

Agile vs. Plan-driven



<http://pixabay.com/en/tent-camping-camper-24500/>



<http://upload.wikimedia.org/wikipedia/commons/5/51/Gotic3d2.jpg>

Motivation: Agile vs. Plan-driven

- Some history
 - Software crisis (1968)
 - Chaos report (Standish group: since 1994, every 1-2 years)
 - Similar reports (e.g. recent study by Forrester)

→ Too many software projects (still) fail!

- Success factors:
 - End-user involvement, Top-Management support, clear requirements
- Failure factors:
 - No End-user involvement, unclear and incomplete requirements, high number of requirements changes

→ Software **Engineering**: Systematic / repeatable approaches to build software.

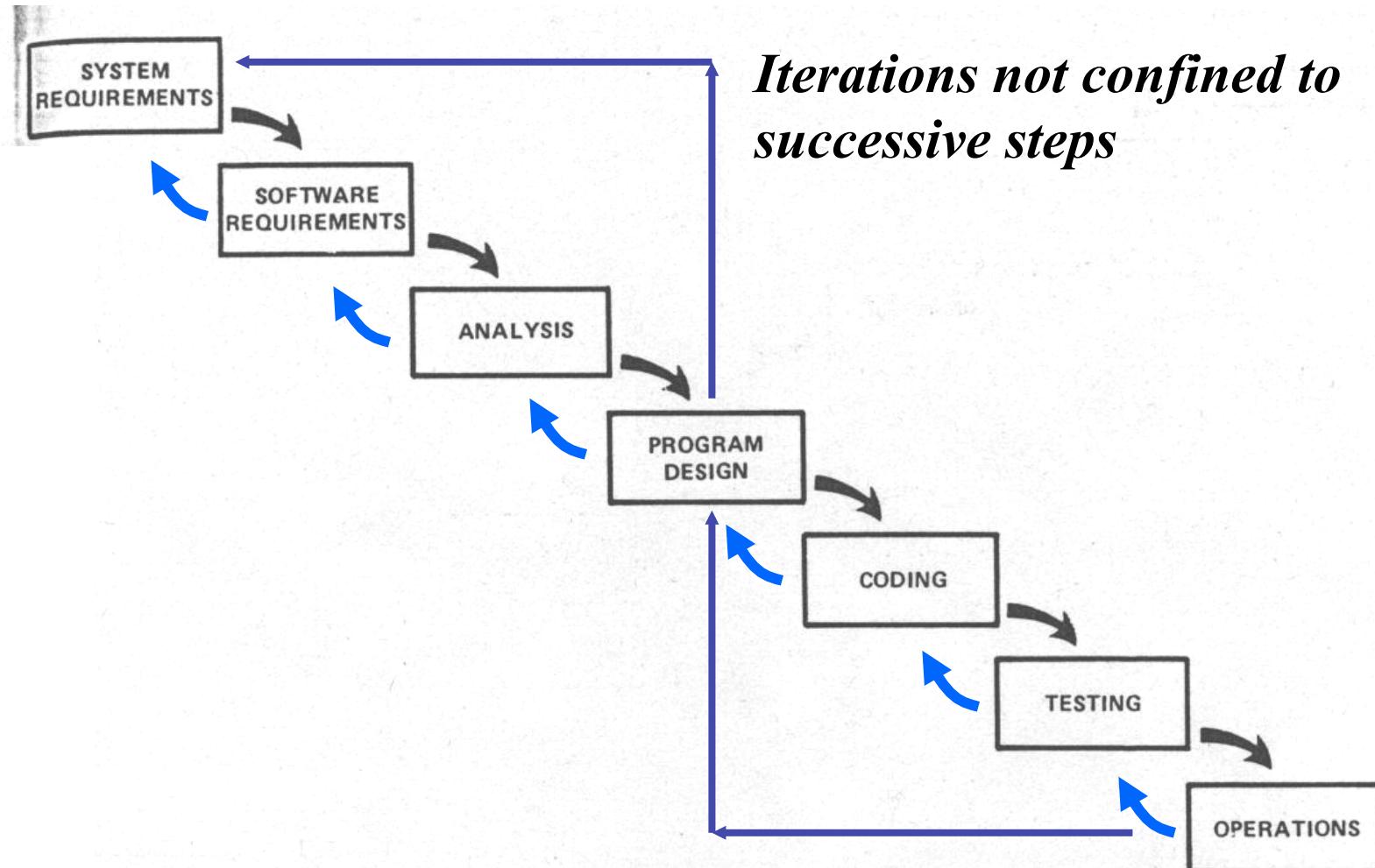


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

Royce, 1970: <http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>

The “unknown” part of the waterfall

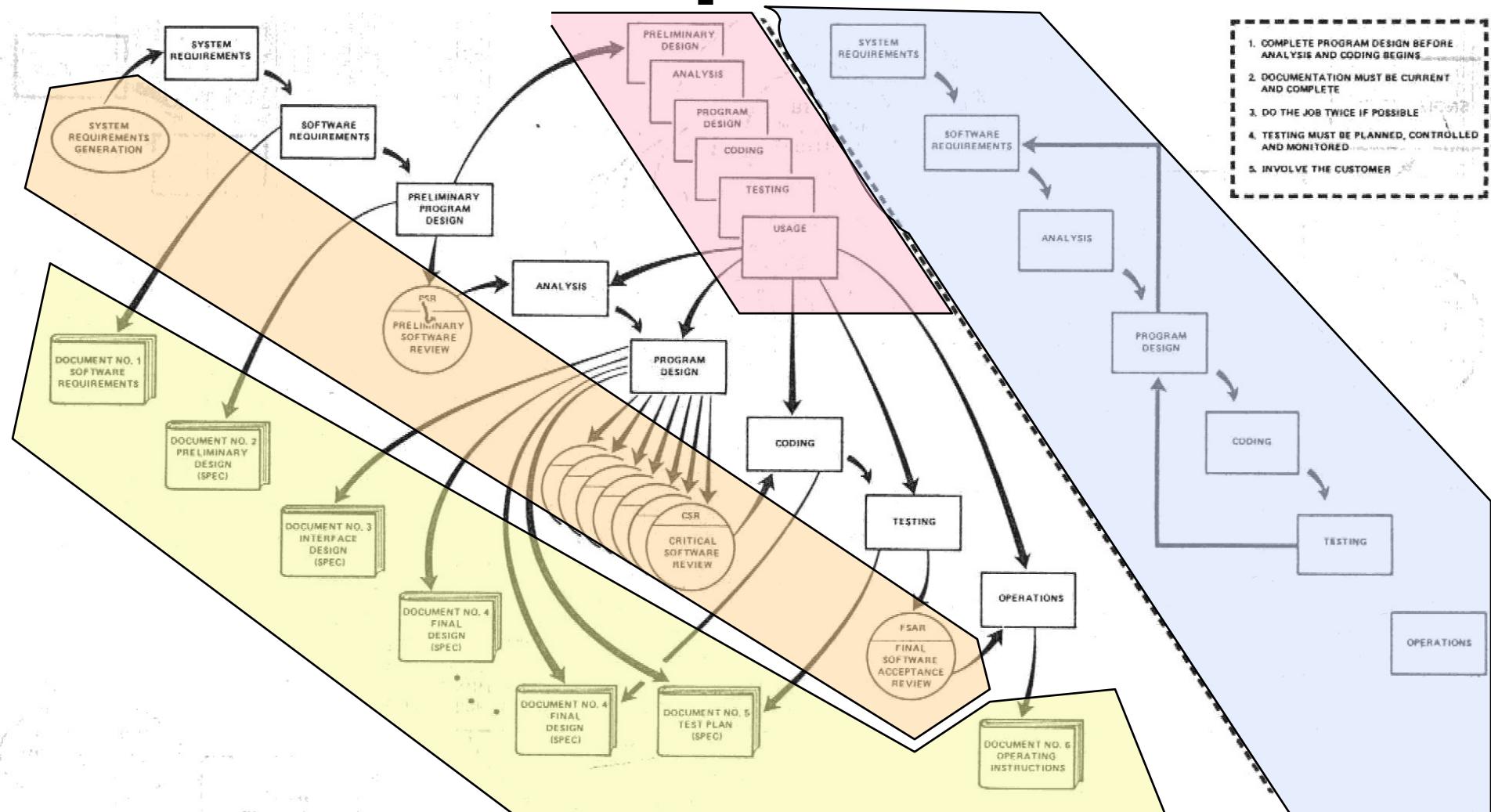


Figure 10. Summary
Agile Software Dev. | Eric Knauss

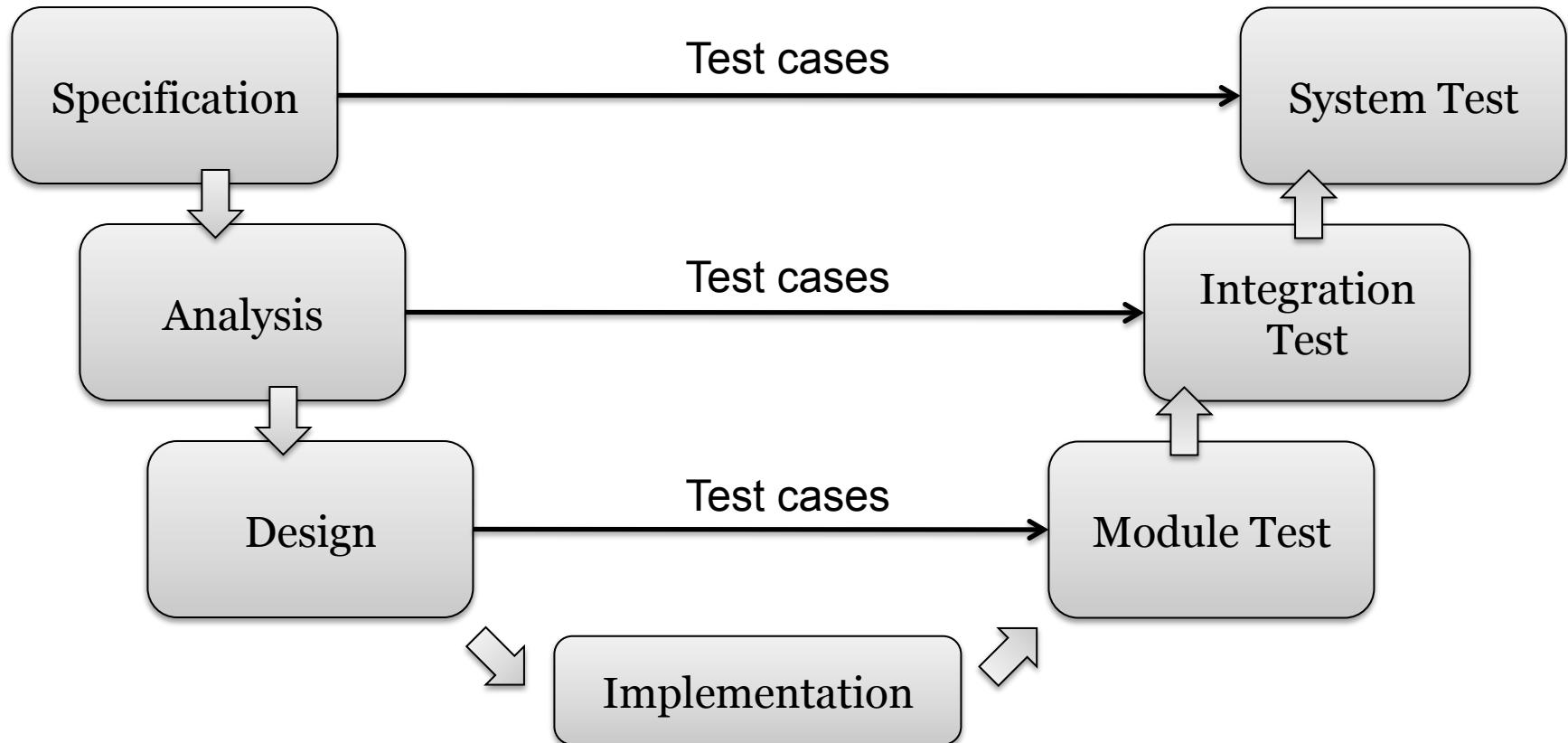
Waterfall Model

- Phases
 - Requirements analysis and definition
 - System and software design
 - Implementation and unit testing
 - Integration and system testing
 - Operation and maintenance
- Main principle: One phase has to be completed before the next phase can be started
- Changes are hard due to implied dependencies between artifacts from different phases

Waterfall Model

- Restrictions
 - Customer's change requests cannot be adapted easily by the ongoing development process
 - The process should only be applied when requirements are pretty complete, well understood by all stakeholders (customer, project manager, developers, testers, ...), and changes are not expected
 - Only a few projects fulfill all these preconditions at the beginning
 - Waterfall model is used for large engineering projects, which might be spread over several development sites

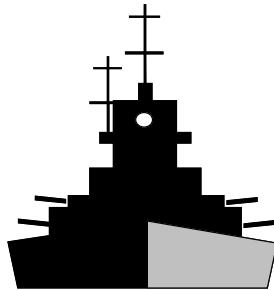
V-Model



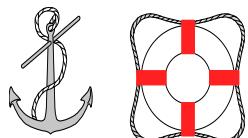
V-Model

- Applicability
 - Automotive industry
 - Governmental projects
 - Hierarchical development projects with subcontractors
 - Large engineering projects which are spread over several development sites
- Restrictions
 - Change requests due to modified requirements could not be adapted easily
 - Sequential development
 - Tayloring only possible with V-Model XT

Field of Tension



Different needs and priorities



*Suitability of means
(Checklists, Templates, ...)
low overhead*

*Mature processes
Disciplined approach*

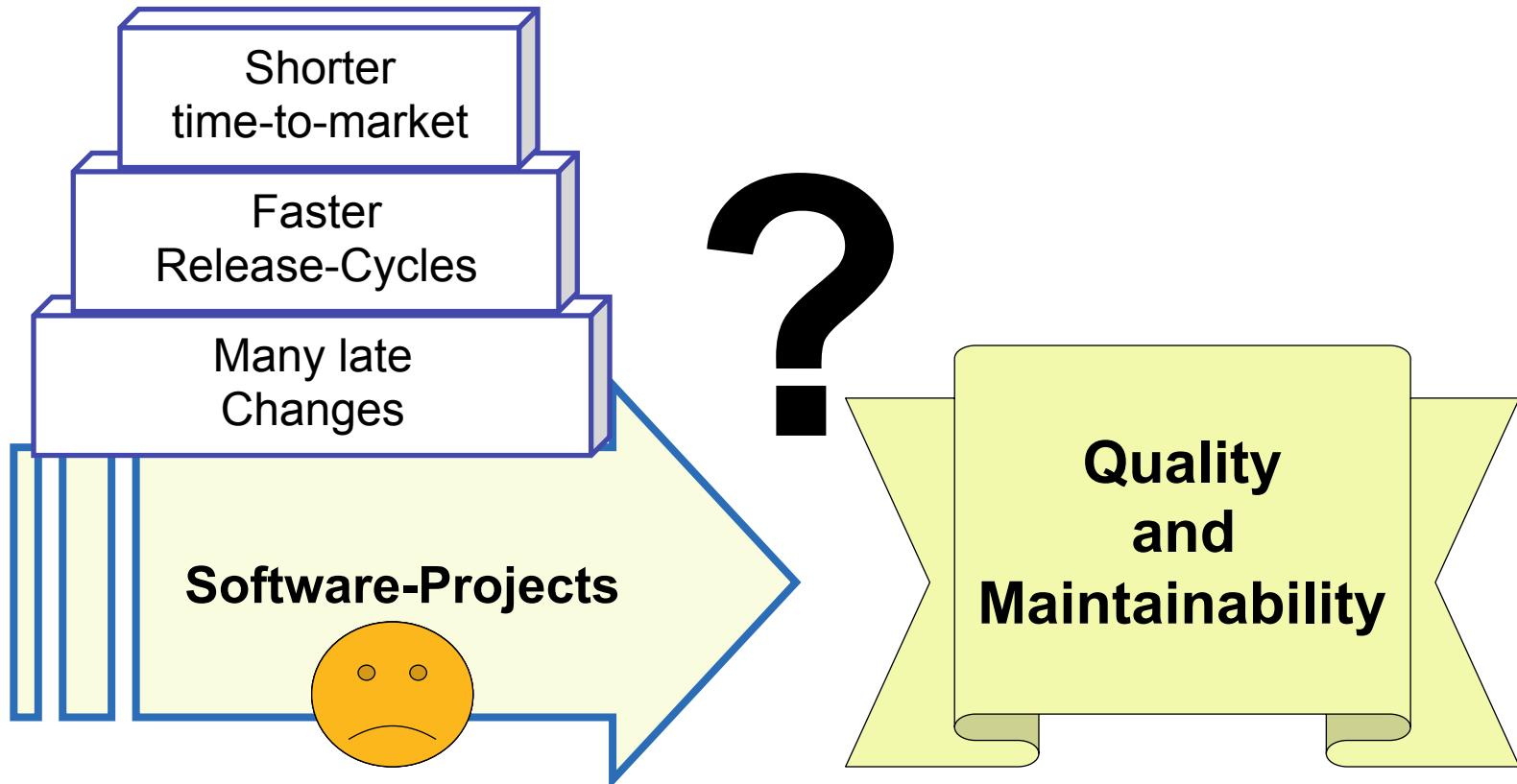
**Improvement in one area
without too much deterioration in others**



*Good ability to react
Quick and flexible*

**Where is the biggest
utility
for a company/project?**

Typical problems in software development



Projektplan (klassisch)

c.f. IEEE Std 729-1983

Defines 6 questions

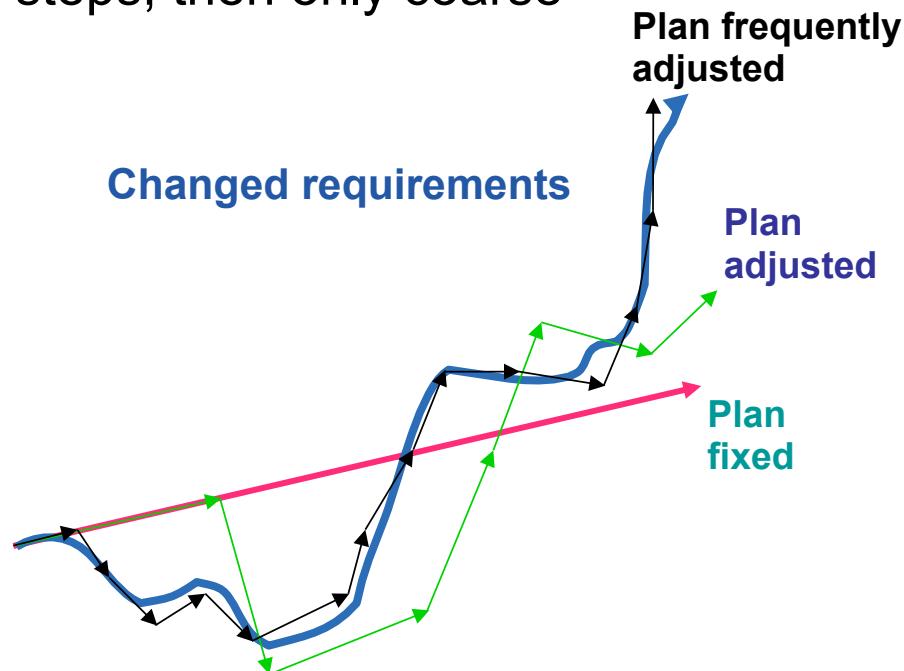
- What will be done
- Why
- For how much money
- By whom
- When
- With what resources (Development- and Management-aids and - techniques)

To be defined

- Responsibility of Project Manager
- Necessary tasks (as far as they can be known)
- Expected results
 - In detail: Software, Documentation, Servicesdetaillierter
 - E.g. for Doc.: table of contents, intended readers, level of detail
 - Ideal: Acceptance criteria, even for Documents

What is different in agile?

- Observation: Planning has limitations
- Therefore: Fine-grained plan for next steps, then only coarse
- Precondition:
Continuous progress control
and feedback



- Example for agile “Micromanagement”: SCRUM

Agile Manifesto

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 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.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

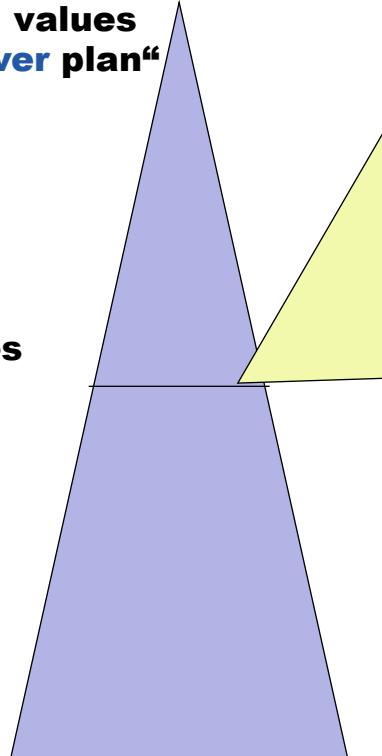
Structure of agile methods

Generally

Fundamental values
„change over plan“

Principles

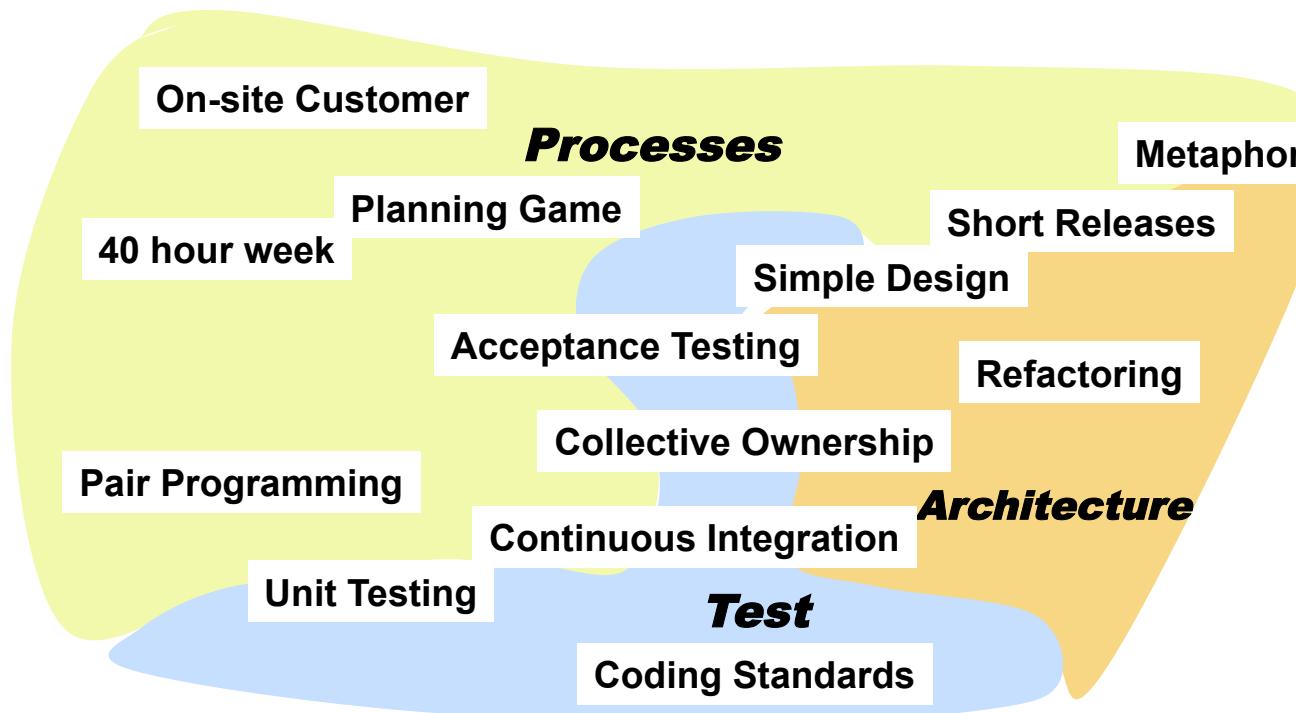
Practices
(Implement., well known)



- **Highest priority: Customer Satisfaction**
 - early and continuous delivery of valuable software
- **Simplicity**
 - Avoid unnecessary work
- **Even late changes are welcome**
 - Imply improved service to customer
- ...
- **Direct face-to-face communication**
 - Best way to exchange information
- **Regular reflection on how to become more effective**
 - Feedback on several levels

eXtreme Programming

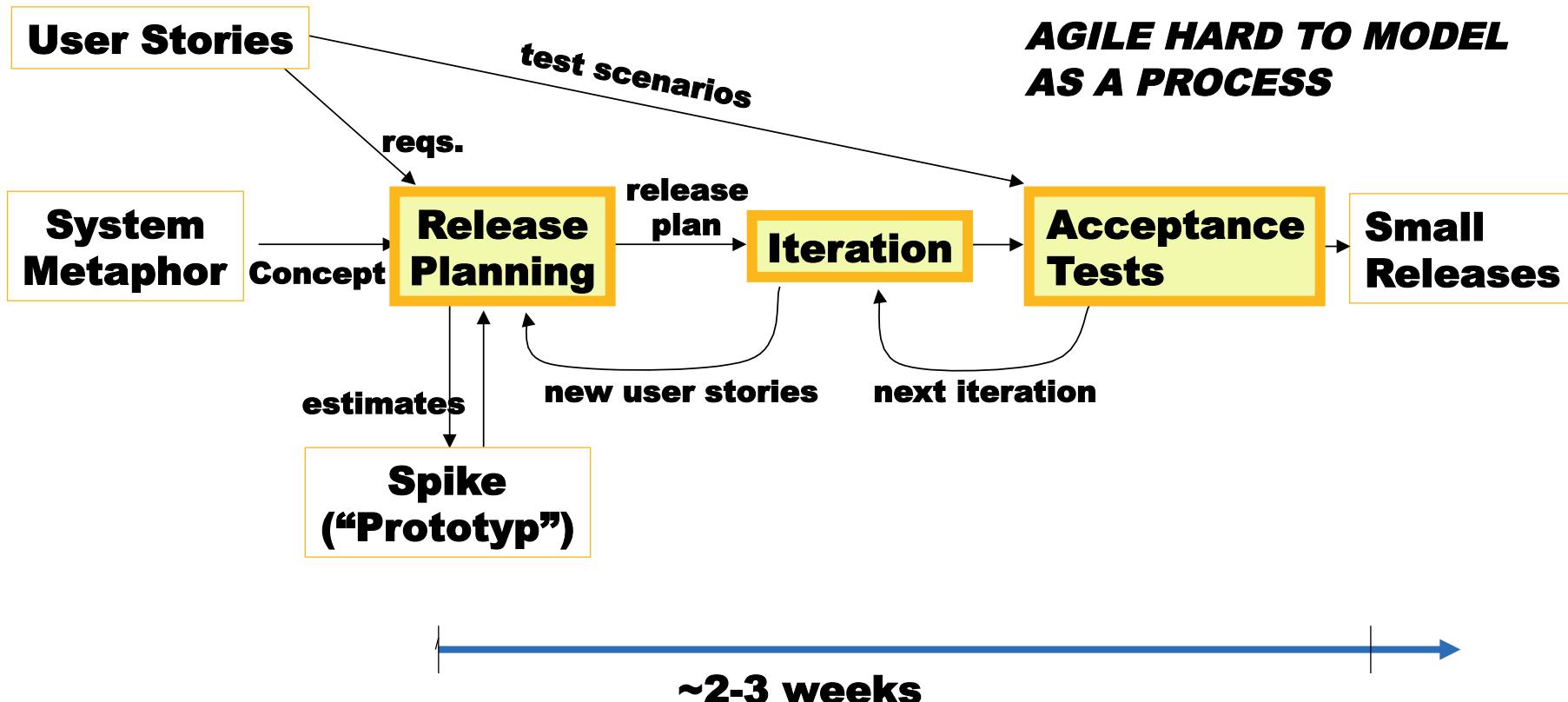
- Praktices
 - Interact and support each other
 - Interplay of Process, Testing, and Architectural issues



Processes in XP

based on Practices

OBS! Many practices are not shown explicitly in this model!

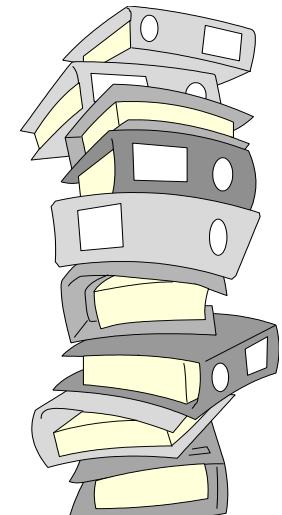


How much planning does a project need?

Inspired by Barry Boehm

Fine grained Contracts

- ++ clear basis for work
- + financial security
- very high effort
- Changes very hard to achieve



How much planning does a project need?

Inspired by Barry Boehm

ad hoc

- + low planning effort
- + individual freedom
- result hard to anticipate
- depending on “Hero”

Fine grained
Contracts



How much planning does a project need?

Inspired by Barry Boehm

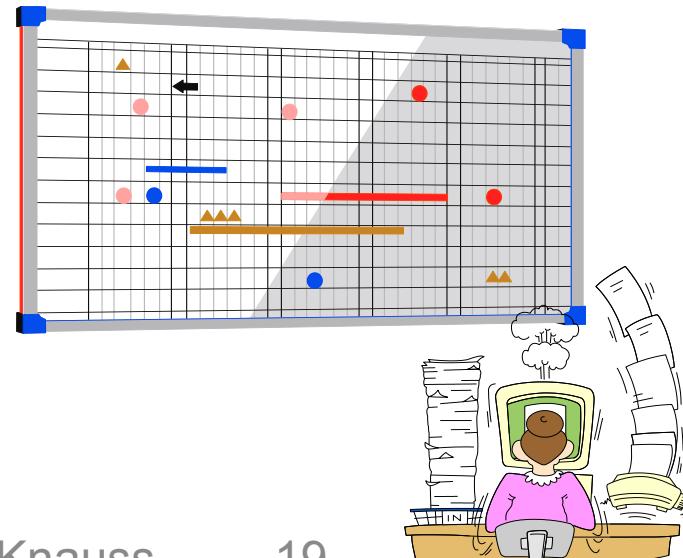
ad hoc



Milestone & Plan Driven

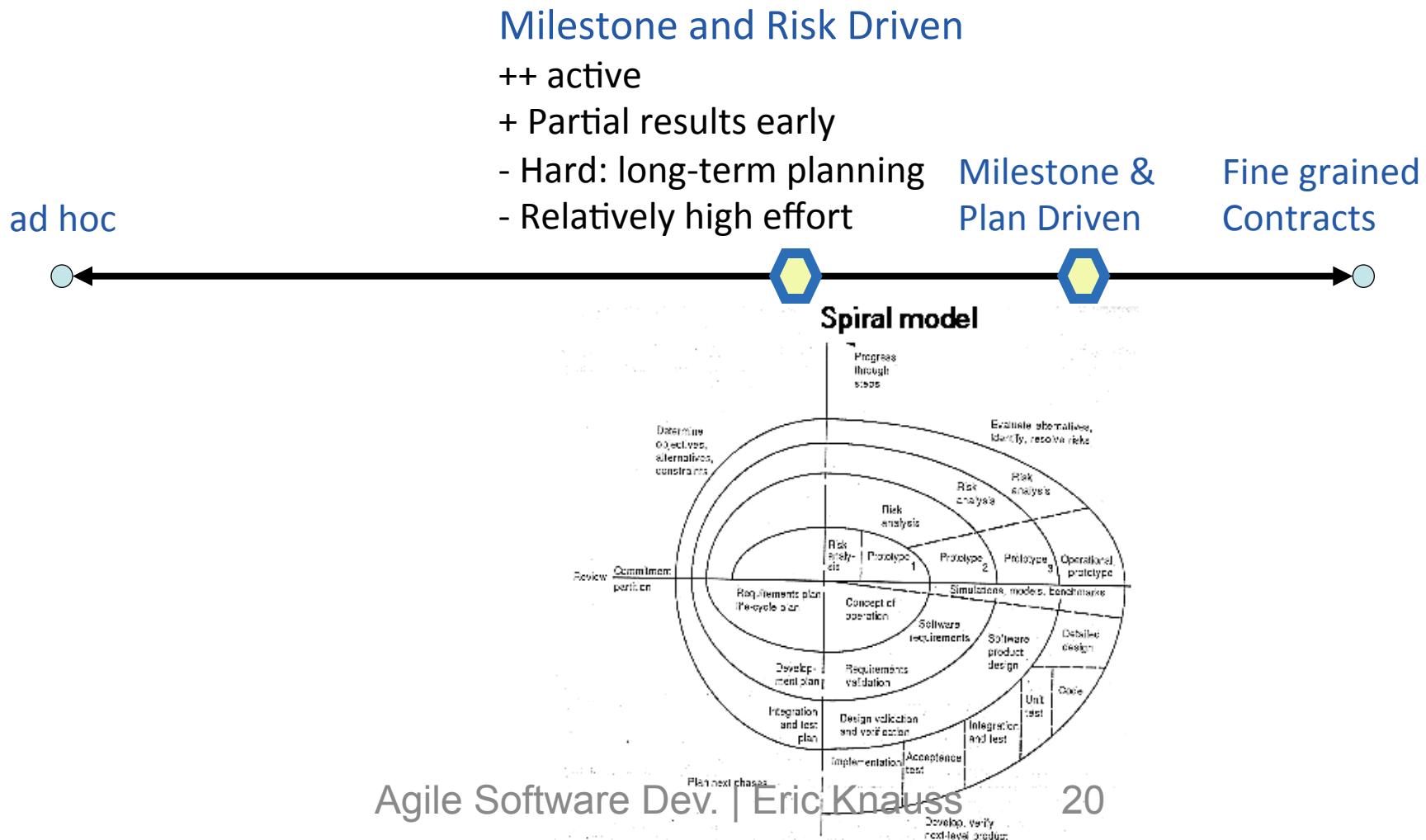
- + long term predictions
- + good status control
- - Changes difficult
- - unrealistic assumptions
hard to eliminate

Fine grained
Contracts



How much planning does a project need?

Inspired by Barry Boehm



How much planning does a project need?

Inspired by Barry Boehm

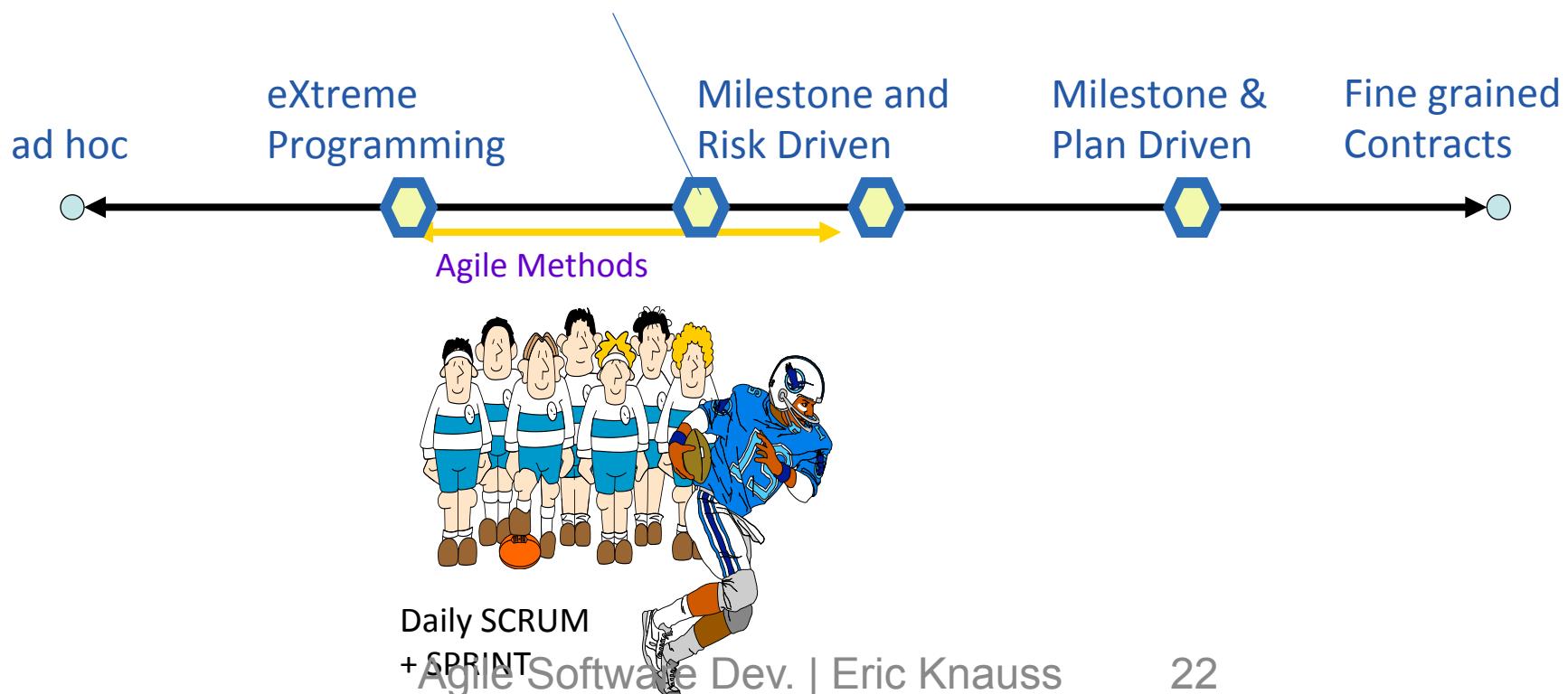


Just enough planning

Inspired by Barry Boehm

Agile Example: SCRUM

- + midterm planning
- + quick reaction
- Relies on team qualification
- End-product not specified



Agile thinking

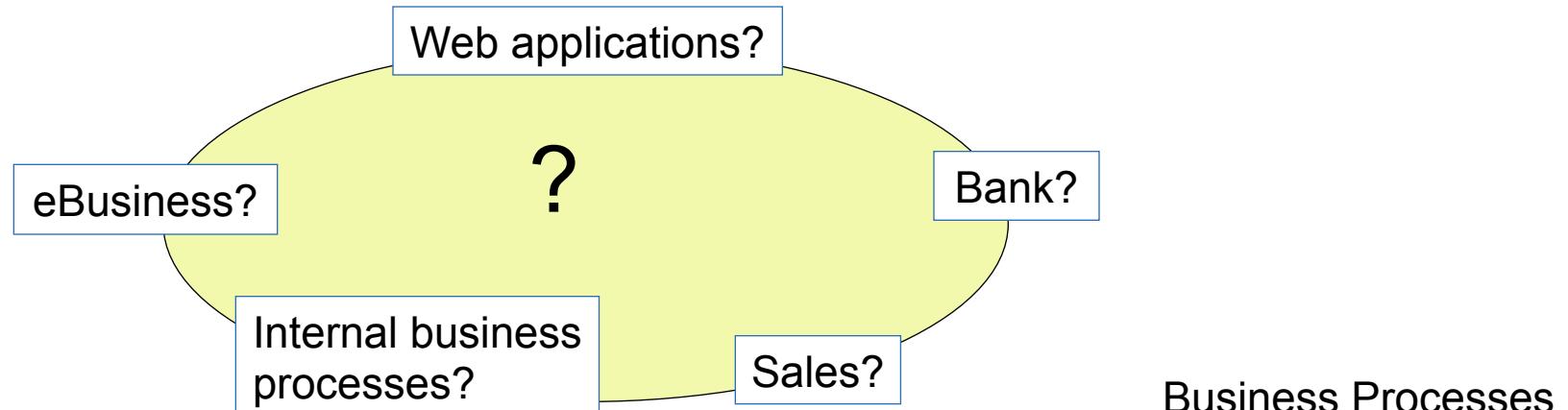
**Based on Fröhlauf,
Conquest 2001**

	Traditional	Agile
Customer collaboration	unlikely	Critical success factor
Delivery of something useful	Only after some (longer) time	At least every 6 weeks
Develop the right system by	Thinking ahead, detailed specification	Develop core, show, improve
Required discipline	formal, low	informal, high
Changes	Create resistance	Are expected and tolerated
Communication	Via documents	Face-to-face
Prepare for changes	By planning ahead	By being flexible

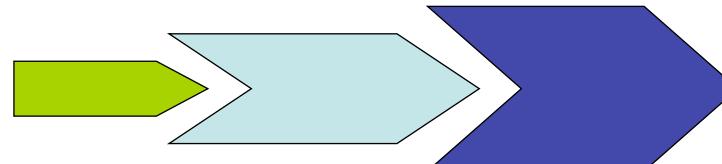
Introduce agile methods_

- How do
 - Line managers
 - Engineers in software maintenance
 - Customers
- with a plan-driven background feel if you propose agile?
- Beware!
 - There are many false pretenses and excuses
 - There are also crucial points that reoccur
- Whoever offers agile methods needs to take these crucial issues very seriously!

Agile in a large company?



Research?



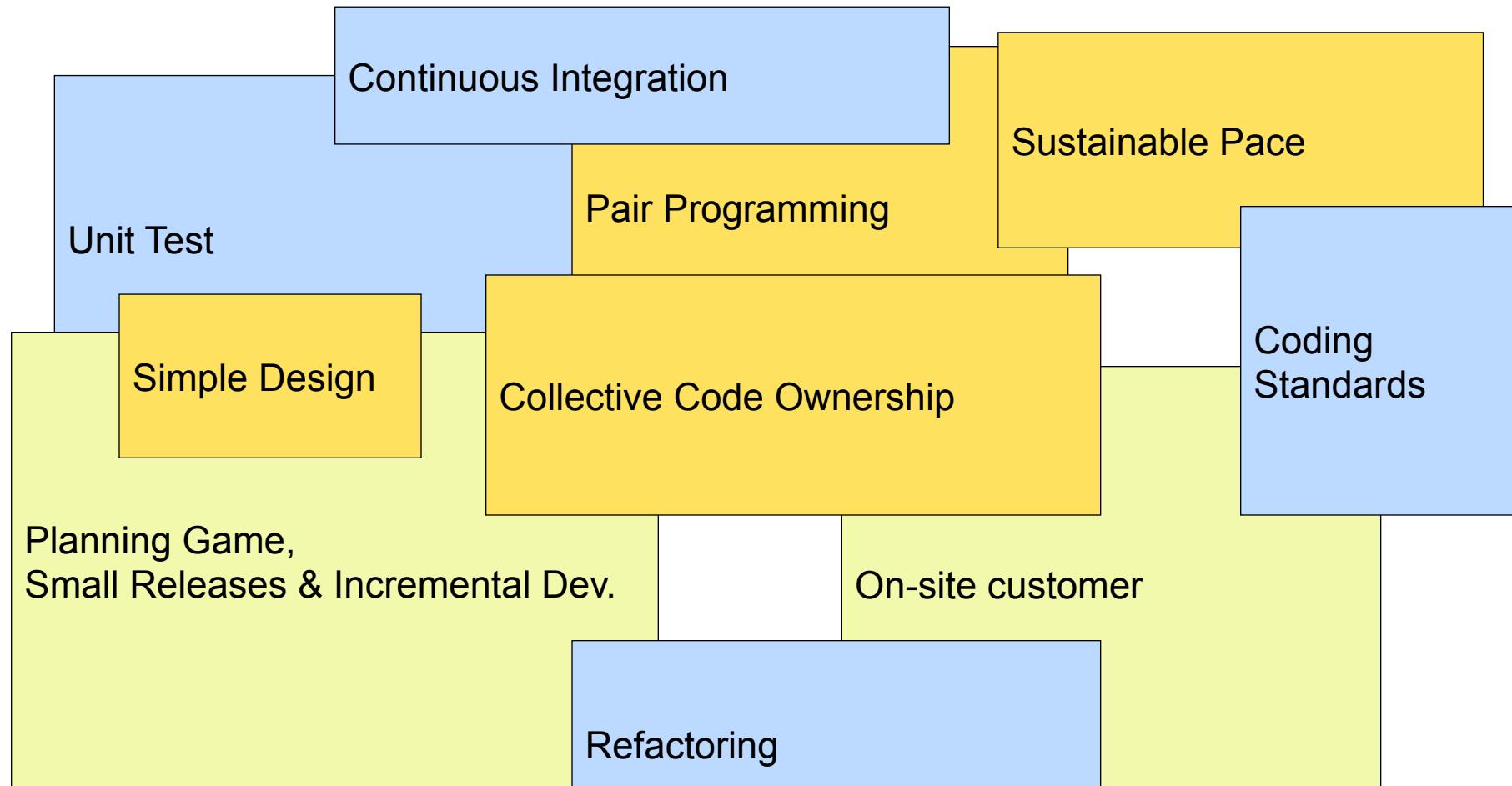
Safety-relevant
Software?

25

Implications of agile

(example here: XP)

„The heart of Agile Modeling is its practices“ Scott W. Ambler



Common objections

Schneider, K. (2003):
SQM congress 2003, Köln, SQS

On the first glance ...

- Working in pairs is a waste of time
- Our large systems do not allow for fast releases
- Sustainable pace? I don't care, we have a 35h week!

Might look like that, but...

Ok, let's check

More serious

- Our people will hate collective code ownership and working in pairs...
- Our type of systems do not allow for refactoring/cont. integration/test first

General Objections

- That is nothing new! 20 years old
- This does not work here.
- We are already doing that – I mean wild hacking, hahaha!

Well known killerphrases

Real crucial issues

Did I get you right?

- You implement without knowing what we want?
- You want a customer representative on site permanently?
- You try to *avoid* documentation?
- You want to make work for you as simple as possible? What about QM?



Be honest!

You feel anxious because so much is changing.
Are you keep the overview?
Or would you prefer failing in a proven way over a risky success?



What do we get?

- We write comprehensive specifications so that we know what we get. And now?
- How do we know what we get in the 5th increment?
- We won't get new money after each increment!
- We work with fixed-price contracts. Does this work here as well?
- Why should we trust this agile contractor?



- Not any more!
 - Specifications are too expensive and outdated while being written. Yes, uncertainty remains.
- We don't know, we assume
 - Make a master plan and change it
 - And: we are not stupid!
- Right; apply for all of it now.
- Yes:
 - Based on exploration phase
 - Based on a good domain description
 - Simple changes without changing contract
- Based on good experience in exploration phase

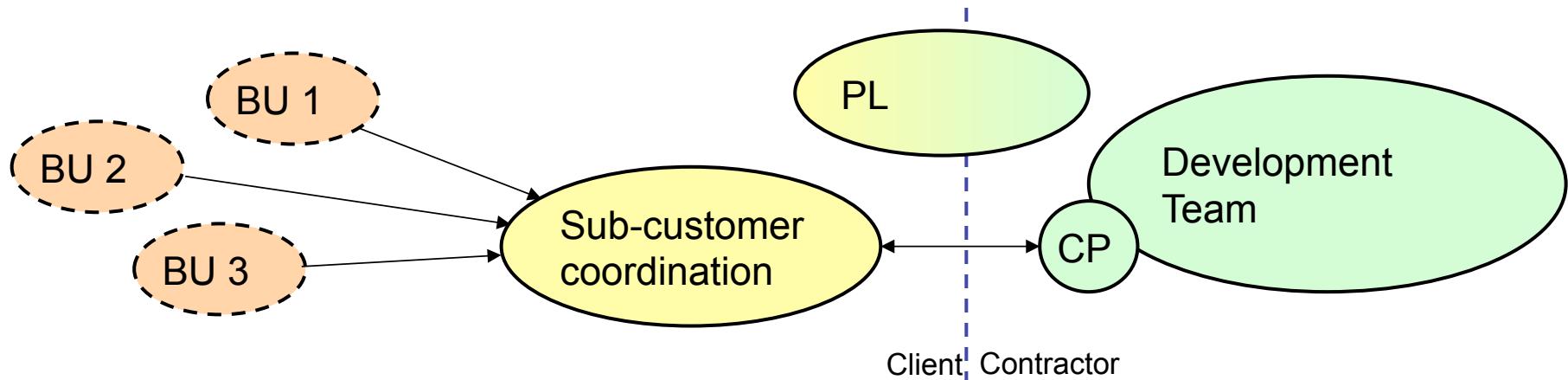
On-Site Customer

- We cannot do without this person!
- Several business units are part of this.
The one customer does not exist!
- Nobody can know all business units in detail
- To avoid errors, a lot of coordination is required
- Does not the On-site Customer lose attachment to the business unit?
- Every business unit representative will favor *their* unit

- Fulltime is rare
 - Sometimes availability on the phone sufficient
 - Consider Customer Proxy! (below)
- You need to define one!
 - (can be changed later)
- No need to know all the details!
 - Changes are possible, errors allowed
 - Important: Ability to decide – without coordination
- Can happen!
 - In long projects: exchange!
- Customer proxy Concept
 - Product manager plays customer

“Customer Proxy” Concept

- Developer plays “customer”
 - Good domain knowledge
 - Intensive contact with developers
 - Participates in coordination of sub-customers
- Important
 - Quick decision by one person
 - Wrong decisions can be revised
 - We observed this in industry and did it ourselves: It works!



Schneider, K. (2003): SQM congress 2003, Köln, SQS

QM, Documentation (etc.) and internal Rules

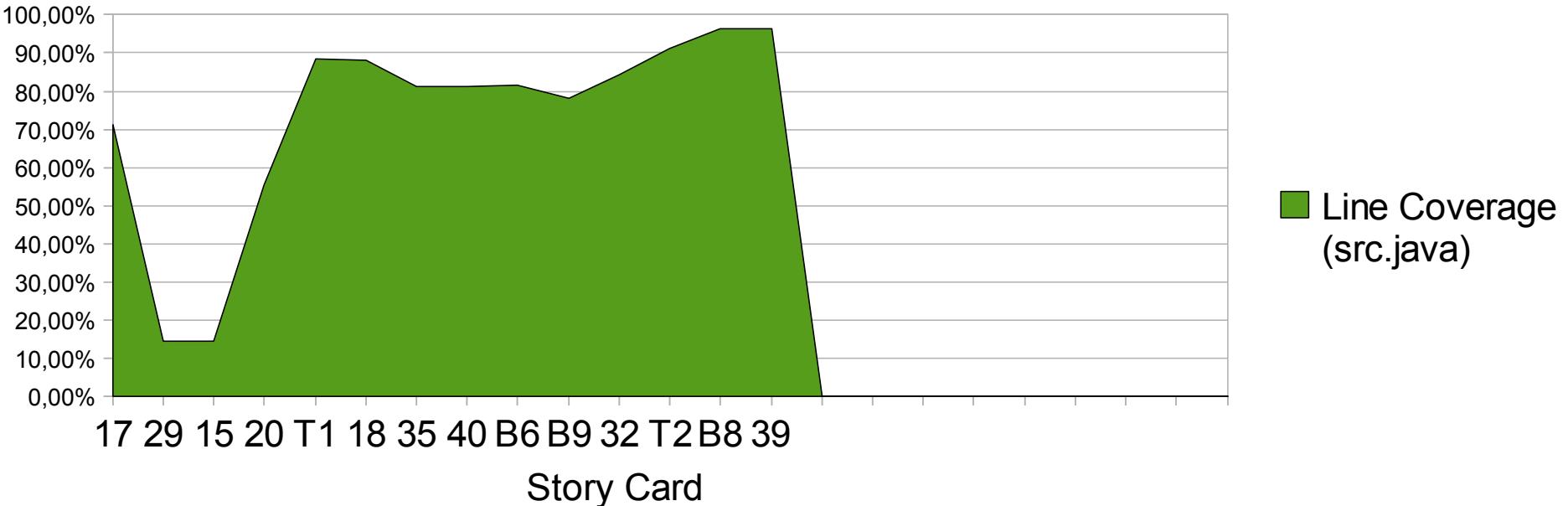
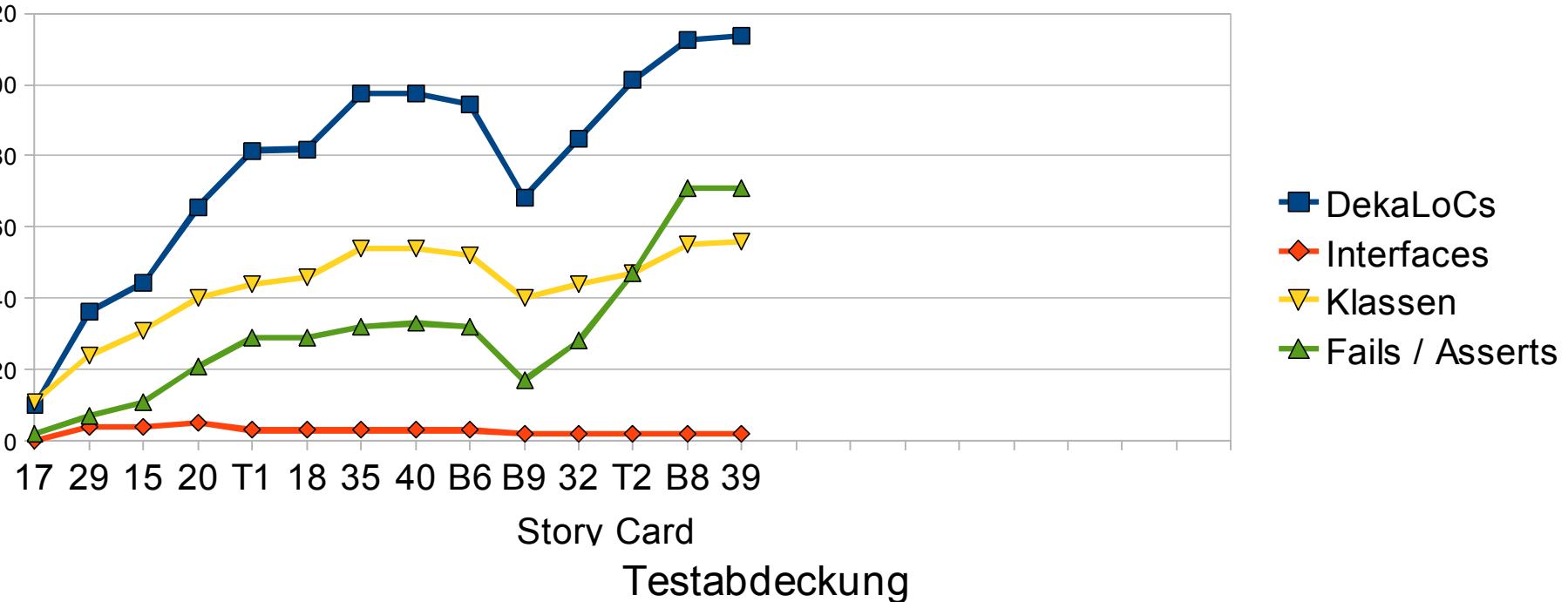


- Software without documentation is useless
- Development teams change – we are lost without technical documentation
- QM-handbooks etc. require many documents, reviews, activities
 - Shall we give up all of that?
 - Do we need a two-class society?
- Can agile approaches adhere to all these rules and still be agile?

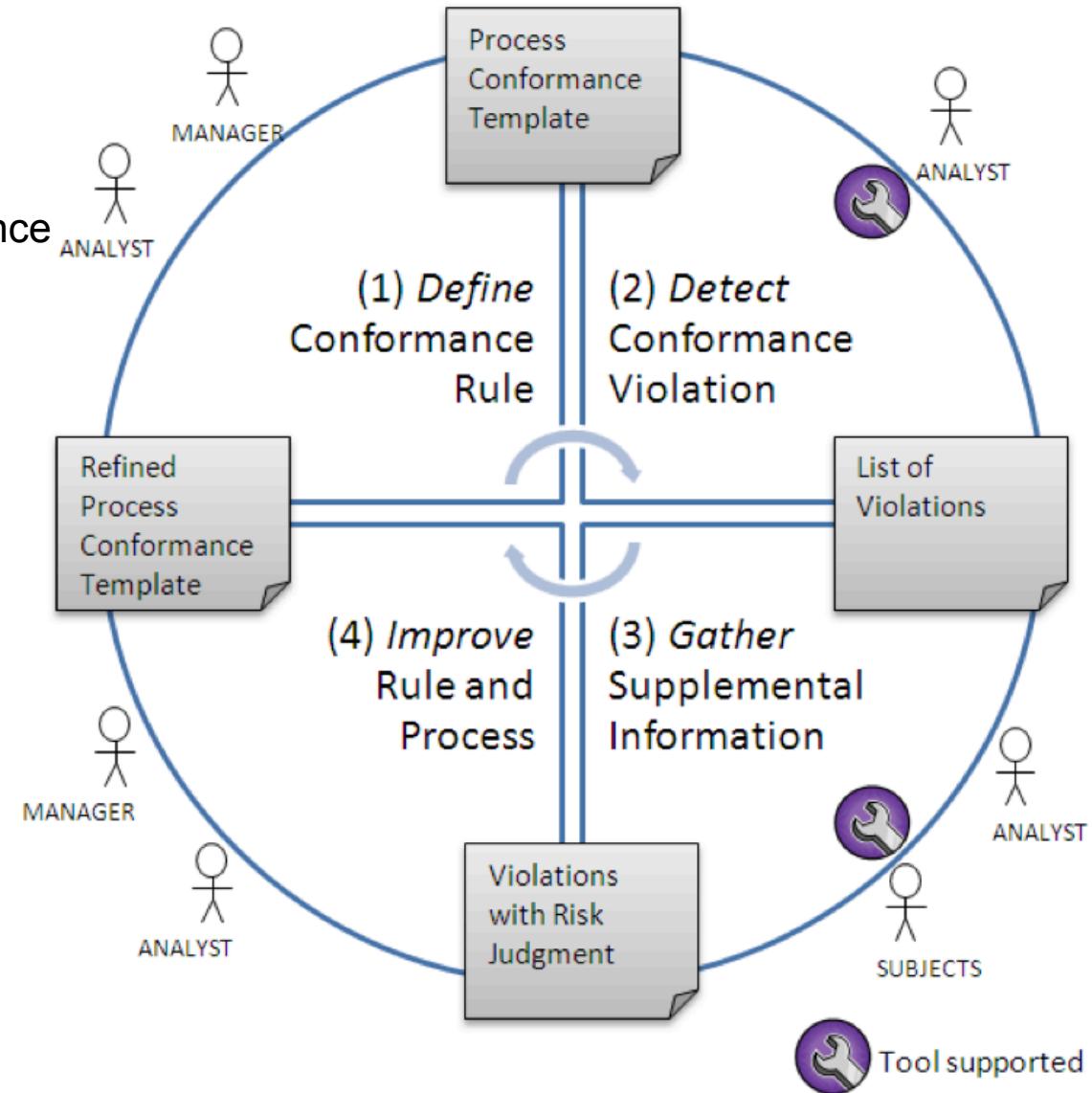
- Right!
 - If you need it, order it (and pay for it!)
- Really?
 - Even with the generated / embedded documentation?
 - If yes: order it and loose agility!
- Four-step approach
 - 1 Map agile activities to rules
 - 2 Include QM unit
 - 3 Negotiate about what is still missing
 - 4 Build an agile variant with QM
- Not if you take it all
 - That would be magic
 - Travel light: Abandon inefficient parts
 - Keep all agile quality aspects!



Process Conformance



- Measure conformance
 - Manage agile teams
 - Control for non-conformance in scientific studies



Example:

Conformance Template

Pr. Name	Test-Driven Development
Pr. Focus	Improved correctness.
Process Description	For each component (i.e. Java class) developers are supposed to create a JUnit test class (collection of test cases) prior to the development of the component.
Collected Data	Subversion code history. Developers are advised to use following file naming scheme for implementation and test classes: Implementation class: SomeName.java Test class: SomeNameTest.java
Process Violations	Syntactic: (1) Implementation classes (but not interface classes) without test classes. Violation detection: Implementation class is checked into the Subversion repository before its according test class. Semantic: (1) The line coverage of the test cases is below 70% (2) The branch coverage of the test cases is below 70%

Example:

Conformance Template

Process Name	Continuous Refactoring
Process Focus	Improved maintainability (extendibility).
Process Description	Refactoring activities should be a continuous part of code development.
Collected Data	<ul style="list-style-type: none">Manually: SVN commit template includes change type (e.g. refactoring)Implicitly: SVN data provides us with information about changes of architecture. Further Code Metrics /Code Smells can provide insight into decay of code.
Process Violations	<p>Syntactic:</p> <ol style="list-style-type: none">No refactoring activities in the commit template at all (during whole project)Large refactoring only in a single stage (e.g. at the end of the project) <p>Semantic:</p> <p>(3) Increasing amount of God Class code smells</p>

Example:

Conformance Template

Process Name	Collective Code Ownership (Pair Programming + Pair Switching)
Process Focus	Code is collectively owned, high Truck Factor
Process Description	Pair Switching: subjects are supposed to switch their pair programming partner with each new story card and between iterations.
Collected Data	Manually: SVN commit template include name of programmers and story card number
Process Violations	Syntactic: (1) The same developer pair working together on two consecutive <i>story cards</i> (2) The same developer pair working together on two consecutive <i>iterations</i> Semantic: (1) The project's Truck Factor (explained later is low)

Collective-Code Ownership

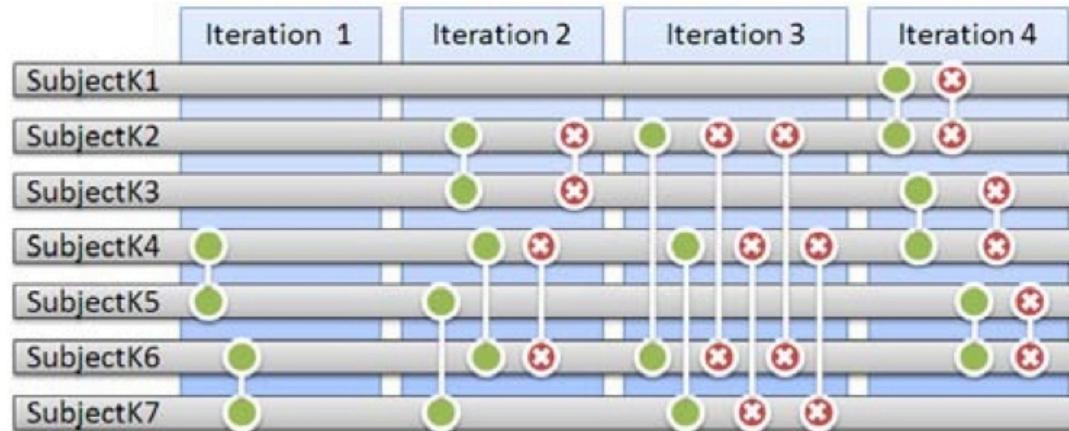


Figure 4: Pair-Switching for team KlaRa.

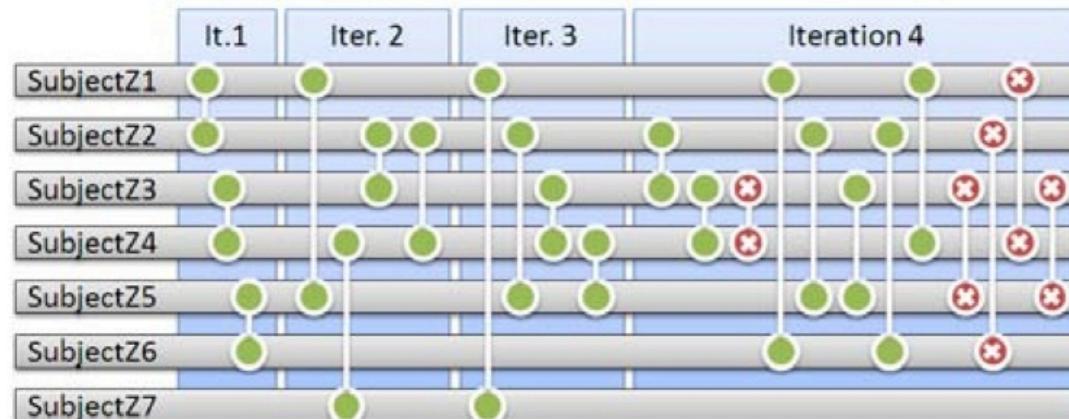
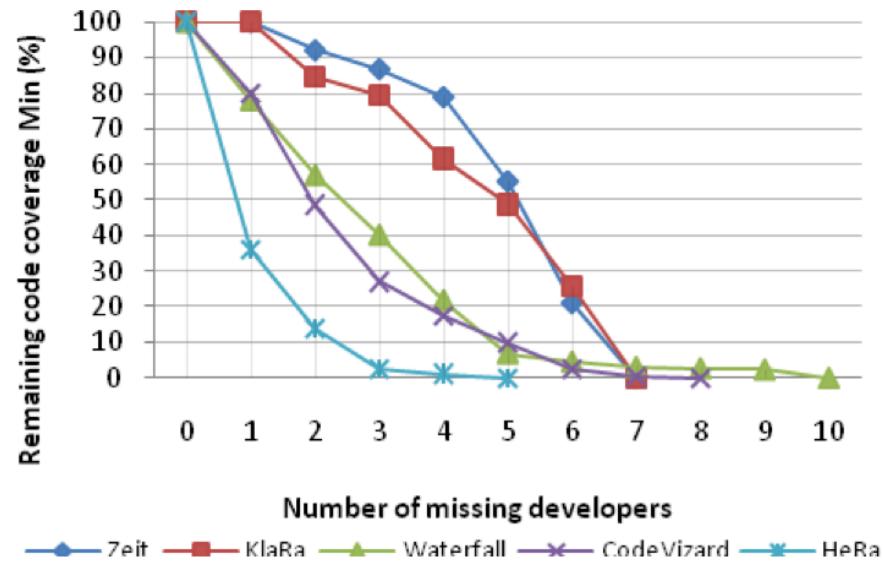
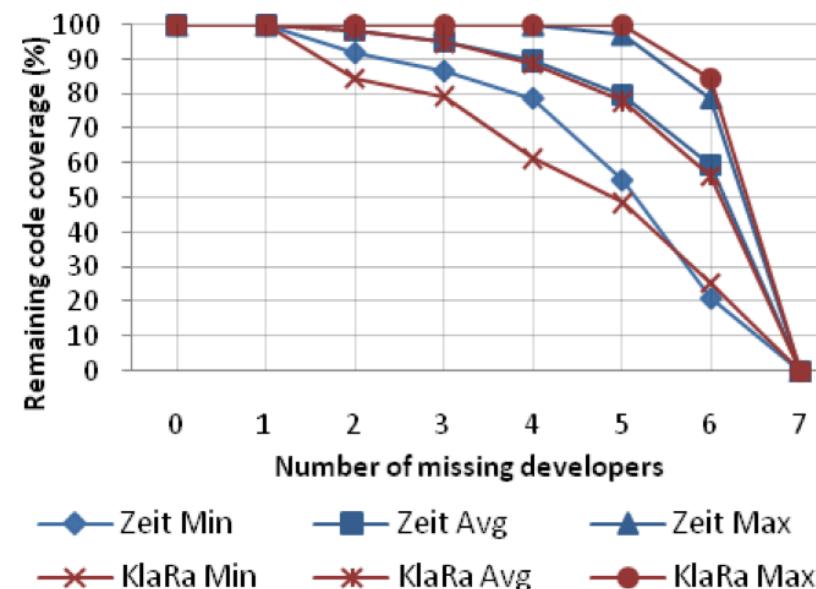


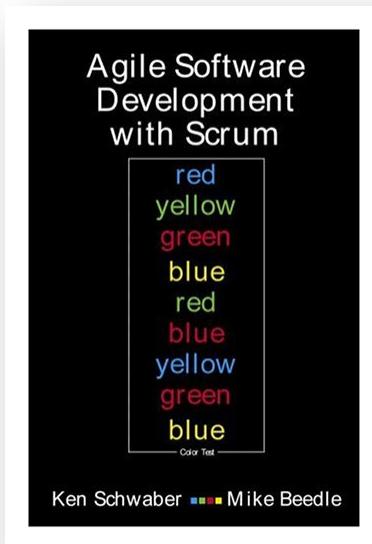
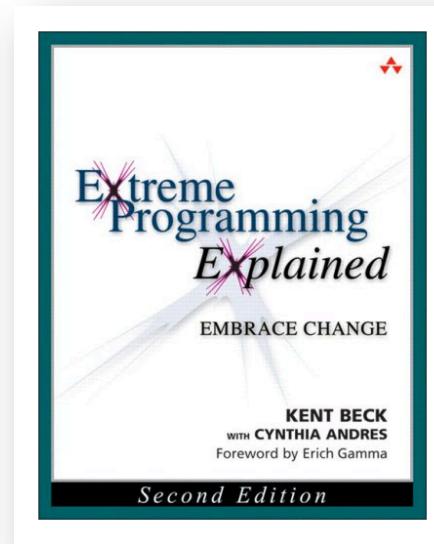
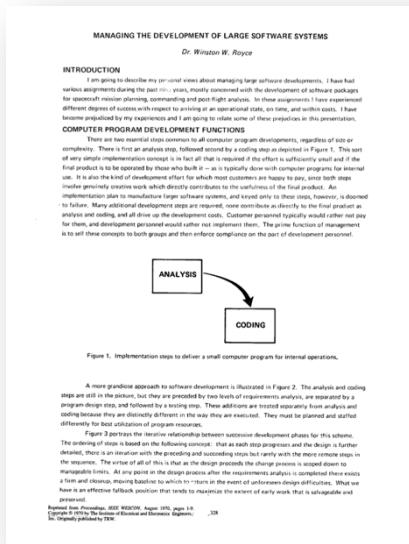
Figure 4: Pair-Switching for team Zeit

Truck factor analysis



Knowledge and understanding	Skills and ability	Judgement and approach
Compare agile and traditional softw. dev,	Forming a team organically	Explain: people/commun. centric dev.
Relate lean and agile development	Collaborate in small software dev. teams	Apply fact: people drive project success
Contrast different agile methodologies	Interact and show progress continuously	Describe: No single methodology fits all
Use the agile manifest and its accompanying principles	Develop SW using small and frequent iterations	Discuss: methodology needs to adopt to culture
Discuss what is different when leading an agile team	Use test-driven dev. and automated tests	
	Refactor a program/design	
Sprint 2	Be member of agile team	
	Incremental planning using user stories	

Thanks! (...and optional further reading)



Winston W. Royce:
Managing the
Development of Large
Software Systems. In:
Proceedings of IEEE
WESCON, pages 1-9,
1970

Kent Beck: Extreme
Programming
Explained. Addison-
Wesley, 2000

Ken Schwaber and
Mike Beedle: Agile
Software
Development with
Scrum, Prentice Hall,
2002

