13. Team evolutionary development

• Most of the software projects require a larger effort than a solo programmer can handle

• Programmers have to organize themselves into teams

• Agile teams

• Directed teams
Agile Iterative Process (AIP)

• Agile development process for small-to-medium-sized teams
• Decisions made by consensus
• No specializations among the programmers
  – Developers are the only programmer role
Model of AIP

- Users request changes, which are added to the product backlog.
- The product backlog is managed by the Product Manager.
- The iteration backlog is managed by the Process Manager.
- Programmers work in parallel, making software changes.
- Changes are built and discussed in daily meetings.
- The iteration meeting/release process concludes the cycle.
Iterations

• Iteration meeting
• Assessing current state of the product
  – all stakeholders participate
  – technical and business point of view
• Planning the next iteration
  – iteration backlog are the changes to be done
    in the next iteration
  – extracted from product backlog
Daily meeting

• Daily problems and challenges
• Programmers build a consensus about the progress
  – discuss the tasks and problems at hand
  – conflicts between the code commits
• Daily assignments of change requests
• Clarify the ambiguities
• Needs for code refactoring
Daily meeting, cont.

• Early warning when anything goes wrong
  – problems with build

• Meetings are short
  – the recommended duration is 15 minutes
  – may be attended by other stakeholders

• After the daily meeting is concluded, the programmers resume their individual work
  – the software changes
Software changes

• Done by programmers in parallel

• Conflict of commits must be resolved
“Agile manifesto”

- Developed in 2001 (17 original authors)
- Signed by numerous people since

<table>
<thead>
<tr>
<th>Individuals and interactions</th>
<th>over processes and tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working software</td>
<td>over comprehensive documentation</td>
</tr>
<tr>
<td>Customer collaboration</td>
<td>over contract negotiation</td>
</tr>
<tr>
<td>Responding to change</td>
<td>over following a plan</td>
</tr>
</tbody>
</table>
Scrum: Example of AIP

• 1995:
  – Scrum by Jeff Sutherland & Ken Schwaber

• 1996:
  – introduction of Scrum at OOPSLA conference

• 2001:
  – textbook “Agile Software Development with Scrum” by Ken Schwaber & Mike Beedle
  – Successful use of Scrum in 50 companies

• Since 2001
  – Wide use of Scrum
Scrum vs. Sprint in Rugby
Chickens and Pigs

Hey Pig, I was thinkin’ we should open a restaurant.

I don’t know. What would we call it?

How about “Ham-n-Eggs.”

No thanks, I’d be committed, but you’d only be involved!

By Clark & Vizdos
Product manager (owner)

- Define the features of the product
- Decide on release date and content
- Be responsible for the profitability of the product (ROI)
- Prioritize features according to market value
- Adjust features and priority every iteration
- Accept or reject work results.
Scrum master
(aka process manager)

• Enacts Scrum values and practices
• Removes impediments
• Ensures that the team is fully functional and productive
• Enables close cooperation across all roles and functions
• Shields the team from external interferences
Scrum team

• Typically 5-10 people
• Cross-functional
  – QA, Programmers, UI Designers, etc.
• Members should be full-time
  – exceptions: System Admin, etc.
• Teams are self-organizing
  – ideally, no titles
• Membership can change only between sprints
Sprint Review

• 4 hours informational meeting
• Team presents accomplishments of the sprint
• Demo of new features or underlying architecture
• Informal
  – 2-hour preparation time rule
• Participants
  – customers
  – management
  – product owner
  – Scrum team
Sprint planning meeting

- Participants
  - customers
  - management
  - product owner
  - Scrum team
- Determine the next Sprint goal
- Sprint backlog is created
Product backlog

• A queue of business and technical functionalities that need to be developed
• Requirements come from stakeholders
  – users
  – customers
  – sales
  – marketing
  – customer service
  – programmers
Sprint backlog

• A selection of tasks and an estimated effort
• Created by stakeholders
• No more than 300 tasks in the list
• If a task requires more than 16 hours, it should be broken into parts
• Team becomes better at Sprint planning after 3rd or 4th Sprint
# Example of sprint backlog

<table>
<thead>
<tr>
<th>Story/task</th>
<th>days in sprint / effort left</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fetch one day temperature data from the weather provider system</strong></td>
<td></td>
</tr>
<tr>
<td>Connect and authenticate server</td>
<td>4 16 12 8 3 3 3 3 3 3 3 3 3</td>
</tr>
<tr>
<td>Read provider's data directory</td>
<td>8 7 7 7 4 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Parse the current temperature out of the data</td>
<td>6 6 4 4 4 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>Push the temperature data to the client</td>
<td>16 16 16 16 16 16 8 2 0 0 0</td>
</tr>
<tr>
<td><strong>Fetch rain, snow, etc details from the provider</strong></td>
<td></td>
</tr>
<tr>
<td>Parse snow/rain data from the provider's data</td>
<td>4 4 4 4 4 4 4 4 0 0 0 0 0</td>
</tr>
<tr>
<td>Push the snow/rain data to the client</td>
<td>4 4 4 4 4 4 4 4 4 4 4 0 0</td>
</tr>
<tr>
<td>Redesign client screen a bit</td>
<td>3 3 3 3 3 3 3 3 3 3 3 3 3</td>
</tr>
<tr>
<td>Refactor the server code</td>
<td>4 4 4 4 4 4 4 4 4 4 4 4 4</td>
</tr>
<tr>
<td><strong>Fetch several days data from the provider</strong></td>
<td></td>
</tr>
<tr>
<td>Parse the weather data in day packs</td>
<td>10 10 10 10 10 10 10 10 10 10</td>
</tr>
<tr>
<td>Push several days data to the client</td>
<td>3 3 3 3 3 3 3 3 3 3 3 3 3</td>
</tr>
<tr>
<td><strong>Auto-refresh feature</strong></td>
<td></td>
</tr>
<tr>
<td>Make the client ping server once per 4 hours</td>
<td>6 6 6 6 6 6 6 6 6 6 6 6 6</td>
</tr>
<tr>
<td>Make the server update the client</td>
<td>2 2 2 2 2 2 2 2 2 2 2 2 2</td>
</tr>
</tbody>
</table>
Backlog graph

- Depicts the number of hours of work remaining until the end of sprint
- Ideally should burn down to zero to the end of the Sprint
- In reality is not a straight line
  - additional work might be required
  - some of the work might be removed because of a bad estimation
Example of backlog graph
Sprint (aka Iteration)

- No inference, no intruders, no peddlers
- All the work is measured and empirically controlled
- The progress is measured through daily product builds
- A product increment is delivered at the end of every sprint
Daily Scrum Meeting

• Format
  – 15-minutes, stand-up
• Chickens and pigs are invited
  – help avoid other unnecessary meetings
  – only pigs can talk
• Three questions:
  1. What did you do yesterday
  2. What will you do today?
  3. What obstacles are in your way?
Daily Scrum Meeting (cont.)

- Is NOT a problem solving session
- Is NOT a way to collect information about who is behind the schedule
- Is a meeting in which team members make commitments to each other and to the Scrum Master
- Is a good way for a Scrum Master to track the progress of the Team
Scrum improves engineering practices

- Improves productivity
  - improve code quality

- If the team doesn’t report any problems within the daily meeting then there is a problem
  - daily builds ALWAYS have problems
  - maybe the daily build is absent – DANGER!!!
  - daily build ensures that the team moves forward
Example of Unpredictable Activity

• Corporate NewsPage (CNP)
  – team is under pressure to make a new release
  – team was porting the system from Sun to HP and IBM platforms
  – HP and IBM released new operating systems
  – technology has changed
    • complexity had skyrocketed
    • change the operating systems in addition to developing new functionality for the release
Example, cont.

- The team adopted Scrum
  - team stopped and reevaluated the priorities
  - sales had not yet sold any sites that would use IBM technology and had to sold only one HP installation
  - team decided to give a low priority to porting the system to HP and IBM
Why does Scrum work?

• Risk of not pleasing customer
  – Scrum allows the customer to see the product on a constant basis

• Risk of not completing all functionalities
  – all the high priority functionalities will be delivered
  – only lower priority functionalities are missed

• Risk of poor estimating and planning
  – daily meetings provide estimates
  – plan is adjusted through Sprint Review and Sprint Planning meeting
Why does Scrum work?

• Risk of not resolving issues promptly
  – through daily scrum the managers control these issues

• Risk of not being able to complete development cycle
  – working version is delivered after every sprint

• Risk of taking too much work and changing expectations
  – Scrum does not allow changing product backlog associated with a sprint
Tacit vs. Explicit knowledge

• Explicit knowledge
  – included in source code, documentation, UML diagrams

• Tacit knowledge
  – programmers experience, their intuition, which cannot be externalized
  – tacit knowledge can be lost
Scrum and the organization

• Scrum allows the organization to detect impediments

• Example of impediments
  – person responsible to purchase software is too busy
  – an engineer was using two workstations because the 15” screen was too small
  – free coffee was not available to programmers, programmers were losing time searching for coins
Scrum Values

• Commitment
  – Scrum provide people all the authority they need to meet their commitments

• Focus
  – focus all your efforts and skills to the work you’ve committed to

• Openness
  – Scrum keeps everything about a project visible to everyone
Scrum Values, cont.

• Respect
  – it is important to respect the people who comprise a team

• Courage
  – have the courage to commit, to be open, and to expect respect
Conclusions Scrum

- Scrum is an agile process
- Scrum has clearly defined rules
- Scrum increases the productivity of a team
- Scrum improves team communication
- Scrum can improve the organization of the company
- Scrum works only if developers adopt Scrum values
Extreme programming (XP)

- Variant of agile
- Takes commonsense practices to extreme levels
XP

- if code reviews are good, review code all the time (pairs)
- if testing is good, test all the time
- if simplicity is good, design with the simplest design that supports its current functionality
- if architecture is important, everybody works on defining and refining the architecture all the time
- if integration testing is important, integrate and test several times a day
- if short iterations are good, make the iterations very short
12 Practices

1. The planning game (scope of the next release).
2. Small releases – simple system into production quickly
3. Metaphor – simple shared story of how the system works
4. Simple design – design simply, remove complexity
5. Testing – by developers and customers
6. Refactoring
7. Pair programming
8. Collective ownership
9. Continuous integration
10. On-site customer
11. Coding standards – communication through code
12. 40-hour week
1. The planning game

- Business decisions
  - scope: which “stories” should be developed
  - priority of stories
  - composition of releases
  - release dates

- Technical decisions
  - time estimates for features/stories
  - elaborate consequences of business decisions
  - team organization and process
  - scheduling
2. Small releases

• Put system into production ASAP
  – fast feedback
• Deliver valuable features first
• Short cycle time
  – planning 1-2 months is easier than planning 6-12 months
• Releases should be
  – as small as possible
  – containing the most valuable business requirements
  – "coherent" (you can't release just for the sake of releasing)
3. Metaphor

- Intuitive overall idea of the system
  - for example, the ATM, the contract
  - metaphor as shared verbal vision of architecture
    - architecture is boxes and connection
    - metaphor is holistic, and can be communicated

- How does the whole system work?
4. Simple design

• The “right” design
  – no code duplication
  – fewest possible classes and methods
  – fulfills all *current* business requirements
  – design for today not the future
5. Testing

• Write tests before production code
  – unit tests $\rightarrow$ developer
  – feature/acceptance tests $\rightarrow$ customer

• Strong emphasis on regression testing
  – unit tests need to execute all the time
  – tests for completed features need to execute all the time

• Unit tests pass 100%
6. Refactoring

• Goal: Keep design simple
  – change bad design when you find it
• Examples:
  – remove duplicate code
  – remove unused code
• Refactoring requires good unit tests and functional tests
7. Pair programming

“All production code is written with two people looking at one machine”
- Person 1: Implements the method
- Person 2: Thinks strategically about potential improvements, test cases, issues

Pairs change all the time

Advantages
- no single expert on any part of the system
- training on the job
- permanent inspections

Problems:
- wasted development time?
- pairs need to function
8. Collective Ownership

• Nobody owns code
• Nobody owns design
• Everybody takes responsibility for the whole system
  – anybody can change or improve anything at any time
  – you can't know what's broken or can be improved unless you have the big picture in your mind
9. Continuous Integration

- Short development cycle with integration at the end of every cycle
- Integration happens after a few hours of development
  - Code is released into current baseline on integration machine
  - All tests are run
  - In case of errors:
    - Revert to old version
    - Fix problems
10. On-site customer

- Many software projects fail because they do not deliver software that meets business needs
- Real customer has to be part of the team
  - defines business needs
  - answers questions and resolves issues
  - prioritizes features
11. Coding standards

• Team has to adopt a coding standard
  – makes it easier to understand other people’s code
  – avoids code changes because of syntactic preferences
12. 40 hour week

- Programming is a hard work
- Reasonable amount of time needed for the rest
  - excessive hours on a regular basis are counterproductive
  - they lead to a drop in productivity
Summary XP

• XP is set of "practices"
• Any one practice doesn't stand on its own. They require the other practices to keep them in balance.
• For example, simple design can't work unless you
  – have a shared vision of the design
  – have the big picture in mind
Directed Iterative Process (DIP)

• Process runs under direction of managers
• Several different specialized roles for the programmers
• The process scales to large teams and large systems
Model of DIP

Product backlog

Iteration backlog

Developers

Parallel software changes

Testers

Build

Iteration review/release

Process managers

Users

Product manager

© 2012 Václav Rajlich  Software Engineering: The Current Practice  Ch. 13
The Roles

• Developers
  – produce code

• Testers
  – accept/reject developer’s commits
  – test and certify new baseline

• There can be additional specialized roles
  – specialized technologies
  – specialized tasks

• Specialization increases effectiveness
Management

- Product managers
  - make strategic decisions
    - resolve the conflicts
    - prioritize tasks
    - direct programmers
  - Guarantee the coordination of the effort
- Process managers
  - assign tasks
  - control the process
Safeguarded Development

- Users
  - requests
  - product backlog
  - Developers
    - parallel software changes
    - permission to commit
  - Architects and Code owners
  - permission to commit
  - Process Manager
    - release

- Product Manager
  - build
  - Testers
Architect

- Guarantees that developers preserve software architecture constrains
- Approves/disapproves commits
Code ownership

• Programmers specialize in certain parts of the code
  – the “owner” must agree to the changes
  – can reject a commit
• Coordination can become a problem
  – some information may not reach other team members
Additional roles

• Quality manager
  – tracks quality data
• Support personnel
• . . .
Variants of safeguarded development

- Open source
- Inner source
- Software with very high quality expectations