Agile Methods: So What?

This talk by Ed Gehringer based on notes by Roy W. Miller
RoleModel Software, Inc.
Why Agile Methods?

• Be more valuable than your peers.
• Be more productive.
• Make you happier.
• Exercise: Your experience with Agile …
The Real Project Lifecycle

- Dream
- Plan
- Capture requirements
- Design a lot, code a little, test if there’s time
- Limp to the finish

Create a comprehensive plan, stick to it at all costs, kill change, hope you survive
The Results

- Junk
- Late
- For a lot of money

Data from 2000 CHAOS Report, Standish Group

<table>
<thead>
<tr>
<th>Year</th>
<th>Succeeded</th>
<th>Failed</th>
<th>Challenged</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>28%</td>
<td>23%</td>
<td>49%</td>
</tr>
<tr>
<td>1998</td>
<td>26%</td>
<td>28%</td>
<td>46%</td>
</tr>
<tr>
<td>1996</td>
<td>27%</td>
<td>40%</td>
<td>33%</td>
</tr>
<tr>
<td>1994</td>
<td>16%</td>
<td>31%</td>
<td>53%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>29%</td>
<td>35%</td>
<td>32%</td>
<td>37%</td>
<td>39%</td>
</tr>
<tr>
<td>Failed</td>
<td>18%</td>
<td>19%</td>
<td>24%</td>
<td>21%</td>
<td>18%</td>
</tr>
<tr>
<td>Challenged</td>
<td>53%</td>
<td>46%</td>
<td>44%</td>
<td>42%</td>
<td>43%</td>
</tr>
</tbody>
</table>
“Modern resolution”

Exercise: Look up the controversy over this report, and submit observations here.

See

- The rise and fall of the Chaos report figures, *IEEE Software*, Jan. 2010
The Source: Taylorism

• Frederick Winslow Taylor, *Principles of Scientific Management* (1911)
• Accepted wisdom by 1950s
• Software began in 1950s
• Software “production” ~= industrial production

**Exercise:** Find an interesting fact about Taylorism. Submit here.

“[I]n each…trade there is always one method and one implement which is quicker and better than any of the rest. And this **one best method** and best implement can only be discovered or developed through a scientific study and analysis of all of the methods and implements in use, together with accurate, minute, motion and time study.”

Making software is like a factory – an efficiency optimization problem
Software Is Different

Exercise: How is software different from other industrial processes? Submit here.

Traditional view...

- Software like industrial production
  - Problem always the same
  - Solution always the same
  - Optimize process
  - Change is disruptive
  - Increase predictability

Reality...

- Software like predicting the weather
  - Problem always different
  - Solution always different
  - Can’t optimize
  - Change is constant
  - Can’t predict accurately

Software is emergent

Copyright © 2002 by RoleModel Software, Inc., © 2017 Edward F. Gehringer
Growing Software

Need a solution that...
- Allows us not to know
- Allows us to explore
- Gives feedback to direct us
- Creates the right conditions, lets software emerge
- Lets us produce the right software at the END

Look up “The Agile debate.”

Agile creates the right conditions for emergent software

Copyright © 2002 by RoleModel Software, Inc., © 2017 Edward F. Gehringer
Agile In a Nutshell

• 4 core values:
  – Simplicity, Communication, Feedback, Courage
• 19 practices
• 1 team
• 3 roles: Customer, Manager, Programmer

What is the simplest thing we can do and still make great software?
The Practices

Joint
- Common Vocabulary
- Iterations
- Open Workspace
- Retrospectives

Development
- Test-First Development
- Pair Programming
- Refactoring
- Collective Ownership
- Continuous Integration
- Just-In-Time Design

Customer
- Storytelling
- Release Planning
- Acceptance Tests
- Frequent Releases

Management
- Accepted Responsibility
- Air Cover
- Quarterly Review
- Mirror
- Sustainable Pace

Agile is about more than programming

Copyright © 2002 by RoleModel Software, Inc., © 2017 Edward F. Gehringer
## Joint Practices

<table>
<thead>
<tr>
<th>Common Vocabulary</th>
<th>Formerly “metaphor” – shared understanding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td>Steering – frequent, regular checkpoints so we can get lots of concrete feedback</td>
</tr>
<tr>
<td>Open Workspace</td>
<td>Easy to communicate and learn</td>
</tr>
<tr>
<td>Retrospectives</td>
<td>Being “Reflective Practitioners” (Donald Schon), learn as we go</td>
</tr>
</tbody>
</table>

**Exercise:** Look up one of these practices (your row number mod 4), and find an interesting fact about it. Submit [here](#).

Create an environment where “one team” can exist and thrive

Copyright © 2002 by RoleModel Software, Inc., © 2017 Edward F. Gehringer
## Customer Practices

<table>
<thead>
<tr>
<th>Practice</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storytelling</td>
<td>Describe each system feature in a small chunk that fits in an iteration</td>
</tr>
<tr>
<td>Release Planning</td>
<td>Tell programmers which features come first</td>
</tr>
<tr>
<td>Customer Tests</td>
<td>Also “acceptance tests” or “functional tests” – tell programmers when they’re done</td>
</tr>
<tr>
<td>Frequent Releases</td>
<td>Get software to users so the team can get feedback to steer with</td>
</tr>
</tbody>
</table>

**Exercise:** Look up one of these practices (your row number mod 4), and find an interesting fact about it. Submit [here](#).

“Drive” the entire process
## Management Practices

<table>
<thead>
<tr>
<th>Practice</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accepted Responsibility</td>
<td>Say what needs to be done, let the team decide who does it and how</td>
</tr>
<tr>
<td>Quarterly Review</td>
<td>Make sure the team knows what it needs to; make sure management knows what it needs to</td>
</tr>
<tr>
<td>Air Cover</td>
<td>Soften up the defenses to make room for the infantry</td>
</tr>
<tr>
<td>Sustainable Pace</td>
<td>Help people avoid burnout</td>
</tr>
<tr>
<td>Mirror</td>
<td>Point out problems, suggest, advise, encourage</td>
</tr>
</tbody>
</table>

**Exercise:** Look up one of these practices (your row number mod 5), and find an interesting fact about it. Submit [here](#).

**Educate, facilitate, stay out of the way**
# Development Practices

<table>
<thead>
<tr>
<th>Practice</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test-First Development</td>
<td>No code without a failing programmer test</td>
</tr>
<tr>
<td>Pair Programming</td>
<td>All code gets two pairs of eyes</td>
</tr>
<tr>
<td>Refactoring</td>
<td>Remove “smells”</td>
</tr>
<tr>
<td>Collective Ownership</td>
<td>Everyone owns all of the code</td>
</tr>
<tr>
<td>Continuous Integration</td>
<td>Integrate many times each day</td>
</tr>
<tr>
<td>Just-In-Time Design</td>
<td>Keep design simple</td>
</tr>
</tbody>
</table>

Copyright © 2002 by RoleModel Software, Inc., © 2017 Edward F. Gehringer
Test-First Development

• Write tests before you write code
  – http://www.junit.org
• Write just enough code to get each test to pass
• All about confidence
• Programmer tests tell you when the code “works”
• Programmer tests must pass 100% all the time
• Test anything you need to be sure it works

Complete test coverage, simplest code that could possibly work, clear intent
Pair Programming

• 2 developers, 1 computer, solving problems together
• One person “drives,” the other “navigates”
• Not Driver/Passenger
• Not Pair Watching
• Pairs should rotate
• Love your pair

Continuous code review, more efficient learning, lower project risk

Copyright © 2002 by RoleModel Software, Inc., © 2017 Edward F. Gehringer
Refactoring

• Changing the design of existing code without changing function
  – http://www.refactoring.com
• All about speed
• Refactor when code “smells”
  – Methods, classes that are too long.
  – Duplicate code (or “almost” duplicate code).
  – Switch statements (instead of polymorphism).
  – “Struct” classes—getters & setters but little else.
• Refactor before adding a feature, and after

Keep code simple, build learning in
Collective Ownership

- Any developer can change any code anytime
- Programmer tests and customer tests tell you if you broke something
- You break it, you fix it

Exercise: Is this a good idea? Look up points pro and con. Submit here.

Convert “my code” to “our code” to lower risk
Continuous Integration

- Integrate changes multiple times each day
- One failing Programmer Test = no integration
- Daily is not enough
- No “Big Bang”

Exercise: Is this a good idea? Look up points pro and con. Submit here.

Maintain speed and spread risk by integrating many times per day
Just-In-Time Design

• Only design for what you’re building
• Always keep the design as simple as possible (OAOO)
• Simplicity allows for change
• Change is constant

*Exercise: Is this a good idea? Look up points pro and con. Submit [here](#).*

Simple design: passes all tests, has no duplication, expresses intent, has least amount of code
All Or Nothing?

• Some practices can stand alone
  – Refactoring, Test-First Development, Pair Programming
• All is better, some often better than none
• All doesn't mean starting all at once

The closer you get to all, the better off you are
Project Success Revisited

Source: Dr. Dobbs, 2010

• Ad-hoc projects
  – 49% are successful, 37% are challenged, and 14% are failures.

• Traditional projects
  – 47% are successful, 36% are challenged, and 17% are failures.

• Iterative projects
  – 61% are successful, 28% are challenged, and 11% are failures.

• Agile projects
  – 60% are successful, 28% are challenged, and 12% are failures.
## Agile vs. Waterfall

### Chaos Resolution by Agile Versus Waterfall

<table>
<thead>
<tr>
<th>SIZE</th>
<th>METHOD</th>
<th>SUCCESSFUL</th>
<th>CHALLENGED</th>
<th>FAILED</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>All Size Projects</strong></td>
<td>Agile</td>
<td>39%</td>
<td>52%</td>
<td>9%</td>
</tr>
<tr>
<td></td>
<td>Waterfall</td>
<td>11%</td>
<td>60%</td>
<td>29%</td>
</tr>
<tr>
<td><strong>Large Size Projects</strong></td>
<td>Agile</td>
<td>18%</td>
<td>59%</td>
<td>23%</td>
</tr>
<tr>
<td></td>
<td>Waterfall</td>
<td>3%</td>
<td>55%</td>
<td>42%</td>
</tr>
<tr>
<td><strong>Medium Size Projects</strong></td>
<td>Agile</td>
<td>27%</td>
<td>62%</td>
<td>11%</td>
</tr>
<tr>
<td></td>
<td>Waterfall</td>
<td>7%</td>
<td>68%</td>
<td>25%</td>
</tr>
<tr>
<td><strong>Small Size Projects</strong></td>
<td>Agile</td>
<td>58%</td>
<td>38%</td>
<td>4%</td>
</tr>
<tr>
<td></td>
<td>Waterfall</td>
<td>44%</td>
<td>45%</td>
<td>11%</td>
</tr>
</tbody>
</table>

The resolution of all software projects from FY2011-2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000.
So What?

Agile doesn’t matter – results do

• Agile reflects the true nature of the problem (complex)
• Agile is change-tolerant
• Agile is realistic
• Agile has the potential to facilitate organizational change

Agile increases likelihood for success
Resources

http://www.xprogramming.com (Ron Jeffries)

http://www.junit.org (JUnit testing framework)

Addison Wesley XP Series:
  Extreme Programming Explained: Embrace Change, Beck
  Extreme Programming Installed, Jeffries, Hendrickson, Anderson
  Planning Extreme Programming, Fowler and Beck
  Extreme Programming Applied: Playing to Win, Auer and Miller

Refactoring, Fowler

IBM developerWorks XP Column, starting in August
(http://www.ibm.com/developerWorks)

Growing Software (working title), Addison Wesley, 2003

http://www.roywmiller.com

https://www.agilealliance.org/ (Agile Alliance)