Course "Softwareprozesse"

Agile Manifesto

Lutz Prechelt
Freie Universität Berlin, Institut für Informatik

- Origin of agile
- Agile Manifesto
  - core part, principles list, background
  - common misunderstanding
  - culture
- Underlying assumptions
- Classical view on agile
  - Too much vs. too little planning
  - XP planning game
- Usage frequency of various agile methods
Learning objectives

- Understand the key ideas of the agile movement
- Be able to detect those ideas at work in processes
- Be able to detect deviations and violations of these ideas
On the origins of Agile: The Waterfall myth

It is often said that


has proposed the proper way to develop software as this:

(a plan-driven approach)
On the origins of Agile: The Waterfall myth (2)

• What it really said:
  • "In my experience, [this method] has never worked on large software development efforts"
  • "To give the contractor free rein between requirement definition and operation is inviting trouble."

• Royce recommends several additions as a repair:
  • Note: The article talks about "spacecraft mission planning, commanding and post-flight analysis" where requirements tend to be well-understood and are usually stable!
  • "preliminary design": Architectural design and its documentation
  • "plan, control and monitor testing": approach QA systematically
    • and take it very seriously
  • "do it twice": iterate; build a throw-away prototype
    • mostly to understand non-functional behavior
  • "involve the customer": get feedback
    • have the customer commit to parts of the effort before final delivery
On the origins of Agile: Waterfall practice

- Subsequent practitioners have often used the basic plan-driven waterfall process with only two of the repairs:
  - Architectural design; Systematic testing
- but missed two crucial ones:
  - **iterative development; feedback/incremental commitment**
- Furthermore, they have often overlooked a crucial element for information systems:
  - **User requirements** are needed before system requirements
    - User requirements are situated in the problem domain (⇒ value!)
    - System requirements are situated in the solution domain (HW/SW)
On the origins of Agile: Effect of the misunderstanding

- As a result, many SW projects got into huge trouble
  - massive plan overruns; sometimes no useful SW produced at all
  - especially for projects with unclear requirements
- As computers got faster, some practitioners developed better ways of developing SW
  - highly iterative; fewer documents; automated testing
- Other projects at least used iterations to reduce requirements risks
  - but many projects continued to work badly
    - presumably because too much in the classical-view culture made it hard to see what process was needed.

- Then in 2001, 17 practitioners got together and formulated their worldview in the Agile Manifesto
  - since then, the modern view has quickly gained acceptance
    - at least seemingly
Prelude: Agile is about priorities

  - Very good introduction into the agile way of thinking
- "Processes, tools, documentation, contracts, and plans are useful. But when push comes to shove — and it usually does — something must give, and we need to be clear about what stays and what gives."
  - (Agile methods were initially called "lightweight" methods, because they attempted to get rid of dispensable process elements)
The Agile Manifesto

http://www.agilemanifesto.org  (2001)

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 & interactions over processes and tools
• Working software over comprehensive documentation
• Customer collaboration over contract negotiation
• Responding to change over following a plan

    (this stays)     (this gives way)

That is, while there is value in the items on the right, we value the items on the left more."
We follow these principles:

1. Our highest priority is to satisfy the customer
   - through early and continuous delivery of valuable software.
     (Value focus, not quality focus!)

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.
[We discuss: When is this a good idea? When is it not? Elements of modern view?]

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

10. Simplicity, the art of maximizing the 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.
Use your common sense!

Believe nothing, no matter where you read it or who said it, not even if I have said it, unless it agrees with your own reason and your own common sense.

Gautama Siddharta Buddha
The still-common misunderstanding

Source: [http://agilemanifesto.org/history.html](http://agilemanifesto.org/history.html)

- "The Agile movement is not anti-methodology,
  - in fact, many of us want to restore credibility to the word methodology.
- We want to restore a balance.
  - **We embrace modeling**, but not in order to file some diagram in a dusty corporate repository.
  - **We embrace documentation**, but not hundreds of pages of never-maintained and rarely-used tomes.
  - **We plan**, but recognize the limits of planning in a turbulent environment."

- Agile Methods are about clear positions in difficult trade-offs
  - **not** about letting all discipline and process go and just hack.
The manifesto folks on culture

Source: [http://agilemanifesto.org/history.html](http://agilemanifesto.org/history.html)

- "we all felt privileged to work with a group of people who held a set of compatible values,
  - a set of values based on trust and respect for each other and promoting organizational models based on people, collaboration, and building the types of organizational communities in which we would want to work."

- "In order to succeed in the new economy, to move aggressively into the era of [...] the web,
  - companies have to rid themselves of their Dilbert manifestations of make-work and arcane policies."
  - Does anybody here not know Dilbert?

Lutz Prechelt, prechelt@inf.fu-berlin.de
Dilbert example

I’ll need a project plan to justify the resources we need to change our software.

I can make those software changes in ten seconds.

Done.

Good work. Now all we need is that plan.

"[...] in the final analysis, the meteoric rise of interest in—and sometimes tremendous criticism of—Agile Methodologies is about the mushy stuff of values and culture."
Assumptions of Agile Methods

1. You cannot foresee the development of a whole project
2. Developers are motivated, technically competent, and capable of good judgement
   - if they do not have to work in Dilbert-like work environments
3. A project can adjust to unforeseen changes
   - In particular, changes to the design are not ruinously expensive
4. Adjustment is easier if everything that need not be in writing, is not done in writing
   - Code must be in writing
   - Some documentation may, too (e.g. for certification)
   - Most other information does not.
     - And where it does, short-lived throw-away writing is often sufficient.

- Agile methods are useful if these assumptions are met,
  - they are problematic if even one of them is not.
  - The most risky assumption is that of cheap design changes
When are the assumptions true?

1. "You cannot foresee the development of a whole project"
   • Almost always true

2. "Developers are motivated, technically competent, and capable of good judgement"
   • Depends critically on available staff and organizational culture

3. "Changes to the design are not ruinously expensive"
   • True only if modern technology (such as reusable components, middleware, modules, object-orientation, development tools etc.) is used in a highly competent fashion
   • False if an "architecture breaker" occurs
     • → almost sure if you do not have a good architecture to begin with

4. "Adjustment is easier if everything that need not be in writing, is not done in writing"
   • Possible only if there is a tightly coupled, fairly stable, and preferably co-located team
   • Not possible in a process-heavy organization:
A very heavy-weight process

DOGBERT, I NEED YOU TO FACILITATE SOME MEETINGS.

WHAT KIND OF MEETINGS?

WE'RE CREATING A PROCESS TO FIX OUR PRODUCT DEVELOPMENT PROCESS. BUT FIRST WE'RE HAVING SOME PREPLANNING MEETINGS...

...TO DECIDE ON A PROJECT NAME.

HOW ABOUT "DEATH SPIRAL"?
"Agile Software Development: The Business of Innovation"

- Agile in a nutshell: "Agile methods stress two concepts:
  - the unforgiving honesty of working code and
  - the effectiveness of people working together with goodwill."

- Generative rules: "Most methodologies provide inclusive rules—all the things you could possibly do under all situations.
  - Agile methods offer generative rules—a minimum set of things you must do under all situations to generate appropriate practices for special situations."

- Feedback: "Because they are most applicable to turbulent, high-change environments, agile approaches recommend a variety of practices for constant feedback
  - on technical decisions, customer requirements, and management constraints."
More from Highsmith/Cockburn (2)

- Customer collaboration: "Using agile development methods requires close customer partnerships. [...]"
  - Poor customers result in poor systems."

- Humanist worldview: "What is new about agile methods is not the practices they use,
  - but their recognition of people as the primary drivers of project success [...]"
SW processes and value generation

- **Optimizing value-generation is not easy**
  - we must find the "right" requirements
- **Therefore, conventional (plan-driven) SW processes run a substantial risk of producing**
  - a **high overhead** (→ increased cost),
    - because they build many functions that are not important
  - perhaps even **low value**
    - because they miss or distort some important requirements

**But it is even worse:**

- Requirements (and hence the value proposition) change much faster and wider today than they did in the past
- Plan-driven processes cannot cope well with serious changes of a project's value proposition
  - due to their high initial investments in requirements and design
The classical view of Agile

- The most obvious difference is
  - much less planning (and in that context
  - fewer written specifications)

- Obviously, there can be too much planning or too little
  - How to think about what is a proper amount?
The planning spectrum

• Barry Boehm: "Get ready for agile methods, with care", IEEE Computer, January 2002

• The defining difference between agile and conventional methods is the amount of planning
  • There can be too much planning ("inch-pebble") as well as too little ("just hacking")
The planning spectrum (2)

- Risk exposure is the product of:
  - $P(L)$ (probability of loss) and
  - $S(L)$ (size of loss)

- High exposure can come from too much planning or from too little

![Diagram showing the planning spectrum](image)
The planning spectrum (3)

- The most relevant factors are dependability requirements and project size
  - More planning is useful if they are high
  - Less if they are low
The "home ground" of agile vs. plan-driven methods

Agile methods:
- Developers:
  - agile-minded, knowledgeable, co-located, collaborative
- Requirements:
  - Largely emergent, rapid change
- Architecture:
  - Designed for current req'mts
- Refactoring:
  - Inexpensive
- Size:
  - Smaller teams and products
- Primary objective:
  - Rapid value or low requirements risk

Plan-driven methods:
- Developers:
  - plan-oriented, adequate skills, access to external knowledge
- Requirements:
  - Knowable early, largely stable
- Architecture:
  - Designed for current and foreseeable requirements
- Refactoring:
  - Expensive
- Size:
  - Larger teams and products
- Primary objective:
  - High assurance of reliability
Too much planning and specification

- Specifications may remain unread if they are too detailed

https://dilbert.com/strip/1999-08-09
"Just enough" planning: The XP Planning Game

- Extreme Programming (XP) is one of several concrete agile development methods
  - It consists of a number of specific practices (see next lecture)
  - One of these is called "Planning Game" and describes the project planning method

- XP project planning occurs on three granularity levels
  - Release ~2-4 project months
  - Iteration ~1-2 project weeks
  - Task ~0.5-3 person days

- Only the current Release, Iteration, and set of Tasks are planned in some detail
  - all future Releases and Iterations (even within the current release) are at most sketched, perhaps not even that
XP Planning Game:
1. Release planning

- 1. Customer enumerates (rough) requirements
  - Each is written on a *Story Card*
  - Cards are collected

- 2. Customer prioritizes stories into E (essential), V (valuable), and N (nice-to-have)

- 3. Developers query Customer to obtain sufficient detail about stories to understand their content and purpose
  - "conversation"

- 4. Developers estimate development cost for each story
  - and categorize their estimates into R (reliable), A (approximate), and U (unknown)

- 5. Customer selects stories for next release, preferring E and V
  - Remaining stories are to be realized in subsequent releases
XP Planning Game: 2. Iteration and task planning

- Before each iteration, the customer can select stories for it
  - Priorities may have shifted
  - Estimation may now be more precise
  - Customer is even allowed to bring in new requirements
    - but must drop (for this release) others of the same weight
- Customer defines acceptance tests for selected stories
  - "confirmation"
- and clarifies remaining details about the stories
  - "conversation"

- Developers turn the set of stories (requirements) into a set of tasks (design and implementation work)
  - A developer adopts a task and then must personally estimate its size
XP Planning Game:
3. Plan tracking and replanning

- Tracking the release-level plan
  - after each iteration, compare expected and actual progress
    → possibly modify release content

- Tracking the iteration-level plan
  - developers report completed tasks daily
    → possibly modify iteration content

- Estimation is always done in terms of "ideal development time"
  - i.e., programming only, without interruptions or additional tasks.
  - After each iteration, a "load factor" is computed for each developer, relating ideal time to elapsed time
  - and is used during the next iteration planning

- Planning focus is always on stories
  - because they represent customer value
Which processes used how often?
State of Agile survey [digital.ai21]

- yearly survey, ~4000 respondents
  - ~40% North Am., ~40% Europe
  - ~50% team members, ~50% managers, consultants etc.

results in 2021
Summary

- Agile approaches expect change and attempt to accommodate it as much as possible
- They attempt to avoid process elements that make change expensive
  - and replace them by something that makes change cheaper

- Agile approaches are most suitable when
  - team sizes are small,
  - dependability requirements are modest, and
  - requirements change is common

- Agile processes apply a very different planning style
  - coarser
    - postponing much detail-clarification to implementation time
  - much shorter-term
  - priority is on ensuring value, not finishing on time
Thank you!
Using Risk for Balancing Agile vs. Plan-Driven


Basic idea:
- Agile methods are susceptible to different risks than plan-driven methods
- Analyzing risk types and strengths helps deciding which method (or balance of methods) to use

Approach:
- Classify risks into
  - environmental risks (unavoidable)
  - agility-oriented risks (reducible by planning)
  - plan-driven risks (reducible by agility)
Using Risk for Balancing…: Examples

- Characteristics of 3 imaginary example projects/systems:

<table>
<thead>
<tr>
<th>Application</th>
<th>Team size</th>
<th>Team type</th>
<th>Failure risks</th>
<th>Clients</th>
<th>Requirements</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event planning</td>
<td>5</td>
<td>In-house venture startup, colocated</td>
<td>Venture capital, manual effort</td>
<td>Single, colocated, representative</td>
<td>Goals generally known, details emergent</td>
<td>Provided by single COTS package</td>
</tr>
<tr>
<td>Supply-chain management</td>
<td>50</td>
<td>Distributed, often multiorganization</td>
<td>Major business losses</td>
<td>Multiple success-critical stakeholders</td>
<td>Some parts relatively stable, others volatile, emergent</td>
<td>Provided by small number of COTS packages</td>
</tr>
<tr>
<td>National crisis management</td>
<td>500</td>
<td>Highly distributed, multiorganization</td>
<td>Large loss of life</td>
<td>Many success-critical stakeholders</td>
<td>Some parts relatively stable, others volatile, emergent</td>
<td>System of systems, many COTS packages</td>
</tr>
</tbody>
</table>
## Using Risk for Balancing...:
Examples risk assessment

<table>
<thead>
<tr>
<th>Risk Items</th>
<th>Event Managers</th>
<th>SupplyChain.com</th>
<th>NISCM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E-Tech</strong>: Technology uncertainties.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>E-Coord</strong>: Many stakeholders.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>E-Cmplx</strong>: Complex system of systems.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A-Scale</strong>: Scalability.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A-Yagni</strong>: Use of simple design.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A-Churn</strong>: Personnel turnover.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A-Skill</strong>: Not enough people skilled in agile methods.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>P-Change</strong>: Rapid change.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>P-Speed</strong>: Need for rapid results.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>P-Emerge</strong>: Emergent requirements.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>P-Skill</strong>: Not enough people skilled in plan-driven methods.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Risk Rating Scale
- **Minimal risk**
- **Moderate risk**
- **Serious but manageable risk**
- **Very serious but manageable risk**
- **Showstopper risk**
Using Risk for Balancing...:
Decision schema

- High risk uncertainty → "buy" information

Step 1. Risk analysis
Rate the project’s environmental, agility-oriented, and plan-driven risks

Uncertain about ratings?
- Yes: Buy information via prototyping, data collection, and analysis
- No: Continue

Step 2. Risk comparison
Compare the agile and plan-driven risks

- Agility risks dominate: Go risk-based agile
- Plan-driven risks dominate: Go risk-based plan-driven
- Neither dominate: Architect application to encapsulate agile parts

Step 3. Architecture analysis

now tailor, execute, and monitor
Using Risk for Balancing…:
Levels of development staff

### Table C. Levels of software method understanding and use.

<table>
<thead>
<tr>
<th>Level Characteristics</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Able to revise a method, breaking its rules to fit an unprecedented new situation.</td>
<td>3</td>
</tr>
<tr>
<td>Able to tailor a method to fit a precededented new situation.</td>
<td>2</td>
</tr>
<tr>
<td>With training, able to perform discretionary method steps such as sizing stories to</td>
<td>1A</td>
</tr>
<tr>
<td>fit increments, composing patterns, compound refactoring, or complex COTS integration. With experience, can become Level 2.</td>
<td></td>
</tr>
<tr>
<td>With training, able to perform procedural method steps such as coding a simple method,</td>
<td>1B</td>
</tr>
<tr>
<td>simple refactoring, following coding standards and CM procedures, or running tests.</td>
<td></td>
</tr>
<tr>
<td>With experience, can master some Level 1A skills.</td>
<td></td>
</tr>
<tr>
<td>May have technical skills, but unable or unwilling to collaborate or follow shared</td>
<td>-1</td>
</tr>
<tr>
<td>methods.</td>
<td></td>
</tr>
</tbody>
</table>

A 5:1 ratio of Level 1A to Level 2 staff can be OK for agile methods, but 1B get in the way.

Plan-driven can function with many 1B.

-1 are always a problem.
Using Risk for Balancing...: Risks and risk levels

[Diagram showing a 3D graph with axes labeled Personnel, Criticality (Loss due to impact of defects), Dynamism (Percent requirements change/month), Size (Number of personnel), Comfort, and Culture (Percent thriving on chaos versus order). Arrows indicate the movement between Plan-driven and Agile styles.]