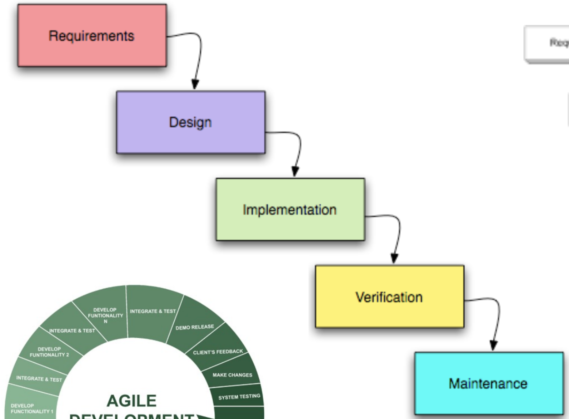
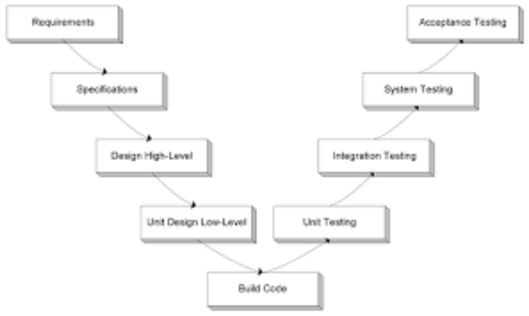
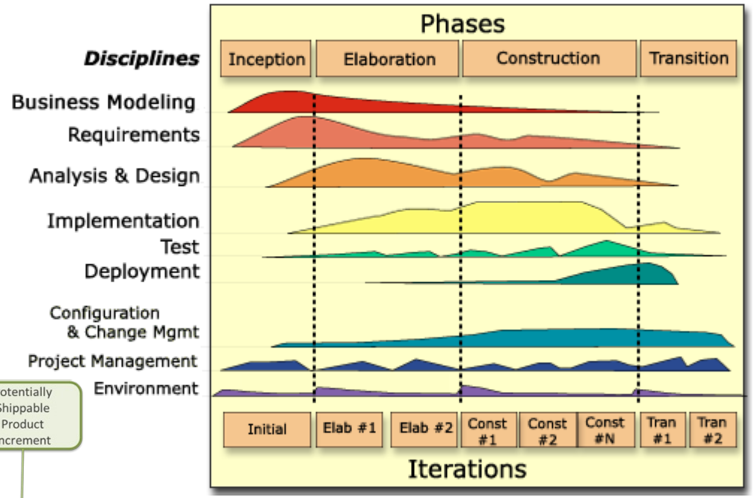
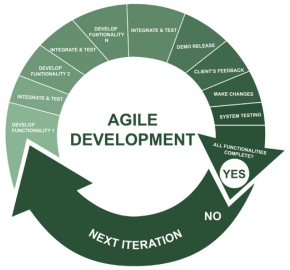
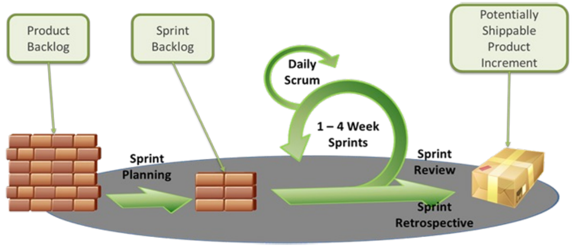


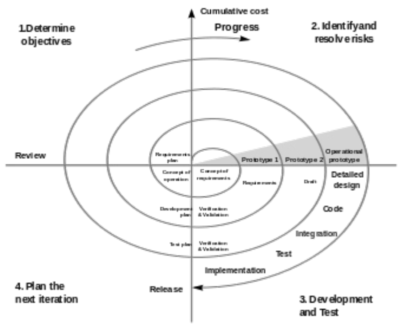


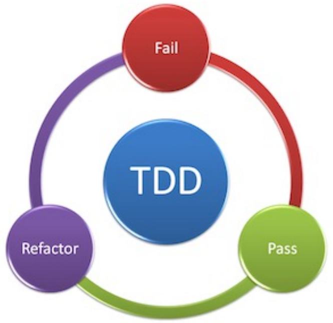






The software development process
Food for thought





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

http://www.ge.infn.it/geant4/training/APC2025/

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!

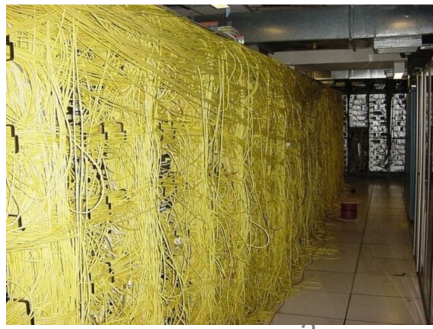
Cowboy programming

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.

...

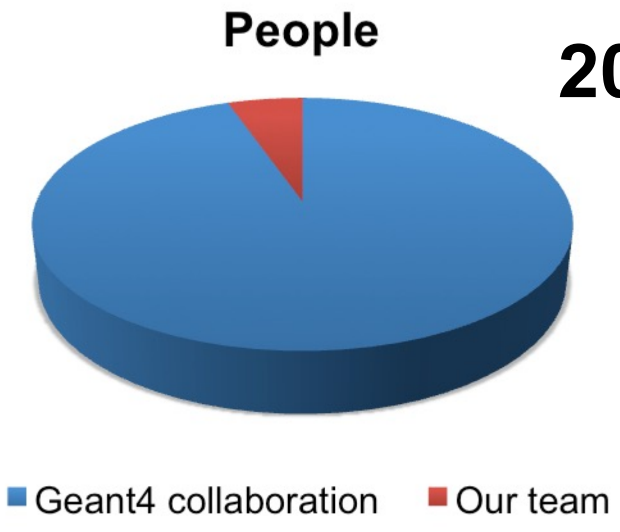
Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any - **no inventions** that will do **for software** productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did **for computer hardware**.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

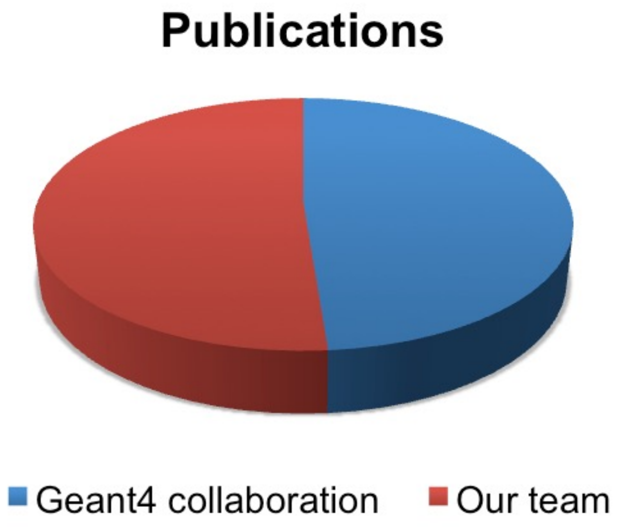
...

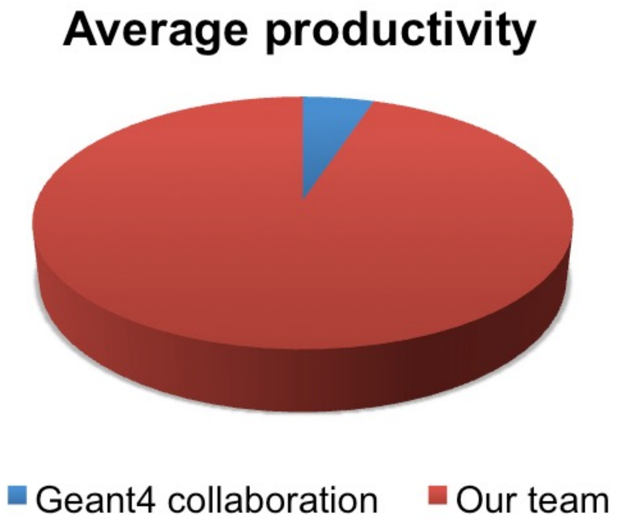
Although we see no startling breakthroughs - and indeed, I believe such to be inconsistent with the nature of software - many encouraging innovations are under way. A **disciplined, consistent effort** to develop, propagate, and exploit these innovations should indeed yield an order-of-magnitude improvement.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

Does it help?

2003-2013







http://geant4.web.cern.ch/geant4/results/publications.shtml
http://www.ge.infn.it/geant4/papers/

Much more than just hacking code…

Implementation

Configuration management

Test

Design

Change management

Requirements

Project management

Software development environment

Business modeling

e.g. in your experiment

Deployment

Life-cycle

early stage, elaboration, construction, use in production…

Activities **Workflows** **Products**

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

Software **specification**

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

Produce software that meets the specification

Ensure that the software **does what one wants**

in a nutshell

Software **verification** and **validation**

Software **evolution**

Deal with **change**

These complex **activities** include other sub-activities

e.g. requirements validation, architectural design, unit testing etc.

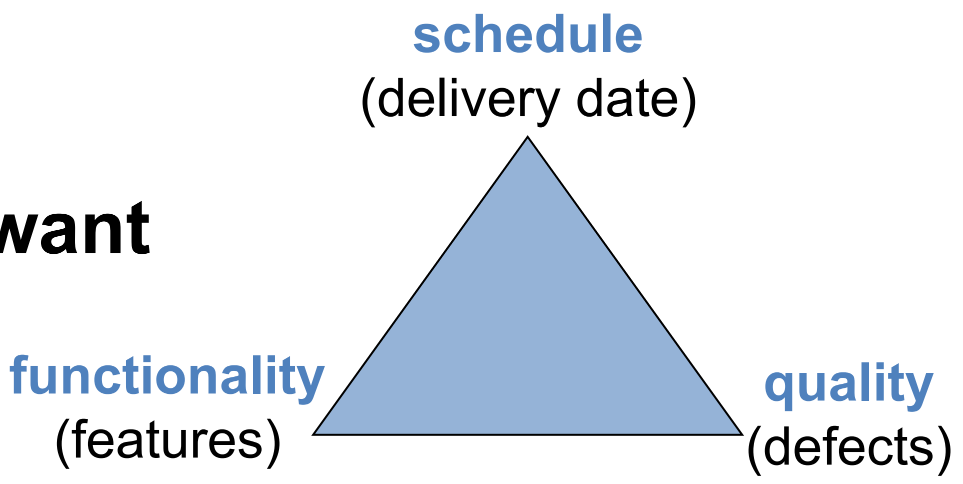
All of the above generate **products**

e.g. code, documentation, design diagrams, test results etc.

and involve responsibilities in various **roles**

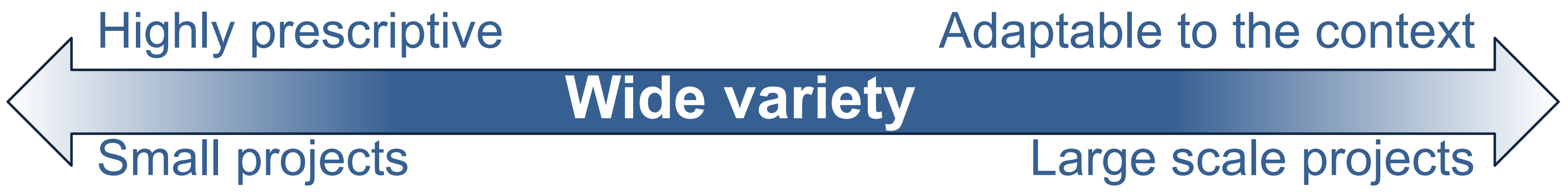
Software development methodologies

What we want

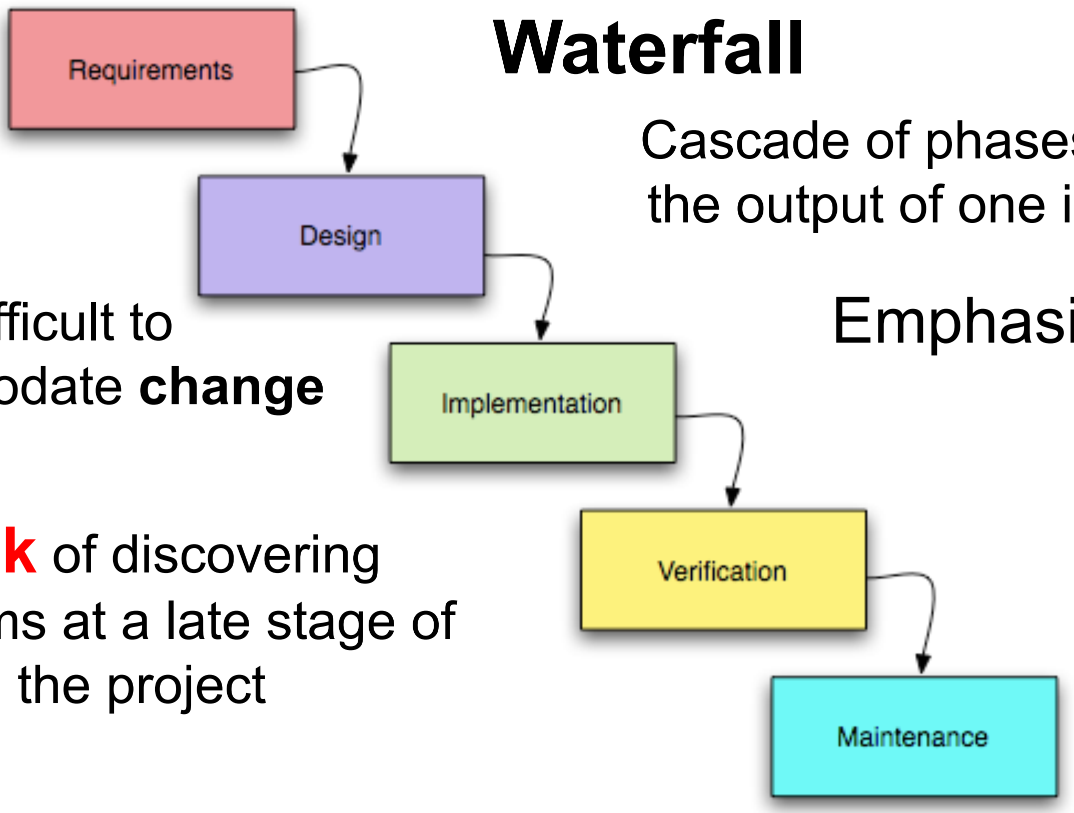


Software development methodologies
are **conceptual frameworks** to **structure**, **plan**, and **control**
the process of developing software

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

Emphasis on **planning**

Difficult to accommodate **change**

Risk of discovering problems at a late stage of the project

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

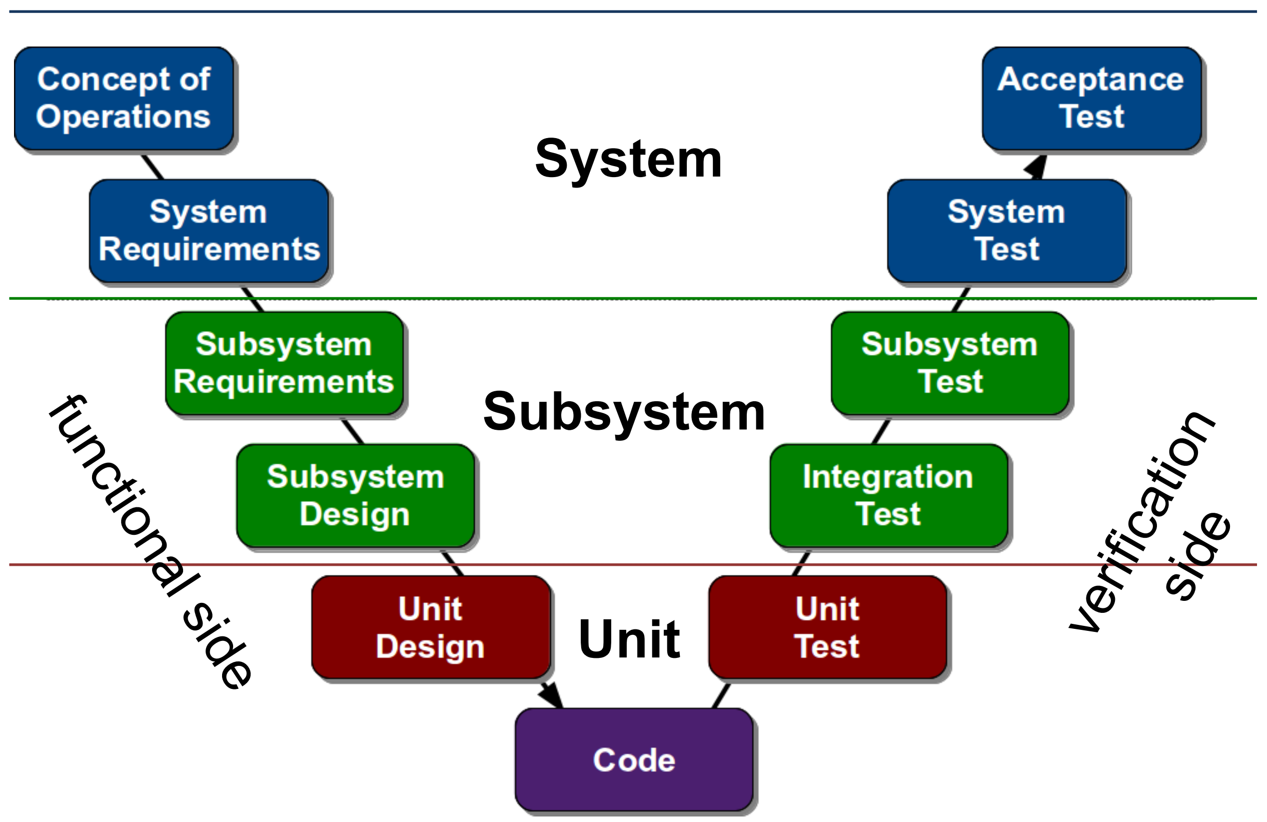
Maria Grazia Pia, *INFN Genova*

“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



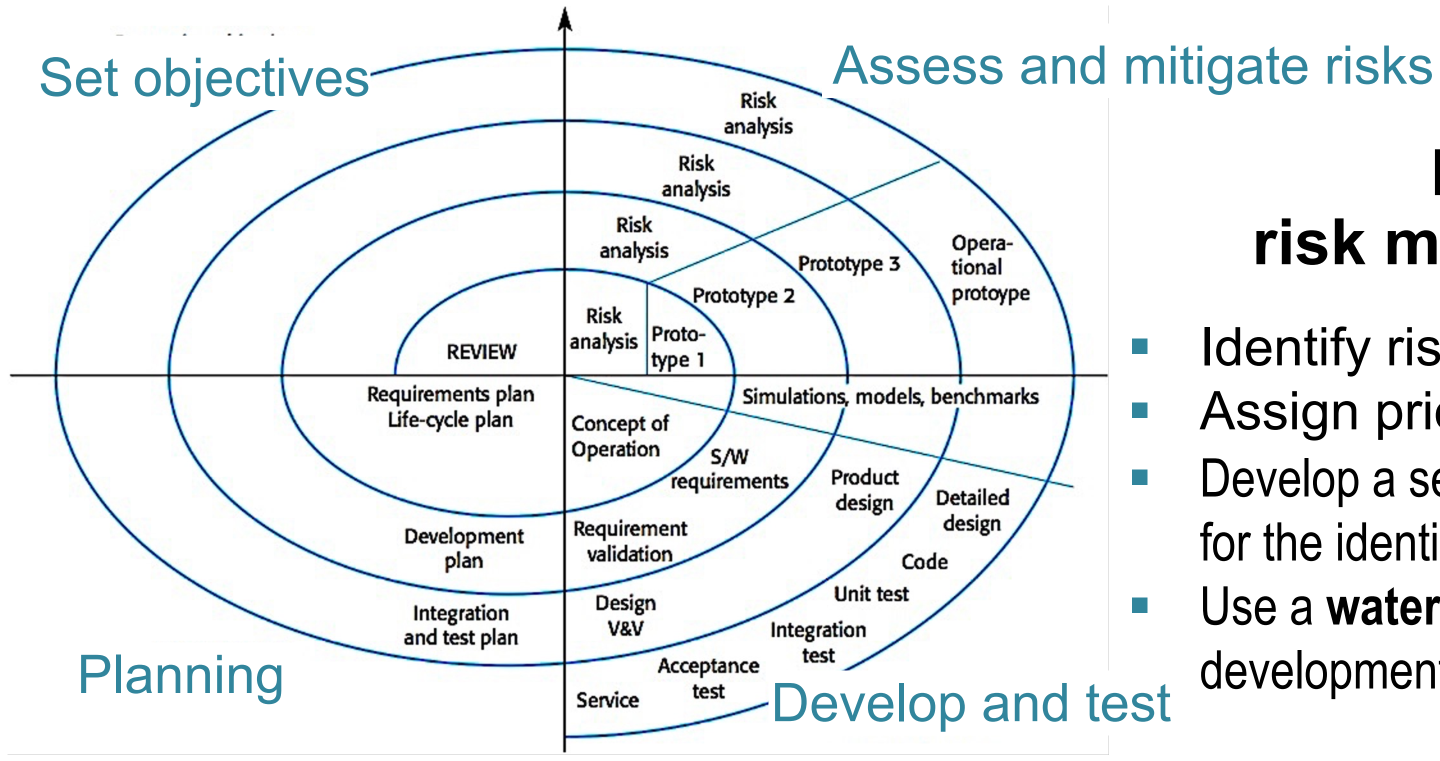
Emphasis on **testing** at all levels of software development

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

Maria Grazia Pia, *INFN Genova*

Spiral development

Non-linear view of the software life cycle



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

Unified Process (UP, USDP, RUP)

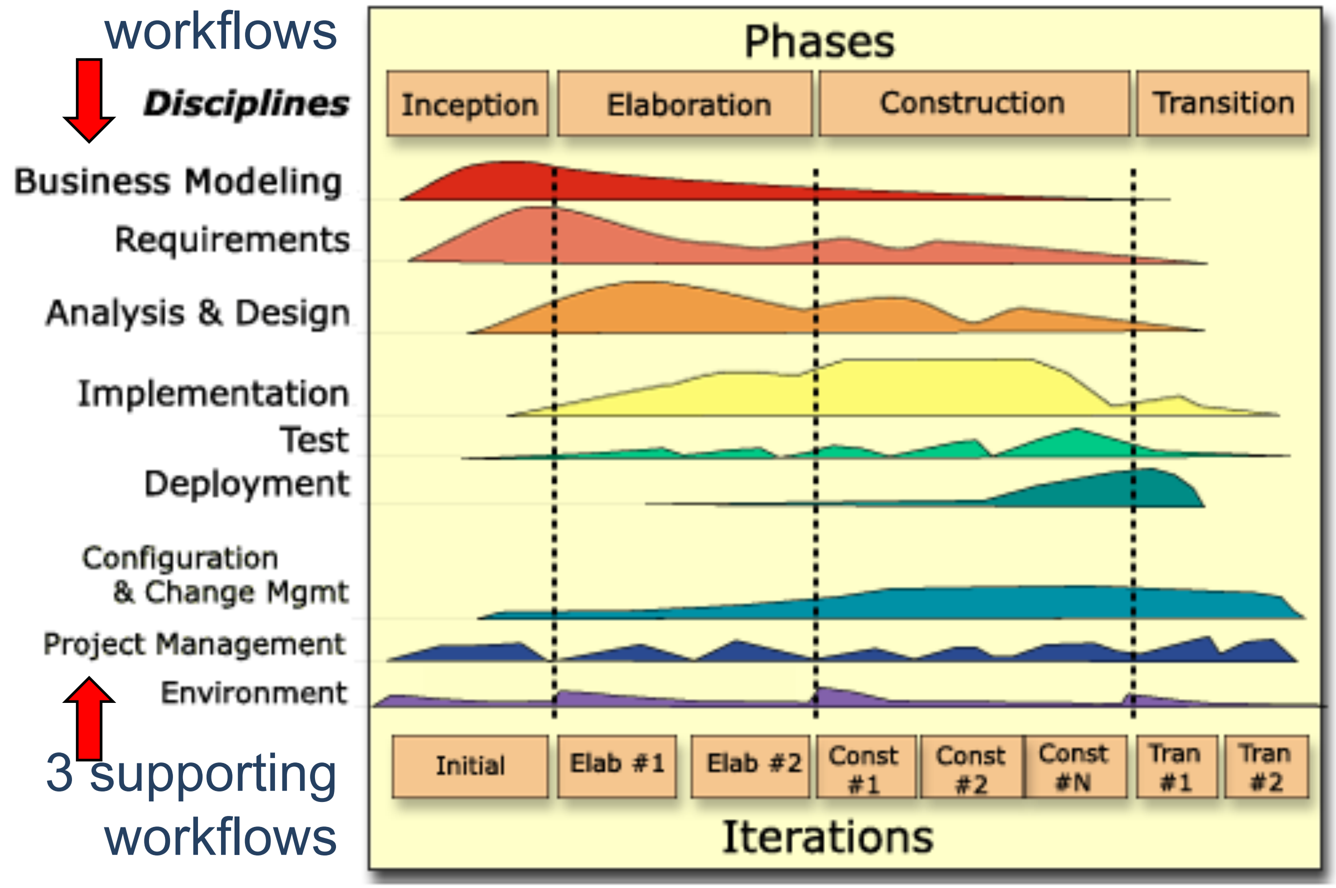
Iterative, incremental process

Static perspective

6 core process workflows

3 supporting workflows

Dynamic perspective



Emphasis on **modeling**

Designed along with UML

Apparently complex, but highly **customizable**

Best practices

Develop software iteratively	• High priority features developed first
Manage requirements	• Document requirements • Keep track of changes to requirements • Analyze the impact of changes
Use component-based architectures	• Structure the system into components
Visually model software	• UML: static and dynamic views
Verify software quality	• Testing (and more)
Control changes to software	• 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
Testing is good	Everybody tests all the time
Design is good	Part of daily business
Simplicity is good	Enough design to meet requirements and no more
Architecture is important	Simple shared story of how the system works
Integration testing is important	Continuously integrate and test
Short iterations are good	Make iterations really short

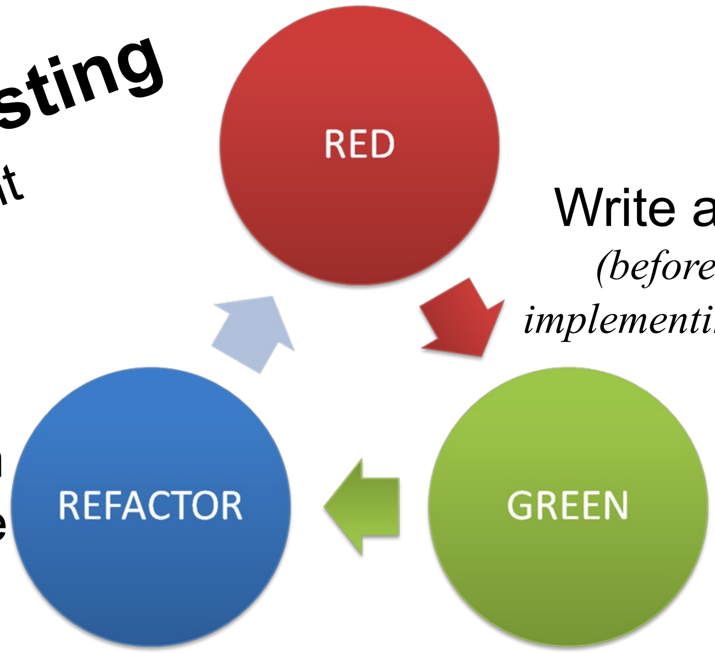
Highly prescriptive, but often organizations adopting XP pick and choose

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

K. Beck, Extreme Programming explained, Addison-Wesley, 2000

Test-driven development (TDD)

Essential: an **automated testing** environment



Write a test that **fails**
(before writing the code implementing new functionality)

Write code that makes the test **pass**

Improve the design of the existing code

Facilitates **regression testing**

Discover problems early during the software development

Limited to unit testing,

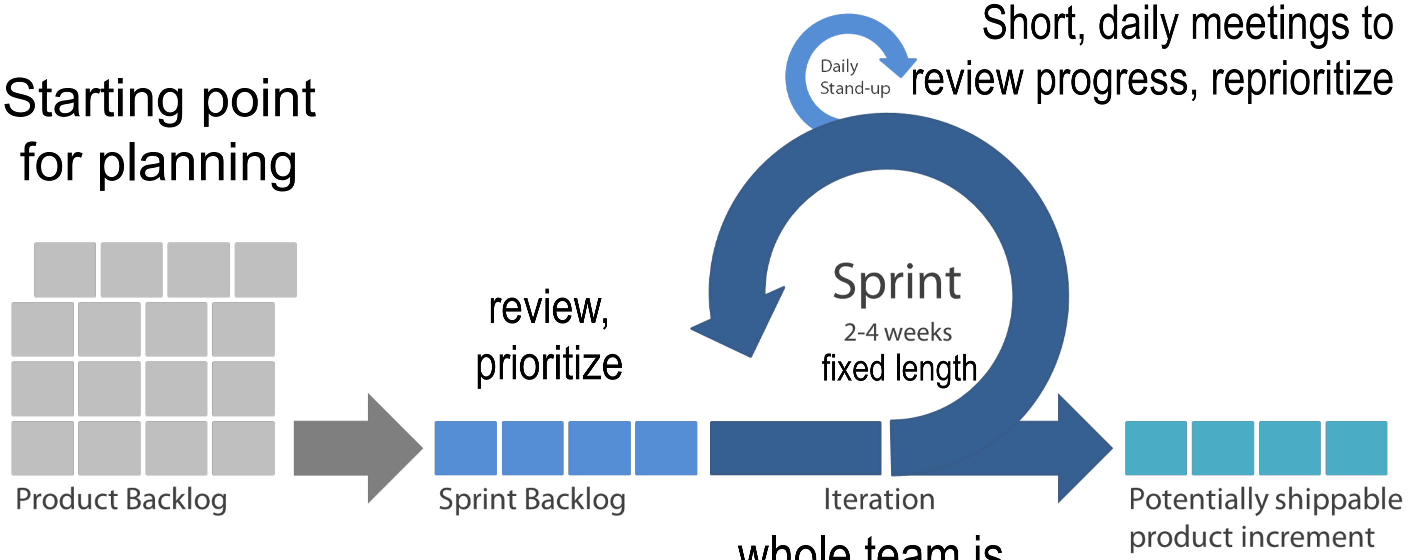
still need system testing, performance, reliability testing etc.

Suitable to **small-size projects**

D. Astels, Test Driven Development: A Practical Guide, Prentice Hall, 2003

Scrum

Project management for agile (incremental) development



Starting point for planning

review, prioritize

Short, daily meetings to review progress, reprioritize

fixed length

whole team is empowered

Who/what defines the product backlog?

Scrum master is a facilitator

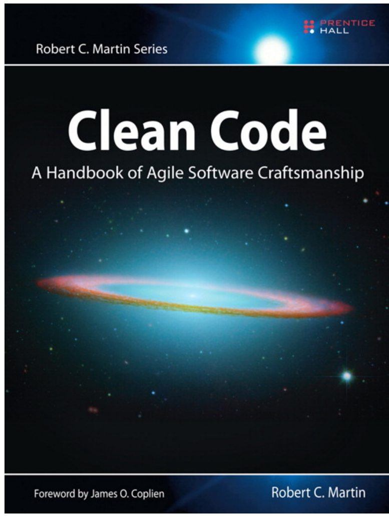
- arranges daily meetings
- tracks the backlog of work to be done
- records decisions
- measures progress against the backlog
- communicates with customers and management

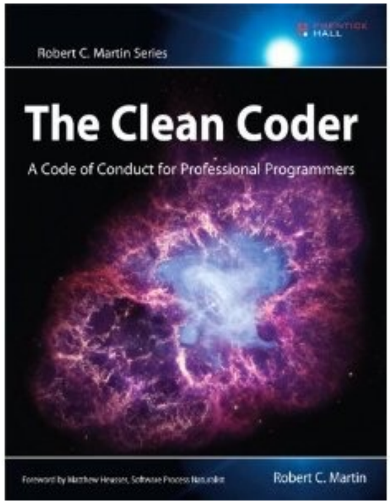
K. Schwaber, and M. Beedle, Agile Software Development with Scrum, Prentice Hall, 2001
K. Schwaber, Agile Project Management with Scrum, Microsoft Press, 2004

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."

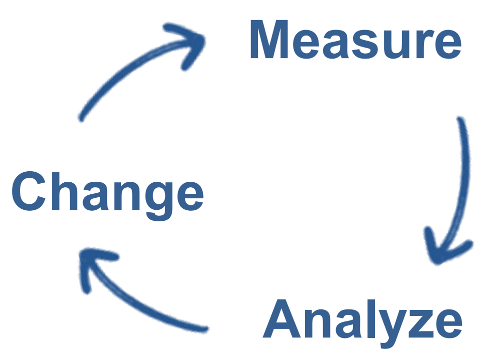
- **Meaningful names**
  - Classes, functions etc.
- **Comments**
  - Do not make up for bad code…
  - Good/bad
- **Functions**
  - Small!
  - Do one thing
  - No side effects
  - Arguments: zero, few
- **Objects**
  - Expose behavior, hide data
- **Classes**
  - Small!
  - Encapsulation
  - Cohesion
  - Single Responsibility Principle
- **Unit testing**
  - Clean
  - Single concept per test
- **Smells**
  - Heuristics
  - More in Refactoring


Robert C. Martin Series
PRENTICE HALL
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin



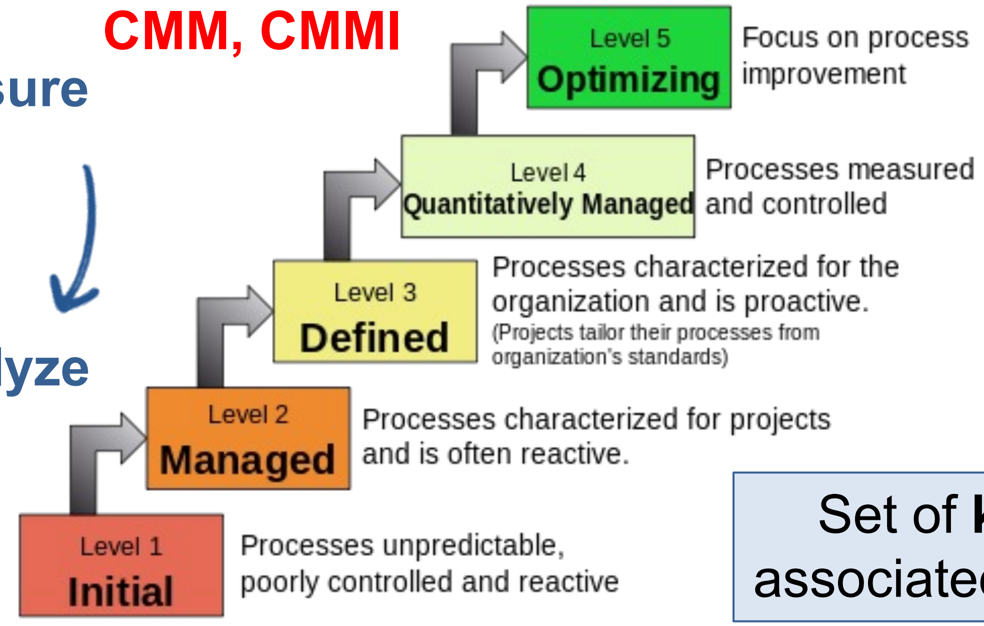
Can we **improve** the way we develop software? How?

Improvement requires **measurement**: before/after



Software Capability Maturity Model

CMM, CMMI


Level 5
Optimizing
Focus on process improvement
Level 4
Quantitatively Managed
Processes measured and controlled
Level 3
Defined
Processes characterized for the organization and is proactive.
(Projects tailor their processes from organization's standards)
Level 2
Managed
Processes characterized for projects and is often reactive.
Level 1
Initial
Processes unpredictable, poorly controlled and reactive

ISO/IEC 15504
standard for process assessment

ISO/IEC 330xx

Set of **key practices** associated with each level

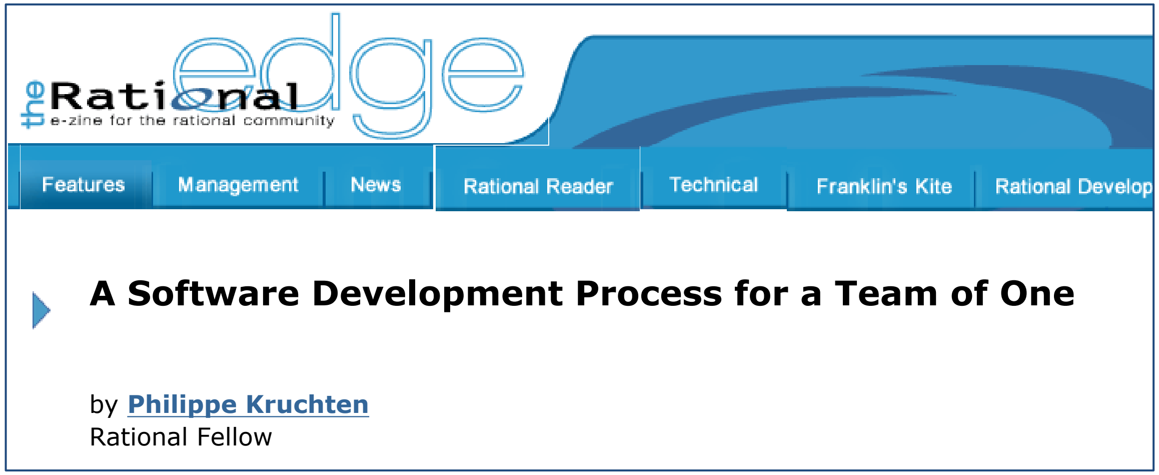
Helpful guidance towards adopting good practices

even without formal assessments

For singles

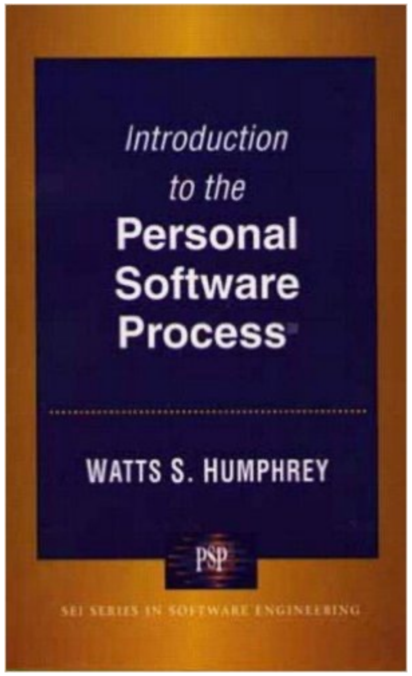
Emphasis is usually on software development **teams**

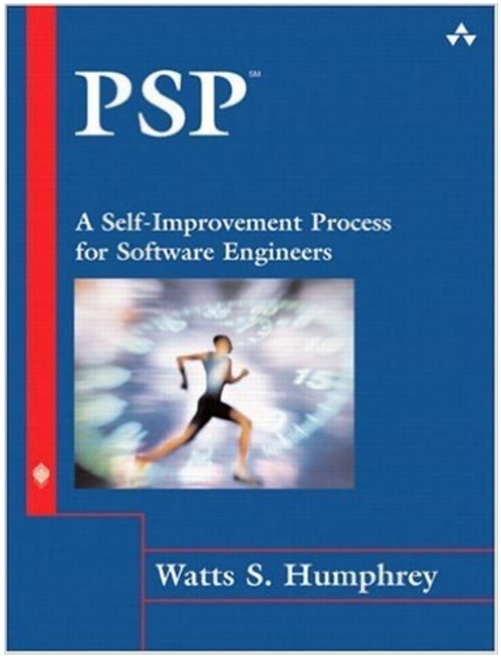
What if I work at a project where I am the only software developer?



A Software Development Process for a Team of One

by Philippe Kruchten
Rational Fellow


Introduction to the Personal Software Process
WATTS S. HUMPHREY
PSP
SEI SERIES IN SOFTWARE ENGINEERING



One size does not fit all

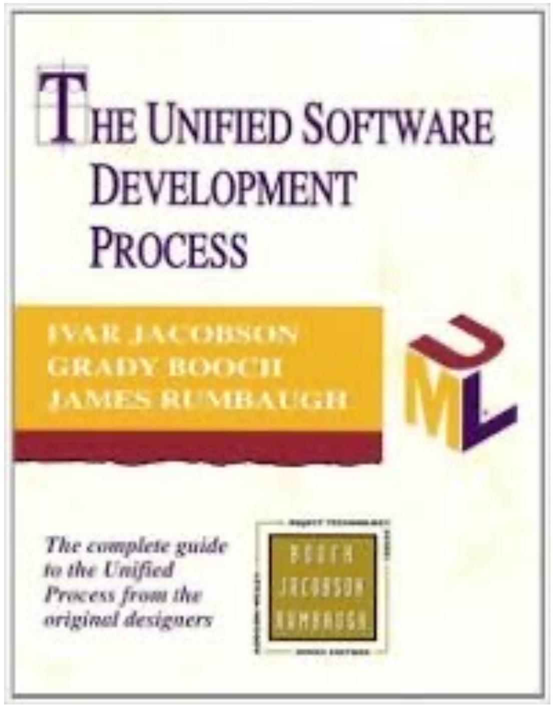
- A software process **model** is a **simplified** representation of a software process
  - From a particular perspective
- **Many different approaches are possible**
  - Positive and negative sides in any of them
- **Good or bad** often depends on the **context**
  - Small/large scale project, short/long lifetime etc.
- Process frameworks may (<u>should</u>) be adapted and extended

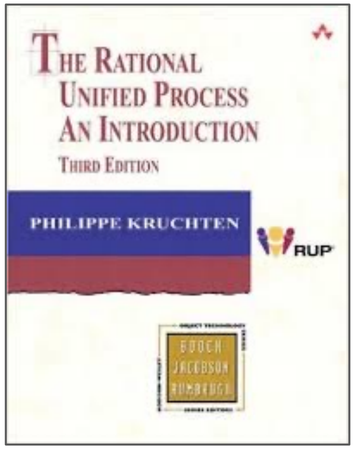
A good software process is **tailored to the project**

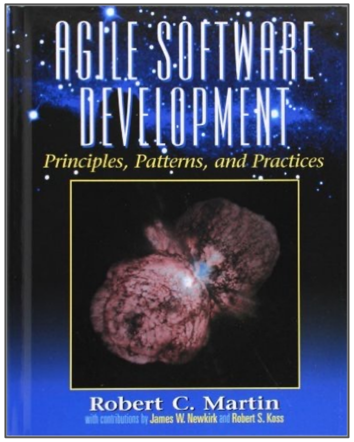
Peculiarities of the **scientific environment**

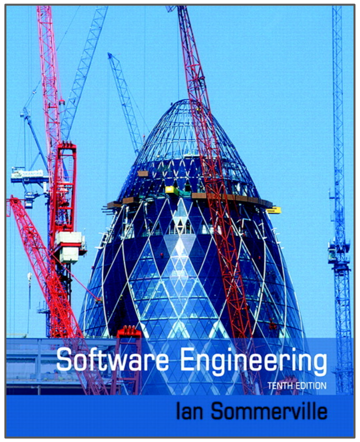
We are both the **developers** and the **customers**!

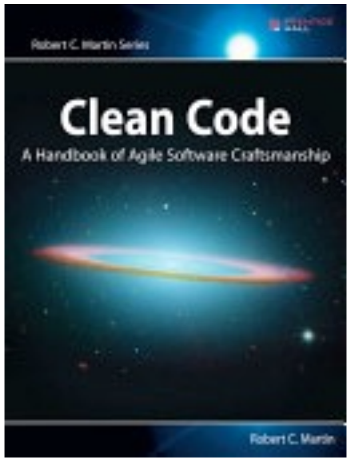
Further learning

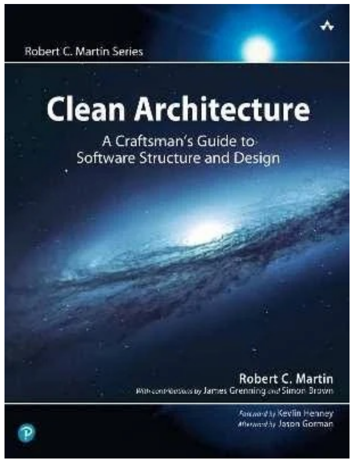

The Unified Software Development Process
Ivar Jacobson
Grady Booch
James Rumbaugh
The complete guide to the Unified Process from the original designers




Agile Software Development
Principles, Patterns, and Practices
Robert C. Martin







Get a mentor!