  - Face-to-face communication is emphasized
    - Suggests the use of “reflection workshops” to review the work habits of the team

Feature Driven Development

- Originally proposed by Peter Coad et al. (1999)
- Distinguishing features
  - Emphasis is on defining features - a client-valued function that can be implemented in 2 weeks or less.
  - Uses a feature template
    - <action> the <result> <by | for | of | to> a(n) <object>
  - Uses a feature set group template
    - <action> <ing> a(n) <object>
  - More emphasis on project management guideline, compared to other agile methods.
    - Milestones: domain walkthrough, design, design inspection, code, code inspection, promote to build.

Feature Driven Development

- Per feature milestones for completion tracking

<table>
<thead>
<tr>
<th>Domain Walkthrough</th>
<th>Design</th>
<th>Design Inspection</th>
<th>Code</th>
<th>Code Inspection</th>
<th>Promote To Build</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>40%</td>
<td>3%</td>
<td>45%</td>
<td>10%</td>
<td>1%</td>
</tr>
</tbody>
</table>

Source: wiki

Agile Modeling

- Originally proposed by Scott Ambler
- Suggests a set of agile modeling principles
  - Model with a purpose
  - Use multiple, essential models
  - Travel light – keep simplicity with long-term value in mind
    - Must be maintained as changes occur
  - Content is more important than representation
    - Allow flaws in models
  - Know the models and the tools you use to create them
  - Adapt locally – nothing is permanent
Scrum

- Originally proposed by Schwaber and Beedle [2001]
  - Focus on managing iterative development rather than process
  - Does not prescribe practices (i.e., can be combined with any practices)
- Distinguishing features
  - Partition development work into packets
  - On-going testing and documentation during construction
  - Start with user stories (called backlogs)
  - Release in “sprints” (2-4 weeks) derived from backlogs
  - Scrum meetings (even without chairs) and timeboxing
  - Burndown chart

Kanban
Problems in Time-boxed Development

- Common problems include:
  - Short time-boxes force development items to be smaller
  - Smaller development items are often too small
  - Overhead between iterations
    - Quality of requirements suffers as analysts rush to prepare for upcoming cycles
    - Quality of current development suffers when busy analysts are unable to inspect software or answer questions during development
    - Quality often suffers as testers race to complete work late in the development time-box

Inside an iteration, Effort across roles is uneven

- Simultaneous activities are under execution

Kanban (看板)

- Meaning: “Visual cards” or “signboard”
  - Toyota used Kanban cards to limit the amount of inventory tied up in “work in progress” on a manufacturing floor

- Core features Visualize workflow
  - Lean
  - Flow is continuous
  - Just-In-Time (JIT)
  - Limit WIP

Kanban

- Practices
  - Drop time-boxed development
  - Use fewer but larger stories (limit numbers)
  - Estimate minimally
  - Measure cycle time (not velocity)

- Benefits
  - Bottlenecks are visible in real-time.
  - More gradual evolution path from waterfall to agile
  - Still agile when sprints do not make sense (at uncertainty and variability)
  - Applicable in other domains in a company

Modified based on Jeff Patton’s slides
1. Define a work process flow

- Look at the typical flow for features, stories, or work packages and describe typical process steps

This simple process flow has the steps:
1. elaboration & acceptance criteria
2. development
3. test
4. deployment

Source: Jeff Patton's slides

2. Lay out a visual Kanban board

- Place a goals column on the left, then a waiting queue, the process steps, and a final "done" column.

Place an expedite track above the main left to right queue
Place "done and waiting" queues between each work queue (in this example they're placed below)

Source: Jeff Patton's slides

3. Decide limits for items in queue & WIP

- A good limit is a factor of the number of people in a role that can work on an item in a given process step.

This board uses painters tape to indicate available "slots" for work in progress

Source: Jeff Patton's slides

4. Place prioritized goals on the left column

- A good goal describes the outcome we hope to achieve after software ships.

Having goals visible:
- promotes focus
- helps us prioritize
- helps us manage feature scope & requirements

Source: Jeff Patton's slides
5. Start by placing stories in queue

- Mark on the story or feature card the date it entered the queue. (Begin measuring cycle time.)

6. Move features through the process flow

- As the story enters, mark that date on the card. As it’s finished, mark that date on the card.

7. Use the dates to calculate cycle time

- Use average cycle time to set wait times. Pay attention to flow and bottlenecks.

Cycle time = finish date – start date
The average cycle time from the date the item enters the board is the wait time from this point in the queue.
Scrum vs. Kanban

<table>
<thead>
<tr>
<th>SCUM</th>
<th>KANBAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cadence</td>
<td>Regular fixed length sprints (6, 2 weeks)</td>
</tr>
<tr>
<td>Release methodology</td>
<td>At the end of each sprint if approved by the product owner</td>
</tr>
<tr>
<td>Roles</td>
<td>Product owner, scrum master, development team</td>
</tr>
<tr>
<td>Key metrics</td>
<td>Velocity</td>
</tr>
<tr>
<td>Change philosophy</td>
<td>Teams are forced to not make changes to the sprint forecast during the sprint. Doing so compromises learnings around estimation.</td>
</tr>
</tbody>
</table>

Source: https://www.atlassian.com/agile/kanban

Agile is the Norm Now

Source: State of Performance Engineering 2015-16

The Most Popular Agile Methods?

Source: Agile Dev Survey
Wrap-up

- Rapid market changes made agile paradigm emerge.
- All agile processes stress:
  - Self-organization (control)
  - Communication and Collaboration
  - Rapid delivery

*Agile development is more culture than process*

*Culture is process.*