Course "Spezielle Themen der Softwaretechnik"

Agile Methods

Lutz Prechelt
Freie Universität Berlin, Institut für Informatik
http://www.inf.fu-berlin.de/inst/ag-se/

- Goals and priorities in SW development
- Principles of agile approaches
  - Misunderstandings about them
- Assumptions of agile methods
- Balancing risk:
  - Too much vs. too little planning
    - The XP planning game
  - Other
- CMMI process areas in agile methods
Learning objectives

- Understand the basic ideas of agile methods
- Contrast them to "conventional" approaches
- Understand the best-fit regions for application of agile methods
- Understand risk as a driver for agility decisions
SW processes and value generation

- As we saw in the lecture on SW engineering economics, optimizing value-generation is not easy
- Therefore, conventional (plan-driven) SW processes run a substantial risk of producing
  - a high overhead (→ increased cost),
  - perhaps even low value

- 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
Purpose of software processes

One possible view (but more facets exist):

- **Plan-driven:**
  1. Anticipate development as much as possible in order to eliminate the cost of changes
  2. Generate high quality at low cost

- **Agile:**
  1. Find ways to reduce the cost of change because eliminating change is impossible
  2. Generate high value at low cost and high speed

  - Very good introduction into the agile way of thinking
Agile is about priorities

Highsmith and Cockburn:

- "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 attempt to get rid of dispensable process elements

- These priorities were spelled out in the oft-misunderstood Agile Manifesto
  - published by a group of 17 high-profile members of the agile process community
  - all of them are practitioners who really know what they are talking about
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."
Agile Manifesto: Principles

*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. **Working software is the primary measure of progress.**  
   - Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

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

4. **The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.**  
   - In particular, business people and developers must work together daily throughout the project.
5. Agile processes promote sustainable development.
   • The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

6. Build projects around motivated individuals.
   • Give them the environment and support they need, and trust them to get the job done.
   • Continuous attention to technical excellence and good design enhances agility.
   • The best architectures, requirements, and designs emerge from self-organizing teams.
   • At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

7. Simplicity is essential.
   • Simplicity is the art of maximizing the amount of work not done.
The common misunderstanding

Source: http://www.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 is about clear positions in really hard trade-offs
  • NOT about letting all discipline and process go and just hack.
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
   - User documentation may, too (Depends!)
   - 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 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
The planning spectrum


- 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 relationship between planning and risk exposure with labels for execution-related and planning-related exposure, and a sweet spot indicating optimal risk exposure.]
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, 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

Your requirements document is the biggest I've ever seen.

It's too big to read, but I can guess from its weight what must be in there.

You know it's a multi-user, global system, right?

No, I'm not getting that.

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.

Good work. Now all we need is that plan.

Done.
Too much planning (2)

---

THE TECHNOLOGY DEMO

THE SOFTWARE ISN'T 100% COMPLETE.

IF IT HAD A USER INTERFACE YOU WOULD SEE SOMETHING HERE... HERE... AND SOMETIMES HERE.

AND THEN YOU'D BE SAYING, "I GOTTA GET ME SOME OF THAT."

ANY QUESTIONS?
"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-within-release are at most sketched, perhaps not even that
XP Planning Game: Release planning

- Customer enumerates (rough) requirements
  - Each is written on a *Story Card*
  - Cards are collected
- Customer prioritizes stories into E (essential), V (valuable), and N (nice-to-have)
- Developers query Customer to obtain sufficient detail about stories to understand their content and purpose
  - "conversation"
- Developers estimate development cost for each story
  - and categorize their estimates into R (reliable), A (approximate), and U (unknown)
- Customer selects stories for next release, preferring E and V
  - Remaining stories are to be realized in subsequent releases
XP Planning Game: 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: 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
  - informally ask developers for progress every ~2 days or so → 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
Other elements of agile processes

Besides lean planning, there are other typical behaviors that help with agility:

- **Documentation:**
  - Whenever you can, rely on oral communication instead of written information
    - Rationale 1: Talking is so much cheaper than writing that you can even tolerate some redundance; in the presence of change, writing costs can probably never pay off
    - Rationale 2: Oral communication is more flexible, understanding is easier to obtain
  - If you have to use writing for clarity, consider throwing it away as soon as understanding has been achieved
    - Rationale: In the presence of change, it is too costly to keep the documentation up-to-date; out-of-date documents are dangerous.

- **Automation:**
  - Use automation wherever possible to remove barriers against change (in particular: build process; testing)
Other elements of agile processes (2)

• **Meetings** and communication:
  • Keep meetings with multiple participants as short as possible
    • Rationale: Long meetings are inefficient
  • Rely a lot on one-to-one discussions with changing partners
    • Rationale: Can replace many meetings; avoids meetings' tendency to get lengthy; improves motivation and initiative
  • Maximize the opportunity for informal communication
    • Rationale: Reduces the need for formal meetings as risk-reduction mechanisms

• Early and intensive **feedback**:
  • Reduce the uncertainty about the quality of requirements (and implementation) by frequent releases
  • Reduce the uncertainty about the consistency of design and implementation by automated builds and tests

• **Simplicity**:
  • Avoid all complexity that is not very clearly warranted
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
  - agility-oriented risks
  - plan-driven risks
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,</td>
<td>Venture capital, manual</td>
<td>Single, colocated, representative</td>
<td>Goals generally known, details emergent</td>
<td>Provided by single COTS package</td>
</tr>
<tr>
<td></td>
<td></td>
<td>colocated</td>
<td>effort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supply-chain management</td>
<td>50</td>
<td>Distributed, often</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></td>
<td></td>
<td>multiorganization</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>National crisis management</td>
<td>500</td>
<td>Highly distributed,</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>
<tr>
<td></td>
<td></td>
<td>multiorganization</td>
<td></td>
<td></td>
<td></td>
<td></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>Environmental risks</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E-Tech: Technology uncertainties.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E-Coord: Many stakeholders.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E-Cmplx: Complex system of systems.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Risks of using agile methods</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A-Scale: Scalability.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A-Yagni: Use of simple design.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A-Churn: Personnel turnover.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A-Skill: Not enough people skilled in agile methods.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Risks of using plan-driven methods</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P-Change: Rapid change.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P-Speed: Need for rapid results.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P-Emerge: Emergent requirements.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P-Skill: Not enough people skilled in plan-driven methods.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using Risk for Balancing...: Decision schema

- High risk uncertainty $\rightarrow$ "buy" information

Diagram details:
1. **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: Go to next step

2. **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
     - Go risk-based agile for agile parts; go risk-based plan-driven elsewhere

3. **Step 3: Architecture analysis**
   - now tailor, execute, and monitor
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 the relationship between different factors such as criticality, dynamism, comfort, personnel, size, culture, and their risk levels.
CMMI process areas in agile methods

- **Level 2: Managed**
  - + Requirements Mgmt
  - o Project Planning
  - + Project Monitoring&Control
  - - Supplier Agreement Mgmt
  - o Measurement and Analysis
  - o Process and Product Quality Assurance
  - + Configuration Management

- **Level 3: Defined**
  - + Req's. Development
  - + Technical Solution
  - o Product Integration
  - - Verification
  - + Validation
  - - Organizational Process Focus
  - - Organ'l Process Definition

- **Level 4: Quantitatively Manag'd**
  - - Organizational Training
  - o Integrated Project Mgmt.
  - o Risk Management
  - o Decision Analysis and Resolution

- **Level 5: Optimizing**
  - - Organizational Process Performance
  - o Quantitative Project Mgmt
  - o Organizational Innovation and Deployment
  - o Causal Analysis and Resolution

+ usually available
o avail. in reduced form
- usually mostly absent
Further resources

- http://www.agilealliance.com
  - A community portal around the agile approach. Has pointers to most everything about it.
- http://alistair.cockburn.us/
  - Alistair Cockburn (one of the best writers-of-agile), Crystal
- http://www.controlchaos.com/
  - Ken Schwaber, SCRUM
- http://www.adaptivesd.com/
  - Jim Hightsmith, Adaptive SW Development
- http://www.xprogramming.com
  - Ron Jeffries, XP

- http://www.cio.com/article/print/464169
  - "When Agile Projects Go Bad": A good article about common mistakes when going agile

- IEEE Computer, June 2003
  - Issue topic "Agile Methods" (7 articles about when and how)
  - http://www.computer.org
A quote

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
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 cheaper.

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

- A disciplined agile process can cover many of the CMMI process areas up to level 5 however in rather unconventional ways.
Thank you!