CS 370
REVIEW: Discipline vs. Agility

DR. MICHAEL J. REALE
FALL 2015
“Discipline” vs. “Agility”

**Discipline (Plan-Driven) → “The Elephant”**
- Implies training, repeatability, and control
  - Ingrains and strengthens through repeated action
- Provides strength and comfort
- Creates well-organized memories, history, and experience

**Agility → “The Monkey”**
- Implies creativity, innovation, and responsiveness to change
- Releases and invents
- Applies memory and history to:
  - Adjust to new environments
  - React and adapt
  - Take advantage of unexpected opportunities
  - Update experience base for the future
“You Need Both to Climb the Tree...”

To be successful, you need both **discipline** and **agility**:

- **Discipline without agility** → bureaucracy and stagnation
  - Resistant to change
  - Excessive focus on process/procedure/Starfleet regulations

- **Agility without discipline** → unencumbered enthusiasm with little direction
  - Chaos rules the day
Process vs. Plan

- **Process**
  - Defined, *generic* set of steps for doing a job
  - Template

- **Plan**
  - Set of steps for a *specific* job, plus other things such as effort, costs, and dates
Process Improvement

- **Process Improvement** = program of activities designed to improve the performance and maturity of the organization’s processes, and the results of such a program
  - I.e., Figuring out how to do what you’re doing better
Process Improvement Cycle

1. Define Process
2. Measure Process
3. Control Process
4. Perform Process
5. Improve Process

The cycle is continuous, starting from Define Process and moving through Measure Process, Control Process, Perform Process, and back to Define Process and so on.
Process Capability

- **Process capability** = inherent ability of a process to produce planned results
  - I.e., Is it capable of doing what you said it would do?

- Improve process capability → process becomes **predictable** and **measurable** → improves **quality** and **productivity**
Process Group

- **Process group** = a collection of specialists that facilitate the definition, maintenance, and improvement of the process(es) used by an organization
  - I.e., people in your organization who figure out (in general) what to do, make sure its being done right, and how we can do things better

- SEPG = software engineering processes only
- EPG = engineering processes
Organizational Maturity

- Organizational Maturity
  - As processes are improved $\rightarrow$ organization “matures”
  - Not just individual projects, but common application of standard procedures across organization
    - Standard process $\rightarrow$ personnel training $\rightarrow$ measure effectiveness $\rightarrow$ improve based on measures
Risk Management

- **Risk management** = organized, analytic process to:
  - Identify **risks** (uncertainties that might cause harm or loss)
  - Assess and quantify risks
  - Develop and apply plans to prevent/handle risks
Software System Architecture

- **Software System Architecture** = defines:
  - (1) A collection of software and system components, connectors, and constraints
  - (2) A collection of system stakeholders’ need statements
  - Rationale which demonstrates that (1) will satisfy (2)
Verification and Validation

- **Verification** = confirms work products (e.g., specifications, designs, models) properly reflect the requirements specified for them
  - I.e., building the product right
  - “Working VERY well”

- **Validation** = confirms the fitness or worth of a work product for its operational mission
  - I.e., building the right product
  - “Is this a VALID solution?”
• **Waterfall model**
  o Early paradigm
  o “Plan to throw one away; you will, anyhow”
    ▶ Originally recommended by Brooks, although later he realized it was a bad idea
  o *Problems:*
    ▶ Puts testing at END of process → only see problems after the whole thing is built
    ▶ Assumes whole thing is built in one shot → doesn’t have to be, and probably SHOULDN’T be
  o Incremental approach is better in general

• **Plan-driven approaches can use either waterfall or incremental**
Origins

- Plan-driven
  - Motivations:
    - “Software Crisis”
  - Derived from methods in hardware engineering
    - DoD, companies like IBM, Hitachi, and Siemens
  - Fit well with academic → math-driven software verification

- Agile
  - Motivations:
    - “Programming is a craft” → disdain for mechanical, dehumanizing application of plan-driven software development
    - Response to rapidly changing nature of Internet-based economy
  - Outgrowth of rapid prototyping/development experiences
  - 2001 – 17 people met in a ski resort in Snowbird, Utah → developed Agile Manifesto
  - Targets “chaordic” work
    - Chaordic = chaos + order
Plan-Driven: Pros and Cons

- Plan-driven
  - Pros:
    - **Quantitative** measurement and improvement of process
    - **Predictability**
    - **Common training across organization** \(\rightarrow\) loss of employees not devastating to project
  - Cons:
    - Mechanical, checklist mentality \(\rightarrow\) customer becomes secondary
    - Less innovation
    - Cost of documentation
    - Cost of standardization
    - Certification over actual improvement
  - Needs:
    - Management support
    - Organizational infrastructure
    - Supportive environment/culture
The Agile Manifesto

- Agile Manifesto
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan

- Weighing of alternatives → NOT binary choice!
The Twelve Agile Principles: 1-6

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
The Twelve Agile Principles: 7-12

7. Working **software** is the primary measure of progress.

8. **Agile processes promote sustainable development.** The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to **technical excellence and good design** enhances agility.

10. **Simplicity**—the art of maximizing the amount of work not done—is essential.

11. The best architectures, requirements, and designs emerge from **self-organizing teams**.

12. At regular intervals, the **team reflects** on how to become more effective, then tunes and **adjusts its behavior accordingly**.
Other Agile Values

• **Pair programming**
  ○ Style of programming in which two programmers work side by side at one computer
  ○ Often used (but not in all Agile methods)

• **Retrospective**
  ○ Post-iteration review of effectiveness of work performed, methods used, and estimates
  ○ Sometimes called “reflection”

• **Test-driven development**
  ○ Module or method tests are incrementally written by developers and customers before and during coding
  ○ Supports and encourages very short iteration cycles
Agile: Pros and Cons

- Agile
  - Pros:
    - Works very well with small teams (5-10) and/or small projects
    - Less time on documentation ➔ more time for development
  - Cons:
    - Potential issues with:
      - Large team size
      - Very complex software development
      - Safety-critical / mission-critical software
    - Analyzing improvement a challenge ➔ nothing written down
  - Needs:
    - Close relationship between customer/users and developers
    - Informed, involved, and authorized on-site customer representative
    - Critical mass of highly motivated, knowledgeable team members
    - Cultural acceptance (e.g., OK with pair programming)
## Cockburn’s Levels of Skill (Modified)

<table>
<thead>
<tr>
<th>Level</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Able to revise a method (break its rules) to fit an <em>unprecedented</em> new situation.</td>
</tr>
<tr>
<td>2</td>
<td>Able to tailor a method to fit a <em>precedented</em> new situation.</td>
</tr>
</tbody>
</table>
| 1A    | With training, able to perform discretionary method steps  
With experience, can become Level 2.  
Need guidance from Level 2 on both agile and plan-driven teams. |
| 1B    | Average-and-below, less-experienced, hard-working developers.  
Work well in plan-driven team  
Likely to slow down agile team  
With training, able to perform procedural method steps  
With experience, can master some Level 1A skills. |
| -1    | May have technical skills, but unable or unwilling to collaborate or follow shared methods.  
Should be rapidly identified and reassigned to work other than performing on either agile or plan-driven teams. |
Each approach is most comfortable in its “home ground”

- **Plan-driven**
  - Large, complex systems
  - Often have safety-critical or other high-reliability attributes
  - Fairly stable requirements
  - Predictable environment

- **Agile**
  - Systems and development teams are smaller
  - Customers/users readily available
  - Requirements and environment are volatile
<table>
<thead>
<tr>
<th>Plan-Driven</th>
<th>Agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predictive mindset</td>
<td>Adaptive mindset</td>
</tr>
<tr>
<td>Tame change</td>
<td>Embrace change</td>
</tr>
<tr>
<td>Standardize (and improve) processes</td>
<td>Use processes as needed</td>
</tr>
<tr>
<td>Focus on software quality</td>
<td>Focus on customer satisfaction</td>
</tr>
<tr>
<td>Heavier processes</td>
<td>Lighter processes (&quot;barely sufficient&quot; mentality)</td>
</tr>
<tr>
<td>Tends towards waterfall/sequential development</td>
<td>Short cycle times and iterative, evolutionary development</td>
</tr>
<tr>
<td>Contract view of requirements</td>
<td>Actively involve customer (on-site) in requirements throughout development</td>
</tr>
<tr>
<td>Documentation for shared knowledge</td>
<td>Tacit knowledge within the team → no &quot;shelfware&quot;</td>
</tr>
<tr>
<td>Defined processes and procedures</td>
<td>Self-organizing teams</td>
</tr>
<tr>
<td>“Big Design Up Front” (BDUP)</td>
<td>Simple design (&quot;YAGNI&quot; = &quot;You Aren’t Going to Need It&quot;)</td>
</tr>
<tr>
<td>Predictive design → plan ahead</td>
<td>Just-in-time redesign → refactoring</td>
</tr>
<tr>
<td>Easier for less-capable developers to contribute</td>
<td>Usually needs highly-skilled developers</td>
</tr>
</tbody>
</table>