High Performance Computing and High Performance humans in Computing

Maria Grazia Pia
INFN Genova

mariagrazia.pia@ge.infn.it
maria.grazia.pia@cern.ch

IAEA (Virtual) **Workshop on**
Computational Nuclear Science and Engineering
12-16 July 2021

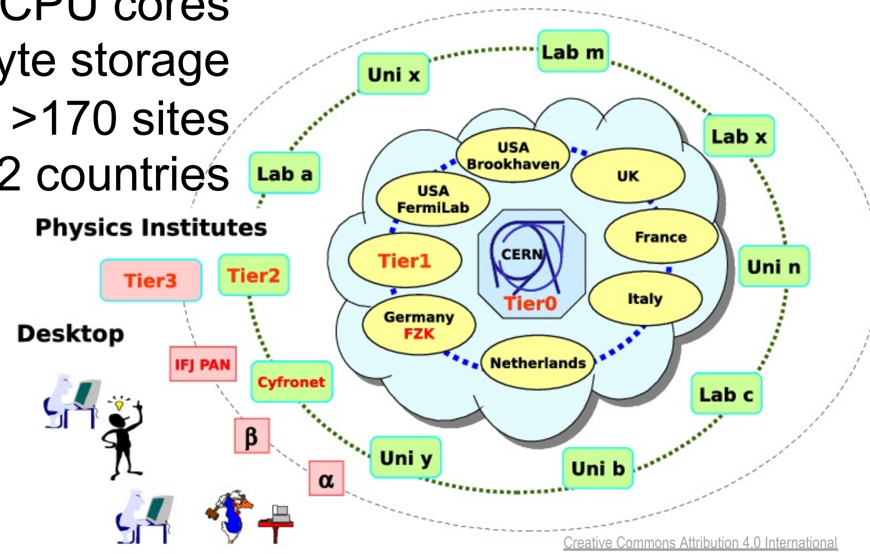
MGP

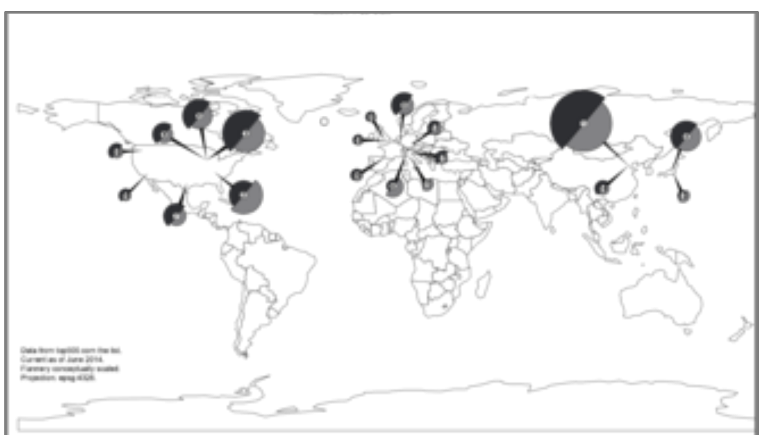
- Does not like writing bios
- Physicist at INFN (Istituto Nazionale di Fisica Nucleare), Section of Genova, Italy
  - Pre-covid: large fraction of time at CERN, looking forward to post-covid era
- Associate Editor of IEEE Transactions on Nuclear Science
- Moderator of arXiv Computational Physics *(physics.comp-ph)*
- High Energy Physics background
  - CERN, FNAL, SLAC
- Monte Carlo development *(Geant4)* and applications, physics data libraries validation
- *Statistical data analysis, scientometrics, epistemology*
- *No Facebook, Twitter, LinkedIn, Instagram… but google me, and you will find track of my research activity*

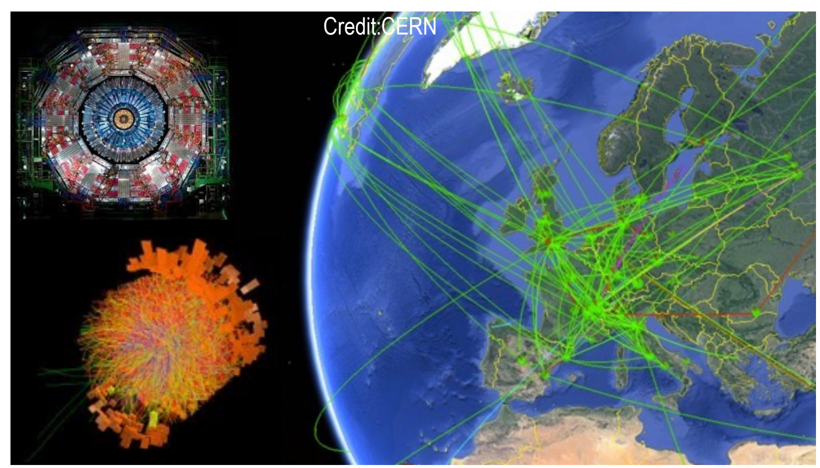
exploit **workload parallelism** to achieve **scalability** ⟵ **HP** ⟶ **manage data** to achieve **efficiency** of operation

What is HPC?

10^6 CPU cores
1 exabyte storage
>170 sites
42 countries







HPC systems

- HPC computers are networks of processors
  - Very fast memories
  - Low-latency, high-bandwidth communication systems
    - between the processors
    - between the processors and the associated memories
- Homogeneous: only CPUs
- Hybrid: CPUs + GPUs
  - **GPU**s can handle millions of threads simultaneously, are more energy efficient, have faster memories, require less data transfer
  - **CPU**s oversee the computation

Key features of **HPC architecture**

- **speed** of operation
- **parallelism** to perform multiple operations
- **efficient** use of critical components
- **electrical power** that it consumes
- **reliability**
- how **easy to program**

Top supercomputers

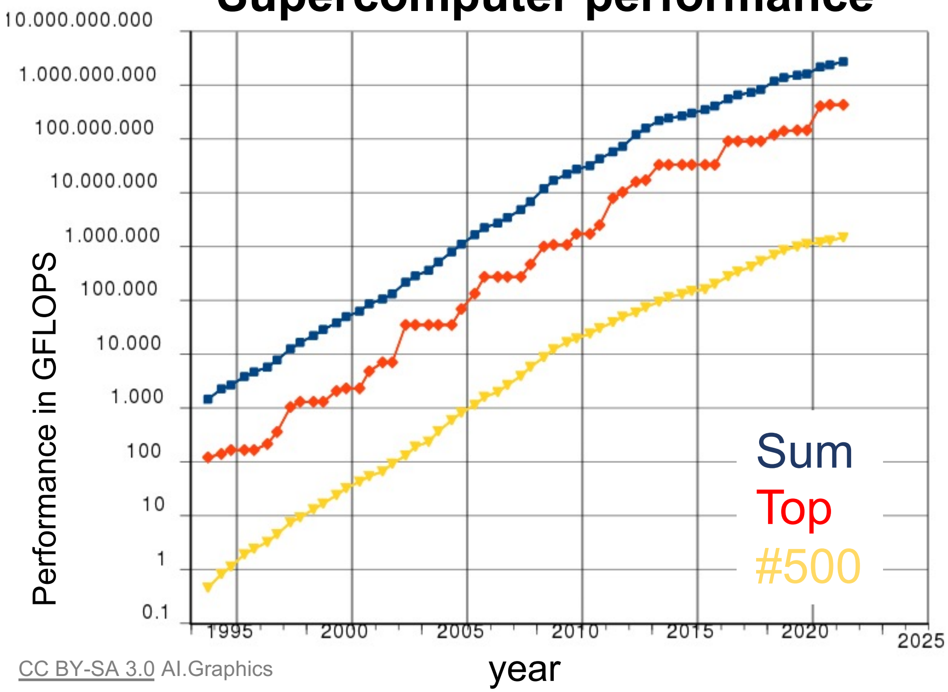
	Rmax PFLOPS	Location	Manufacturer
Fugaku	442.010	RIKEN, Japan	Fujitsu
Summit	148.600	ORNL, USA	IBM
Sierra	94.640	LLNL, USA	IBM
Sunway TaihuLight	93.015	Natl. Supercomp. Centre, China	NRCPC
Perlmutter	64.590	NERSC, USA	Nvidia

Performance

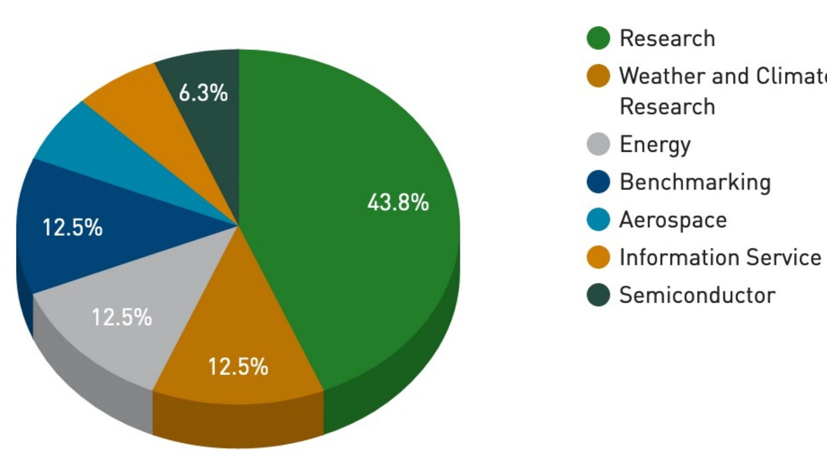
Country	PFLOPS
USA	669
Japan	594
China	564

FLOPS: floating-point operations per second

Supercomputer performance



System share by application area



Commodity clusters

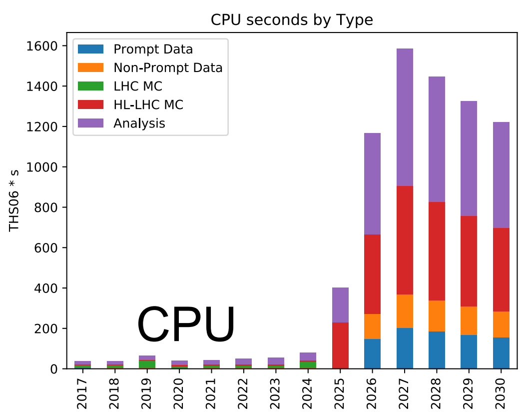
- A group of integrated computer systems
  - Standalone components (COTS), capable of independent operation
  - Integration network is separately developed
  - Off the shelf mass storage
  - Interfaces adhere to industry standards
- Very successful
  - First one in 1997, ~50% of Top 500 systems in 2005, ~ 85% today
- Parallel programming modalities
  - **Throughput computing:** efficiently run a large number of jobs that are independent or require minimal communication
  - **Message-passing:** requires a significant amount of communication and coordination within the application
    - communicating sequential processes model, exemplified by the MPI
  - **Shared-memory multiple-thread applications**
    - exemplified by the OpenMP (open multiprocessing) programming model

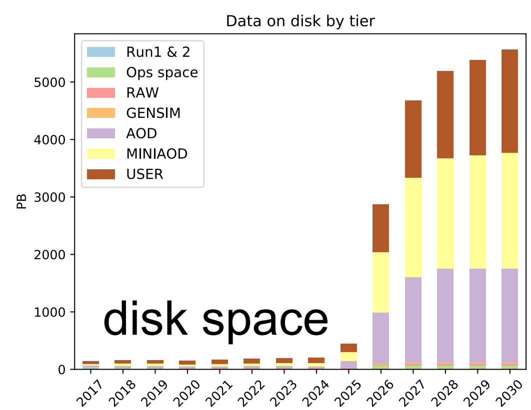
LHC WLCG

LHC data are currently handled by the Worldwide LHC Computing Grid

- >170 sites
- 42 countries
- ~10^6 CPU cores
- 1 exabyte storage
- 2 million tasks per day
- global transfer rates > 60 GB/s
- >12000 physicists around the world

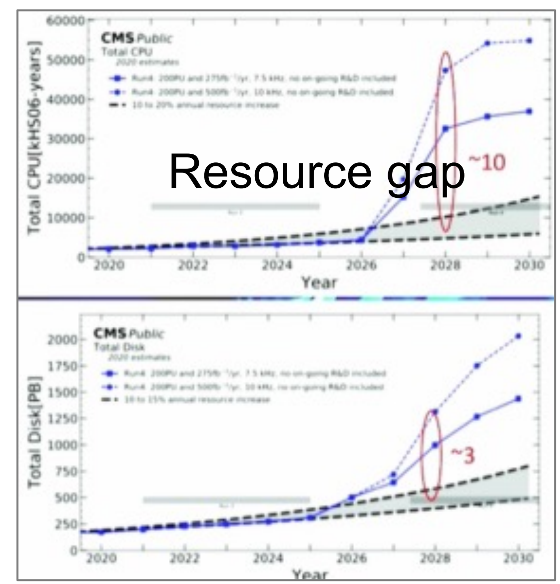
WLCG is a massive **distributed computing** infrastructure





start 2027

High Luminosity LHC



CMS estimated resources required HL-LHC era

J. Albrect et al., A Roadmap for HEP Software and Computing R&D for the 2020s, Comput. Softw. Big Sci. (2019) 3, 7

Data volumes in HL-LHC at the multi-Exabyte scale

LHC experiments are exploring GPUs for accelerated event reconstruction and simulation

Parallel algorithms

Physical parallelism

- SIMD: single-instruction multiple data parallelism
- MIMD: multiple-instruction multiple data
  - shared memory parallelism
  - distributed memory parallelism

- Multiple parts of the workload are performed concurrently to reduce the time to achieve the solution
- Several parallel algorithms are used in scientific computing
  - fork–join
  - divide and conquer
  - halo exchange
  - permutation
  - embarrassingly parallel
  - manager–worker
  - task dataflow

Emerging numerical methods in supercomputing applications:

- graph traversal
- finite state machines
- combinational logic
- statistical machine learning

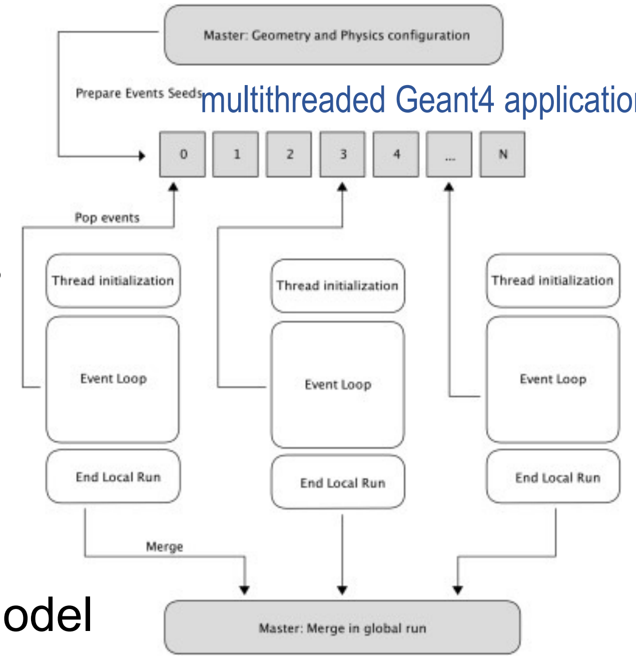
- Some algorithms are better suited for one kind of physical parallelism versus another

Chances are that you would deal with HPC and parallel computing in

Monte Carlo simulation in particle/nuclear physics

- Embarrassingly parallel
  - Parallelism with essentially no inter-task communication
  - Highly partitionable workload with minimal overhead
  - Concurrency is trivially extracted from the workflow
  - Often require gathering the results at the end into a manager process
- Monte Carlo transport is naturally suitable to event-level parallelism
- Methods for embarrassing parallel simulation documented in most popular Monte Carlo particle transport codes
  - **Threading:** OpenMP threading on a single multicore computer or on a single node of a server or cluster
  - **Message-passing** between nodes on a cluster using MPI environment
  - Used separately or together

Multi-threaded simulations

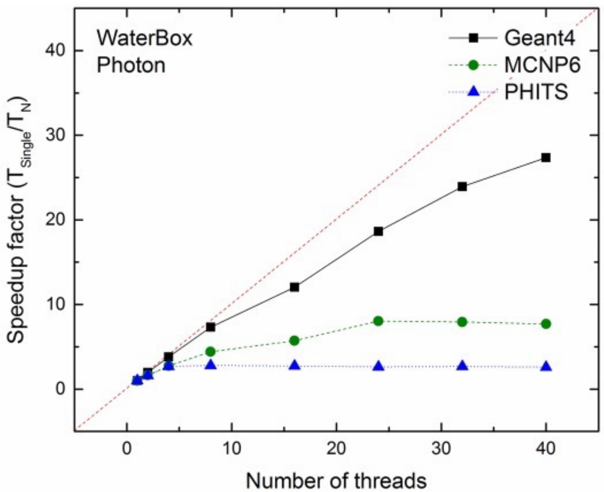


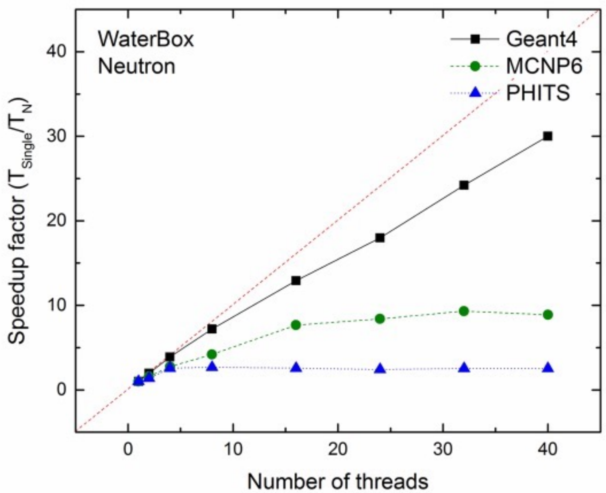
multithreaded Geant4 application

Ability to exploit hardware multi-threading capabilities

Goal: **reduce** the **memory** footprint of parallel applications, while preserving the **linear speedup** as a function of the number of physical cores

Based on a master–worker model





- master thread prepares geometry and physics setups
- worker threads compete for the next events to be simulated

an application example

Min Cheol Han *et al,* **Multi-threading performance of Geant4, MCNP6, and PHITS** Monte Carlo codes for tetrahedral-mesh geometry, 2018 *Phys. Med. Biol.* 63 09NT02

Quantum computing is a computing paradigm that exploits quantum mechanical properties (superposition, entanglement, interference...) of matter to do calculations

Quantum computing

Quantum supremacy using a programmable superconducting processor

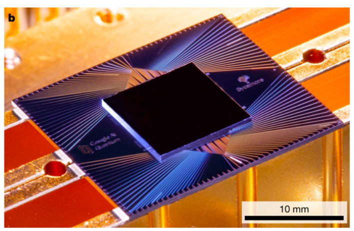
Nature volume 574, pp. 505–510 (2019)

https://doi.org/10.1038/s41586-019-1666-5

Received: 22 July 2019

Accepted: 20 September 2019

Published online: 23 October 2019



Frank Arute[1], Kunal Arya[1], Ryan Babbush[1], Dave Bacon[1], Joseph C. Bardin[1,2], Rami Barends[1], Rupak Biswas[3], Sergio Boixo[1], Fernando G. S. L. Brandao[1,4], David A. Buell[1], Brian Burkett[1], Yu Chen[1], Zijun Chen[1], Ben Chiaro[5], Roberto Collins[1], William Courtney[1], Andrew Dunsworth[1], Edward Farhi[1], Brooks Foxen[1,5], Austin Fowler[1], Craig Gidney[1], Marissa Giustina[1], Rob Graff[1], Keith Guerin[1], Steve Habegger[1], Matthew P. Harrigan[1], Michael J. Hartmann[1,6], Alan Ho[1], Markus Hoffmann[1], Trent Huang[1], Travis S. Humble[7], Sergei V. Isakov[1], Evan Jeffrey[1], Zhang Jiang[1], Dvir Kafri[1], Kostyantyn Kechedzhi[1], Julian Kelly[1], Paul V. Klimov[1], Sergey Knysh[1], Alexander Korotkov[1,8], Fedor Kostritsa[1], David Landhuis[1], Mike Lindmark[1], Erik Lucero[1], Dmitry Lyakh[9], Salvatore Mandrà[3,10], Jarrod R. McClean[1], Matthew McEwen[5], Anthony Megrant[1], Xiao Mi[1], Kristel Michielsen[11,12], Masoud Mohseni[1], Josh Mutus[1], Ofer Naaman[1], Matthew Neeley[1], Charles Neill[1], Murphy Yuezhen Niu[1], Eric Ostby[1], Andre Petukhov[1], John C. Platt[1], Chris Quintana[1], Eleanor G. Rieffel[3], Pedram Roushan[1], Nicholas C. Rubin[1], Daniel Sank[1], Kevin J. Satzinger[1], Vadim Smelyanskiy[1], Kevin J. Sung[1,13], Matthew D. Trevithick[1], Amit Vainsencher[1], Benjamin Villalonga[1,14], Theodore White[1], Z. Jamie Yao[1], Ping Yeh[1], Adam Zalcman[1], Hartmut Neven[1] & John M. Martinis[1,5*]

The Sycamore processor can run a test computation in 200 seconds that would take the world's biggest supercomputers 10000 years to complete

IBM researchers claim Google's challenge would take a classical computer just two and half days

In quantum circuits:
- data = qubits
- operations = quantum gates
- results = measurements

NewScientist

China beats Google to claim the world's most powerful quantum computer

5 July 2021

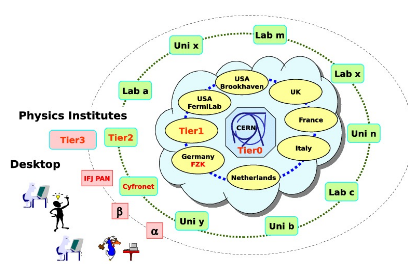
TECHNOLOGY 5 July 2021

By Matthew Sparkes


The Zuchongzhi quantum computer
University of Science and Technology of China/quantumcomputer.ac.cn

High Performance humans in Computing

Technology
Methodologies



Producing results

Measurements over 10 years
Publicly accessible data

Fred 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.

…

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.

Introduction to concepts and methods

- Pills of wisdom
- Food for thought
- Curiosity
- Background for further learning

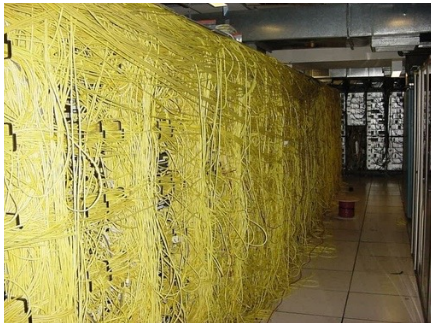
Software development **methods** and **techniques** are seldom part of academic programs for physics and engineering degrees

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

Much more than just hacking code…

Implementation

Requirements

Change management

Design

Environment

Project management

Configuration management

Business modeling

Deployment

Test

These complex **disciplines** include **activities**, generate **products** and involve responsibilities in various **roles**

articulated over the software **life-cycle**: get-go, elaboration, construction, use, maintenance…

*Built on **best practices** derived from experience*

Software development methodologies

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

Highly prescriptive — Adaptable to the context

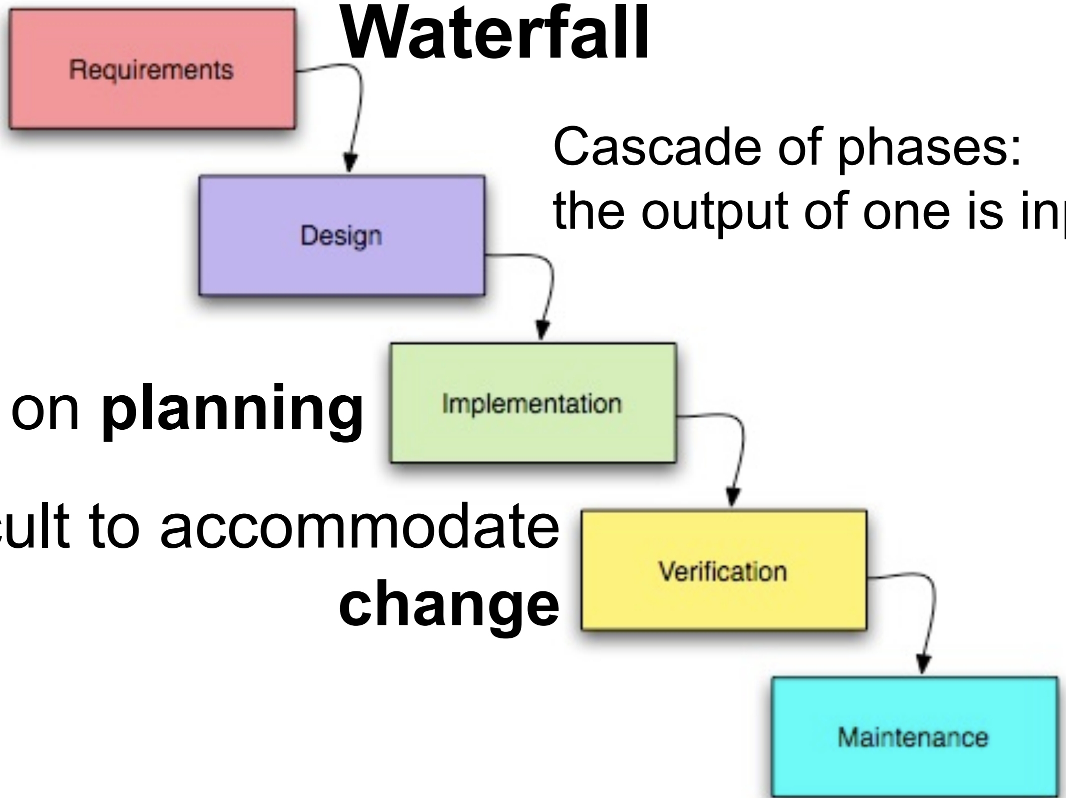
Wide variety

Small projects — Large scale projects

Old, risky… and most common

Waterfall

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

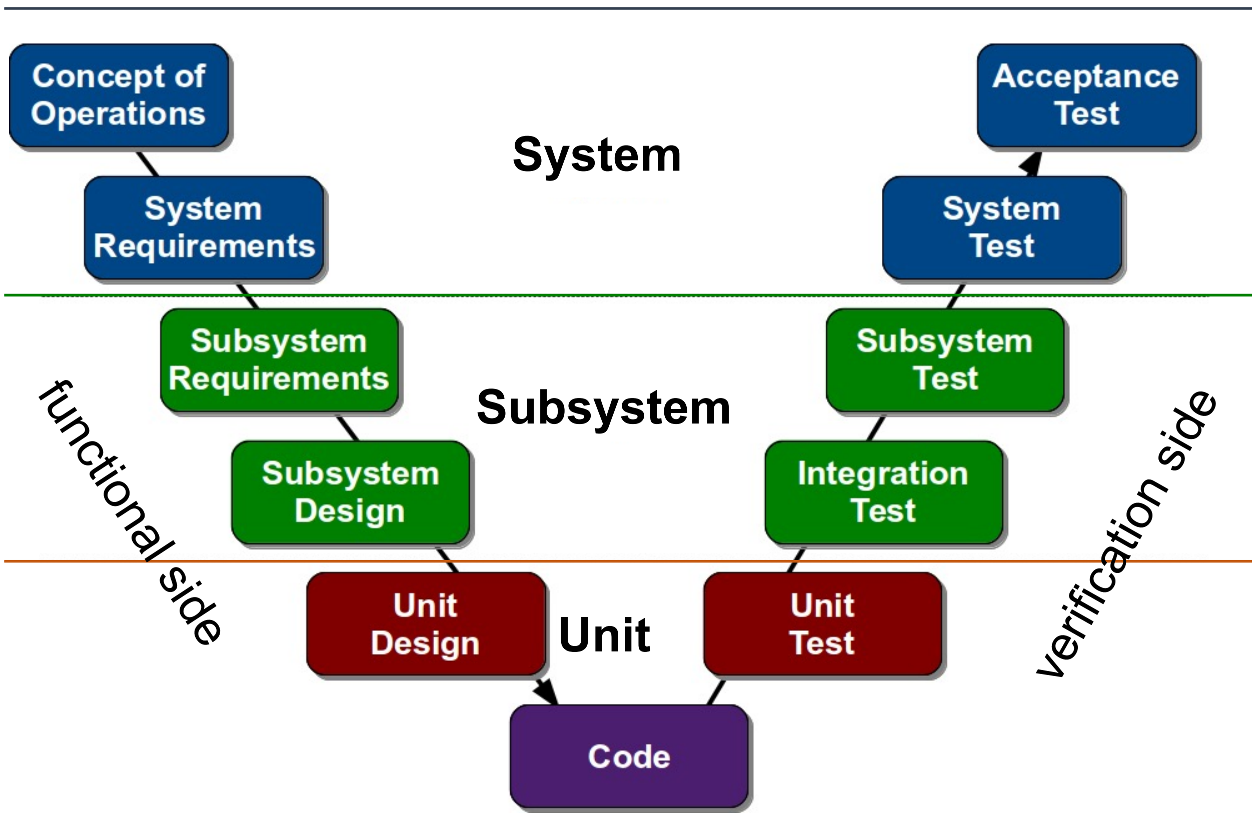


Emphasis on **planning**

Difficult to accommodate **change**

Risk of discovering problems at a late stage of the project

Variant of waterfall: V-model

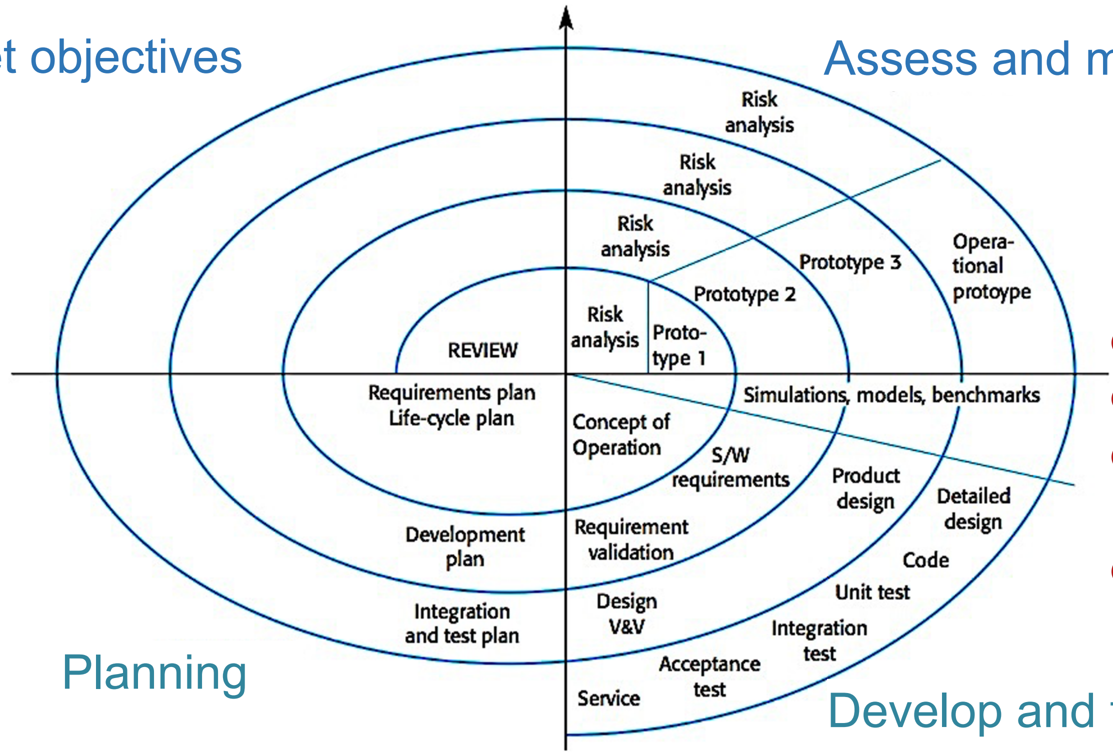


Emphasis on **testing** at all levels of software development

Each development phase is associated with a **testing phase**

Spiral development

basis of modern methodologies



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

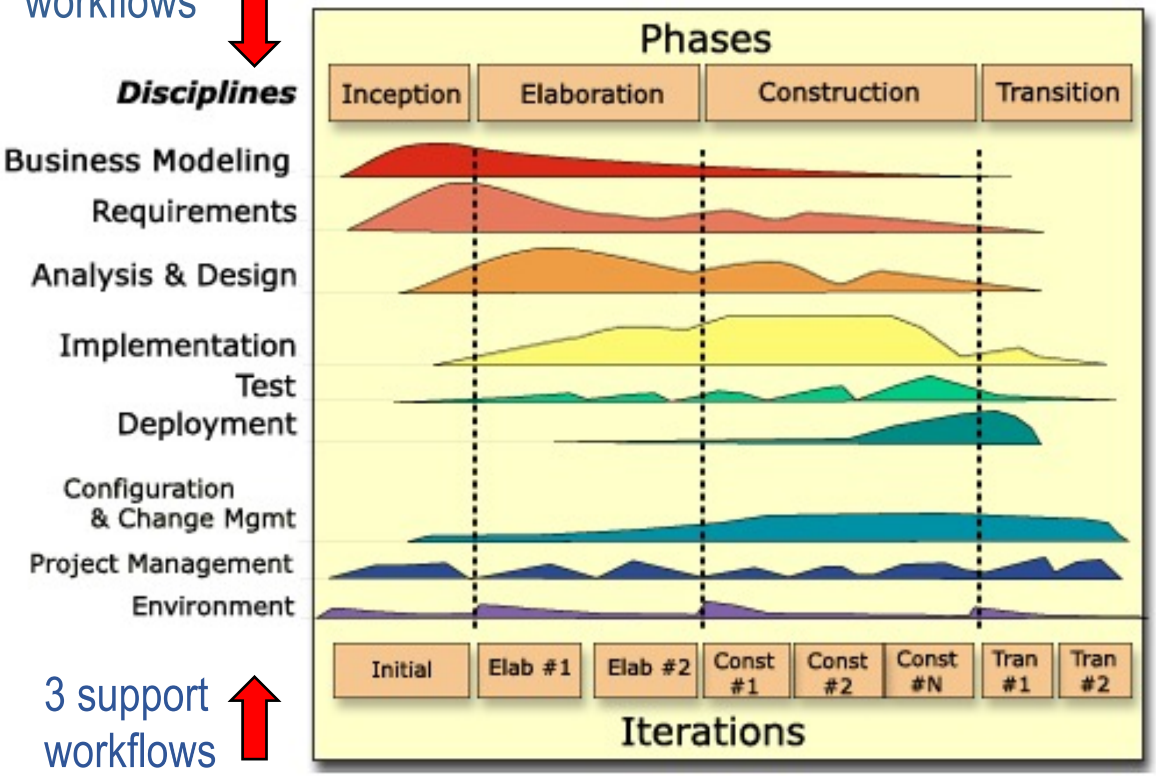
loop in the spiral = phase of software development

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, with emphasis on **modeling**

6 core process workflows



3 support workflows

Complex, but highly **customizable**

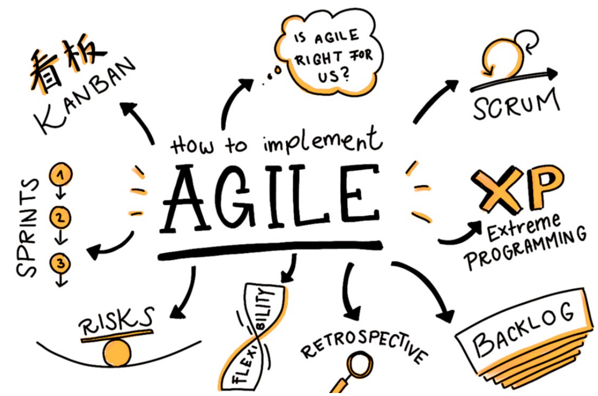
Best practices

- Develop software iteratively
  - High priority features developed first
- Manage requirements
  - Document requirements
  - Keep track of changes
  - Analyze the impact of changes before accepting them
- Use component-based architectures
  - Structure the system into components
- Visually model software
  - UML (Unified Modeling Language)
- Verify software quality
  - Test (and more)
- Control changes to software
  - Change management system
  - Configuration management and tools

Emphasis on

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

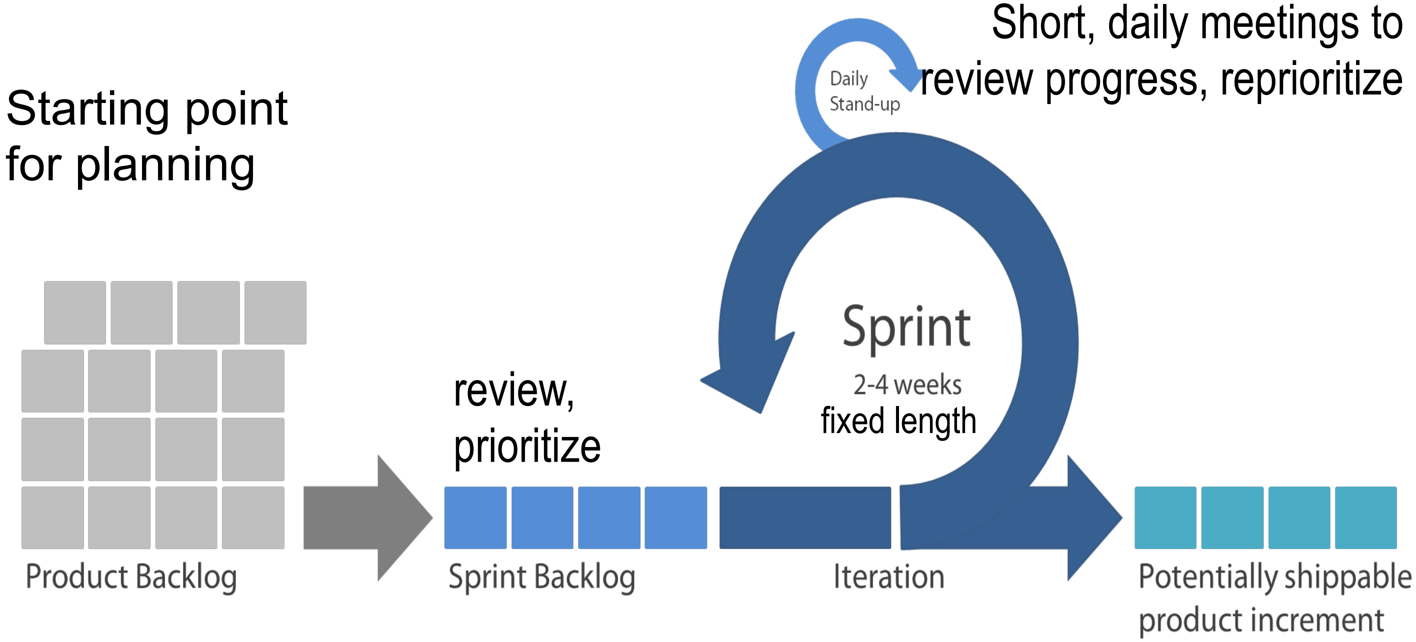
Agile Methods



B. Boehm, "**Get Ready for Agile Methods, With Care**", *IEEE Computer*, 2002, http://dx.doi.org/10.1109/2.976920
B. Meyer, **Agile!: The Good, the Hype and the Ugly**, Springer, 2014
R. C. Martin, **Clean Agile: Back to Basics,** Prentice Hall, 2019

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

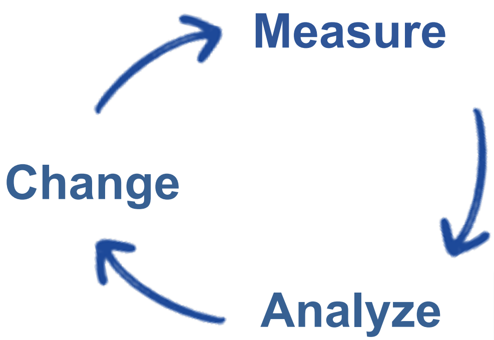
K. Schwaber and M. Beedle, Agile Software Development with Scrum, Prentice Hall, 2001
S. Ockerman and S. Reindl, Mastering Professional Scrum: A Practitioners Guide to Overcoming Challenges and Maximizing the Benefits of Agility, Addison-Wesley, 2019

- **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 often we are not only the developers, but also the customers!

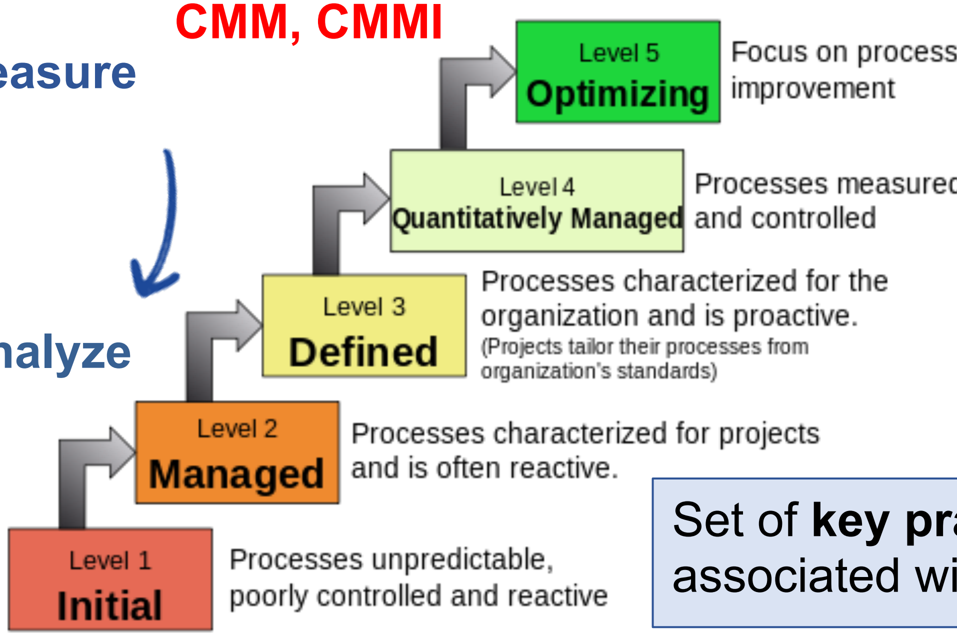
How to improve the way we develop software?

Improvement requires **measurement**: quantify *before/after*

Software Capability Maturity Model



CMM, CMMI



ISO/IEC 33001:2015
revision of ISO/IEC 15504
Software Process Improvement
and Capability Determination
(SPICE)

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

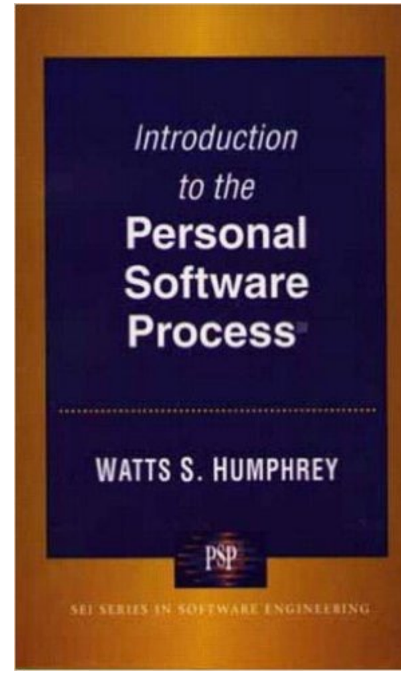
Helpful guidance towards adopting good practices

Capability Immaturity Model: A. Finkelstein, ACM SIGSOFT Software Eng. Notes 17 (4) 1992 pp 22–23
0 **Negligent** -1. **Obstructive** -2. **Contemptuous** -3. **Undermining** https://dl.acm.org/doi/10.1145/141874.141878

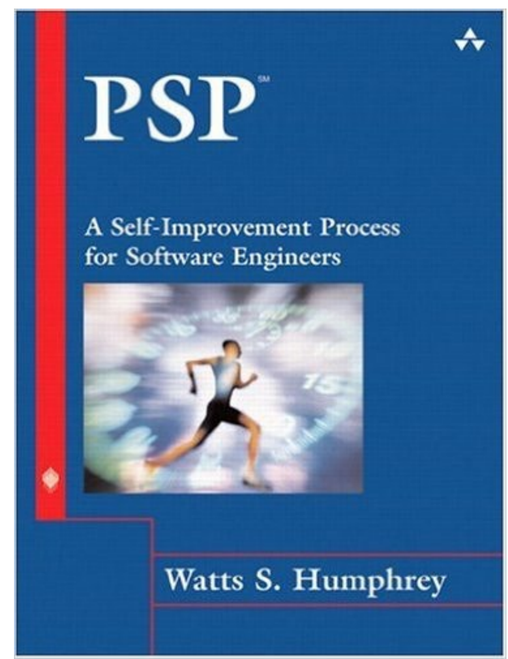
For singles

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

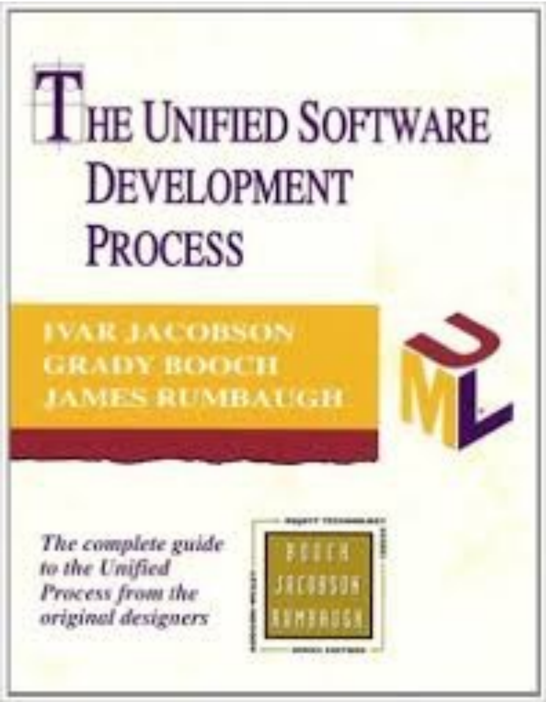
The benefits of sound methodologies are not restricted to large scale projects or sizeable teams

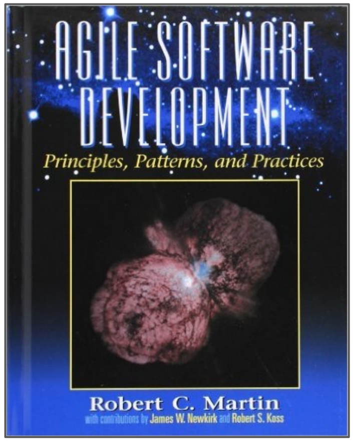


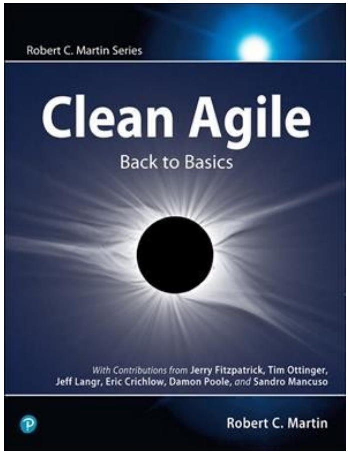
https://doi.org/10.1184/R1/6585197.v1

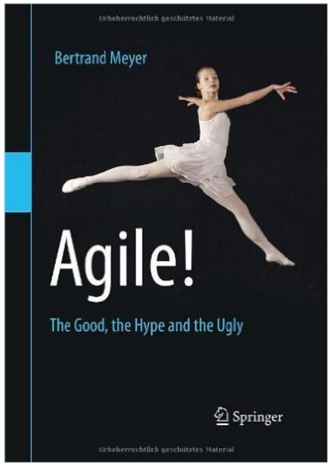


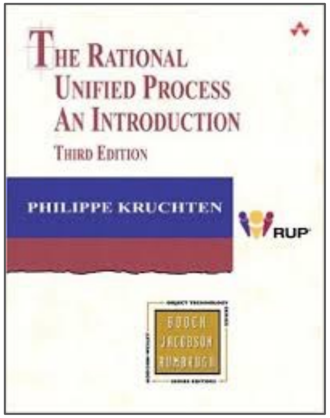
Further learning

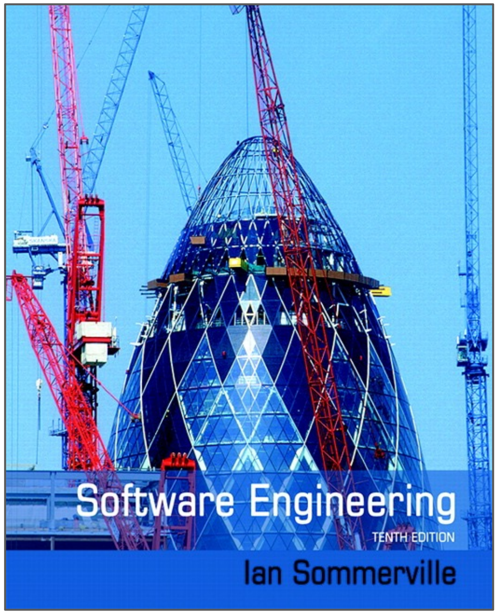












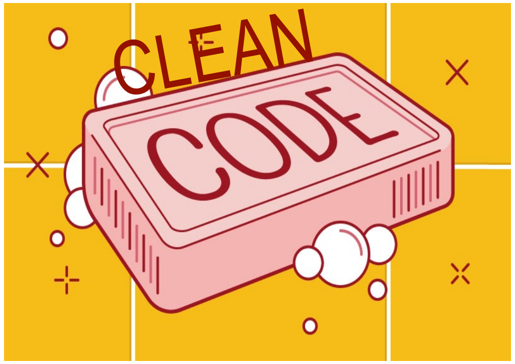
Get a mentor!

Technology

- No time for an extensive overview and in-depth analysis
- A few highlights on key technologies and tools

Modelling | **Testing** | **Tools**

Dealing with legacy code



Basic techniques of programming hygiene

"agile software craftmanship"

"If you have been a programmer for more than two or three years, you have probably been significantly slowed down by someone else's messy code." *R. C. Martin, Clean Code*

[Your own?]

Best practices for clean code

ays better
ble

nd cleaner than

for the root

Design rules

Keep configurable data at high levels

Prefer polymorphism to if/else or switch/case

Separate multi-threading code

Prevent over-configurability

Use dependency injection

Follow Law of Demeter A class should know only its direct dependencies

Understandability

Be consistent If you do something a certain way, do all similar things in the same way

Use explanatory variables

Encapsulate boundary conditions. Boundary conditions are hard to keep track of Put the processing for them in one place

Prefer dedicated value objects to primitive type

Avoid logical dependency. Don't write methods which works correctly depending on something else in the same class.

Avoid negative conditionals

Names

Choose descriptive and unambiguous names

Make meaningful distinction

Use pronounceable names

Use searchable names

Replace magic numbers with named constants

Avoid encodings. Don't append prefixes or type information

Functions

Small

Do one thing

Use descriptive names

Prefer fewer arguments

Have no side effects

Don't use flag arguments. Split method into several independent methods that can be called from the client without the flag

Comments

Always try to explain yourself in code

Don't be redundant

Don't add obvious noise

Don't use closing brace comments

Don't comment out code Just remove

Use as explanation of intent

Use as clarification of code

Use as warning of consequences

Source code structure

Separate concepts vertically

Related code should appear vertically dense

Declare variables close to their usage

Dependent functions should be close

Similar functions should be close

Place functions in the downward direction

Keep lines short

Don't use horizontal alignment

Use white space to associate related things and disassociate weakly related

Don't break indentation

Object and data structure

Hide internal structure

Prefer data structures

Avoid hybrids structures (half object and half data)

Should be small

Do one thing

Small number of instance variables

Base class should know nothing about their derivatives

Better to have many functions than to pass some code into a function to select a behavior

Prefer non-static methods to static methods

Tests

One assert per test

Readable

Fast

Independent

Repeatable

Code smells

Rigidity. The software is difficult to change. A small change causes a cascade of subsequent changes

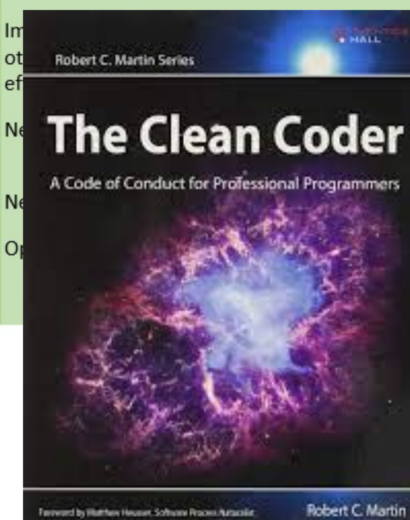
Fragility. The software breaks in many places due to a single change

Im in
ot gh
ef

Ne

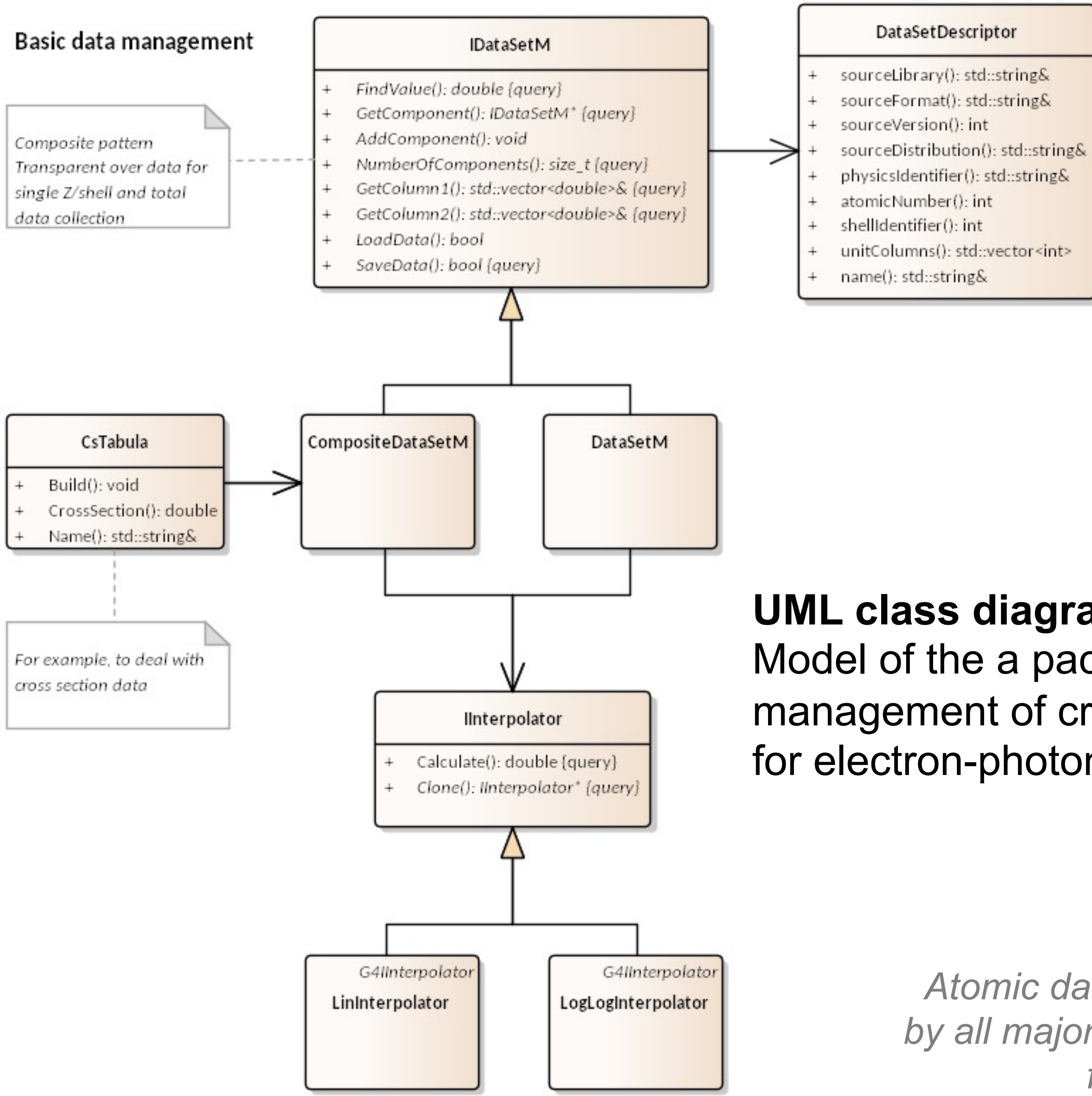
Ne

O



[Visual] Modeling

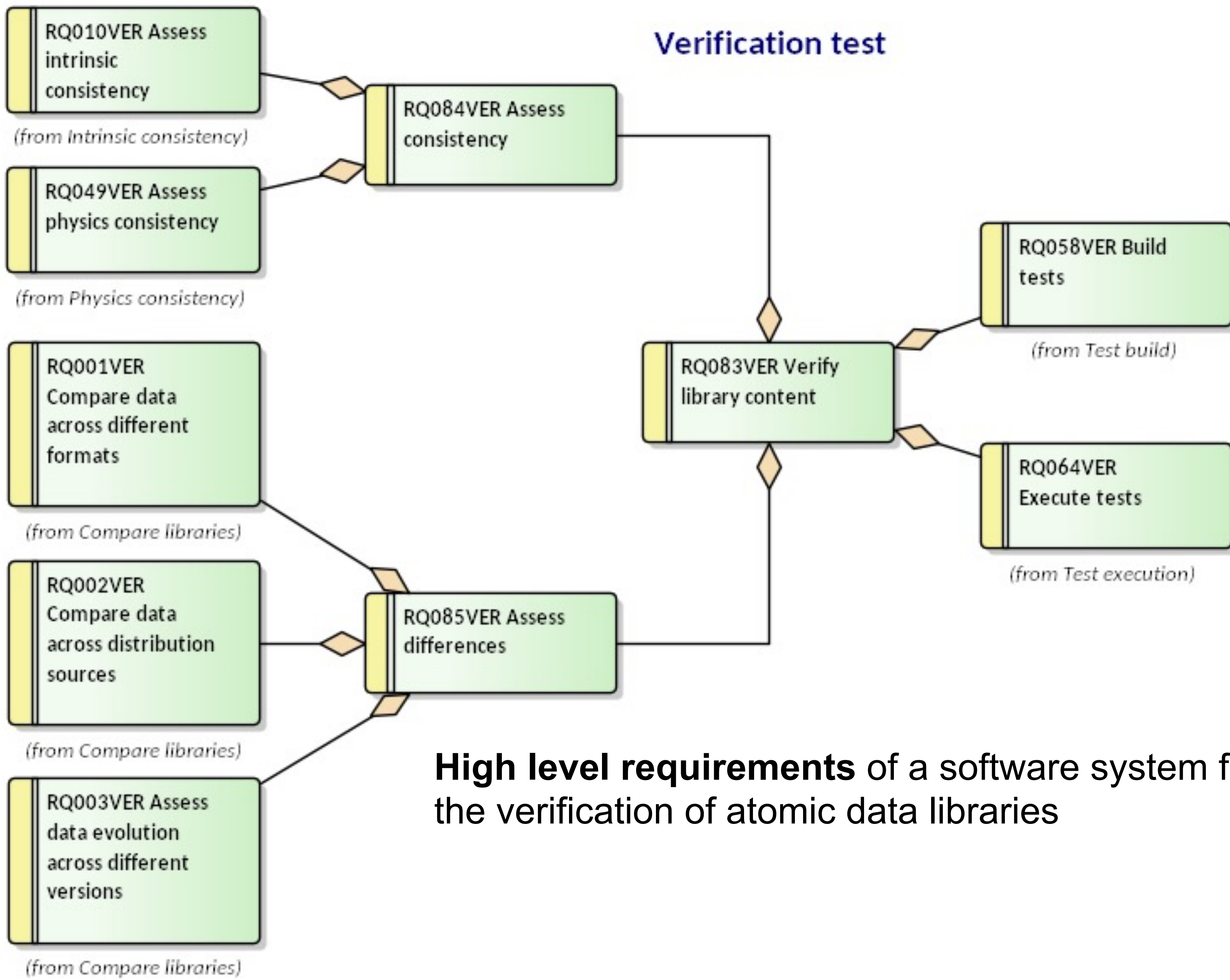
- Enormously helpful
  - **Think before typing** implementation code on the keyboard
  - Bird's-eye view of the software (and of the hardware)
  - Visualize relationships among the players, which may not be easy to catch just by looking at source code
- **UML** (Unified Modelling Language)
  - Standard, lingua franca for the communication of models
  - Plenty of educational material and tools, from simple to very powerful
  - Umbrello (https://umbrello.kde.org/) …➠… Sparx Enterprise Architect™
- **Diagrams**: class, package, component, deployment, object, state, activity, sequence, interaction, use case...
- **Design patterns**
  - elements of reusable object-oriented software



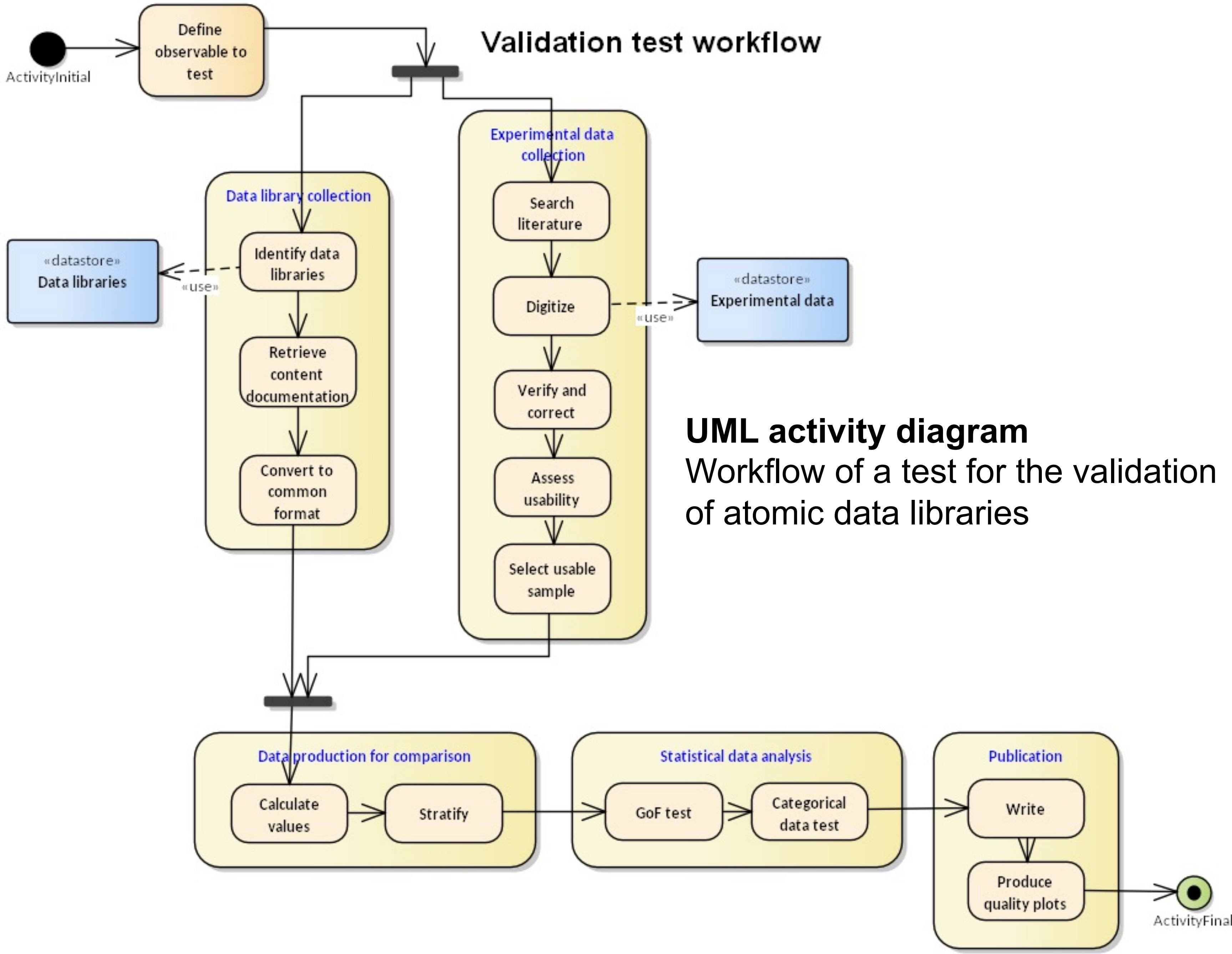
UML class diagram

Model of the a package for the management of cross sections data for electron-photon interactions

Atomic data libraries are used by all major Monte Carlo codes for particle transport

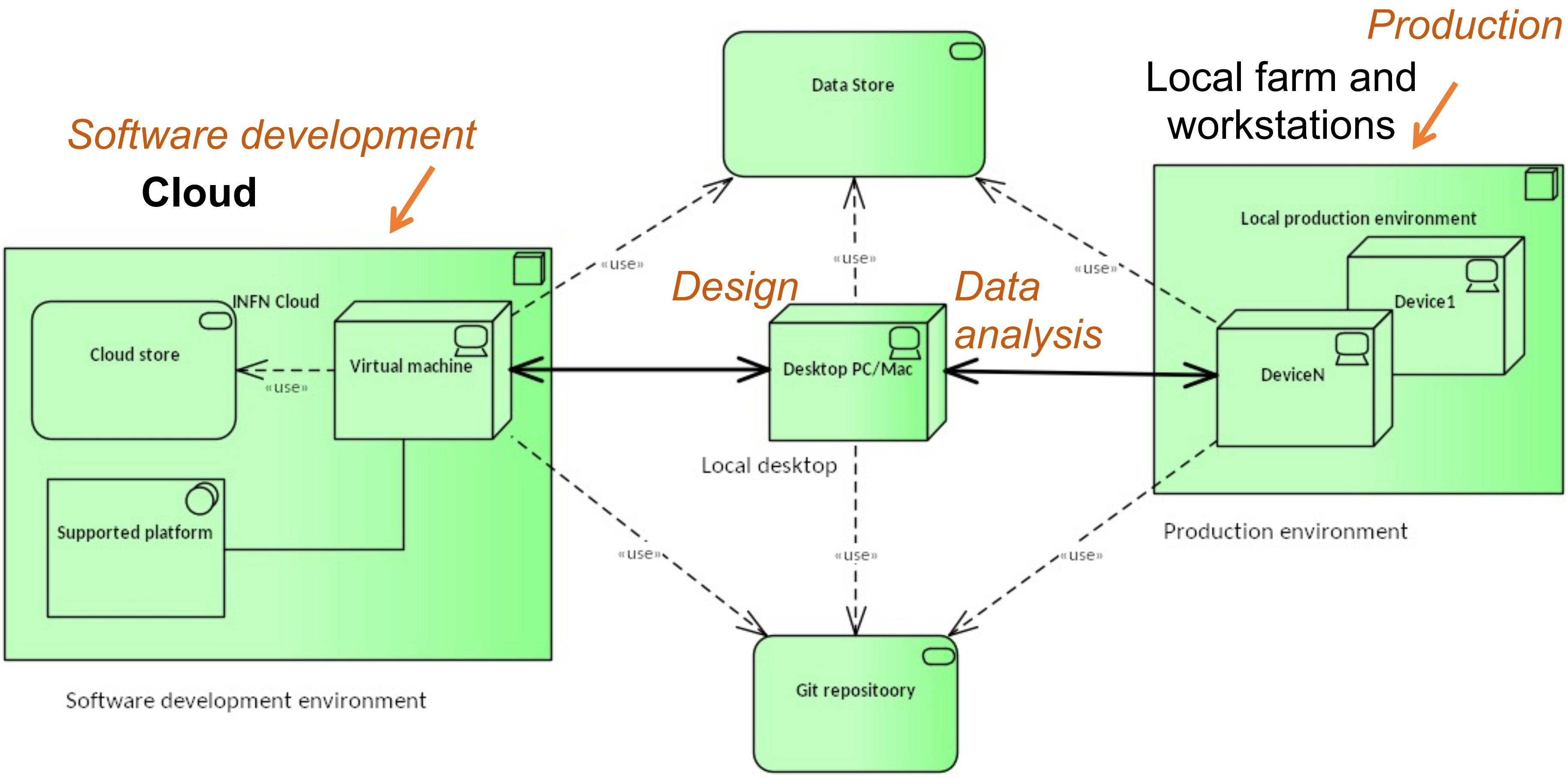


High level requirements of a software system for the verification of atomic data libraries



UML activity diagram

Workflow of a test for the validation of atomic data libraries



ArchiMate diagram of the computational environment of a software system for the verification test of atomic data libraries

How do you trust the software you use?

"Testing shows the presence, not the absence, of bugs." In other words, a program can be proven incorrect by a test, but it cannot be proven correct. All that tests can do, after sufficient testing effort, is allow us to deem a program to be correct enough for our purposes.

Edsger Wybe Dijkstra

Test

No time to cover this discipline in depth!

Levels of testing
- Unit
- Integration
- System
- Acceptance

Regression testing

Performance testing

Stress testing

Configuration testing

Security testing

Verification test

Functional/non-functional testing

Validation test

Black/white-box testing

Test **coverage**

Test **harness**

Test **automation**

Test **cases**

Test **planning**

Test **frameworks**

pytest

nunit

JUnit 5

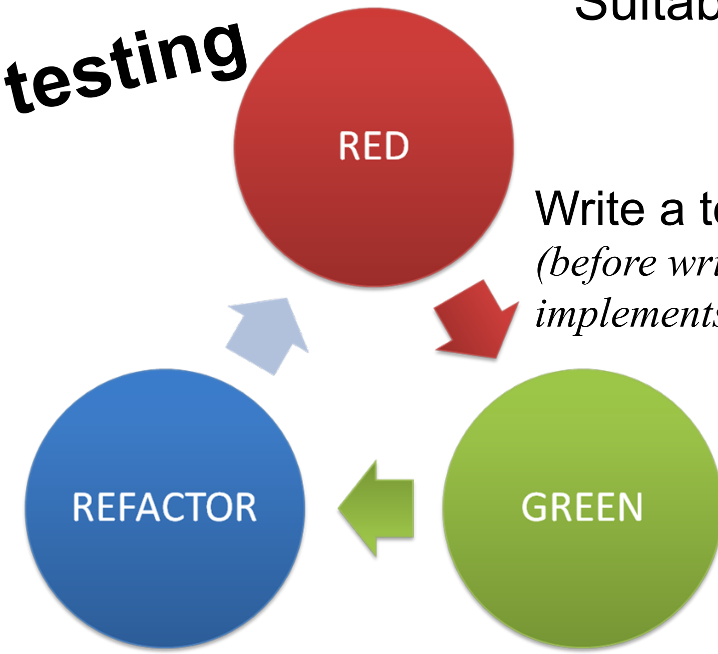
googletest
Google C++ Testing Framework

…etc.

Test-driven development (TDD)

Suitable to **small-size projects**

Essential: an **automated testing** environment



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

Write code that makes the test **pass**

Improve the design of the software

Facilitates **regression testing**
Discover problems early during the software development

Limited to unit testing,
still need system testing, performance, reliability testing etc.

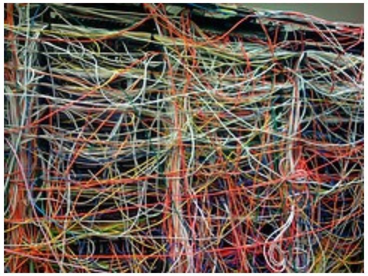
D. Astels, Test Driven Development: A Practical Guide, Prentice Hall, 2003 *conventional wisdom*

"If it ain't broken, don't fix it"

A piece of software can be broken in many ways

Functional it no longer delivers the function it is designed to perform

Maintenance it can no longer be maintained



- Obsolete or no **documentation**
- Missing **tests**
- Original **developers** or users have left
- **Inside knowledge** about the system has disappeared
- Limited understanding of the **entire system**
- Too long to turn things over to **production**
- Too much time to make **simple changes**
- Need for constant **bug fixes**
- Big **build times**
- Difficulties **separating** products
- **Duplicated** code
- Code **smells**

Warnings you are heading into trouble

usually do not occur isolated

S. Demeyer, S. Ducasse, O. Nierstrasz,
Object Oriented Reengineering Patterns

New Functionality

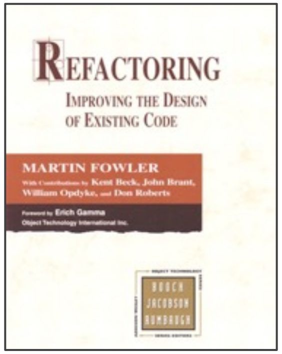
Hack it in ?

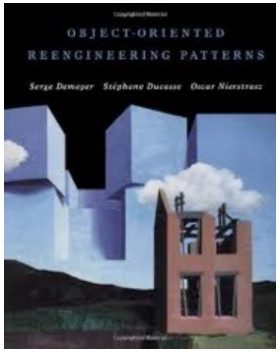
- duplicated code
- complex conditionals
- abusive inheritance
- large classes/methods

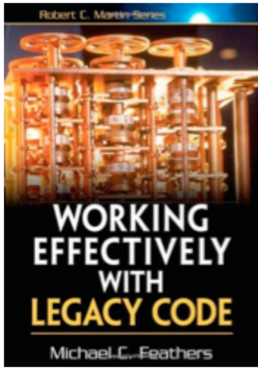
First …
- refactor
- restructure
- reengineer

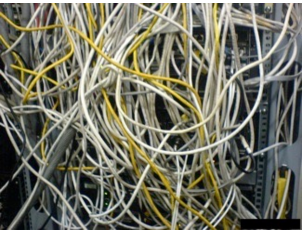
Take a loan on your software
⇒ pay back via reengineering

Investment for the future
⇒ paid back during maintenance









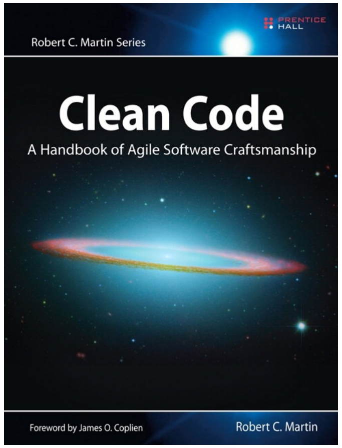
Refactoring is a disciplined technique for **improving the design of an existing code**

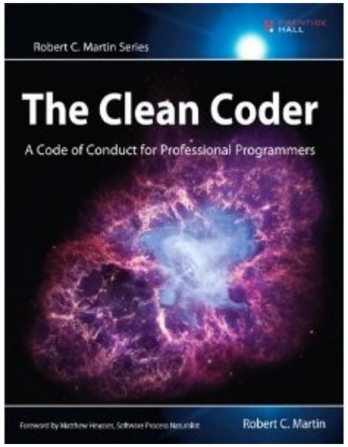
In the ideal world there would be **hardly any need** for **refactoring**

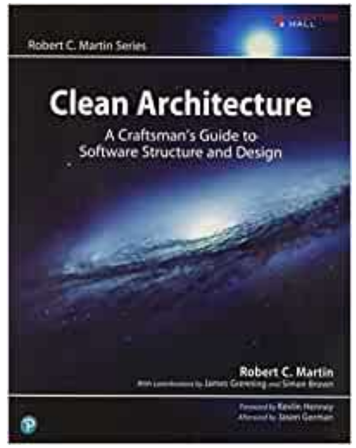
In the real world **most software needs to be refactored**

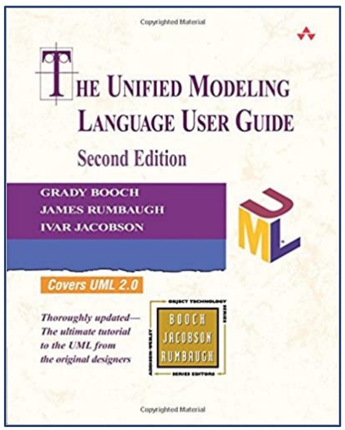
By learning refactoring you also learn writing code that minimizes the need to be refactored

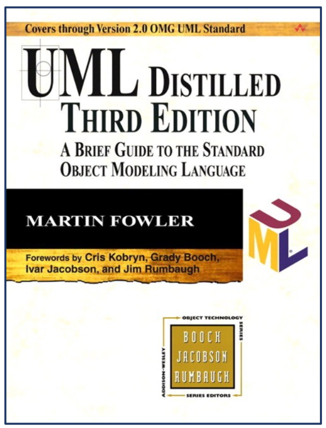
Further learning

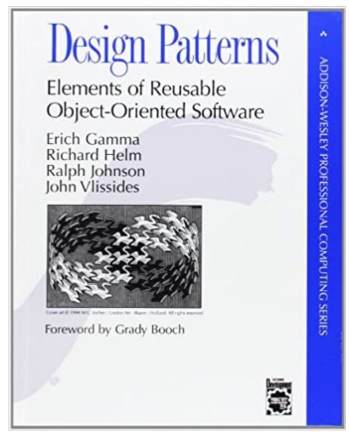


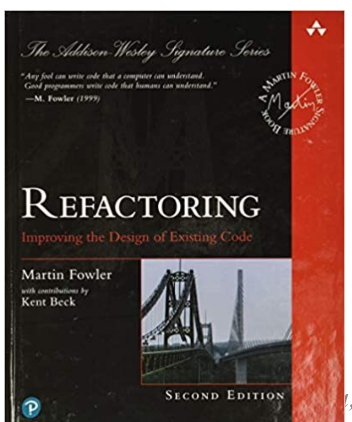


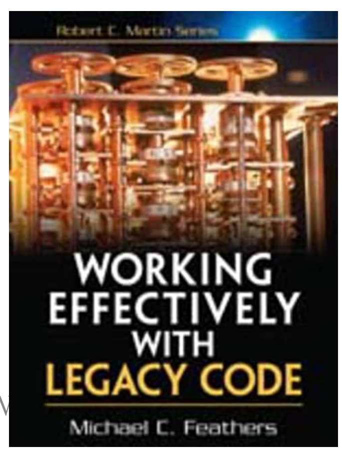


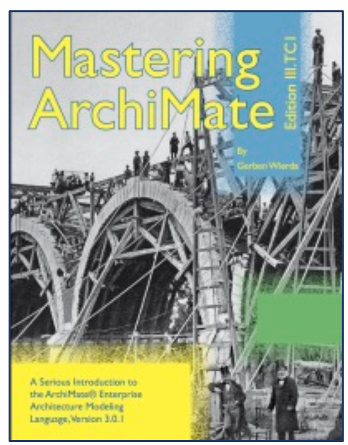








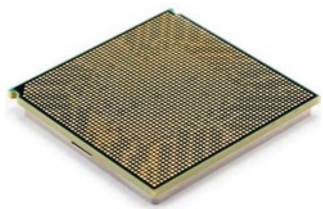




Get a mentor!

The End

HPC =



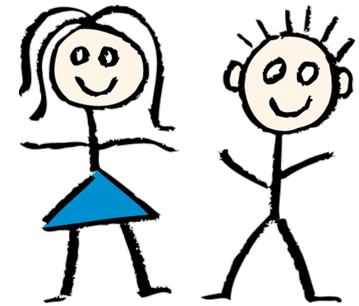
+

```
Mat DecodeGrayCPU(
  const vector<Mat>& Ig, // imaged Gray patterns
  const Mat& Il, // image under no illumination
  const Mat& Ih) // image under full illumination
{
  Mat Cp = Mat::zeros(Il.size(), CV_16U);
  Mat Bp = Mat::zeros(Il.size(), CV_16U);
  Mat Id = Ih - Il;

  for (vector<Mat>::const_reverse_iterator Ik =
       Ig.rbegin(); Ik != Ig.rend(); Ik++)
  {
    Mat Rk = (*Ik - Il) / Id;
    Mat Bk;
    threshold(Rk, Bk, 0.5, 1, THRESH_BINARY);

    Bk.convertTo(Bk, CV_16U);
    Bp = Bk ^ Bp;
    Cp = 2 * Cp + Bp;
  }
  return Cp;
}
```

+



Technology

Methodology

Master them, so that you can choose what is most appropriate to your research problem

This 40' introduction only scratches the surface,
more to follow next year

Feel free to contact me for further information and for suggestions for next year