Bauhaus-Universität Weimar

Software Engineering

Lecture 05 – Agile Methods

© 2015-20 Dr. Florian Echtler
Bauhaus-Universität Weimar
<florian.echtler@uni-weimar.de>

General problems with IT projects

Source (FU): McKinsey, 2012, http://www.mckinsey.com/...IT_projects_on_time_budget_and_value.ashx

- On average, large IT projects (> $15 million):
  - 45 % over budget
  - 7 % over time
  - 56 % less value than predicted
- 17 % of large projects fail so spectacularly as to threaten company survival

Plan-driven methods: Problems

- Emerged in 1960s from other engineering disciplines
- Business cycles getting ever faster
  - → requirements change quickly
  - → plan-driven methods require rework
- For small projects: massive overhead
  - → often more management than development

- → starting in 1990s: shift to *agile* methods

The Agile Manifesto

Source (FU): Kent Beck et al., 2001, http://agilemanifesto.org/

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

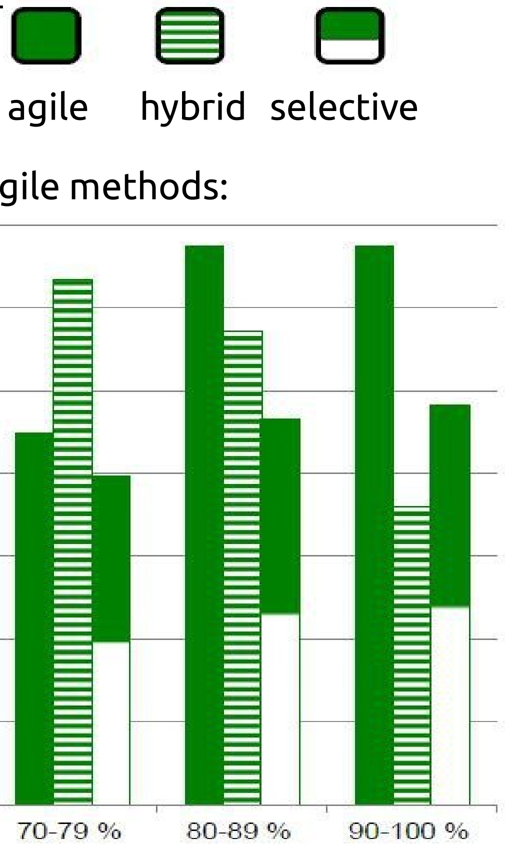
- ***Individuals and interactions*** over processes and tools
- ***Working software*** over comprehensive documentation
- ***Customer collaboration*** over contract negotiation
- ***Responding to change*** over following a plan

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

Popularity of agile methods

Image source (FU): Prof. Dr. Komus, HS Koblenz, Studie Status Quo Agile 2014

Estimate your success quota for projects conducted with agile methods:

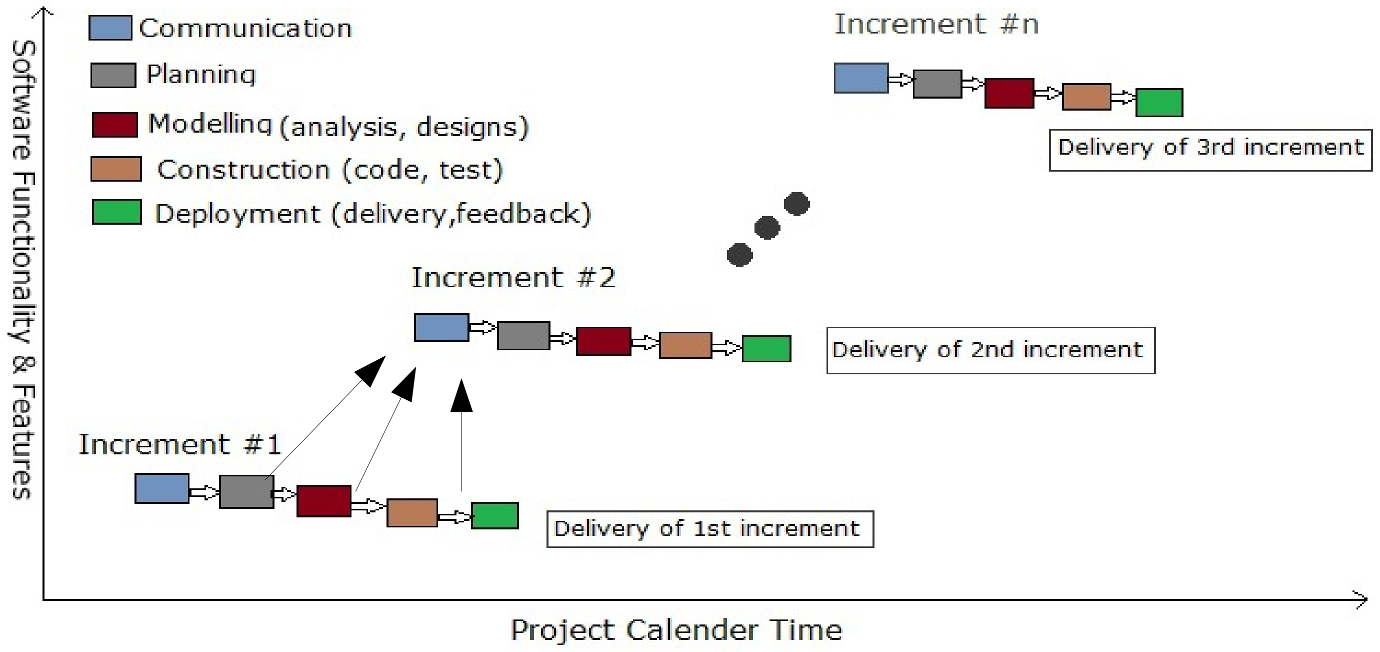


Key aspects of agile models

- Interleaving of activities
  - Specification, design and implementation interleaved/concurrent
- Incremental development
  - Regular releases with customer involvement
- Prototyping tools
  - e.g. user interfaces quickly built with visual editors

Incremental Development

Image source (CC): https://en.wikipedia.org/wiki/Incremental_build_model


Communication
Planning
Modelling (analysis, designs)
Construction (code, test)
Deployment (delivery,feedback)
Increment #n
Delivery of 3rd increment
Increment #2
Delivery of 2nd increment
Increment #1
Delivery of 1st increment
Software Functionality & Features
Project Calender Time

Key agile principles

- *Customer involvement* throughout the development process
- *Incremental delivery* based on customer's priorities
- *People, not process:* leave the development team to their own self-organizing ways
- *Embrace change:* expect requirements to change from the start
- *Maintain simplicity* by constant refactoring

Possible issues with agile methods

- Strong commitment & background knowledge (also in IT) from customer required
- Conflicting priorities from many stakeholders
- Team members' personalities?
- Cultural conflict with processes/contracts in large companies

Possible issues with agile (2)

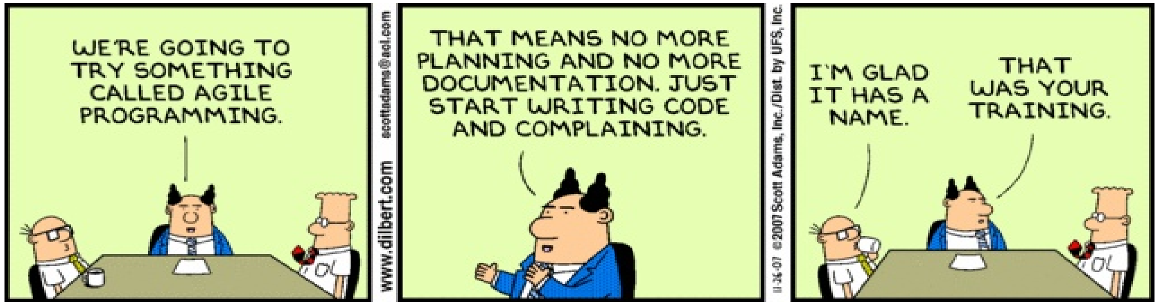
- Constant refactoring means extra work
  → often neglected under time pressure
- Team members often change after each cycle
  → less stable environment
- Maintenance more difficult due to lack of documentation/original team

Things to consider

Best for...	***Agile***	***Plan-driven***
Development team	small, co-located, highly skilled	large, distributed, lower (average) skill
System size	small, monolithic	large, distributed
Regulation compliance or formal analysis needed?	no	yes
System lifetime and iterations	short/many	long/few
Organization	small startup	large corporation

Agile Methods

- Extreme Programming [Beck1999]
- Scrum [Schwaber/Beedle2001]
- Feature Driven Development [DeLuca1997]
  http://www.nebulon.com/
- (Open) Kanban [Hurtado2013]
  http://agilelion.com/agile-kanban-cafe/open-kanban


WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.
scottadams@aol.com
www.dilbert.com
©2007 Scott Adams, Inc./Dist. by UFS, Inc.

Extreme Programming (XP)

- Focused on development aspects
- Key components:
  - User stories define requirements
  - Pair programming, test-first development
  - Multiple new builds/versions per day
  - All tests must pass before integration
  - New releases delivered every ~ 2 weeks
  - No delays, remove features if necessary

XP: Planning & Design

- Incremental planning
  - Requirements based on “user stories”
  - Select stories for each release based on available time & priority
- Simple design
  - Only design for the current requirements
- On-site customer
  - Customer representative as full-time team member

XP: Planning & Design (2)

- Conventional wisdom in SE: “design for change”
  → anticipating changes *now* saves time and effort *later*

- XP perspective: changes cannot be reliably anticipated, leads to unnecessary effort for generalization

  → constant code improvement/refactoring

XP: User Stories

Source (FU): Sommerville, Software Engineering, Chapter 3

- Sample user story: *Prescribing medication*

  Kate is a doctor who wishes to prescribe medication for a patient attending a clinic. The patient record is already displayed on her computer, so she clicks on the medication field and can select “current medication”, “new medication” or “formulary”.

  If she selects “current medication”, the system asks her to check the dose. If she wants to change the dose, she enters the new value and then confirms the prescription.

  […]

  The system always checks that the dose is within the approved range. If it isn't, Kate is asked to change the dose.

  After Kate has confirmed the prescription, it will be displayed for checking. She either clicks 'OK' or 'Change'. If she clicks 'OK', the prescription is recorded on the audit database. If she clicks on 'Change', she reenters the 'Prescribing Medication' process.

XP: User Stories (2)

Source (FU): Sommerville, Software Engineering, Chapter 3

- User stories broken down into tasks
  - Task 1: Change dose of prescribed drug [...]
  - Task 2: Formulary selection [...]
  - Task 3: Dose checking

    Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.

    Using the formulary ID for the generic drug name, look up the formulary and retrieve the recommended maximum and minimum dose.

    Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low. If within the range, enable the 'Confirm' button.

XP: Development

- Pair programming
  - Two people on one workstation
  - Provides instant code review
- Constant refactoring, e.g.
  - Rename methods/classes with descriptive names
  - Move long code section to separate method
- Collective ownership
  - All developers work on all system components
  - No “islands of expertise”

XP: Development (2)

- Test-first development
  - Automated unit tests
  - Written *before* the code itself
- Continuous integration
  - Task completed → integrate into whole system
  - Whole build, all tests must still pass
- Sustainable pace
  - Significant overtime not acceptable
  - Reduces code quality and long-term performance

XP: Testing

- Incremental development
  - → no clear/fixed specification
  - → no external testing team possible
- Tests created by developers *before* the code
  - → *implicitly defines interface & spec.*
  - → *avoids test lag*
- Tests built on tasks from user stories
- Customer involvement through test cases

XP: Testing - Example

Source (FU): Sommerville, Software Engineering, Chapter 3

- Test: Dose Checking
  - Input:
    - A number in mg representing a single dose of the drug.
    - A number representing the number of doses per day.
  - Tests:
    - Input where single dose is correct but freq. too high.
    - Input where single dose is too high or too low.
    - Input where single dose * freq. is too high or too low.
    - Input where single dose * freq. is in permitted range.
  - Output:
    - OK or error message that dose is outside of safe range.

XP: Testing – possible issues

- Code coverage
  - Developers may skip certain test cases
  - Refactoring can cause classes to be missed
- “Incremental testing”
  - Interaction between classes difficult to test
  - Even 100% unit test coverage can't catch all bugs

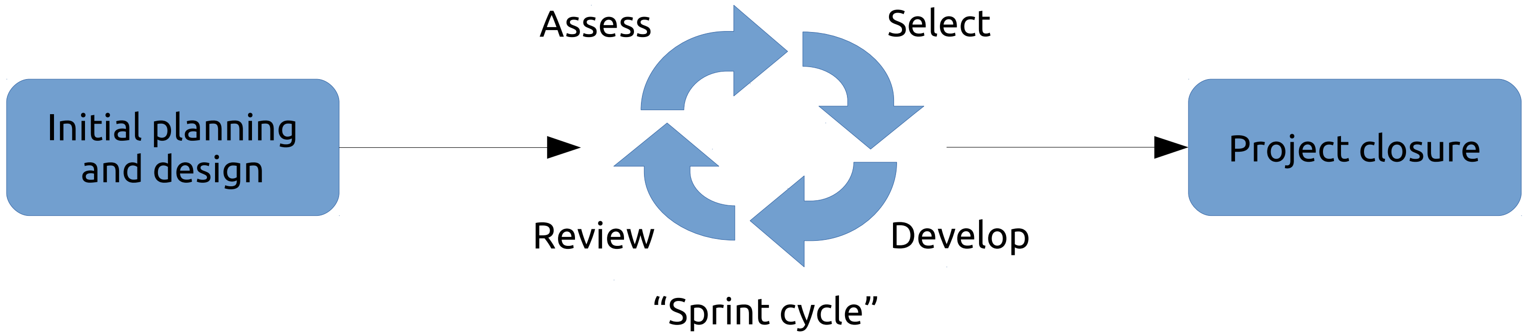
XP: Pair Programming

- Two developers on one workstation
  - Supports collective ownership/responsibility, “egoless programming”
  - Informal review process: each line of code is looked at by at least 2 persons
- Total productivity comparable to 2 persons working alone: continuous discussion →
  - Implicit knowledge sharing
  - Fewer false starts/mistakes
- However: depends on individual personality!

Scrum

- Focused on management aspects
    - Definition – British (*informal),* orig. from Rugby:
        - a disorderly crowd of people or things.
          "there was quite a scrum of people at the bar"
- Designed to enable management of iterative development processes (*contradiction!*)
- 3 phases:
    - Outline – general objectives, architecture
    - Sprint cycles – develop one release per cycle
    - Project closure – wrap-up & documentation

Scrum


Initial planning and design
Assess
Select
Review
Develop
"Sprint cycle"
Project closure

Scrum

- Sprint cycle (~ 2 weeks = one release in XP)
  - *Assess* product backlog (work to be done)
  - *Select* features/functionality (with customer)
  - *Develop* release (team isolation by Scrum master)
  - *Review* and present to stakeholders

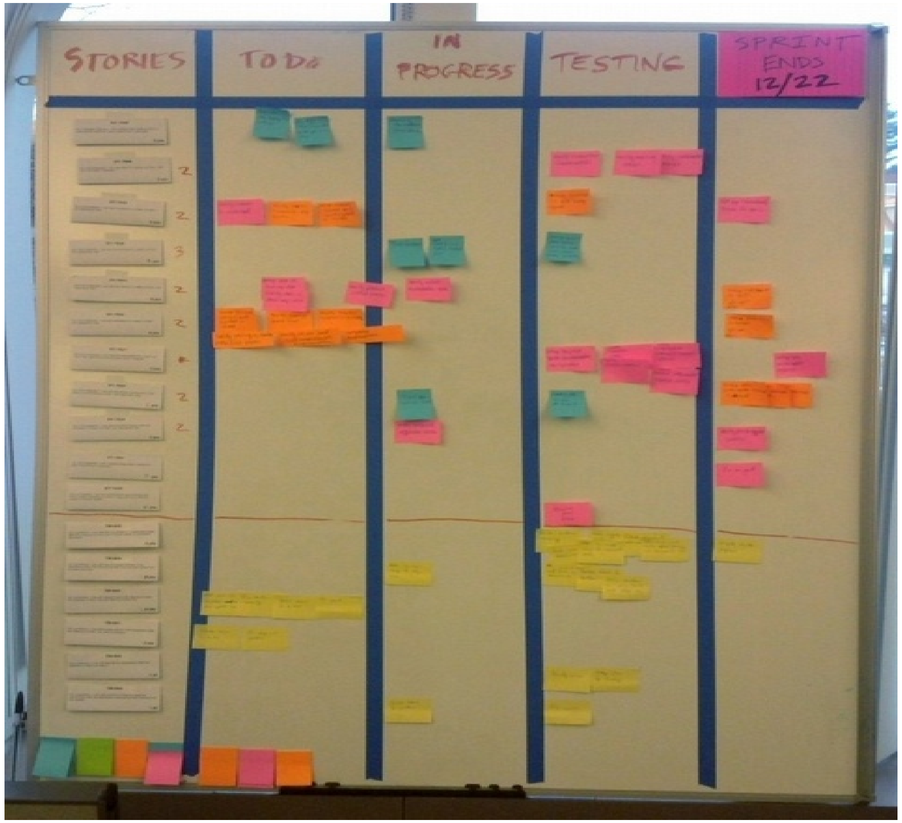
Scrum Master/Meetings

- Facilitator – main tasks:
  - Arrange daily stand-up meetings (~ 15 min.)
  - Track backlog/work to be done
  - Record decisions
  - *Communicate with customers/upper management*
- Daily meetings
  - Everyone briefly describes progress/problems/ plan for the day
  - Everyone knows about project state

Scrum Board

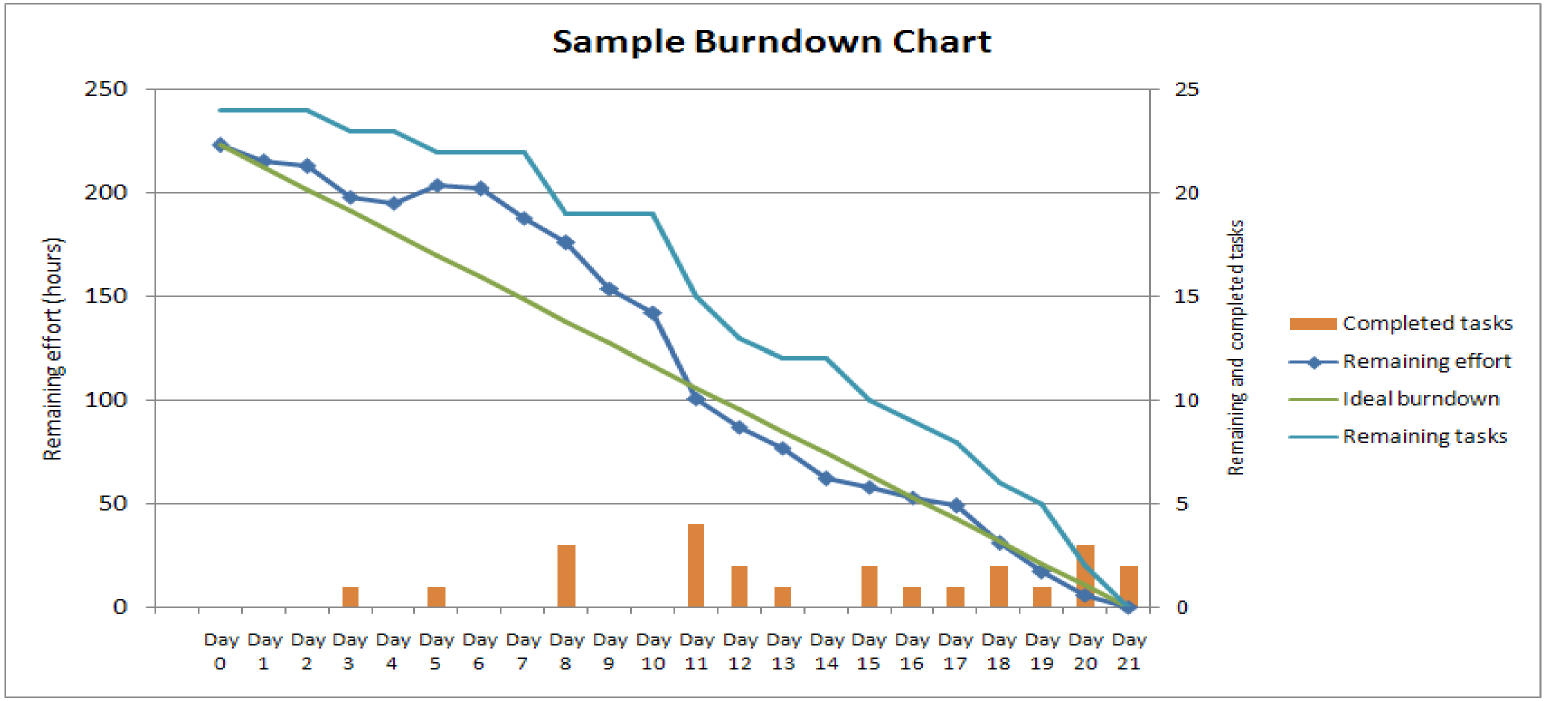
Image source (CC): https://en.wikipedia.org/wiki/Scrum_%28software_development%29

- Often used to track progress during sprint cycle
- Categories: ToDo, In Progress, (Testing), Done



Scrum: Burn-down chart

Image source (CC): https://en.wikipedia.org/wiki/Scrum_%28software_development%29



Scaling agile methods *up*

- Scaling up: agile methods for large systems
  - Consist of separate, communicating subsystems
  - Include/interact with other existing systems
  - Have to follow regulations (e.g. aircraft)
  - Long procurement/development/deployment time
  - Multiple stakeholders

Scaling agile methods *up* (2)

- Agile adaption requires …
  - Multiple teams (too large for single small team)
  - At least some up-front architecture design
  - Communication mechanisms between teams (e.g. Wikis, video calls, group chat, …)
  - More detailed documentation
  - Continuous integration perhaps not possible due to build duration

Scaling agile methods *out*

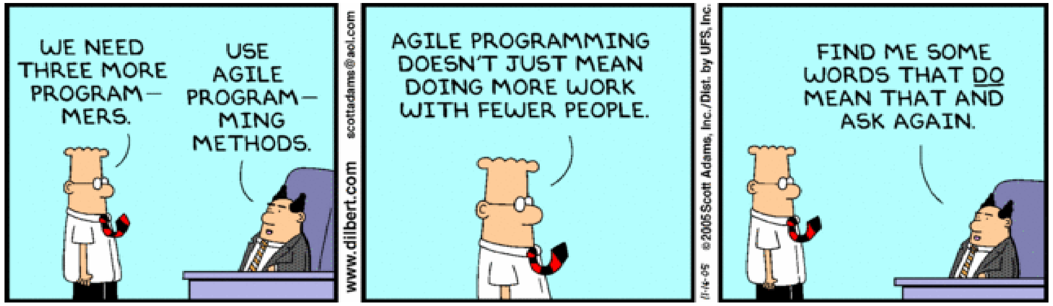
- Scaling out: agile methods in large companies
- Needs to deal with …
  - Reluctance in upper management
  - Incompatible with bureaucratic procedures/regulations
  - Wide range of skill levels in teams
  - Cultural resistance

Summary

- Both plan-driven and agile have merits
- Textbook examples are unlikely to work in reality without adaptation

Remember: one size does *not* fit all.

Questions/suggestions?


WE NEED THREE MORE PROGRAM-MERS.
USE AGILE PROGRAM-MING METHODS.
AGILE PROGRAMMING DOESN'T JUST MEAN DOING MORE WORK WITH FEWER PEOPLE.
FIND ME SOME WORDS THAT DO MEAN THAT AND ASK AGAIN.
www.dilbert.com scottadams@aol.com
© 2005 Scott Adams, Inc./Dist. by UFS, Inc.