CS 370
REVIEW: Overview of Software Engineering Approaches

DR. MICHAEL J. REALE
FALL 2015
### Methods Covered Here

<table>
<thead>
<tr>
<th>Agility Rank</th>
<th>Method</th>
<th>Very Low</th>
<th>Low</th>
<th>Med.</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scrum</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Adaptive Software Development (ASD)</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Lean Development (LD)</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Crystal</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>eXtreme Programming (XP)</td>
<td>● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Dynamic Systems Development Method (DSDM)</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Rational Unified Process (RUP)</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Team Software Process (TSP)</td>
<td>● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Feature-Driven Development (FDD)</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Capability Maturity Model Integration (CMMI)</td>
<td>● ● ● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Capability Maturity Model for Software (SW-CMM)</td>
<td>● ●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Personal Software Process (PSP)</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Cleanroom</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Scrum

- **Scrum**
  - Project management technique
  - **Scrum Master** = Project/team management
  - **Daily Scrum meetings** = 30 min. team meetings to monitor status and communicate problems
  - **Product Backlog** = All project features/functions/tech enhancements
  - **Sprint** = 30-day development cycle
    - Before Sprint → pick features and plan work
    - **Sprint Goal** = minimum success criterion for Sprint
  - Can introduce Agile to Plan-driven environment
  - Can be scaled up
  - Need Cockburn Level 3 coaches/personnel
Scrum Development Cycle
Adaptive Software Development (ASD)

- Acknowledges **uncertainty**
- Repeating cycles of:
  - **Speculate** = explore and experiment
  - **Collaborate** = work together, share knowledge, make decisions
  - **Learn** = project retrospectives, customer focus groups
- **Characteristics:**
  - Mission-focused
  - Feature based
  - Iterative
  - Timeboxed
  - Risk driven
  - Change tolerant
- **Non-prescriptive method** → difficult to apply to large, critical, relatively stable development efforts
- Need Cockburn Level 3

http://jimhighsmith.com/
https://twitter.com/sambayer
Lean Development

- Proprietary approach by Bob Charette (ITABHI)
- Business strategy and project management
  - Not really development process
- Focus on software reuse → product lines
- Involves entire enterprise
- Most explicit focus on assessing and achieving business value
  - Deliver value sooner rather than later:
  - Buy rather than build if you can
Crystal

- Crystal
  - Not single method → framework/family of methods
  - Vary approach based on environment and project (“crystal metaphor”)
    - “Color” = # of people involved
      - Clear, Yellow, Orange, Red
    - “Hardness” = criticality in terms of type of loss
      - Comfort (C), Discretionary money (D), Essential money (E), Life (L)
  - Characteristics:
    - People and communications centric
    - “Ultra-light” and “Stretch-to-fit”
      - Use harder version ONLY if significant consequences/risks involved
  - Needs Cockburn Level 2

eXtreme Programming (XP)

- **eXtreme Programming**
  - Most widely recognized Agile method
    - Often used in tandem with Scrum
  - Four values:
    - **Communication** → force in positive fashion
    - **Simplicity** → simplest product that meets customer’s needs
    - **Feedback** → customers and developers
    - **Courage** → make hard decisions
**Planning game** = customers and developers negotiate requirements
- Requirements $\rightarrow$ “user stories”
- Design
  - Emphasis on **simple design** (YAGNI)
  - “**Metaphor**” = capture system concept
  - If you need to redesign $\rightarrow$ **refactoring**
  - **Spike** = very simple program/prototype to explore solutions
    - Also helps team get better estimates
    - **Architectural spike** = helps develop project metaphor

**Plan multiple iterations**
**XP: Development**

- **Small** teams with *customer continuously present* on-site
- **Short cycle time** → no more than 3 weeks
- **Project velocity** = how much work you are getting done per iteration
- **Collective ownership** of product
  - Anyone can work on anything
  - **Pair programming** often employed
    - Two programmers sit next to each other and work on same problem/code
- Coding standards established by team
- Emphasis on 40-hour work week
XP: Quality Assurance

- Quality assured through:
  - Test-first approach
    - Customers defines **acceptance tests**
    - Pass test → requirement met
  - Continuous integration
    - Make simpler but complete versions as you go along
    - Build on previous iterations
Advantages:
- Works very well with small applications

Disadvantages:
- Scaling an issue $\rightarrow$ max. 20-person team
- Simple design/YAGNI $\rightarrow$ inappropriate for stable systems with predictable evolution
- Issues when teams not collocated
Dynamic Systems Development Method (DSDM) Atern

- Framework for developing software rather than particular method

- Phases:
  - Feasibility + Foundations
    - Scope, requirements, schedule
  - Exploration
    - Develop partial solution
  - Engineering
    - Make partial solution robust for use
  - Deployment
    - Solution put into use
    - Either:
      - *Scope larger* → go back to Foundations
      - *More requirements to implement* → go back to Exploration
      - *Technical issues* → go back to Engineering
      - *All requirements done* → go to Post-Project

- Timeboxing and MoSCoW priorities (Must have, Should have, Could have, Want)
- Designed for small teams → can be scaled up
- Clear project roles
- Traditional feel but uses risk management (Agile-ish)
RUP

- Rational Unified Process (RUP)
  - “Serial in the large; Iterative in the small”
    - 4 Phases (some with multiple iterations):
      - Inception
      - Elaboration
      - Construction
      - Transition
    - Activities overlap, but more emphasis in certain phases
    - Clear phase exit criteria and milestones
  - Heavier process with Agile attributes
Team Software Process (TSP)

- Five objectives of TSP:
  1) Self-directed teams
  2) Managers that coach/motivate and sustain peak team performance
  3) Make SW-CMM Level 5 (Optimizing) normal and expected
  4) Provide improvement guidance to high-maturity organizations
  5) Facilitate university teaching of industrial-grade team skills

- Pros:
  - Can be scaled up using “team of teams”
  - Very thoroughly defined process

- Cons:
  - Undisciplined teams
Feature-Driven Development (FDD)

- Five phases:
  1. Architecture
  2. Establish list of features based on business needs
  3. Create development plan based on list of features
  4. Develop design packages and work packages for features assigned to current iteration
  5. Build features in software
     - Implement, inspect, test at unit level
     - When complete → integrate into current build of overall system
  - Repeat phases (4) and (5) each iteration

- “Object-oriented” mindset
  - Specifically defines process and roles in OO fashion
  - Individual code ownership

- Get architecture right the first time

- Problems:
  - Issues with unpredictable, “architecture-breaker” changes
  - Relies on “good people” (Cockburn Levels 2 and 3) to recognize when to backtrack to previous phases
CMM

- **CMM** = Capability Maturity Model
  - Organized set of practices to improve *process capability* in one or more discipline areas
    - *Example areas*: software engineering, systems engineering
SW-CMM

- Capability Maturity Model for Software (SW-CMM)
  - Mostly management model
  - Developed as framework for U.S. Government to assess software contractors
    - SEI (Watts Humphrey → “Father of Software Quality”) and MITRE corporation
  - Five maturity levels:
    1) Ad hoc, chaotic
    2) Repeatable
    3) Defined
    4) Managed
      - I.e., measuring processes
    5) Optimizing
  - Key Process Areas (KPAs)
    - Describe minimum activities and common features of a process for a particular area
    - I.e., the stuff you should be doing at a given maturity level
**CMMI**

- **Capability Maturity Model Integration (CMMI)**
  - Provide standard for CMMs
    - Includes systems, not just software
    - Replaces SW-CMM
  - NOT: development life cycle
  - Includes:
    - **Process Areas (PAs)**
      - Sets of activities you should be doing
    - **Generic Practices**
      - General activities in process areas
      - Capability scale:
        - 1) Not performed
        - 2) Performed
        - 3) Managed
        - 4) Defined
        - 5) Quantitatively managed process
        - 6) Optimizing
  - Can be viewed as:
    - **Staged**
      - Process Areas organized by maturity levels, like SW-CMM
    - **Continuous**
      - Process Areas evaluated and improved separately, similar to ISO
Personal Software Process (PSP)

- Focuses on improving individual programming skills
- Four levels:
  - PSP0 → establish personal baseline
  - PSP1 → adds planning and disciplined approach to testing
  - PSP2.0 → focus on code/design quality
  - PSP3 → start of improvement cycle
- Complementary with TSP:
  - TSP → team processes
  - PSP → individual processes
Cleanroom

- **Cleanroom**
  - Focus on **defect-free** code
    - Program = complex mathematical function → can “certify” it is reliable
  - **VERY** heavy processes
    - Initial **black box** approach to system
    - **Stringent design/code reviews**
    - Incremental approach with **strict construction rules**
    - **Statistical testing** (probabilistic input/test cases)
  - **Advantages:**
    - Excellent for developing highly reliable software
  - **Disadvantages:**
    - Need highly skilled practitioners
    - Difficult to scale up
    - Problematic in highly dynamic environments