Agile Models

Isaac Griffith
Software Engineering Laboratory (SEL)
Department of Computer Science
Montana State University
Additional Copyright Information

Some Slides


• For non-profit educational use only

• May be reproduced ONLY for student use at the university level when used in conjunction with Software Engineering: A Practitioner's Approach, 7/e. Any other reproduction or use is prohibited without the express written permission of the author.

• All copyright information MUST appear if these slides are posted on a website for student use.
Class Activity: The Ballpoint Game

• Rules:
  – You play in one or several Teams
  – Starting point equals endpoint
  – No passing of the ball to your *direct* neighbor
  – The ball *must* have air-time
  – Everyone in the room participates
  – One minute per iteration
  – One minute for continuous improvement & new estimate
  – You play three iterations
The 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.”

– Kent Beck et al
What is “Agility”?  

• Effective (rapid and adaptive) response to change  
• 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 delivery of software.
Agility and the Cost of Change

Cost of change using conventional software processes

Cost of change using agile processes

Idealized cost of change using agile processes
Agile Principles

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**.
Agile Principles

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.
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
Agile Processes

Prevalent Agile Processes

• XP
• Scrum
• Kanban
• And their combinations:
  – Scrumban
  – Scrum and XP
  – XP and Kanban

Other Agile Processes

• Feature Driven Development – FDD
• Agile Unified Process – AUP
• Agile Modeling
• Story-driven Modeling
• Velocity tracking
• Software Development Rhythms
• Graphical System Design – GSD
• Adaptive Software Development – ASD
• Dynamic Systems Development Method – DSDM
• Crystal Methods
• Crystal Clear
• Lean software development
Scrum

If a scrum is like a huddle, then is a hum like a scruddle.

Scrum is all about the Pigs and the Chickens.

A Pig and a Chicken are walking down the road.
The Chicken says: “Hey Pig, I was thinking we should open a restaurant!”

Pig replies: “Hm, maybe, what would we call it?”
The chicken responds: “How about ‘han-n-eggs’?”
The Pig thinks for a moment and says: “No thanks. I’d be committed, but you’d onlyl be involved!”
Scrum

• Originally proposed by Schwaber and Beedle
  – Emphasizes a set of management values and practices rather than focusing on requirements, implementation, etc..
  – Emphasizes the *empirical* rather than defined process

• Promotes the Demming Cycle: Plan-Do-Study-Act
Scrum – Distinguishing Features

• Work is partitioned into “packets”
• Continuous testing and documentation
• “sprints” and the “backlog”
• Standup meetings
• “demos” are delivered
• Has been proven on small to large projects and those with safety critical concerns.
No powerpoints here. Just an operational system demonstration.

Scrum

- Daily Build
- Integration and Regression Tests

### Planning
- Establish vision
- Set Expectations
- Secure Funding

### Staging
- Identify more requirements
- Prioritize enough for first iteration

**Sprint Review**

- Every 24 hours
- Product Backlog: Prioritized product features desired by the customer
- Sprint Backlog: Feature(s) assigned to sprint
- Backlog items expanded by team

- 30 days
- New functionality is demonstrated at end of sprint

- 15 minute daily meeting. Team members respond to basics:
  1) What did you do since last Scrum Meeting?
  2) Do you have any obstacles?
  3) What will you do before next meeting?
Scrum Roles

• Product Owner (Customer)
  – Creates and prioritizes Product Backlog
  – Sets goals for the next Sprint
  – Reviews system at the end of a sprint

• Scrum Master (Management) – 50% Dev
  – Reinforces project and iteration vision and goals
  – Mediator between management and developers
  – Monitors progress and removes blocks
  – Conducts the daily scrum
  – Conducts the sprint review
Scrum Roles

• Scrum Team (7 ± 2 Developers)
  – Work on items in the Sprint backlog

• Chickens (Everyone else)
  – Observe but do not interfere
How to Fail with Scrum

• Not using a self-directed team, allowing the Scrum Master to organize and direct.
• Not updating the Sprint Backlog daily.
• Changing the requirements of an iteration during development.
• Having an uninvolved Product Owner.
• Not conducting a Sprint Review.
• More than one Scrum Master.
How to Fail with Scrum

• Assuming that documentation is bad or not allowed.
• Assuming that design or diagramming is bad or not allowed.
• Poor understanding of Scrum by the dev team.
• Scrum meetings taking too long/unfocused.
• Software not integrated and tested at completion of iteration
• Use of predictive planning -> PERT charts.
Extreme Programming - XP

• Originated with Kent Beck.

• Values
  – Communication
  – Simplicity
  – Feedback
  – Courage
Distinguishing Features

• Has been proven on smaller to medium projects which are not safety critical.
• Developer Centric
• Limited production of artifacts beyond tests and code
• Automated continuous integration
XP Process

Planning
- Release Planning Game
- Story Card Writing/Estimating

Exploration
- Prototypes
- Story Card Writing and Estimating

Iterations
- Iteration Planning Game
- Task Writing/Estimating
- Simple Designs
- CRC Cards

Coding
- Pair Programming

Refactoring

Testing
- Unit Tests
- Acceptance Tests

Release
- Software Increment
- Velocity Computation

Productionizing
- Deployment
- Document
- Training
- Marketing

Maintenance
- Enhance
- Fix
- Build

Task Writing/Estimating
- Simple Designs
- CRC Cards

Pair Programming
XP Roles

• Customer
  – Writes stories and acceptance tests
  – Selects stories for release and iteration

• Development
  – Programmer
    • Writes tests, designs, code
    • Refactors
    • Identifies tasks and estimates
  – Tester
    • Helps customer write and develop tests
XP Roles

• Management
  – Coach
    • Process conscience and process customizing
    • Intervention and teaching
  – Tracker
    • Collect metrics and tracks progress
    • Provides feedback on poor estimates
Extreme Programming (XP)

- XP Planning
  - “User stories”
  - Assigns a cost
  - Deliverable increment
  - Delivery date
  - “Project Velocity”
Extreme Programming (XP)

• XP Design
  – Follows the KIS principle
  – CRC (Class, Responsibility, and Collaborations) Cards
  – “Spike solutions”

• XP Coding
  – Test first or Test Driven Development (TDD)
  – “Pair programming”

• XP Testing
  – Unit testing
  – “Acceptance tests” - ATDD
CRC Card Example

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td>Student number</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>Address</td>
</tr>
<tr>
<td></td>
<td>Phone number</td>
</tr>
<tr>
<td></td>
<td>Enroll in a seminar</td>
</tr>
<tr>
<td></td>
<td>Drop a seminar</td>
</tr>
<tr>
<td></td>
<td>Request transcripts</td>
</tr>
<tr>
<td>Collaborators</td>
<td>Seminar</td>
</tr>
</tbody>
</table>

Pair Programming in Action
Agile 2.0

Ultra-Light Methods

Less is more
Kanban
Kanban

• What is a Kanban?
  – A signaling card put into circulation
  – Control method to reduce WIP

• How does this work for Software Development?
  – Virtual Kanban – cards are work items
  – WIP is controlled by capacity constraints
The Kanban Board

[Diagram showing a Kanban board with columns for Goals, Story Queue, Elaboration, Acceptance, Development, Test, Deployment, and Done.]
Kanban

• Originally formulated by David J. Anderson

• Four Basic Principles
  1. Start with what you do now
  2. Agree to pursue incremental, evolutionary change
  3. Respect the current process, roles, responsibilities and titles
  4. Leadership at all levels
Kanban – Core Practices

1. Visualize
2. Limit WIP
3. Manage flow
4. Make policies explicit
5. Implement feedback loops
6. Improve collaboratively, evolve experimentally (using models and the scientific method)
Things to Keep in Mind

• Kanban is not a development lifecycle
  – Requires a process to already be in place

• Recipe for Success
  – Focus on Quality
  – Reduce WIP
  – Deliver often
  – Balance demand against throughput
  – Prioritize
  – Reduce variability to increase predictability
Questions?
Design Patterns: An Introduction

Isaac Griffith
Software Engineering Laboratory (SEL)
Department of Computer Science
Montana State University
Objectives

- Today you accomplish
  - Understanding what a Design Pattern Is
  - Roles patterns play in the SDLC
  - Understand how to select design patterns
  - Understand what Refactoring and Code Smells are.
What is a Pattern

• A pattern describes a problem that frequently occurs in software design and implementation, and then describes the solution to that problem in such a way that it can be reused.

• Patterns are a means to document good design practices, thus they serve to preserve existing expert knowledge.
Your OO Toolbox

Programmer
- Abstraction
- Encapsulation
- Polymorphism
- Inheritance
- Delegation

Developer/Designer
- Design Principles
- Design Patterns

- Encapsulate What Varies
- Favor Composition over Inheritance
- Program by Contract

Strategy Pattern
Role of Patterns in Software Development

• Promise reuse benefits early in the development lifecycle.

• Why?
  – Reduce development effort
  – Assure higher software quality
  – Provide a common design vocabulary

• Issue:
  – Designing applications by systematically deploying design patterns is not a trivial task.
Pattern-Oriented Design Challenges

• What qualifies a pattern as a design component?
• Can we compose applications solely from design patterns?
• How can we systematically develop applications using design patterns?
Design Patterns in the Software Lifecycle

• Reuse
• Level of reuse
• Development phase classification:
  – Analysis Patterns
  – Architectural Patterns
  – Design Patterns
  – Idioms
Selecting a Pattern

• Consider how design patterns solve design problems
• Scan the Intent sections of patterns
• Study how patterns interrelate
• Examine causes of redesign
• Consider what should be variable in your design
Figure 1.1: Design pattern relationships
Causes of Redesign

• Creating an object by specifying a class explicitly
• Dependence on specific operations
• Dependence on hardware and software platform
• Algorithmic dependencies
• Tight coupling
• Extending functionality by subclassing
• Inability to alter classes conveniently
Guidelines on Design Pattern Use

• Read the pattern once through for an overview
• Go back and study the Structure, Participants, and Collaborations
• Look to the example code
• Choose names for pattern participants that are meaningful
• Define the classes
• Define application-specific names for operations in the patterns
• Implement the operations to carry out the responsibilities and collaborations in the pattern
A Final Note on Patterns

• Design patterns should not be applied indiscriminately.
• They often achieve flexibility and variability by introducing additional levels of indirection which can complicate design or cost you some performance.
• Look to the consequences section to evaluate the benefits and liabilities of a pattern.
Questions?
References


• E. Gamma, R. Helm, R. Johnson, and J. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, Boston, MA.
References

- XP Tools and Techniques image from http://www.xprogramming.com
- CRC Card example images from: http://www.agilemodeling.com/artifacts/crcModel.htm
- Pair programming image from: http://en.wikipedia.org/wiki/File:Pair_programming_1.jpg
- Refactoring examples from: http://www.refactoring.com