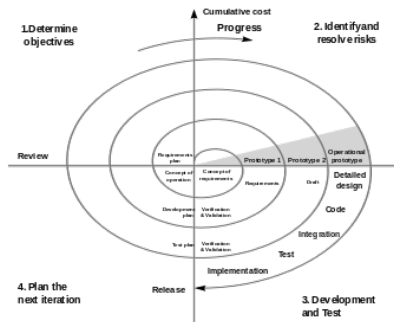
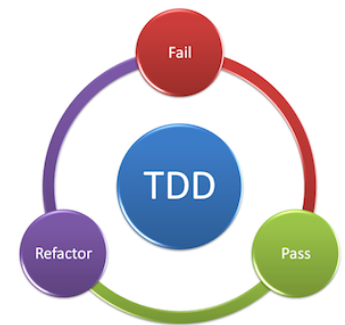


The software development process

Food for thought



Maria Grazia Pia
INFN Genova, Italy
 Maria.Grazia.Pia@cern.ch



<http://www.ge.infn.it/geant4/training/APC2016/>

MISSION: IMPOSSIBLE

the software development process in ½ hour

Introduce concepts and methods,
which will be discussed in following lectures

Pills of wisdom

Food for thought

Curiosity

Background for further learning

...feel free to contact us after the school!

Cowboy programming

Emphasis on ingenious artistry

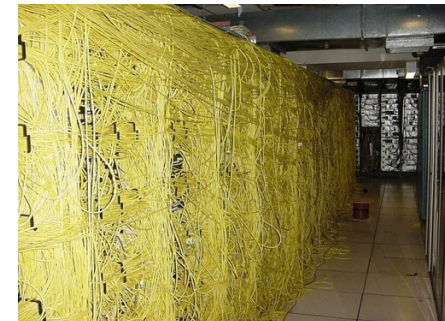


- Galloping off on one's own without a prior plan
- Brute-force programming
- Uncertain design requirements, code rewrite
- Quick and dirty: code and fix later
- Lack of comments, documentation, reviews
- Reinventing the wheel

**The results are often spotty
and difficult to duplicate**

Inexperienced developers are unfamiliar with
technologies and **methodologies**
that support producing quality software effectively

Software development methods and techniques are
seldom part of academic programs for physics degrees



F. P. Brooks,

“**No Silver Bullet** - Essence and Accidents of Software Engineering”

IEEE Computer, vol. 20, no. 4, pp.10-19, April 1987

As we look to the horizon of a decade hence, we see **no silver bullet**. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity.

...

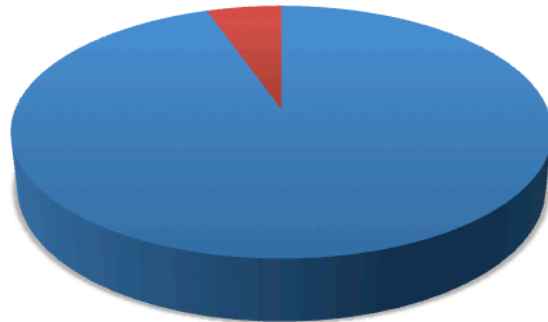
Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any - **no inventions** that will do **for software** productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did **for computer hardware**. We cannot expect ever to see twofold gains every two years.

...

Although we see no startling breakthroughs - and indeed, I believe such to be inconsistent with the nature of software - many encouraging innovations are under way. A **disciplined, consistent effort** to develop, propagate, and exploit these innovations should indeed yield an order-of-magnitude improvement. There is no royal road, but there is a road.

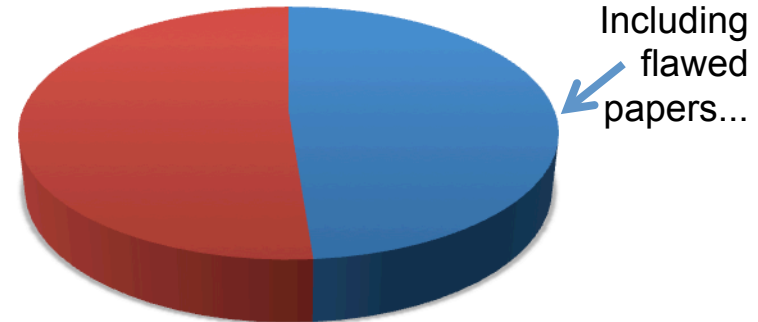
Producing results

People



■ Geant4 collaboration ■ Our team

Publications



■ Geant4 collaboration ■ Our team

2003-2013

Geant4 core subjects
(no applications)

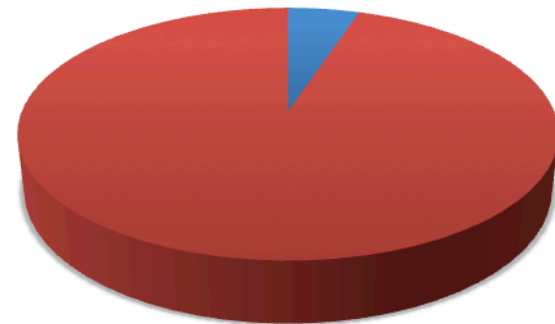
2 Geant4 general papers excluded

Sources:

<http://geant4.web.cern.ch/geant4/results/publications.shtml>

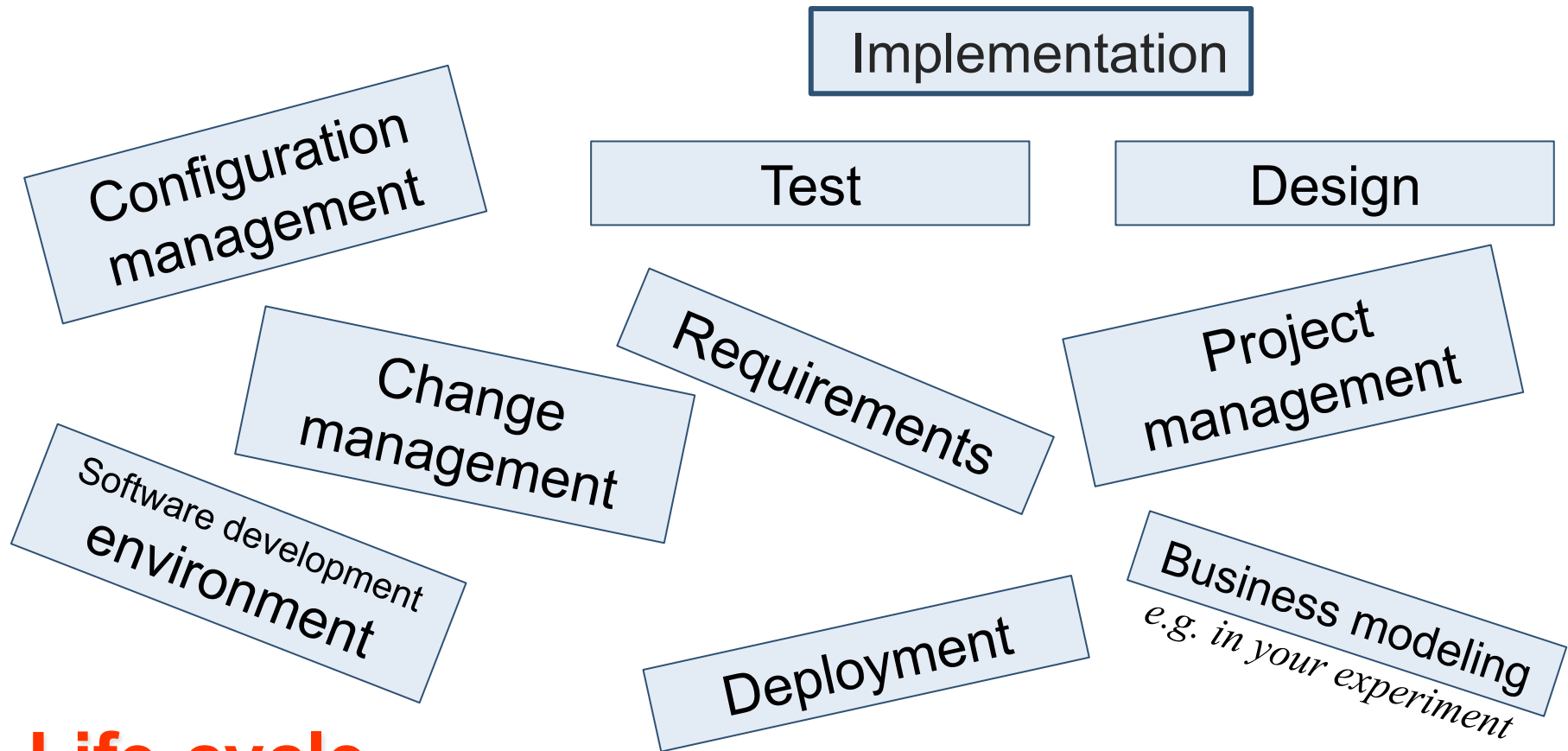
<http://www.ge.infn.it/geant4/papers/>

Average productivity



■ Geant4 collaboration ■ Our team

Much more than just hacking code...



Life-cycle

early stage, elaboration, construction, use in production...

Activities

Workflows

Products

Software **specification**

Define the **functionality**
of the software and
constraints on its operation

Software **design** and **implementation**

Produce software that
meets the specification

in a nutshell

Ensure that the software
does what one wants

Software **validation**

To meet **change**

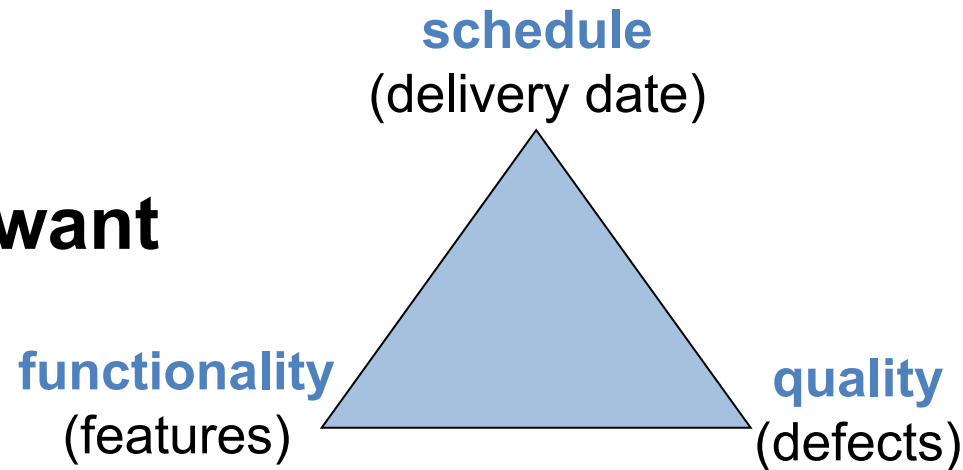
Software **evolution**

These complex **activities** include other sub-activities
e.g. requirements validation, architectural design, unit testing etc.

All of the above generate **products**
e.g. code, documentation, design diagrams, test results etc.
and involve responsibilities in various **roles**

Software development methodologies

What we want



Software development methodologies are **conceptual frameworks** to **structure**, **plan**, and **control** the process of developing the software

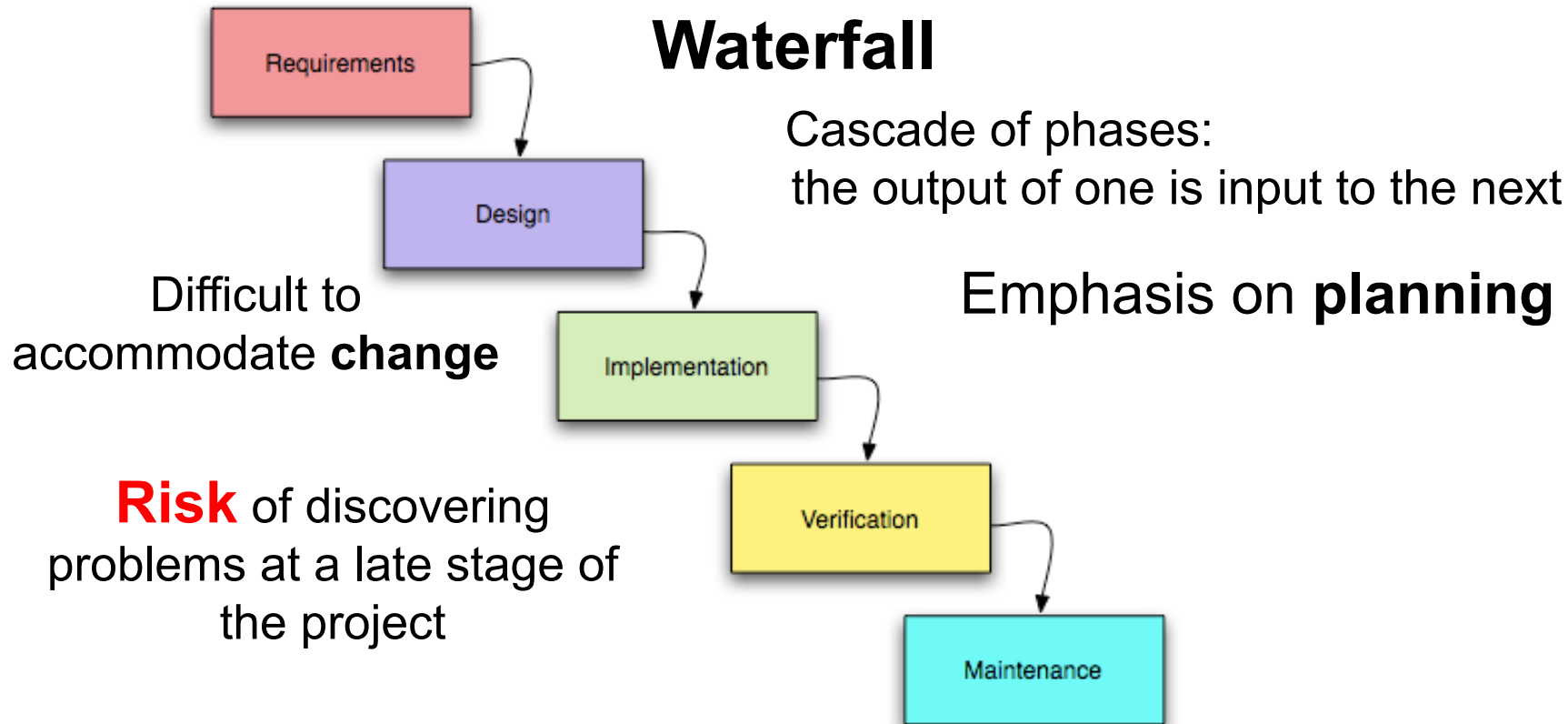
Usually built on **best practices** derived from experience on the field



Old, risky... and most common

Waterfall

Cascade of phases:
the output of one is input to the next



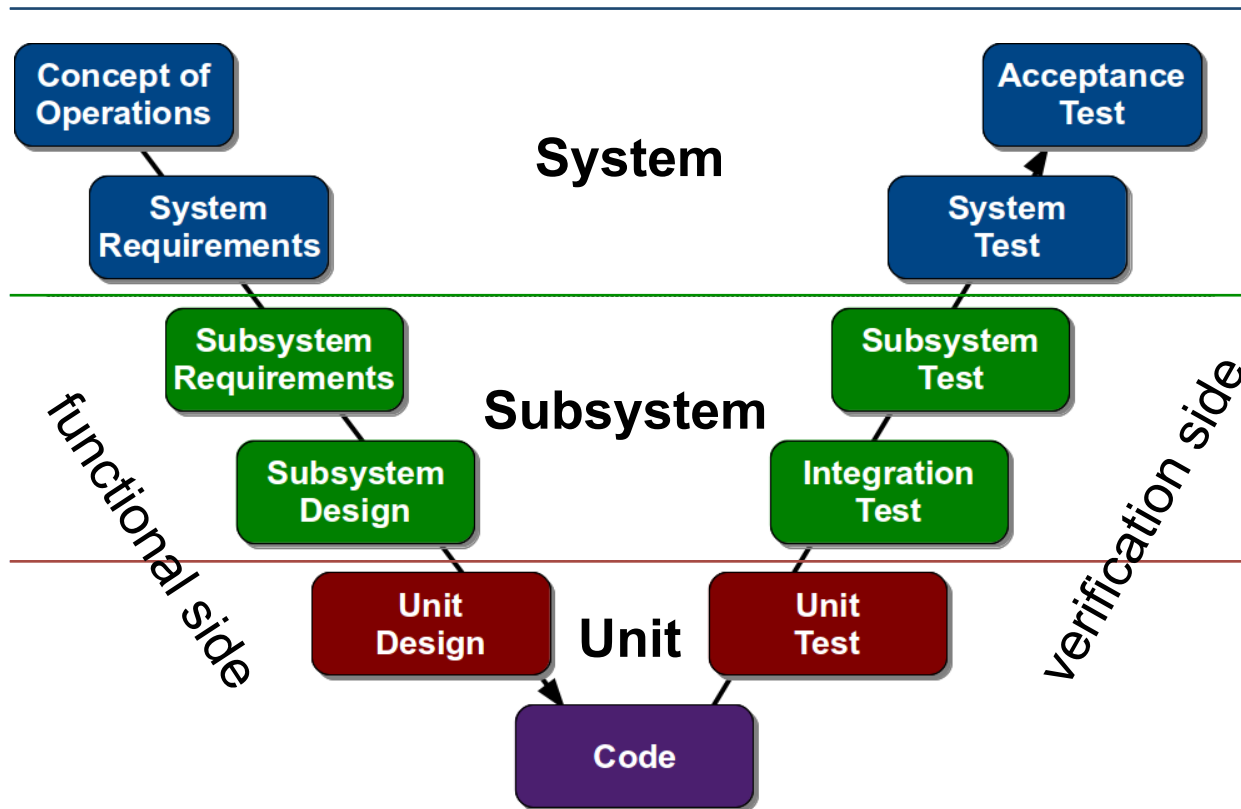
Best-suited to solving **well-understood problems**

“For most projects, the first system built is barely usable: too slow, too big, too hard to use, or all three.

Plan to **throw one away**; you will, anyhow.”

Fred Brooks, *The Mythical Man-Month*, Addison-Wesley, 1975-1995

Variants of waterfall development



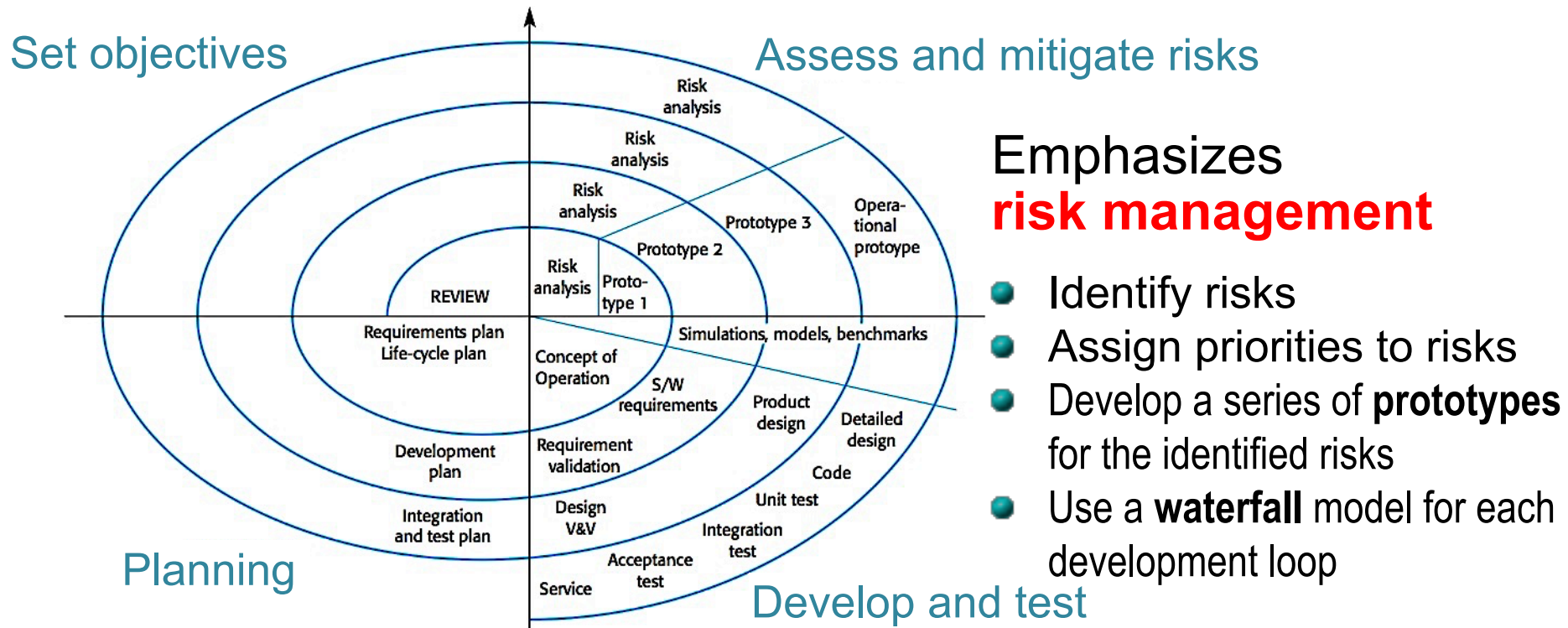
Emphasis on **testing** at all levels of software development

The software development proceeds once the details have been defined both on the functional side and on the verification side

Spiral development

Non-linear view of the software life cycle

It is a **meta-model**:
it can incorporate
other models



Emphasizes
risk management

- Identify risks
- Assign priorities to risks
- Develop a series of **prototypes** for the identified risks
- Use a **waterfall** model for each development loop

loop in the spiral = phase of software development

Barry W. Boehm, A Spiral Model of Software Development and Enhancement, *IEEE Computer*, vol. 21 no. 5, pp. 62-72, 1988

Multiple perspectives

Dynamic perspective

How the process is articulated over time

Static perspective

How activities are enacted

Practice perspective

Suggests good practices
to be adopted in the software
development

Risk management
Software reuse

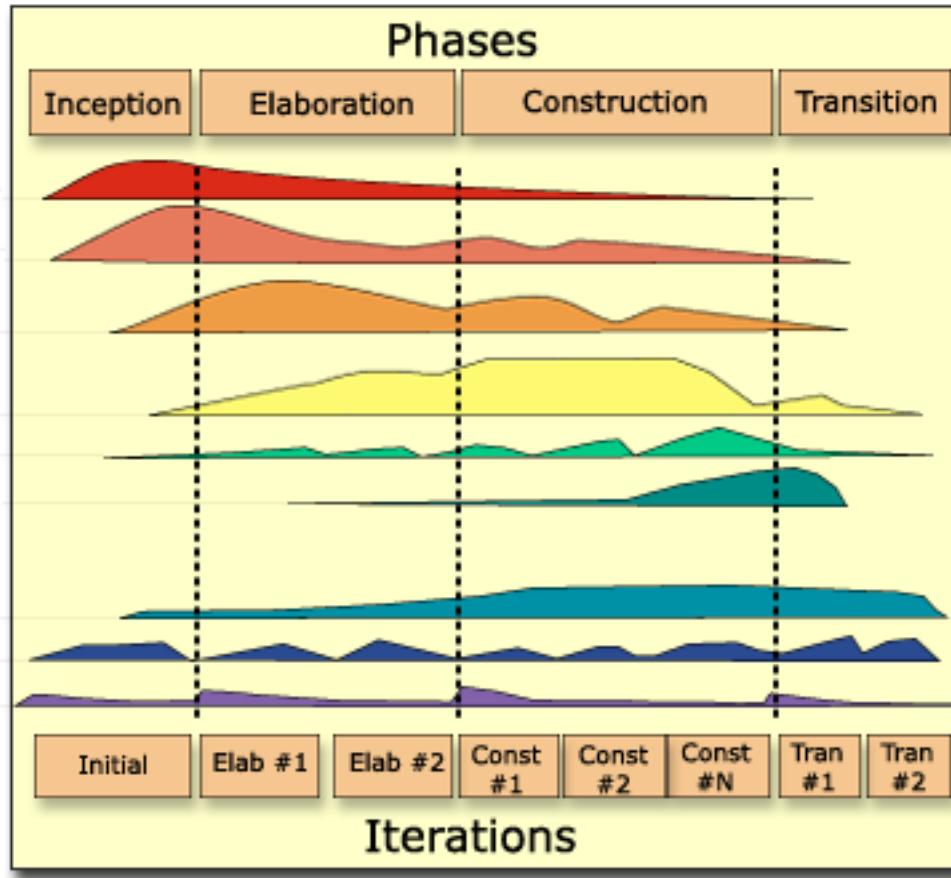
Unified Process (UP, USDP, RUP)

6 core process
workflows



Disciplines

Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration
& Change Mgmt
Project Management
Environment



3 supporting
workflows



“use case driven, architecture-centric”

**Iterative,
incremental
process**

Emphasis on
modeling
UP designed along
with UML

Apparently complex,
but highly
customizable

Best practices

Develop software
iteratively

Manage requirements

Use component-based
architectures

Visually model software

Verify software quality

Control changes to
software

- High priority features developed first
- Document requirements
- Keep track of changes to requirements
- Analyze the impact of changes before accepting them
- Structure the system into components
- UML: static and dynamic views
- Testing (and more)
- Change management system
- Configuration management procedures and tools

The agile manifesto

Kent Beck et al. (2001)

“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.”

Emphasis on

- Effective **communication** among all stakeholders
- **Adaptive** response to change
- **Rapid, incremental** delivery of software

B. Boehm, “**Get Ready for Agile Methods, With Care**”, *IEEE Computer*, 2002, <http://dx.doi.org/10.1109/2.976920>

A thoughtful critique of agile methods, their strengths and weaknesses, written by a very experienced software engineer

B. Meyer, **Agile!: The Good, the Hype and the Ugly**, Springer, 2014

Extreme Programming (XP)

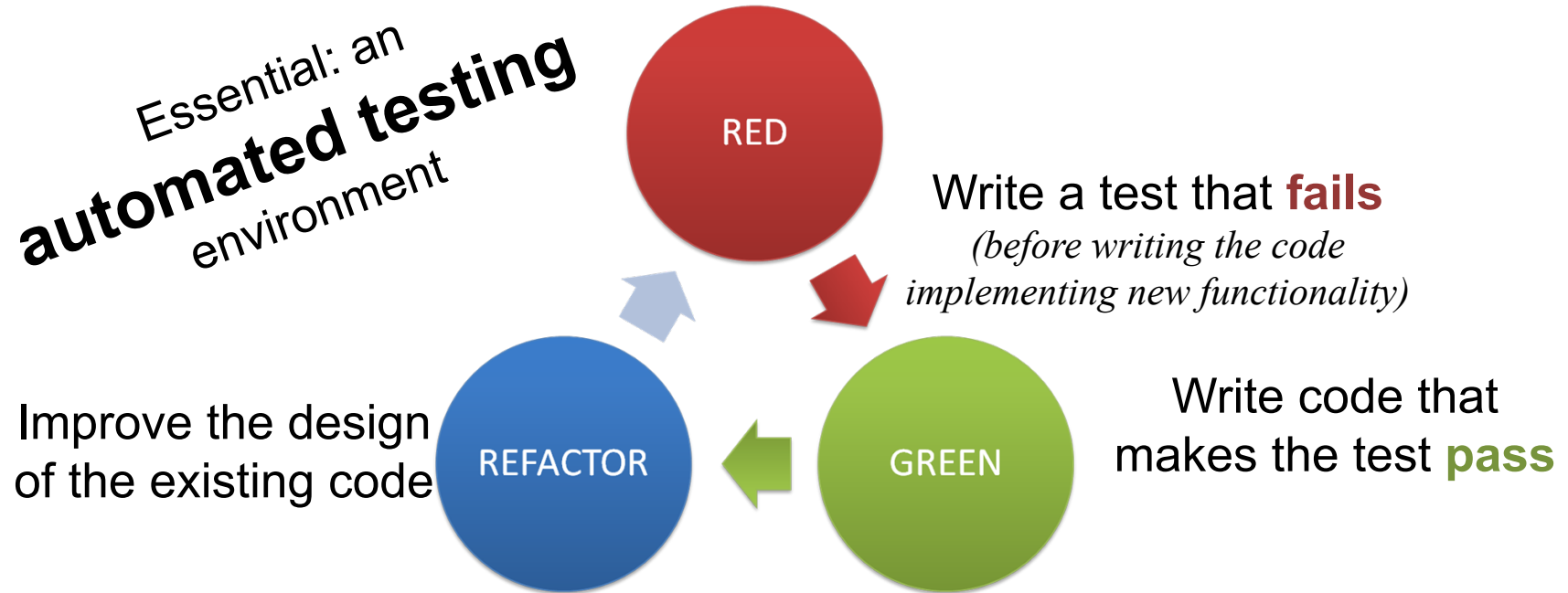
Pushes recognized good practices to the extreme

Software Engineering Practice		XP Principles
Code reviews are good	Review code all the time	Pair programming
Testing is good	Everybody tests all the time	Unit tests, TDD
Design is good	Part of daily business	Refactoring
Simplicity is good	Enough design to meet requirements and no more	
Architecture is important	Simple shared story of how the system works	Metaphor
Integration testing is important	Continuously integrate and test	Continuous integration
Short iterations are good	Make iterations really short	Planning game

Highly prescriptive, but often organizations adopting XP pick and choose

Emphasis on **quick, incremental, test-first** development

Test-driven development (TDD)



Facilitates **regression testing**

Discover problems early during the software development

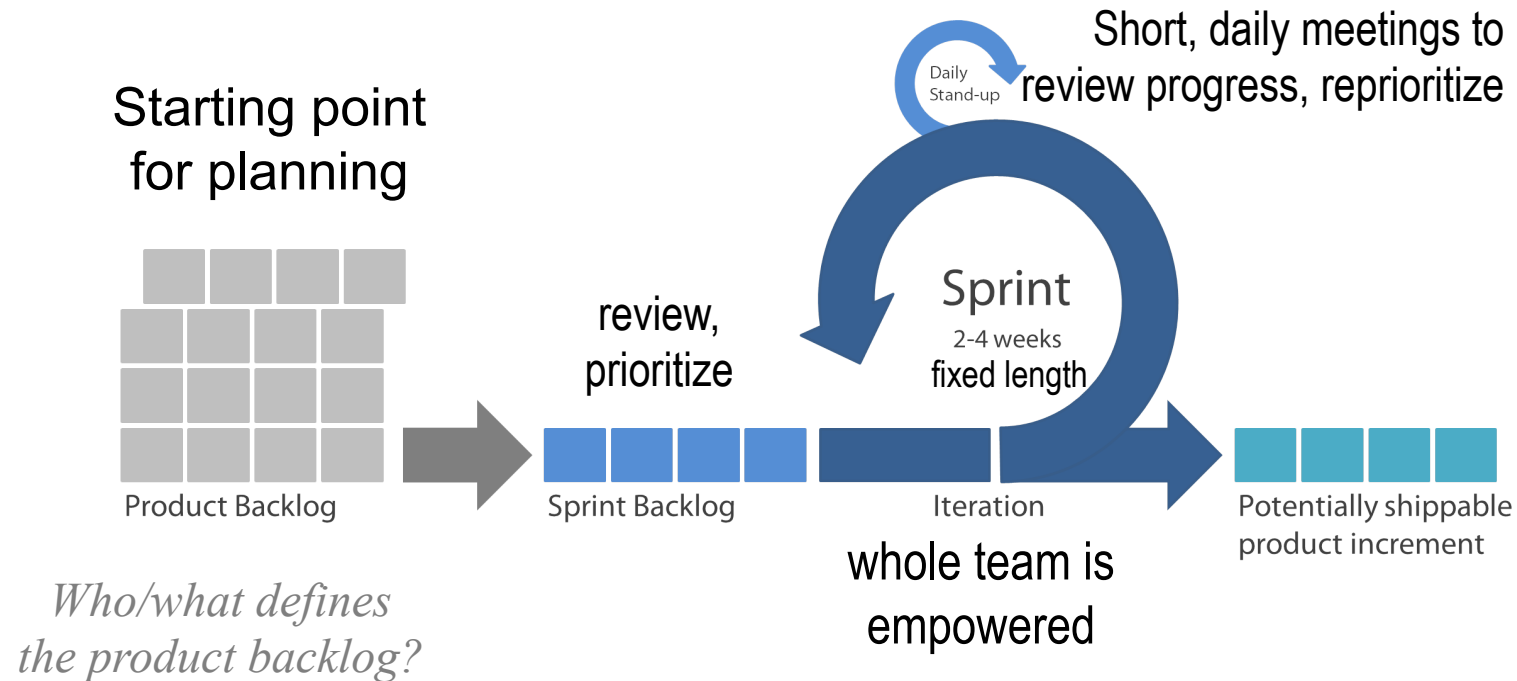
Limited to unit testing,

still need system testing, performance, reliability testing etc.

Suitable to **small-size projects**

Scrum

Project management for agile (incremental) development



Scrum master
is a facilitator

- arranges daily meetings
- tracks the backlog of work to be done
- records decisions
- measures progress against the backlog
- communicates with customers and management

K. Schwaber, and M. Beedle, Agile Software Development with Scrum, Prentice Hall, 2001

K. Schwaber, Agile Project Management with Scrum, Microsoft Press, 2004

Clean code

Programming hygiene

“Even bad code can function. But if code isn't clean, it can bring a development organization to its knees. Every year, countless hours and significant resources are lost because of poorly written code. But it doesn't have to be that way.”

● Meaningful names

- Classes, functions etc.

● Comments

- Do not make up for bad code...
- Good/bad

● Functions

- Small!
- Do one thing
- No side effects
- Arguments: zero, few

● Objects

- expose behavior, hide data

● Classes

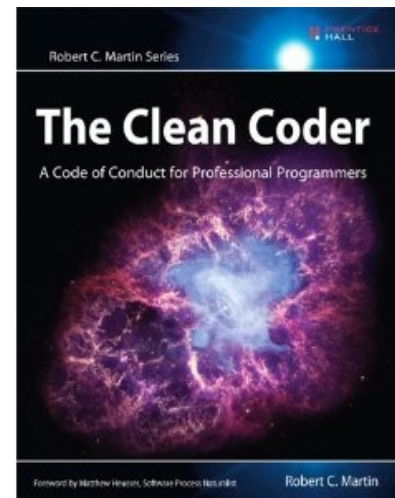
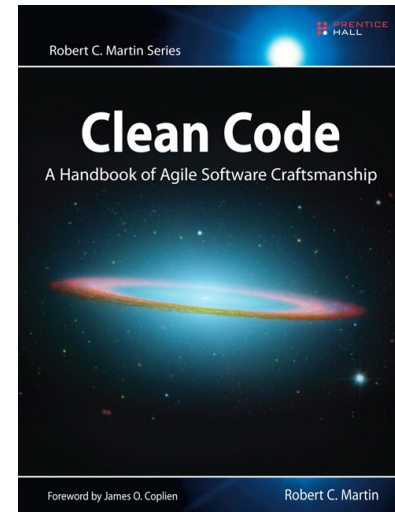
- Small!
- Encapsulation
- Cohesion
- Single Responsibility Principle

● Unit testing

- Clean
- Single concept per test

● Smells

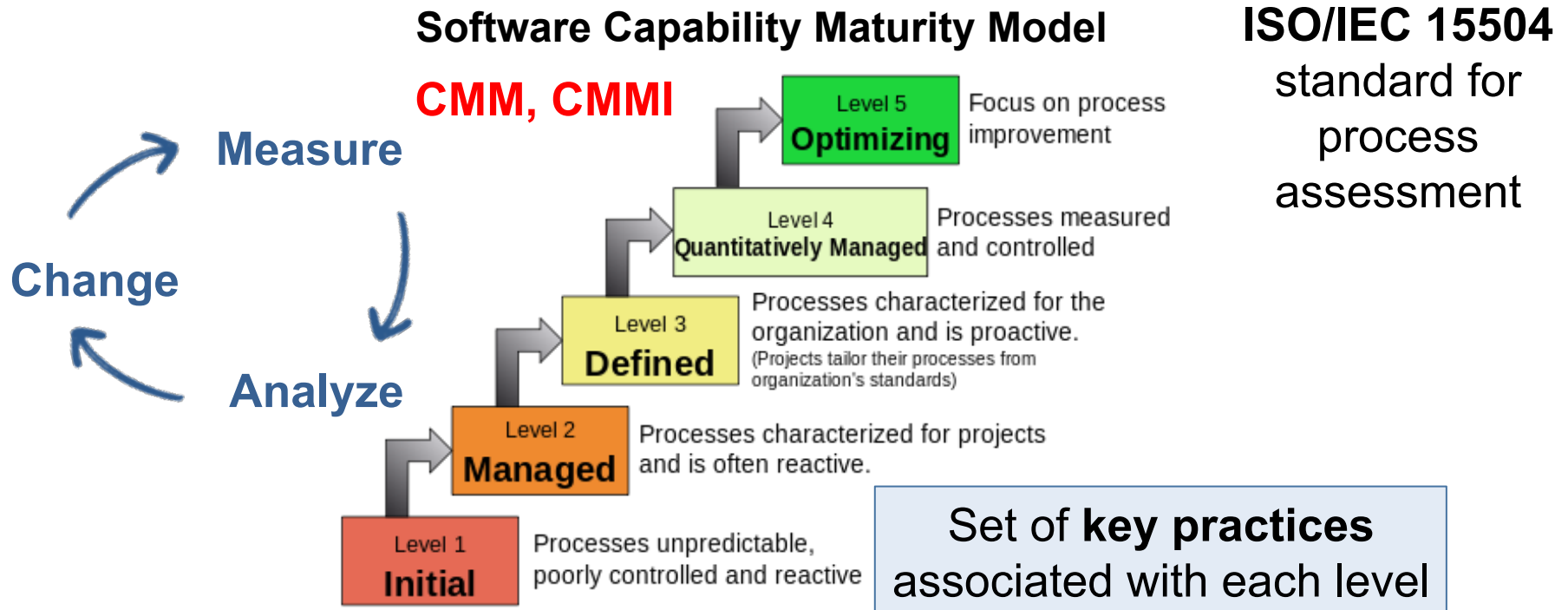
- Heuristics
- More in Refactoring



Improvement

Can we improve the way we develop software? How?

Improvement requires **measurement**: before/after



Helpful guidance towards adopting good practices
even without formal assessments

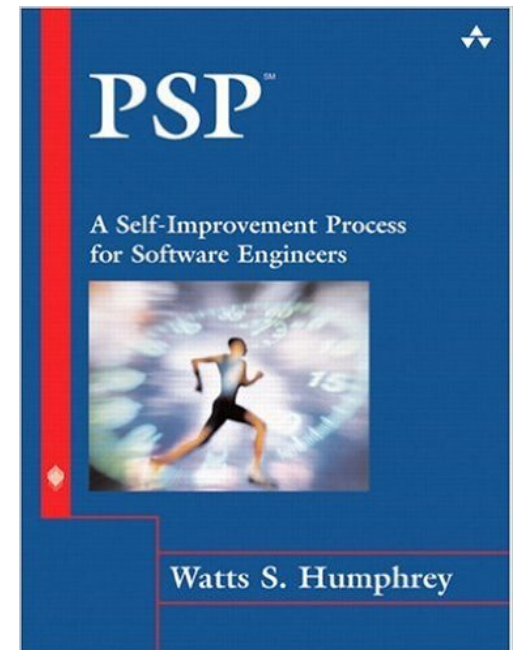
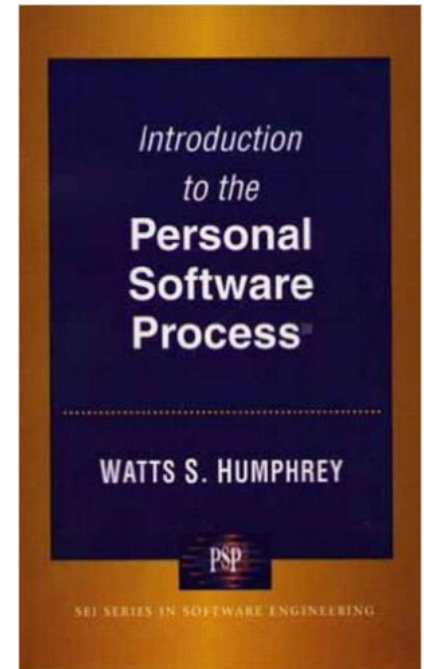
For singles

Emphasis is usually on software development
teams

What if I work at a project where I am
the only software developer?



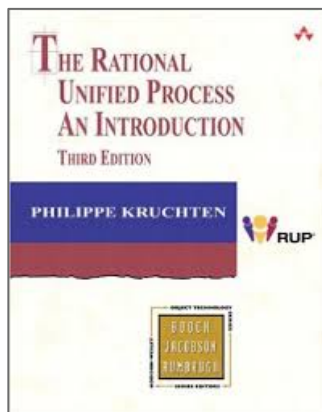
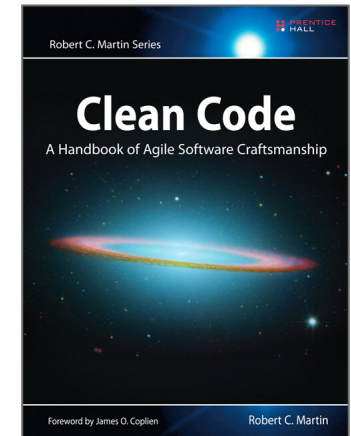
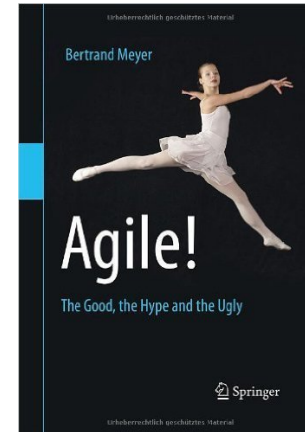
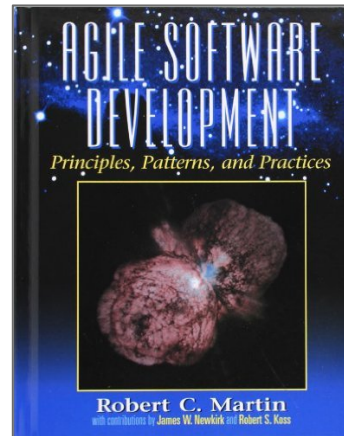
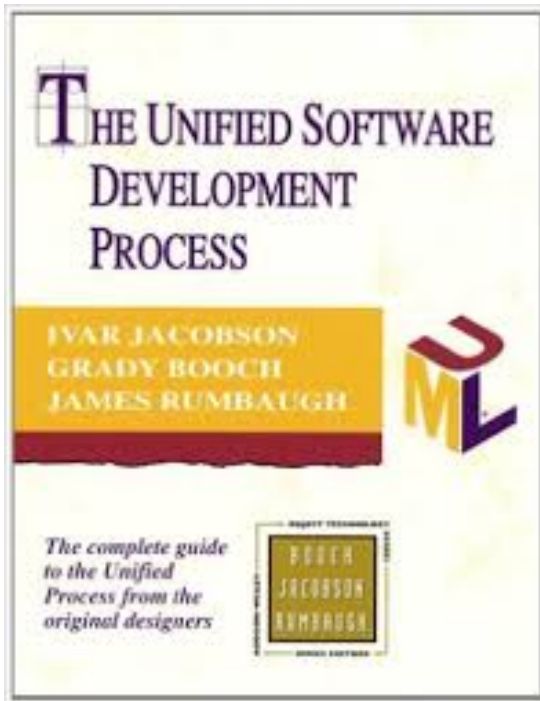
The screenshot shows the Rational Edge website. The header features the 'the Rational edge' logo with the tagline 'e-zine for the rational community'. Below the logo is a navigation bar with links: Features, Management, News, Rational Reader, Technical, Franklin's Kite, and Rational Develop. The main content area displays an article titled 'A Software Development Process for a Team of One' by Philippe Kruchten, Rational Fellow. A small blue triangle icon is to the left of the title.



One size does not fit all

- A software process **model** is a **simplified** representation of a software process
 - From a particular perspective
- **Many different approaches are possible**
 - Positive and negative sides in any of them
- **Good or bad** often depends on the **context**
 - Small/large scale project, short/long lifetime etc.
- Process frameworks may (should) be adapted and extended
 - A good software process is **tailored to the project**
- Grain of salt
 - Commercial vs. scientific environment
 - ...let's not forget that we are not only the developers, but also the customers!

Further learning



Get a mentor!