Agile and Evolutionary Software Development

Václav Rajlich
Wayne State University
Department of Computer Science
rajlich@wayne.edu

© 2015 Václav Rajlich
Agile and Evolutionary Software Development
Contents

• Staged model of software lifespan
• Agile and evolutionary development
• Software change
• Software maintenance
Staged model of software lifespan

- **Initial development**
  - *first version*

- **Evolution**
  - *evolutionary changes*

- **Maintenance (Servicing)**
  - *servicing patches*
  - *servicing discontinued*

- **Development**
  - *evolution stops*

- **Phase-out**
  - *switch-off*

- **Close-down**
Initial development

- Fundamental project decisions
  - Technologies, architecture, GUI design, …
- First (incomplete) version
- Phases: requirements, design, implementation
Initial development

First version

Evolutionary changes

Evolution

Development

evolution stops

- Repeated software changes

- Each change adds/modifies functionality or property of existing (and running) software
Evolutionary Software Development (ESD)

• Bulk of development is done in evolution
• Offers stakeholders regular feedback about the progress of the project
• Allows the developers to react to the volatility
  – Volatility of requirements, technology, or knowledge
Waterfall

• Bulk is done in initial development

• Little or no evolution

• Suitable for projects without volatility
  – Stable technology and domain
  – Stable knowledge → small projects
Contents

• Staged model of software lifespan

• Agile and evolutionary development (ESD)

• Software change

• Software maintenance
ESD

• Current mainstream

• There are numerous ESD processes with track record of success
  – Example: Agile
  – There are other effective ESD processes
Iterative development

• Iterations are milestones with specific goals
• Volatility is frozen during iteration
  – Allows planning of iteration
• Stakeholders assess the past iteration and plan the next one
  – Iteration meeting of all stakeholders
  – Volatilities are taken into account during iteration meeting
Agile development

• Autonomous decision makings by developers during the iterations
  – Requires intensive communication within the team
  – Very short iterations
• Suitable for small teams
  – SCRUM, eXtreme
Safeguarded development

• Presence of guardians of the code base
  – They accept or disallow code changes

• Software with high quality expectations
  – Avionics
Open source development

• Open source development
  – Safeguarded
  – Code ownership
  – Wide community of developers, variable skills

• Inner source development
  – Leverages the experience of open source
  – Practiced within a corporation
Directed development

• Default process
• Managers assign tasks to the developers
• Allows different specialized roles
  – Developers, testers, architects, technology experts, managers, …
• Suitable for large or delocalized projects
Exploratory development

• Features are establish by trial-and-error

• Common in research projects
Solo development

• Single developer

• More and more common
  – Mobile apps
Success of small projects*

• Budget of less than $1 M

• 76 % of small projects are successful
  – Only 4% of small projects are cancelled
  – Historically high success rate

*Chaos Manifesto 2013
Future directions of ESD

• Associate individual practices with specific project circumstances
• Tailor new processes that exactly fit specific project needs
## Example

<table>
<thead>
<tr>
<th>Project circumstances</th>
<th>Practice suitable for the circumstances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploratory programming is necessary</td>
<td>Developers are domain experts</td>
</tr>
<tr>
<td>Gap between programmer capability and expected quality</td>
<td>Code guardians, permission to commit</td>
</tr>
<tr>
<td>Frequent turnover of developers</td>
<td>Concept location practices</td>
</tr>
<tr>
<td>High volatility of requirements, technologies, or knowledge</td>
<td>Short initial development, short iterations</td>
</tr>
<tr>
<td>Low volatility</td>
<td>Long initial development</td>
</tr>
<tr>
<td>. . . (we need a very large table)</td>
<td>. . .</td>
</tr>
</tbody>
</table>

© 2015 Václav Rajlich           Agile and Evolutionary Software Development 19
Problem with large projects*

- Budget larger than $10 M
- Low (10%) probability of success
  - 38% of them were cancelled
  - Large financial losses
  - Embarrassment to our community
  - Something needs to be done

*Chaos Manifesto 2013
Confused debate about processes

• False dichotomy: “waterfall vs. agile”
  – Note waterfall is suitable for stable small projects
  – Agile is suitable for small teams
• Contributing to the low success rate of large projects
Better debate

• What is the appropriate size of the initial development?
  – Volatility requires small initial development
  – Stability allows large initial development

• After initial development, which ESD process is the most appropriate?
  – Which combination of ESD practices is the most appropriate?
Contents

• Staged model of software lifespan
• Agile and evolutionary development
• Software change
• Software maintenance
Phased Model of Software Change (PMSC)

- Process model for changes with limited impact
- Enactment of PMSC contains a subset of phases
Phased Model of Software Change (PMSC)

Initiation

Concept
Location

Impact Analysis

Prefactoring

Actualization

Postfactoring

Conclusion

. . . Requirements
Phased Model of Software Change (PMSC)

- Initiation
- Concept Location
- Impact Analysis
- Prefactoring
- Actualization
- Postfactoring
- Conclusion

Requirements

Finds what needs to change
Phased Model of Software Change (PMSC)

- Requirements
  - Finds what needs to change
- How big is the change?
Phased Model of Software Change (PMSC)

- Requirements
  - Finds what needs to change
- How big is the change?
  - Prepare for the change
Phased Model of Software Change (PMSC)

- Requirements
  - Finds what needs to change
- How big is the change?
  - Prepare for the change
- Do the change

© 2015 Václav Rajlich

Agile and Evolutionary Software Development
Phased Model of Software Change (PMSC)

- Requirements
  - Finds what needs to change
- How big is the change?
- Prepare for the change
- Do the change
- Clean up
Phased Model of Software Change (PMSC)

- Requirements
  - Finds what needs to change
- How big is the change?
- Prepare for the change
- Do the change
- Clean up
- Commit
Impact analysis (IA)

- IA predicts prefactoring and actualization
  - Currently unable to predict postfactoring
  - Postfactoring often resolves technical debt after it accumulates beyond acceptable limit
Incremental IA

Color codes:
- Unknown
- Changed
- To be inspected
- Inspected and unchanged
- Propagating

© 2015 Václav Rajlich
Agile and Evolutionary Software Development
Incremental IA
Incremental IA

Color codes

- Unknown
- Changed
- To be inspected
- Inspected and unchanged
- Propagating

© 2015 Václav Rajlich  Agile and Evolutionary Software Development
Incremental IA

Color codes

Unknown | Changed | To be inspected | Inspected and unchanged | Propagating

Unknown | Changed | To be inspected | Inspected and unchanged | Propagating
Incremental IA

Color codes

- Unknown
- Changed
- To be inspected
- Inspected and unchanged
- Propagating

© 2015 Václav Rajlich  Agile and Evolutionary Software Development
Incremental IA

Color codes

<table>
<thead>
<tr>
<th>Unknown</th>
<th>Changed</th>
<th>To be inspected</th>
<th>Inspected and unchanged</th>
<th>Propagating</th>
</tr>
</thead>
</table>
Incremental IA

Color codes:
- Unknown
- Changed
- To be inspected
- Inspected and unchanged
- Propagating

© 2015 Václav Rajlich        Agile and Evolutionary Software Development
Incremental IA

Color codes

- **Unknown**
- **Changed**
- **To be inspected**
- **Inspected and unchanged**
- **Propagating**
Incremental IA

Color codes
- Unknown
- Changed
- To be inspected
- Inspected and unchanged
- Propagating

© 2015 Václav Rajlich
Agile and Evolutionary Software Development
Incremental IA

Unknown | Changed | To be inspected | Inspected and unchanged | Propagating

© 2015 Václav Rajlich Agile and Evolutionary Software Development
Incremental IA

Color codes

<table>
<thead>
<tr>
<th>Unknown</th>
<th>Changed</th>
<th>To be inspected</th>
<th>Inspected and unchanged</th>
<th>Propagating</th>
</tr>
</thead>
</table>

© 2015 Václav Rajlich  Agile and Evolutionary Software Development
Incremental IA

Color codes:
- Unknown
- Changed
- To be inspected
- Inspected and unchanged
- Propagating

© 2015 Václav Rajlich
Agile and Evolutionary Software Development
http://jripples.sourceforge.net

• Eclipse plug-in
  – Java
  – 15,000 LOC
  – 150 Classes

• Keeps track of marks
  Changed, Propagating, Unchanged, Next
Research in IA

• Navigates program dependencies
• Some dependencies propagate the change while others do not
  – Heuristics and tools that identify “likely to propagate” dependencies - need further attention
  – Heuristics for determining when the incremental impact analysis is completed
  – Hidden dependencies
Seamless IDE for PMSC

• PMSC currently needs multiple uncoordinated tools
  • JRipples for concept location, impact analysis
  • Mylyn for the workflow
  • Tasktop for timing data
  • JUnit for unit tests
  • Abbot for functional tests
  • Refactoring tools
  • . . .

• Future IDE will provide seamless support
Measuring impact of PMSC

• Introducing well-defined process often improves productivity and quality

• Preliminary data
  – PMSC may improve developer productivity up to 50%!
  – Needs additional measurements

• Seamless IDE may offer additional gains
Contents

• Staged model of software lifespan
• Agile and evolutionary development
• Software change
• Software maintenance
Software maintenance

• **Maintenance ≠ evolution !!**

• Maintenance also consists of repeated changes to software
  – Objectives are drastically reduced
  – The only goal is to keep software usable in a cost-effective way
  – No new functionality

• Software enters maintenance (servicing) stage when it is transferred to users
Parallel evolution and maintenance

• Several branches
  – One is evolved into new version
  – Other branches (released versions) are just maintained
How does evolution stop:

• Managerial decision
  – Expensive evolution stops if it is no longer needed

• Stabilization
  – No longer any volatility

• Technical debt gets out of hand 😞
  – Code is no longer evolvable
  – Can happen by accident
Future of maintenance

• Accidental code decay is a serious issue

• Research:
  – Keep software evolvable, clean technical debt

• Complementary approach:
  – Extend software evolution (PMSC) to the code that is currently considered non-evolvable
Conclusions

• Software evolution was historically an unexpected and peripheral phenomenon
• Moved into the center of software development
• It gained enormous importance to both developers and researchers
Conclusion

• We covered
  – Staged model of software lifespan
    • Includes evolution
  – Evolutionary software development
    • Many processes, some suitable for large projects
  – Phased model of software change
    • For changes that have limited impact
  – Software maintenance
    • Keeping software usable, no new functionality
Book on this topic

SOFTWARE ENGINEERING
THE CURRENT PRACTICE

VÁCLAV RAJLICH