Communication

If you haven't yet ...

- Sign up to mailing list
- Join Slack, teaching team is available
- All links are on the course website
- Slides are uploaded there too
Next Weeks’ Schedule

Previous weeks
- Introduction lectures
- Testing

Week Nov 5 – Nov 9
- Finish intro exercise
- Finalize teams + meeting times
- POs: Customer meeting!
  - Write initial user stories

Nov 9 Lecture
- 1st slot: Scrum LEGO Exercise!
  - Room D.E-9/10
- 2nd slot:
  - Kick-off
  - Present vision
  - Start of project

November 15, 2018
1. The Case for Agile
2. The Scrum Process
3. Scaling Scrum
How Traditional Projects Fail

- Delivering late
- Delivering over budget
- Delivering the wrong thing
- Unstable in production
- Costly to maintain
Why Traditional Projects Fail

- Smart people trying to do good work
- Stakeholders are well intended

Process in traditional projects

- Planning
- Analysis
- Design
- Code
- Test
- Deploy

- Much effort for
  - Documents for formalized hand-offs
  - Templates
  - Review committees
"The later we find a defect, the more expensive it is to fix it!"

Does front-loading a software development process make sense?

Reality shows:

- Project plans are wonderful
- Adjustments & assumptions are made during analysis, design, code
- Re-planning takes place
- Example: Testing phase at the end
  - Tester raises a defect
  - Programmer claims he followed the specification
  - Architect blames business analyst etc.
  - Exponential cost
Why Traditional Projects Fail

- People are afraid of making changes
- Unofficial changes are carried out
- Documents get out of sync
- ...

Again, why do we do that!?

To minimize the risk of finding a defect too late...
A Self-Fulfilling Prophecy

- We conduct the front-loaded process to minimize exponential costs of change
  - Project plan
  - Requirements specification
  - High-level design documents
  - Low-level design documents
- This process causes the exponential costs of change!
  ➔ A self-fulfilling prophecy

*This makes sense for a bridge, ship, or a building*

*but software (and Lego) are easy to change!*
The Agile Manifesto

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

That is, while there is value in the items on the right, we value the items on the left more.  

[http://agilemanifesto.org/](http://agilemanifesto.org/)
How Agile Methods Address Project Risks

No longer late or over budget
- Tiny iterations
- Easy to calculate budget
- High-priority requirements first

No longer delivering the wrong thing
- Strong stakeholder communication
- Short feedback cycles
How Agile Methods Address Project Risks

No longer unstable in production
- Delivering each iteration
- High degree of automation

No longer costly to maintain
- Maintenance mode starting with Sprint 2
- Maintenance of multiple versions during development
The Cost of Going Agile

Outcome-based planning
- No complete detailed project plan

Streaming requirements
- A new requirements process

Evolving design
- No complete upfront design → flexibility required
- Emergent Design

Changing existing code
- Need for refactoring
The Cost of Going Agile

Frequent code integration
- Continuous integration

Continual regression testing
- Add nth feature; test n-1 features

Frequent production releases
- Organizational challenges

Co-located team
- Easy communication, keep momentum
Discuss!

Pros and Cons

■ Short planning horizon
■ No up-front design
■ Stories instead of requirement documents
■ Extreme ideology
1. The Case for Agile
2. The Scrum Process
3. Scaling Scrum
Scrum

Product Backlog

Sprint Backlog

Sprint

Working increment of the software

Daily Scrum

24 h

2 weeks – 1 month

Review/Retrospective

Planning

Team

Product Owner

Scrum Master

Product Owner

Team

Scrum Master
The Team

Customer → Product Owner → Scrum Master → Management

Developers
Product Owner

Responsibilities

- **Customer** communication
  - Contact person for team
- Product Backlog
  - **User Stories**
  - Priorities
- Acceptance Criteria & Tests
Scrum Master

Responsibilities

- **Process** manager
  - Moderator in meetings
- Management communication
  - Remove *impediments*
- Enabler, not boss
Developers

Responsibilities

■ Communication
  □ **Critically** discuss all inputs
  □ Honestly share important information
  □ Represent team as expert

■ Sprint Backlog

■ Developing ;(-)
List of work items

- Requirements (modification requests)
  - Features
  - Bug fixes
- Ordered/prioritized
In Scrum, requirements are often defined as user stories:

“As <role>, I want <feature> to <reason>”

Requirements need to fulfill INVEST properties:

- I – Independent
- N – Negotiable
- V – Valuable
- E – Estimable
- S – Small
- T – Testable

Filling the sprint
- Estimate Backlog items
- Move items from Product to **Sprint Backlog**

Defining the work
- **Break down** Backlog items into tasks
- PO not required

Total time: 2 hours per week of sprint
For better planning, stories are broken down into tasks

Tasks should be **SMART:**
- **S** – Specific
- **M** – Measurable
- **A** – Achievable
- **R** – Relevant
- **T** – Time-boxed

List of tasks for a sprint

- Tasks are **signed-up** for, not assigned
- During the sprint
  - No new features
  - Team may change/add tasks
Daily Scrum Meeting

Status update
- Last achievements
- Next steps
- Problems

Max. **2 min** per person

Discussions?
- Schedule **subsequent** expert’s meeting
Acceptance of Features
- Demo to PO
  - PO should be prepared
  - Optional: invite other stakeholders
- Comments by developers
Retrospective Meeting

Internal team evaluation
- PO not required
- Discuss process and problems
- **Measure** improvements
Product Increment

Potentially shippable increment

- Complete according to **Definition of Done**
  - Even if not actually released
- **No regrets** if project ended now
Scrum

Team
- Product Owner
- Scrum Master
- Developers

Meetings
- Planning
- Daily Scrum
- Review
- Retrospective

Artifacts
- Product Backlog
- Sprint Backlog
- User Stories
- Software Increment
Effort, Schedule, and Cost Estimation

- Depends on software engineering process
- Highly uncertain, must be negotiated and revised with stakeholders
- Waterfall effort estimation
  - Methods: calibrated estimation model based on historical size
    (Function Points, LOC, ...); expert judgment; ...
  - Output: X man-months
- Agile effort estimation
  - Iterative methods, shorter planning horizon
  - Output: functionality to be implemented in the next iteration
  - Different methods exist
Effort Estimation: “Planning Poker”

Participants
- Everyone operationally involved in creating the software product
- Product Owner (and Scrum Master) are not playing

Preconditions
- Product backlog is complete and prioritized
- Backlog items are known by the team
- The effort for a small backlog item was determined as a reference
- Every participant has a set of sizing cards
Product owner explains backlog item
Product owner answers questions of team members
Participants estimate complexity of item and choose a card (hidden)
All cards shown simultaneously
Participants with highest and lowest number explain choices
The arguments are discussed in the group
A new vote is conducted
- **Team agrees** on item size
  - Most occurring or average value is acceptable
  - If not, another round is played
- The moderator notes size of backlog item in the product backlog
- The game ends if all backlog items are sized or **time is over**
Effort Estimation: “Affinity Estimation”

- **Participants**
  - *Everyone* operationally involved in creating the software product
  - Product Owner (and Scrum Master) are not participating, but are present for questions

- **Preconditions**
  - Product backlog is complete, *prioritized* and understood
  - A shared space to work in
  - User Stories in physical form (e.g. post-it notes or printed)
Affinity Estimation 1/2

**Step 1: Silent Relative Sizing**
- Team members place backlog items on scale of “smaller” to “larger”
- No discussion at this point

[Image of a scale with categories: Smaller, Larger, Tiny items, Small items, Bigger items, Giant items]
Affinity Estimation 2/2

- **Step 2: Editing**
  - Team members rearrange stories on the scale, discuss changes
  - Clarifications from PO

- **Step 3: Place stories into categories**
  - Place size categories (e.g. Fibonacci sequence) above scale
  - Assign each story a size based on location
After the Planning Meeting

**Begin the sprint**
- Select stories until sprint is full
- Break down stories into tasks and fill your **Scrum Board**
- Assign stories to developer(s)
- **Implement** the stories task by task
Project Workflow: Product Owner

Talk to User / Review Existing System

- Informal List of Desired Functionality

Create and Prioritize User Stories

- GithubTickets
- Acceptance Tests

Present User Stories to Team

- List of User Stories that the Team will tackle

Reiterate every Sprint
Project Workflow: Developers

Estimate User Story Effort (Planning Poker) → Create and Estimate Tasks per User Story → Create Unit Test & Implement Task

Repeat until Feature is finished, Run tests frequently → Update Tickets, Create Documentation → Push Feature

Done and sprint is not over, yet?
- **Help** your teammates
- Refactor, **write tests**, document
- Ask the Product Owner for more work

---

Scrum — Software Engineering II
Scaling Scrum
Recap: SWTII High-level Overview
Implications of the Setup

What’s needed in such an environment?

- Development process
- Communication on multiple levels
- Infrastructure for collaboration
Start small and grow organically
- Single Scrum (teaching) team for preparation
- Work out foundation for the first sprints
- Scale when it becomes necessary

We are now at the first scaling point
- SWT II participants take over!
Product Owner / Backlog Hierarchy

Update at the end of each sprint to consolidate team results

Just-in-time update before the synchronized planning

Main Product Backlog

PO Team A

PO Team B

PO Team C

Team Product Backlogs

[Christoph Mathis, Scrum Center]
Scaling Scrum: Sprint Planning

■ Preparation
  □ Individual review and retrospection meetings
  □ Meeting of all teams with 1-2 members each:
    – Review of the last sprint
    – Input dependencies (What is needed)
    – Output dependencies (What needs to be delivered)

■ Execution
  □ Individual plannings (strict timeboxing)
  □ Discussion of identified additional input or output dependencies
  □ Final sprint planning

■ Problem: Time consuming & high degree of coordination needed!
Scaling Scrum: Sprint Planning

Another Option: Co-located planning

Team 1
Team 2
Team 3
Team 4
POs
Team 5
Team 6
Team 7
Team 8
One Room
Scrum of Scrums

Goal: Synchronize team effort with minimal coordination overhead

- Regular meeting of all Scrum masters.
  - Developers join if necessary (ambassador principle)

- Scrum masters
  - Share their learnings
  - Report completions & next steps
  - Coordinate inter-team dependencies
  - Negotiate responsibility

- Developers discuss technical interfaces across teams
- Distribute information back into the teams
1. The Case for Agile
2. The Scrum Process
3. Scaling Scrum
Image Credits

- “ST vs Gloucester - Match – 23” by PierreSelim (CC BY SA 3.0) via Wikimedia Commons
- “Scrum process” by Lakeworks. (CC BY SA 3.0) via Wikimedia Commons
- “Wien - Seestadt, SW-Areal 2013 (2)” by Bwag (CC BY SA 3.0) via Wikimedia Commons
- “Planning Poker! I’ve a straight flush!” by Joel Bez (CC BY 2.0) via flickr
- “Rubbermaid FastTrack Garage Organization System” by Rubbermaid Products (CC BY 2.0) via flickr
Next: Scrum Exercise

- Practical Scrum Exercise
- 09:15
- Room D.E-9/10, D-School building, next to Villa