Teaching Agile Methods

Dr. Mark C. Paulk
CSEE&T 2017, Savannah, Georgia
Teaching Agile Methods

Agile methods are increasingly being used in industry and studied in academia, but relatively little teaching of how to do agile right occurs at the undergraduate or graduate student level. Although many student teams may claim to be using an agile method on their project, an objective assessment is likely to note violations of fundamental agile practices. For example, I observed MSE student projects spanning multiple terms where, for the first year, no code was being written... yet one of the agile principles is that working software is the fundamental measure of progress. Applying what was learned in class on the project did not align with how agile works in this context. Many students find agile practices such as peer programming and collective code ownership (from Extreme Programming) personally uncomfortable. And changing the requirements for the project midway is a violation of academic norms! Exposing students to the agile paradigm can be a challenging experience for both the teacher and the student.
What Is An “Agile Method”? 

A software engineering “methodology” that follows the Agile Manifesto?

A method that supports responding rapidly to changing requirements?
- Mark Paulk

Does an agile method necessarily imply
- Evolutionary / iterative / incremental development?
- Empowerment / participation of the development team?
- Active collaboration with the customer?
- …
Scrum

My “agile method of choice” for teaching is Scrum.
- simple
- basis for several hybrids (XP, Kanban)
- practices can be added from other agile methods (XP, FDD, etc.)

Roles: ScrumMaster, Product Owner, Developer

Ceremonies: Sprints, Sprint Planning Meeting, Daily Scrum Meeting, Sprint Review Meeting, Sprint Retrospective

Artifacts: Product Backlog, Sprint Backlog, (User Stories), Taskboard, Burndown Chart
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 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  
Mike Beedle  
Arie van Bennekum  
Alistair Cockburn  
Ward Cunningham  
Martin Fowler

James Grenning  
Jim Highsmith  
Andrew Hunt  
Ron Jeffries  
Jon Kern  
Brian Marick

Robert C. Martin  
Steve Mellor  
Ken Schwaber  
Jeff Sutherland  
Dave Thomas
Principles Behind the Agile Manifesto

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.
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.
**Sweet Spots for Agile**

- **Dedicated developers**
  - 100% of time to project

- **Experienced developers**
  - 5 years / 10,000 hours

- **Small, co-located team**
  - less than 10
  - visible, audible, isolated

- **Automated regression tests**

- **Easy access to users**
  - talk to customer / users on day-to-day basis

- **Short increments and frequent delivery to real users**
  - two week iteration is most common agile increment
Does Agile Fit Your Needs?

Manifesto Implications

Interactions → Teams
- The agile methods class should be based on “a” team project.

Working software → Programming
- The teams need to build software to internalize the lessons to be learned from agile.

Customer collaboration → A customer
- The instructor will role-play as the customer, interacting with the student teams during the project.

Responding to change → Project requirements change
- The project requirements will change (significantly) over the course of the term.
Implications of the Agile Principles

People-Centric

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

→ Is a grade the motivation for your students?

→ Lack of experience can be an issue.

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

→ Sustainable pace implies not overloading the students – not completely under instructor control.
Implications of the Agile Principles

Working Software

Working software is the primary measure of progress.

Continuous attention to technical excellence and good design enhances agility.

Simplicity – the art of maximizing the amount of work not done – is essential.

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

→ Non-critical software, not anticipating future features.
Implications of the Agile Principles
Iterative-and-Incremental Development

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

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

⇒ Multiple iterations during the team project

⇒ Minimum of three, preferably four over a semester
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

- Each feature (requirement) needs to be prioritized by the customer (instructor)

- With the thought of upcoming changes...
Implications of the Agile Principles

Requirements Volatility

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

- **Culture clash** with university norms
- **Most classes don’t change assignment requirements** “in the middle”
- **Change requirements** between iterations
Implications of the Agile Principles
Communication

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

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

- Instructor plays the role of customer.

- Must be available to answer questions about the requirements... as well as prioritize and change the requirements.
Implications of the Agile Principles

Process Improvement

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

➔ Part of the learning process is learning what works well for this team.

➔ Try a variety of agile practices and decide what works.
  • pair programming
  • test-driven development
  • refactoring
  • continuous integration is problematic…
Summarizing

Must
• write software
• talk to customer
• have frequent releases, iterations
• have changing requirements
• be team-based
• be a small team
• be co-located (not distance ed)
• use automated testing tools
Should be, but practically cannot be
• dedicated full-time to the project
  - affects practices such as continuous integration
• be experienced professionals
  - students with more experience will get more out of the class by contrasting with what they know

May
• have quite different understandings of software development
• know diverse programming languages
• may not shared a language/IDE
Questions and Answers