Good Coding Style

1st PE Mini Lecture

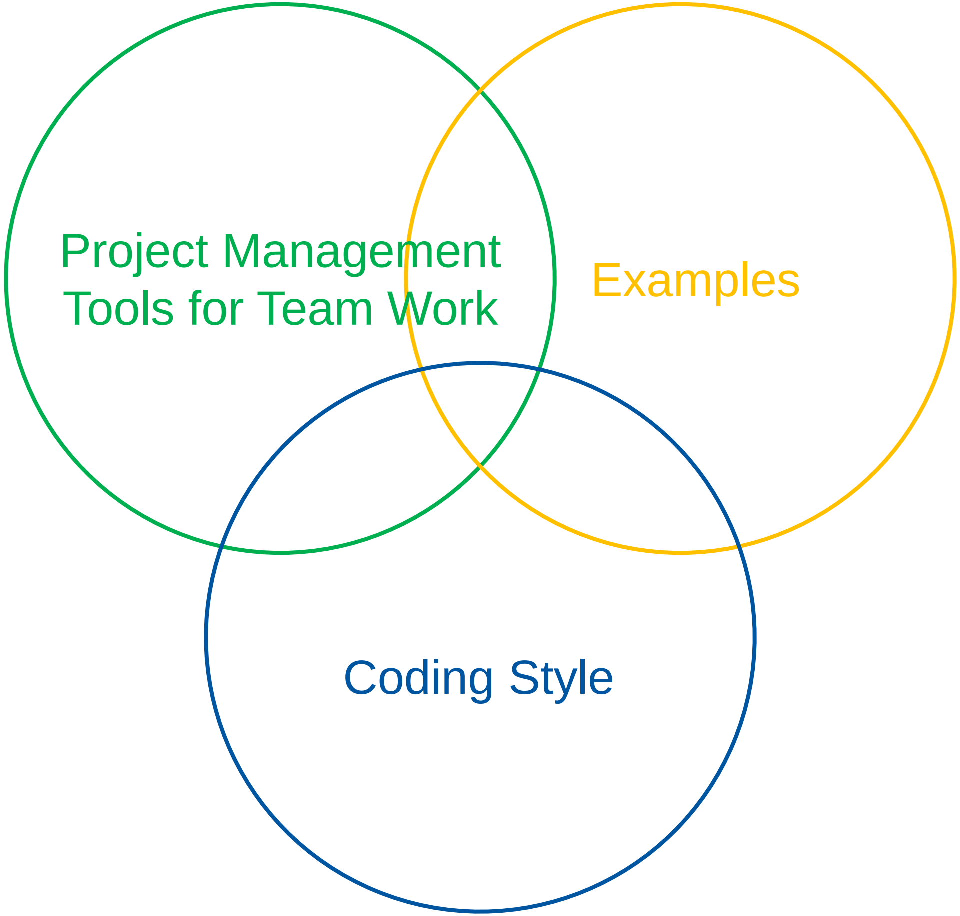
16.05.2019

Michał Maciejewski, Lorenzo Bortot, Marco Prioli, Bernhard Auchmann, Arjan Verweij
on behalf of the

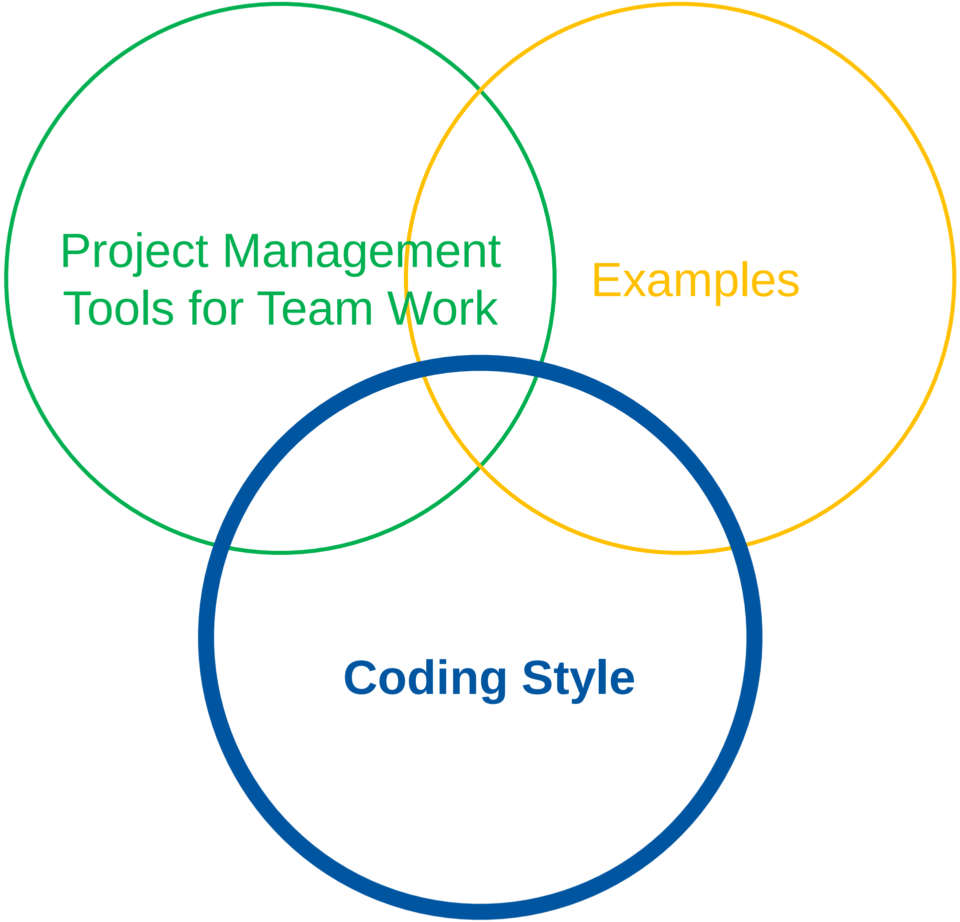
STEAM

TE-MPE-PE, TE-MPE-MS

Presentation Outline



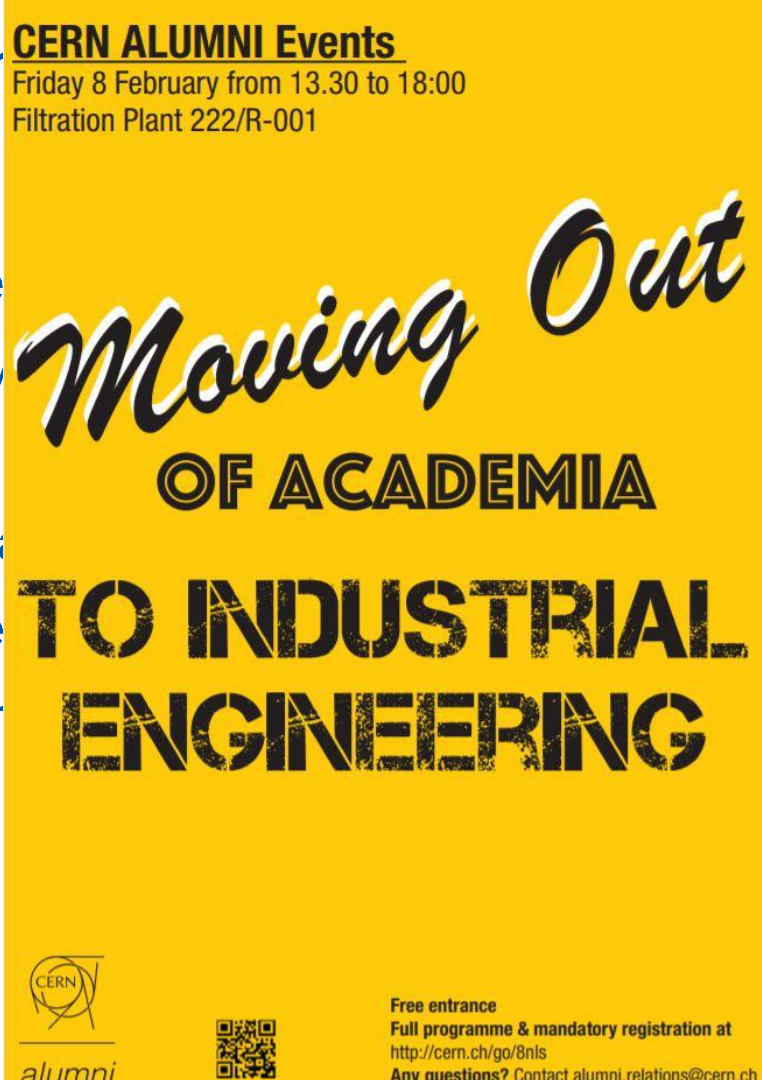
Presentation Outline



Some observations - Ch

We are

- a mix of people with broad expertise in multiple
- dealing with complex problems (simultaneously
- developing quite a lot of software
- a team people with undefinite and limited contra
- developing analysis tools beyond our stay here
- **potentially moving to industry where certair**



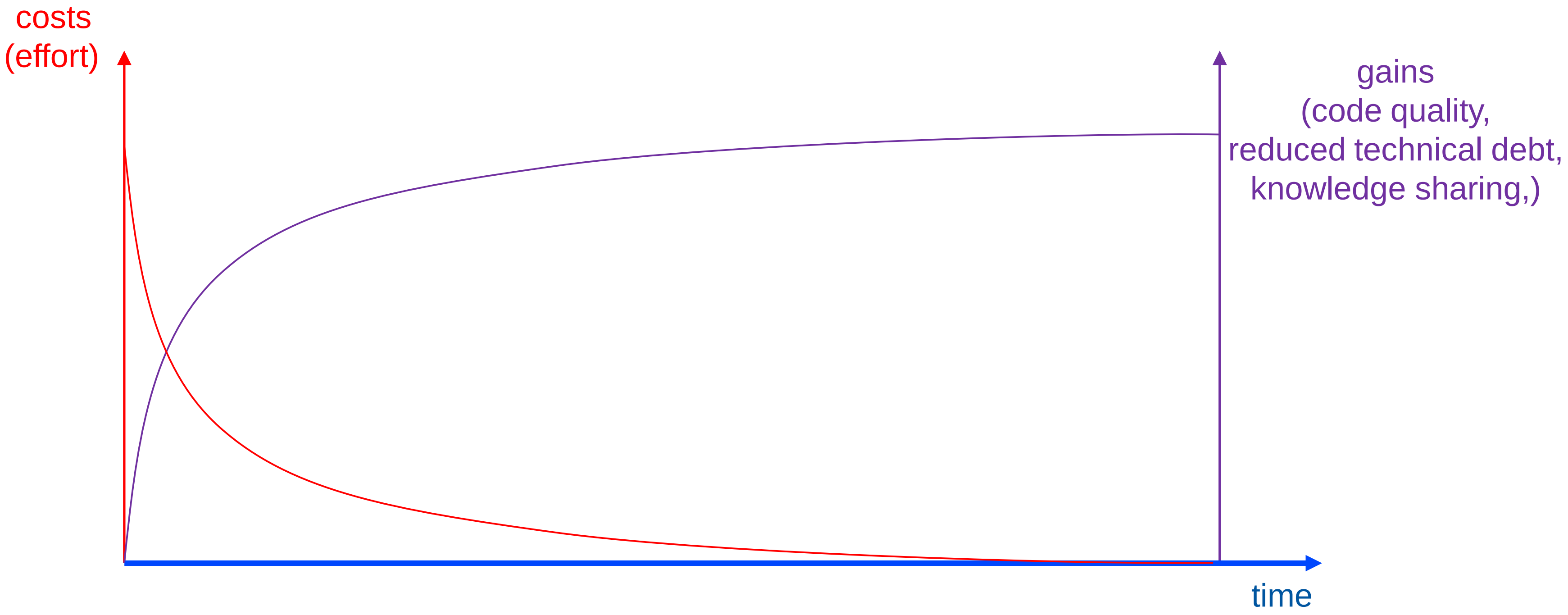
Some experience – **Solution and Results**

Since 2015, STEAM introduced good coding style. This allowed us to

- develop a hierarchical co-simulation framework (4 official releases)
- cover the accelerator needs and support multiple studies (LHC, HL-LHC,FCC)*
- present results at several conferences and publish several papers

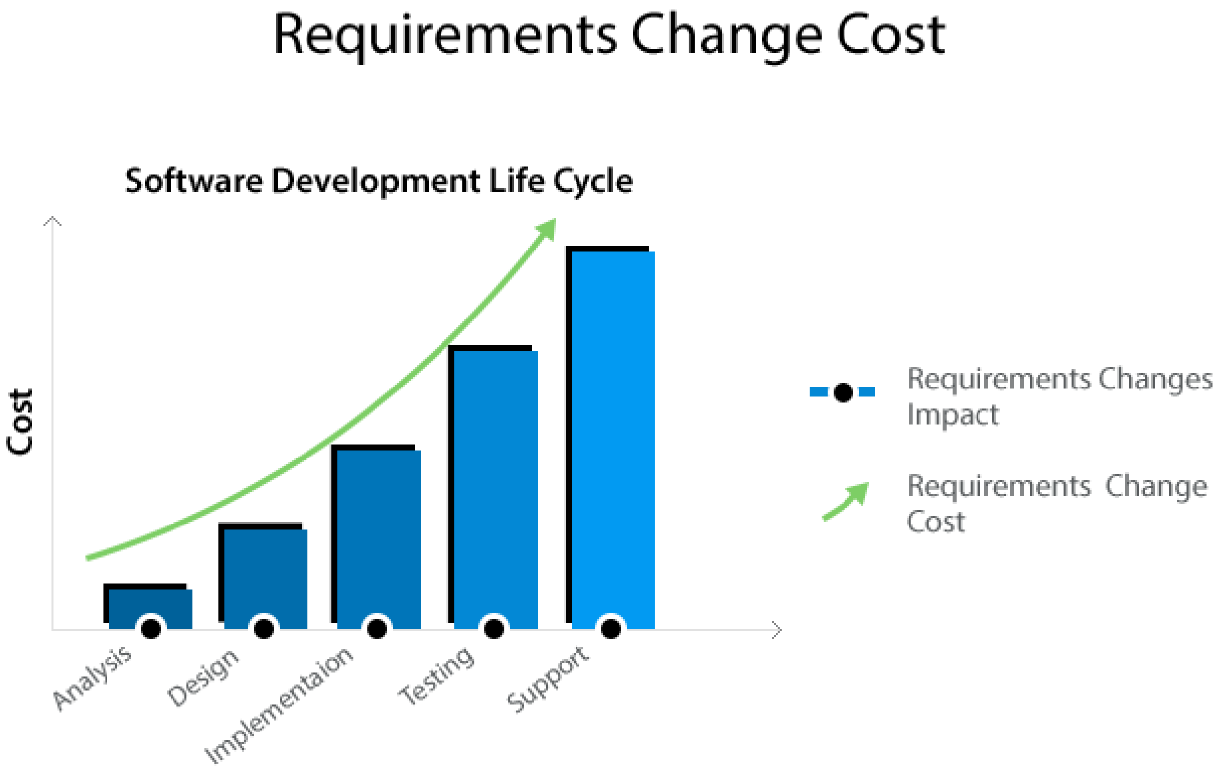
*For a more detailed summary please see Arjan's presentation at the TE Technical Meeting.

Gains vs. costs



The concepts to be presented are based on the common sense and we've been doing some of them.
The goal of this presentation is to structure these concepts and provide a common work strategy.

Costs in the Software Industry



One of the motivations for the reliability and availability studies in PE.

Costs in Research

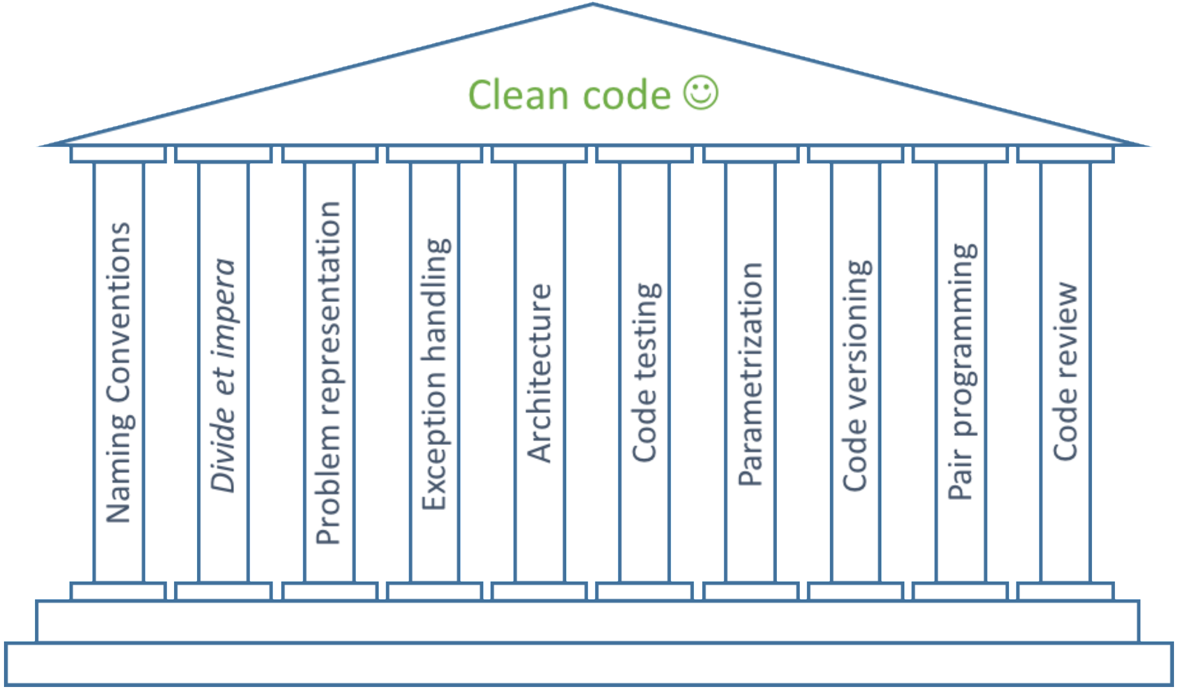
A prerequisite to use simulations is a proper understanding of the underlying process and development of a mathematical model. Further challenges include verifying correctness of obtained results and dealing with the computational complexity associated with large-scale simulations. The importance of verification was shown in a recent paper about inflated false-positive rates in fMRI studies [9]. The authors found a 15 year old software bug in one of the most popular tools that could have an impact on thousands of research papers. The breakdown of single core performance improvements around 2004 accompanied by industries shift to more and more parallelism made the design of complex simulations increasingly difficult [10].

Source: L. Breitwieser, From Cortex3D to BioDynaMo The Birth of a New Platform for Large-Scale Reproducible Biological Simulations

```
// When I wrote this, only God and I understood what I was doing
// Now, God only knows
```

-anonymous

The Temple of Clean Code



9 July 2015, Clean code development workshop, jointly with MPE-MS
13 Aug 2015, Object oriented programming workshop, jointly with MPE-MS

```
DatabaseRow row = Database.getRow();
int a = row.column1;
int b = row.column2;

double c = b / a;
data.add(new Measurement(t, c));
```

AbcCd

Naming covention (Oracle style for Java)

- Name variables meaningfully and in unified way – **exampleDescriptiveName**
- Choose function names that self describe themselves – **exampleFunction()**
- Use only well know acronyms – **Qps, Lhc,** avoid less known – **Dqamcnmb**,

	Physical Variable (use of underscores)	**Regular Variables (CamelCase)**
Class field/local variable	t_fpa	fileName
Constant*	TIME_FPA	FILE_NAME
Setter**	setT_fpa(t_fpa)	setFileName(fileName)
Getter**	getT_fpa()	getFileName()

* (to avoid confusion with temperature

** here we follow IntelliJ IDEA default formatting

We are using English names of greek letters: alpha, rho, omega, etc.

Dictionary for a description of terms used in the code: see SIGMA Magnet glossary for naming unification

```java
DatabaseRow row = Database.getRow();
int current = row.column1;
int b = row.column2;

double c = b / current;
data.add(new Measurement(t, c));
```

```
DatabaseRow row = Database.getRow();
int current = row.column1;
int voltage = row.column2;

double c = voltage / current;
data.add(new Measurement(t, c));
```

```
DatabaseRow row = Database.getRow();
int current = row.column1;
int voltage = row.column2;

double resistance = voltage / current;
data.add(new Measurement(time, resistance));
```

```java
double varianceEnExp = Math.variance(tauEnExp);
double sigmaEnExp = Math.sqrt(varianceEnExp);
```

```java
double varianceEnExp = Math.variance(tauEnExp);
double sigmaEnExp = Math.sqrt(varianceEnExp);

double varianceEnLog = Math.variance(tauEnLog);
double sigmaEnLog = Math.sqrt(varianceEnLog);
```

```
double varianceEnExp = Math.variance(tauEnExp);
double sigmaEnExp = Math.sqrt(varianceEnExp);

double varianceEnLog = Math.variance(tauEnLog);
double sigmaEnLog = Math.sqrt(varianceEnLog);

double varianceDer = Math.variance(tauDer);
double sigmaDer = Math.sqrt(varianceDer);
```

f(x)

Use of functions (*divide et impera*)

- use of functions –
  - code reuse and readability
  - one function does one operation
  - code testing
- object-oriented programming/modelling
  - structured problem representation
  - encapsulation
  - inheritance

```java
double varianceEnExp = Math.variance(tauEnExp);
double sigmaEnExp = Math.sqrt(varianceEnExp);

double varianceEnLog = Math.variance(tauEnLog);
double sigmaEnLog = Math.sqrt(varianceEnLog);

double varianceDer = Math.variance(tauDer);
double sigmaDer = Math.sqrt(varianceDer);

double calculateSigma(double[] tau) {
  double variance = Math.variance(tau);
  return Math.sqrt(variance);
}
```

```java
double sigmaEnExp = calculateSigma(tauEnExp);
double sigmaEnLog = calculateSigma(tauEnLog);
double sigmaDer = calculateSigma(tauDer);

double calculateSigma(double[] tau) {
  double variance = Math.variance(tau);
  return Math.sqrt(variance);
}
```

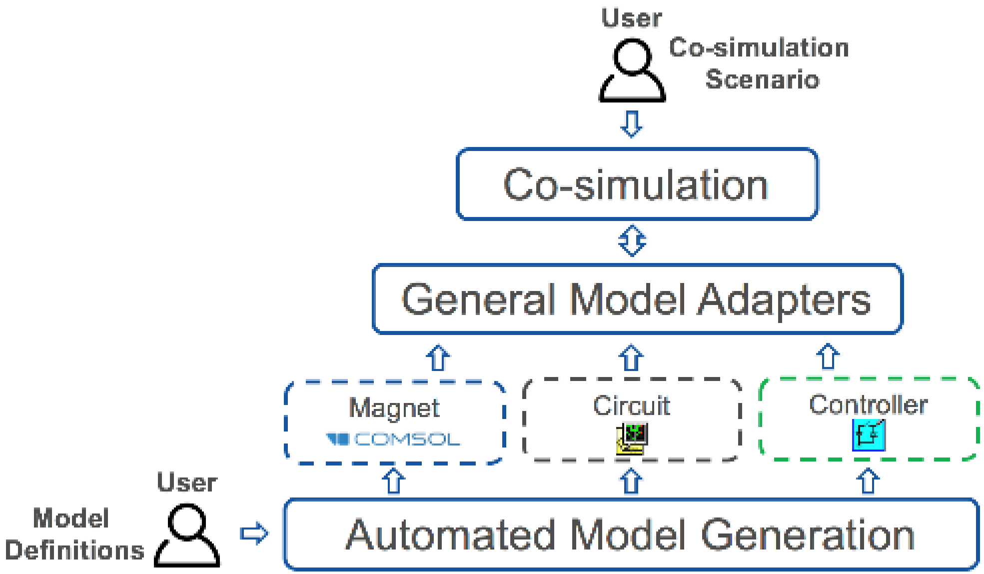
Problem representation – pseudo code

- Pseudo code is a technology independent code sceleton
- Uses words to describe intuitively the problem
- Very helpful at the initial stage for short reviews with other people

```
For each signal from signalValues
    multiply signal by SCALAR
    assign signal to
modifiedSignalValues
end for
```

Problem representation – diagrams

- With graphical diagrams we can map our problem observe relations, hierarchy and flow of data.
- Both pseudo code and graphical diagrams is already a solid documentation.



```java
private double[] readFile(String filePath) {
  FileReader.openFile(filePath);
  double[] signalValues = FileReader.readDoubles();
  FileReader.closeFile();
  return signalValues;
}
```

What if a file does not exist?!

Exception handling

- Input/Output statements has to be covered with *try…catch* statements
- Never an easy fix of logic mistake in an algorithm

```
try {
        /* perform operation that might
throw exception */
} catch ( /* expected exception */ ) {
        /* perform exceptional operation-
come back from error state */
}
```

What if file does not exist?

```
private double[] readFile(String filePath) {
  try {
    FileReader.openFile(filePath);
  } catch(FileNotFoundException exception) {
            logError("File doesn't exist!", exception);
            FileUtils.createNewFile();
  }
  double[] signalValues = FileReader.readDoubles();
  FileReader.closeFile();
  return signalValues;
}
```

This situation is expected to happen so we can prepare countermeasures.
This exception is handled!

Handling exceptional situations

- Example of bad usage

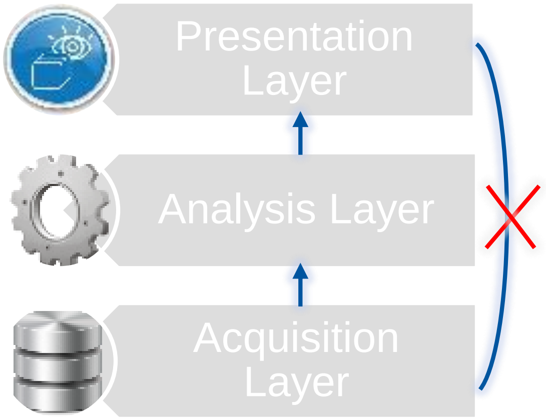
```
try {
        performSimulation(5);
        } catch (Exception e1) {
            try {
                performSimulation(4);
            } catch (Exception e2) {
                try {
                    performSimulation(3);
                } catch (Exception e3) {
                    try {
                        performSimulation(2);
                    } catch (Exception e4) {
                    }
                }
            }
        }
```

```
private void readProcessAndPlotData() {
  /* Open and read file with voltage signal */
  FileReader.openFile(INPUT_FILE_PATH);
  double[] voltageValues= FileReader.readDoubles();
  FileReader.closeFile();

  /* Divide voltage by constant current to get resistance. */
  int[] resistanceValues = new int[voltageValues.length];
  for (int i = 0; i < voltageValues.length; i++) {
       resistanceValues[i] = voltageValues[i] / CURRENT;
  }

  /* Plot resistance on the chart. */
  Chart chart = ChartUtils.createChart();
  chart.setValues(resistanceValues);
  chart.display();
}
```

Is it a clean code?

Architecture

- Acquire/Calculate -> Analyse -> Present/Store
- If at least two of those steps are present, they should be clearly separated



```java
private void readAndPlotData() {
  /* Open and read file with voltage signal */
  FileReader.openFile(INPUT_FILE_PATH);
  double[] voltageValues= FileReader.readDoubles();
  FileReader.closeFile();

  /* Divide voltageby constant current to get resistance. */
  int[] resistanceValues= new int[voltageValues.length];
  for (int i = 0; i < voltageValues.length; i++) {
       resistanceValues[i] = voltageValues[i] / CURRENT;
  }

  /* Plot resistance on the chart. */
  Chart chart = ChartUtils.createChart();
  chart.setValues(resistanceValues);
  chart.display();
}
```

```java
private void readAndPlotData() {
  double[] voltageValues= readFile(INPUT_FILE_PATH);

  /* Divide voltage by constant current to get resistance. */
  int[] resistanceValues= new int[voltageValues.length];
  for (int i = 0; i < voltageValues.length; i++) {
      resistanceValues[i] = voltageValues[i] / CURRENT;
  }

  /* Plot values on the chart. */
  Chart chart = ChartUtils.createChart();
  chart.setValues(resistanceValues);
  chart.display();
}

private double[] readFile(String filePath) {
  FileReader.openFile(filePath);
  double[] signalValues = FileReader.readDoubles();
  FileReader.closeFile();
  return signalValues;
}
```

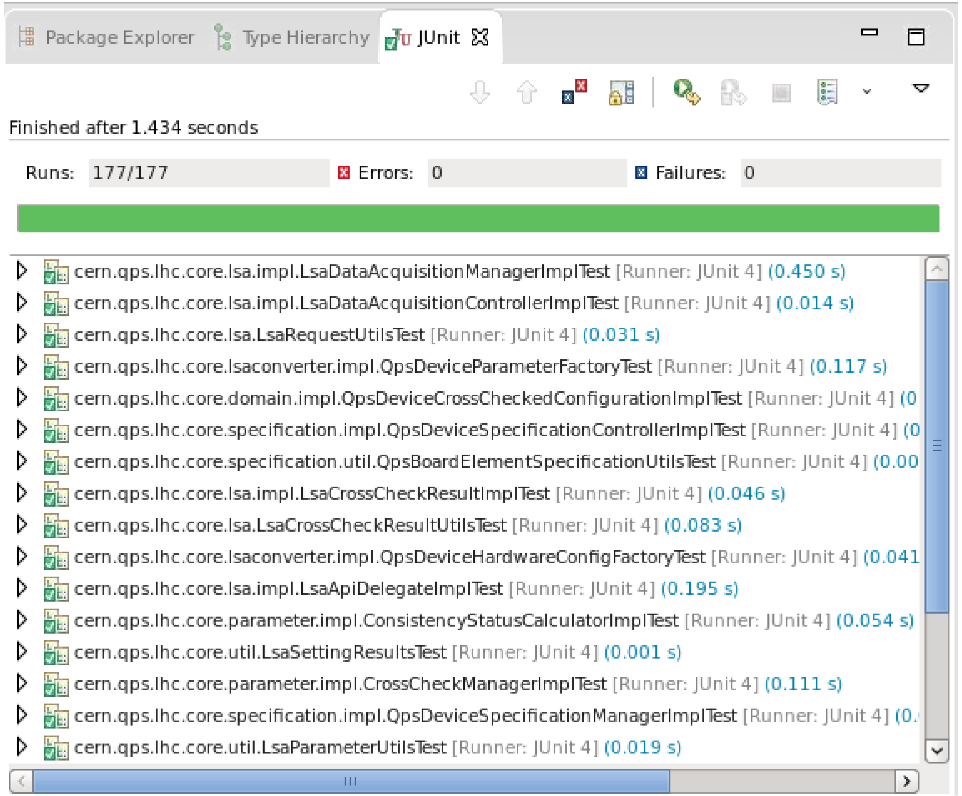
```java
private void readAndPlotData() {
  double[] voltageValues= readFile(INPUT_FILE_PATH);
  double[] resistanceValues= calculateResistance(voltageValues, CURRENT);

  /* Plot values on the chart. */
  Chart chart = ChartUtils.createChart();
  chart.setValues(resistanceValues);
  chart.display();
}

private double[] readFile(String filePath) {
  FileReader.openFile(filePath);
  double[] signalValues = FileReader.readDoubles();
  FileReader.closeFile();
  return signalValues;
}

private double[] calculateResistance(double[] voltageValues, int current) {
  double[] resistanceValues= new int[voltageValues.length];
  for (int i = 0; i < voltageValues.length; i++) {
     resistanceValues[i] = voltageValues[i] / current ;
  }
  return resistanceValues;
}
```

```
private void readAndPlotData() {
  double[] voltageValues= readFile(INPUT_FILE_PATH);
  double[] resistanceValues= calculateResistance(voltageValues, CURRENT);
  plotValues(resistanceValues);
}

private double[] readFile(String filePath) {
  FileReader.openFile(filePath);
  double[] signalValues = FileReader.readDoubles();
  FileReader.closeFile();
  return signalValues;
}

private double[] calculateResistance(double[] voltageValues, int current) {
  double[] resistanceValues= new int[voltageValues.length];
  for (int i = 0; i < voltageValues.length; i++) {
     resistanceValues[i] = voltageValues[i] / current ;
  }
  return resistanceValues;
}

private void plotValues(double[] signalValues) {
  Chart chart = ChartUtils.createChart();
  chart.setValues(signalValues);
  chart.display();
}
```

```java
public double calculateResistance(double circuitVoltage, double circuitCurrent) {
        return circuitVoltage / circuitCurrent;
}
```

Code testing

- Benefit from tested project in case
- Understand code better reading e
- Early find possible problems
- Crucial models with external depe
- Requirements -> Code –> Tests
- Our code is as good as tests dem



http://www.agilemodeling.com/essays/executableSpecifications.htm

```
public double calculateResistance(double circuitVoltage, double circuitCurrent) {
          return circuitVoltage / circuitCurrent;
}
```

```java
public double calculateResistance(double circuitVoltage, double circuitCurrent) {
        return circuitVoltage / circuitCurrent;
}

@Test
public void testCalculateResistance() {
        assertEquals(5, calculateResistance(10, 2));
}
```

```java
public double calculateResistance(double circuitVoltage, double circuitCurrent) {
        return circuitVoltage / circuitCurrent;
}

@Test
public void testCalculateResistance() {
        assertEquals(5, calculateResistance(10, 2));
}

@Test(expected = ArithmeticException.class)
public void testCalculateResistanceWithException() {
        calculateResistance(10, 0);
}
```

Parametrization

- Single parameters as inputs for main function
- Paths, configuration settings, references as .json, .csv file
- NO HARDCODED PARAMETERS IN CODE!

```
/* Uncomment if you want to calculate [illegible]m voltage measured at 07Jun2015. */
FileReader.openFile(“voltage07Jun[illegible]v”);
/* Uncomment if you want to calcul[illegible]res[illegible]ce fr[illegible]voltage measured at 17Jun2015. */
// FileReader.openFile(“voltage17J[illegible]015.c[illegible]
/* Uncomment if you want to calcula[illegible]sistan[illegible] voltage measured at 01Aug2014.
*/
// FileReader.openFile(“voltage01Aug201[illegible]”);
```

```
[kkrol@cwe-513-vml248]$ ./dataAnalysis voltage07Jun2015.csv
Analysis output: SUCCESS
[kkrol@cwe-513-vml248]$ ./dataAnalysis voltage17Jun2015.csv
Analysis output: SUCCESS
[kkrol@cwe-513-vml248]$ ./dataAnalysis voltage01Aug2014.csv
Analysis output: FAILURE
```

V1.txt
V2.txt
V3_A.txt

Code versioning – http://gitlab.cern.ch

- Management of changes made to documents
- Facilitates cooperation between developers
- Unified storage for changes, simple look into the history, everything in one place
- Version code/input text-based files at every stage at every project
- **Do it on daily basis**

Pair programming

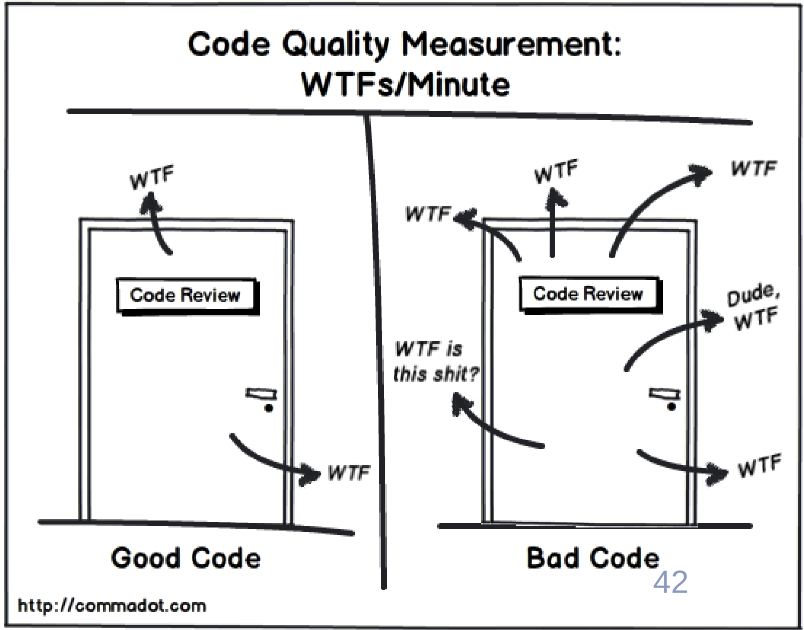
- At every stage of every project within group as well as with our software team
- Very helpful in code merging
- Two developers, one keyboard
  - One writes code
  - One thinks about further steps, looks from wide perspective, finds conceptual problems

Code review

- Well structured code (naming conventions) with graphical representation is very helpful
- Supports knowledge sharing among team members
- Often times we were inviting experts from TE-MPE-MS

For a code review, a developer should

- provide a code in good shape (naming convention, tests)
- indicate main areas to be reviewed
- indicate deadline



Best Practices for Scientific Computing

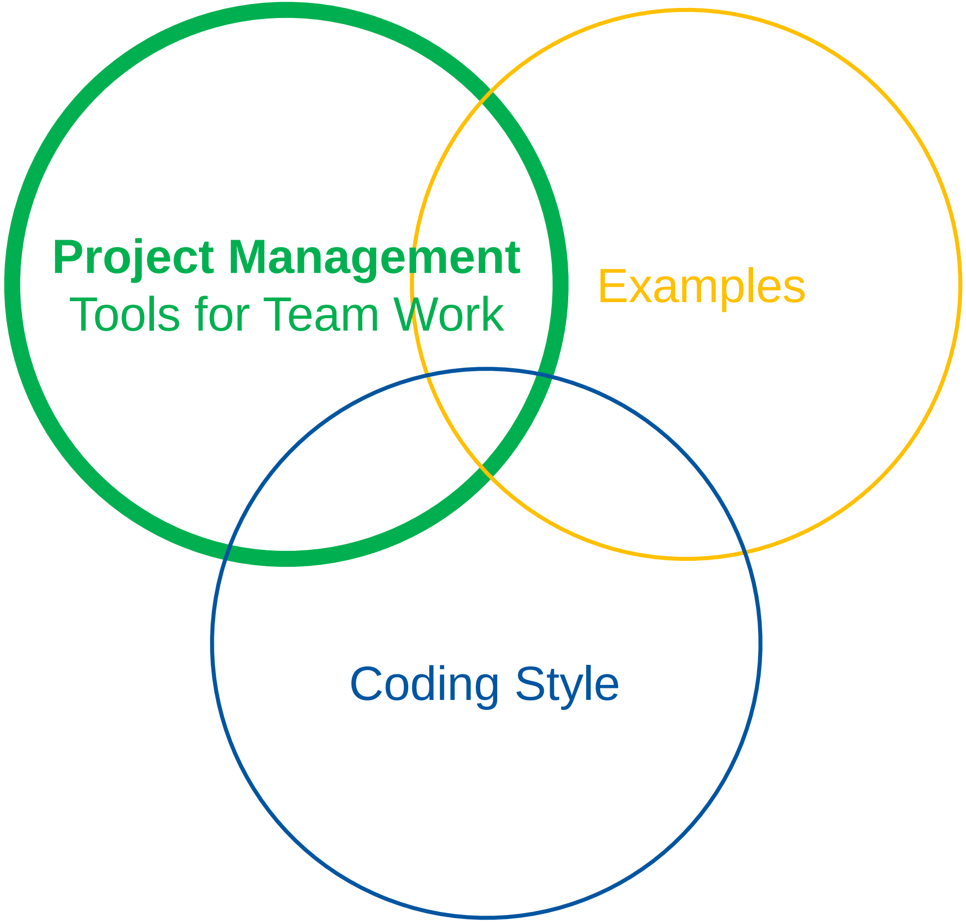
http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745

Box 1. Summary of Best Practices

1. Write programs for people, not computers.
   a. A program should not require its readers to hold more than a handful of facts in memory at once.
   b. Make names consistent, distinctive, and meaningful.
   c. Make code style and formatting consistent.
2. Let the computer do the work.
   a. Make the computer repeat tasks.
   b. Save recent commands in a file for re-use.
   c. Use a build tool to automate workflows.
3. Make incremental changes.
   a. Work in small steps with frequent feedback and course correction.
   b. Use a version control system.
   c. Put everything that has been created manually in version control.
4. Don't repeat yourself (or others).
   a. Every piece of data must have a single authoritative representation in the system.
   b. Modularize code rather than copying and pasting.
   c. Re-use code instead of rewriting it.
5. Plan for mistakes.
   a. Add assertions to programs to check their operation.
   b. Use an off-the-shelf unit testing library.
   c. Turn bugs into test cases.
   d. Use a symbolic debugger.
6. Optimize software only after it works correctly.
   a. Use a profiler to identify bottlenecks.
   b. Write code in the highest-level language possible.
7. Document design and purpose, not mechanics.
   a. Document interfaces and reasons, not implementations.
   b. Refactor code in preference to explaining how it works.
   c. Embed the documentation for a piece of software in that software.
8. Collaborate.
   a. Use pre-merge code reviews.

Citation: Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, et al. (2014) Best Practices for Scientific Computing. **PLoS Biology** 12(1): e1001745. https://doi.org/10.1371/journal.pbio.1001745

Presentation Outline



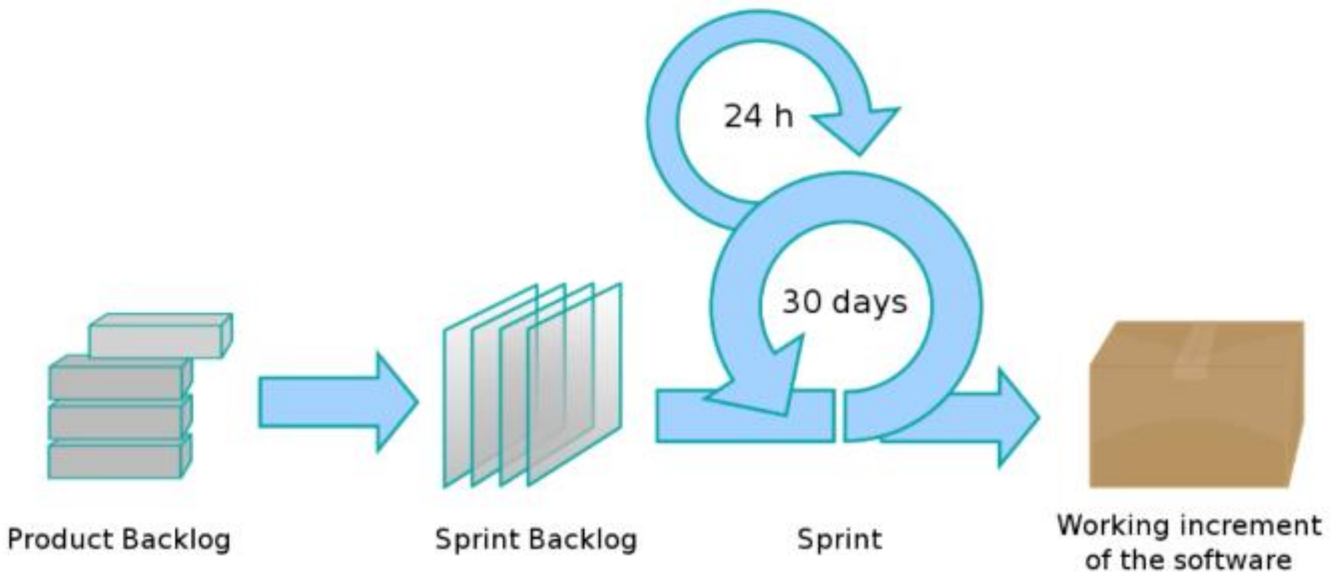
Agile manifesto

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

CERN

SCRUM methodology

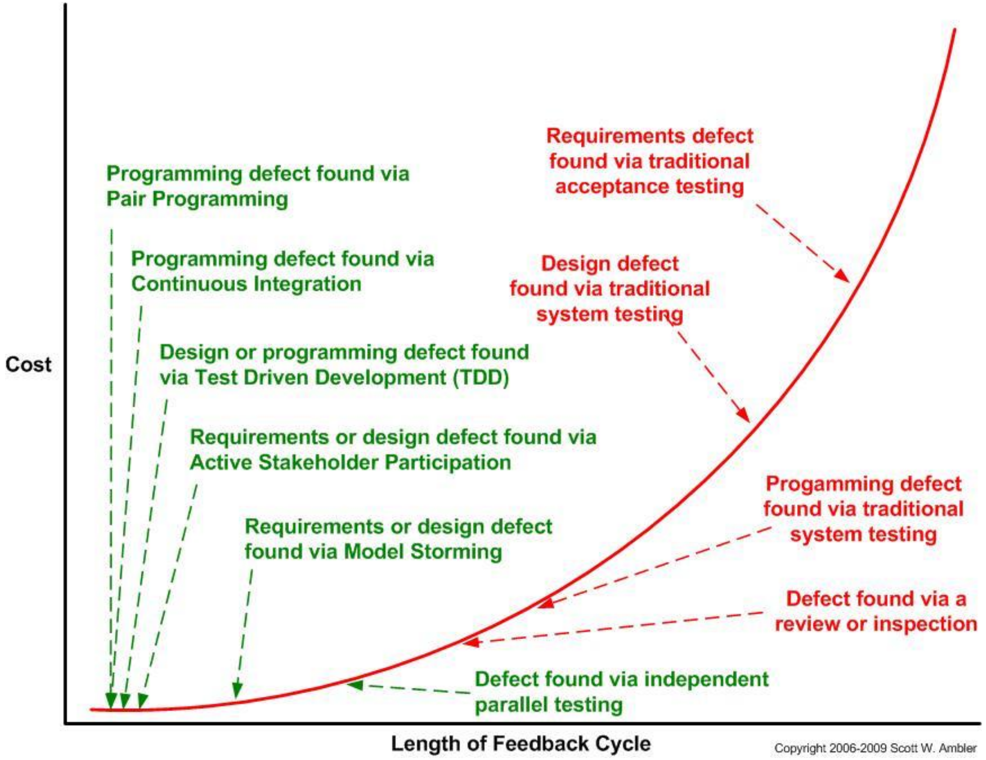
1. Sprint planning
2. Sprint execution
3. Sprint summary



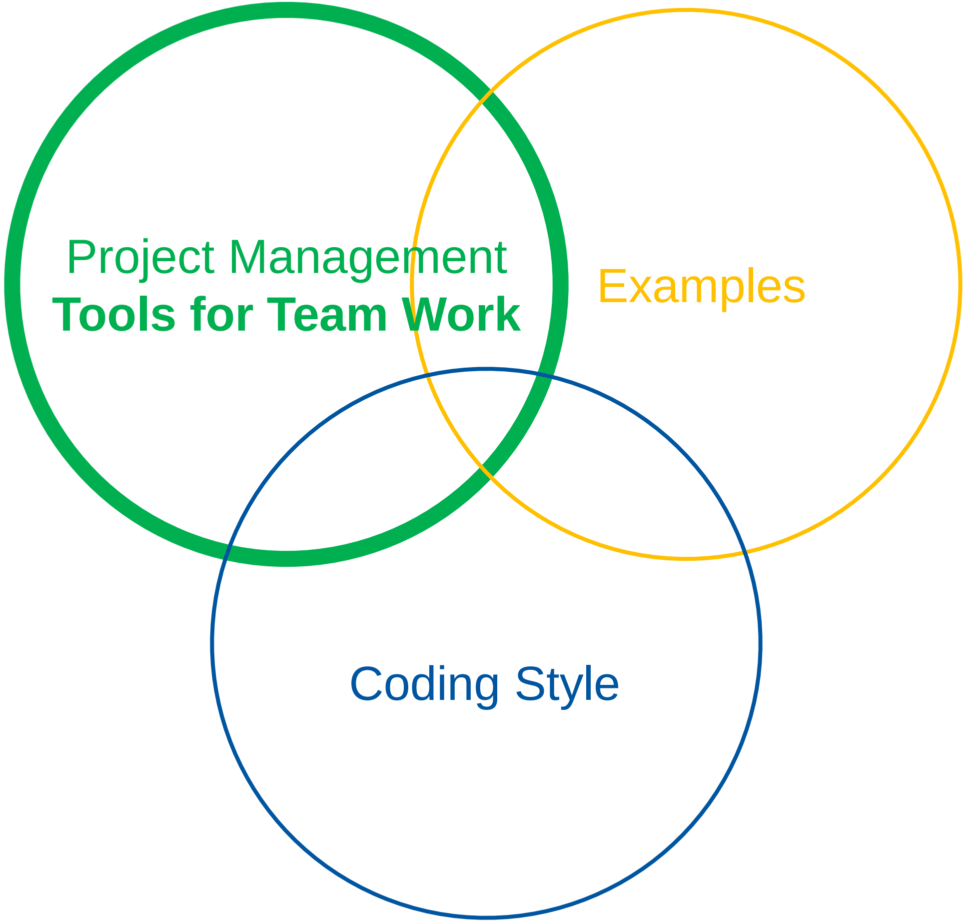
Priorities!!!

Source: Introduction to Agile Project Management, On-line course, Stanford Continuing Studies

Agile Approach Promises Cost Reduction

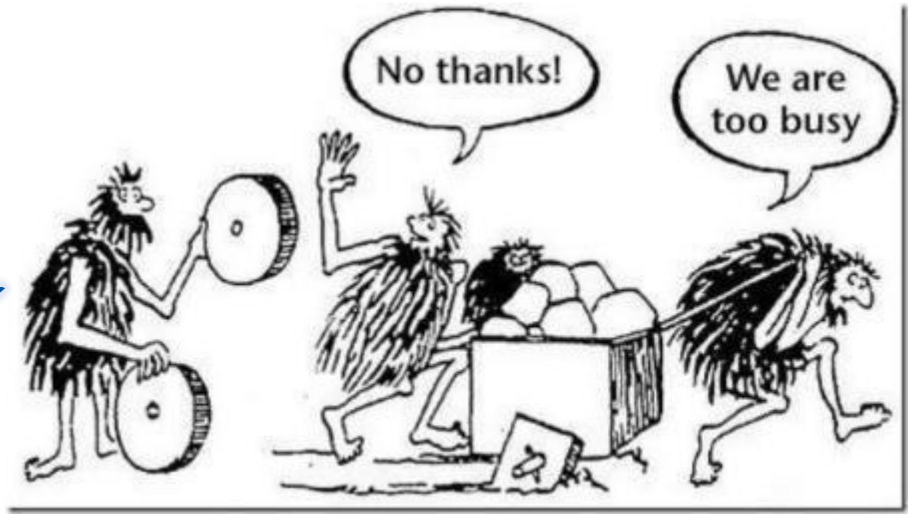


Presentation Outline



Motivation

- We have experience using some of the tools
- We (people in general) can be divided into two groups: *Those who **do** backups and those who **will do** backups*



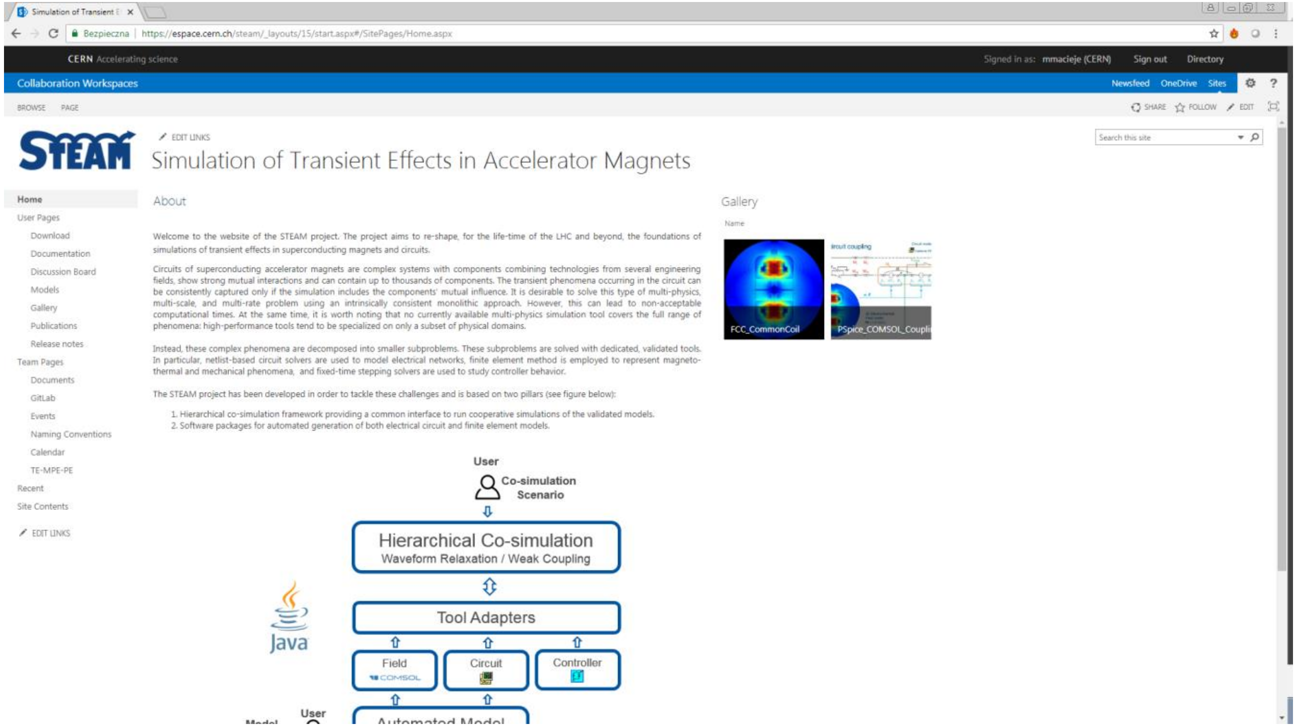
It's Me
(Me how)

Source: https://www.quora.com/I-work-really-very-hard-for-CSIR-life-science-but-still-my-paper-was-not-so-good-What-should-I-have-to-do-so-I-can-crack-it

Section website (http://cern.ch/te-mpe-pe)

- Our core product are analyses covering the accelerator needs (papers, posters, presentations, internal notes, design reports, …)
- Great platform to share our activities internally and externally!

Sharepoint* (cern.ch/steam)



*NB: CERN is in the process of Mexit

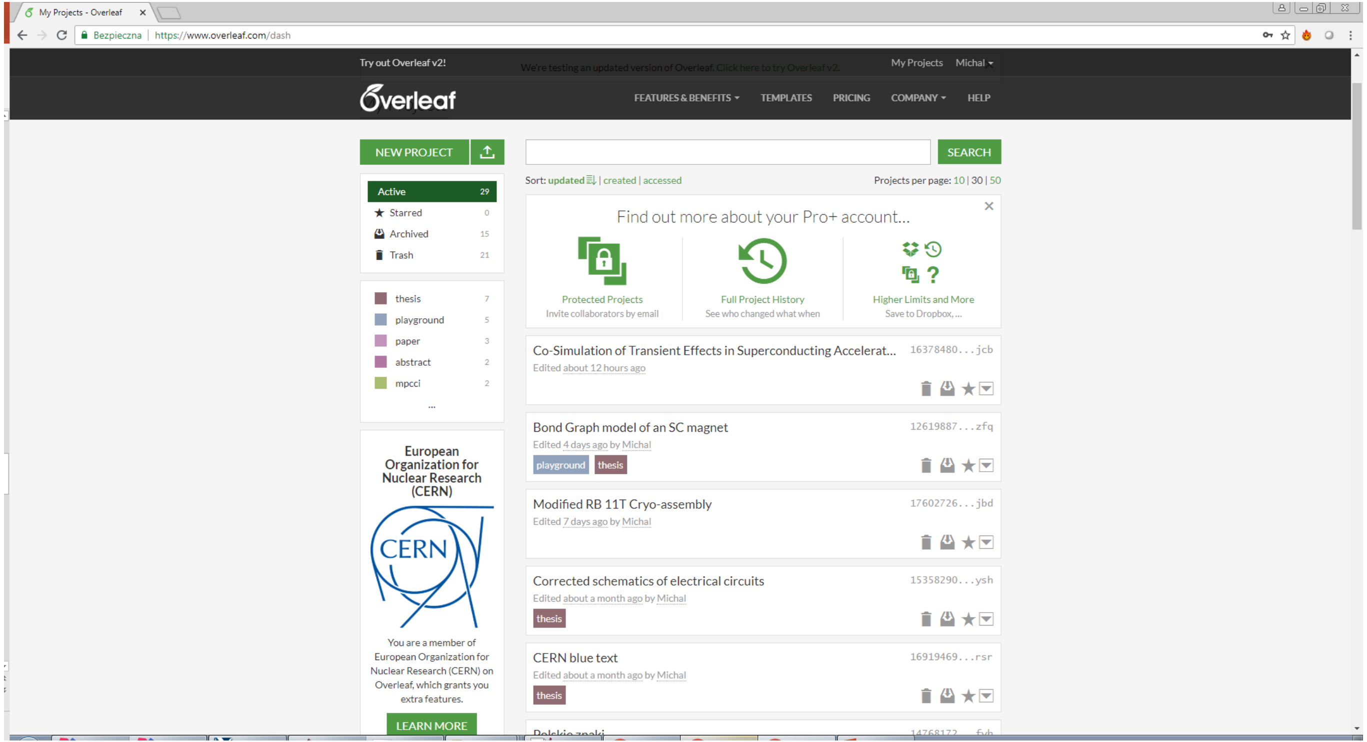
CERNBox (cernbox.cern.ch)

- Use of google drive, onedrive, dropbox, etc. is discouraged
- 1 TB per user (also personal files) / project
- Multiplatform (Android, iOS, Windows, OS, linux)
- Synchronises across multiple locations
- Stores 10 latest versions of a file

Multiple users in our section: Arjan, Bernhard, Matthieu, Michał, Christoph, Per, Zinur, Akrivi, Lorenzo, Marco, ...

Overleaf (http://overleaf.com)

- A platform for cooperative writing of papers + paper repository!
- No more hand written corrections, multiple versions of files: *Nature_paper_MM*.pdf, *Nature_paper_MM_v1*.pdf,…

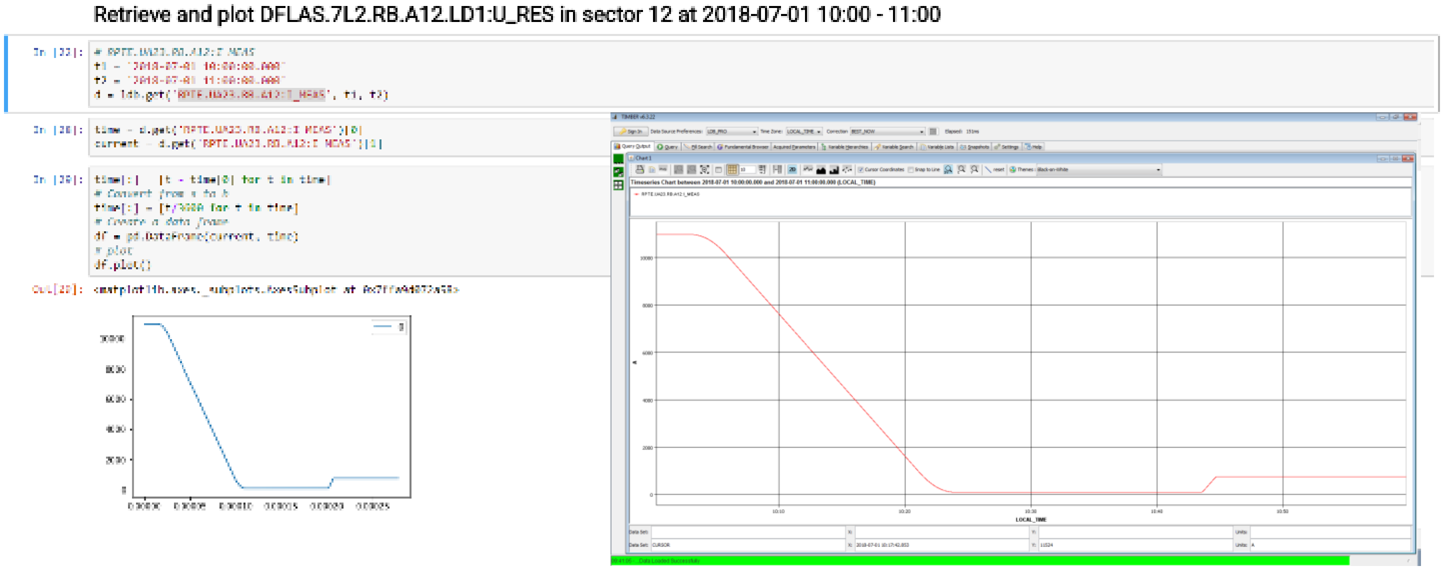


Overleaf (http://overleaf.com) + ShareLaTeX

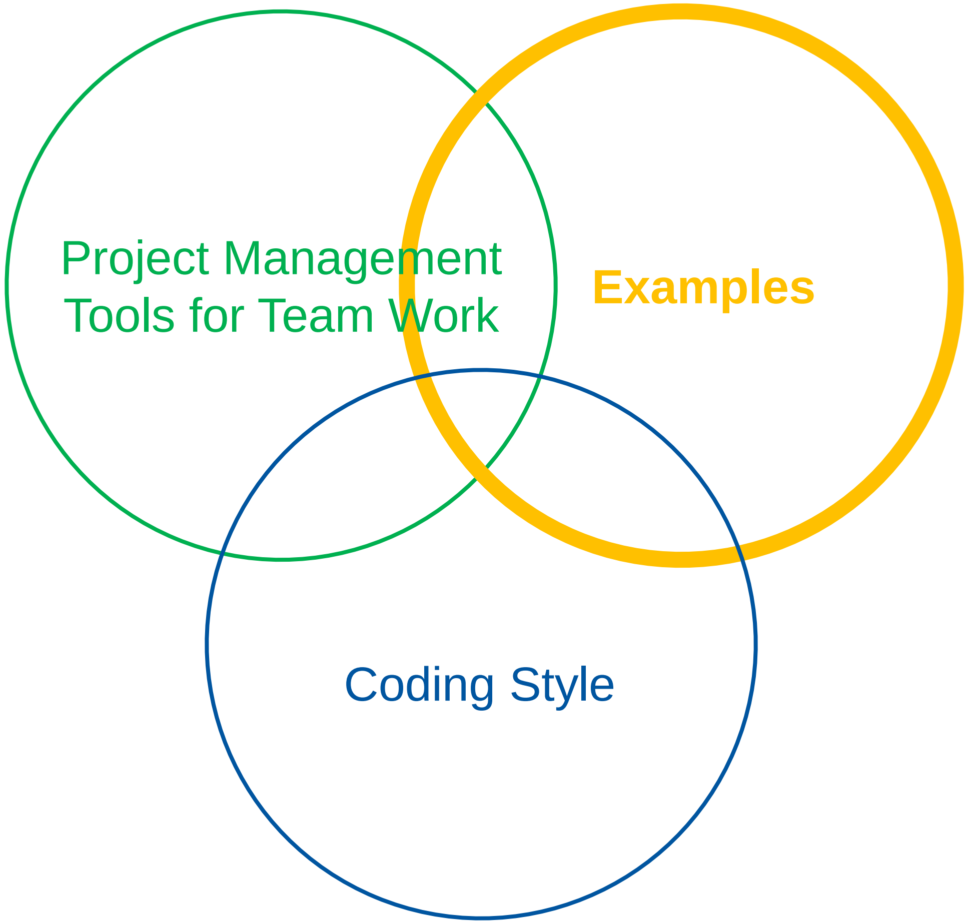
- A platform for cooperative writing of papers + paper repository!
- No more hand written corrections, multiple versions of files: *Nature_paper_MM*.pdf, *Nature_paper_MM_v1*.pdf,…

SWAN (http://swan.cern.ch)

- Analyse data without the need to install any software
- Access experiments' and user data in the CERN cloud – PM, CALS, NXCALS
- Share with colleagues
- **Notebook = code + output (in one file!)**



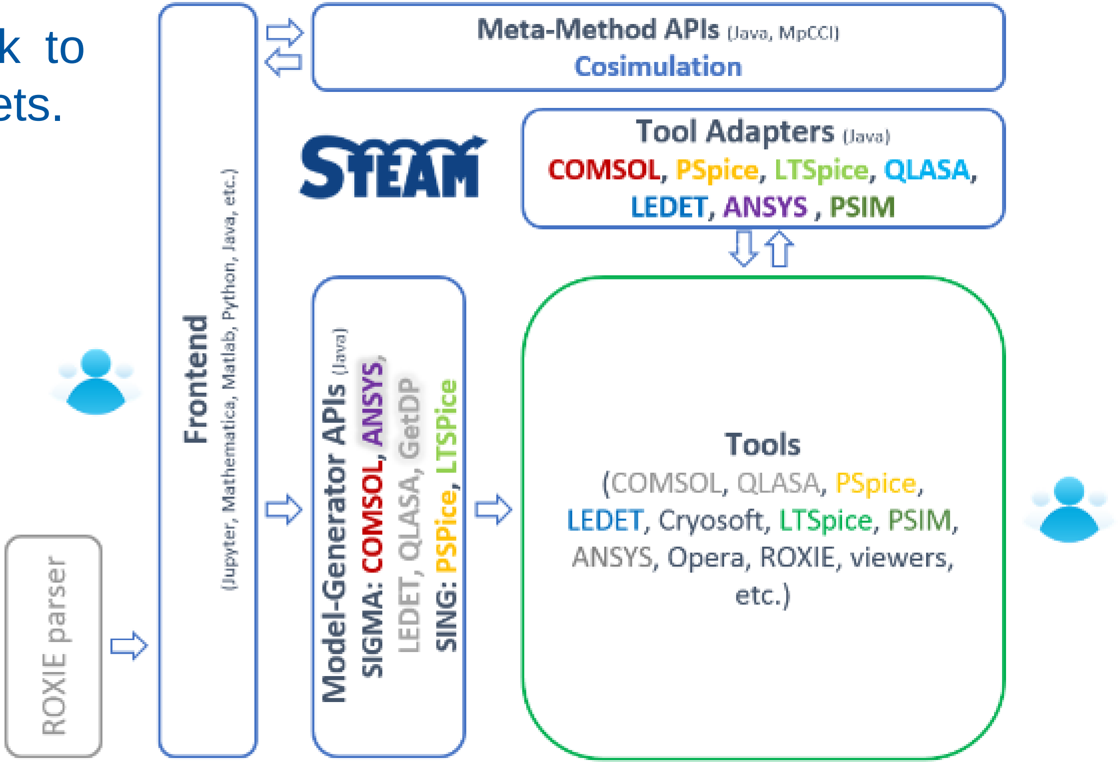
Presentation Outline



STEAM Overview (Bernhard's view)

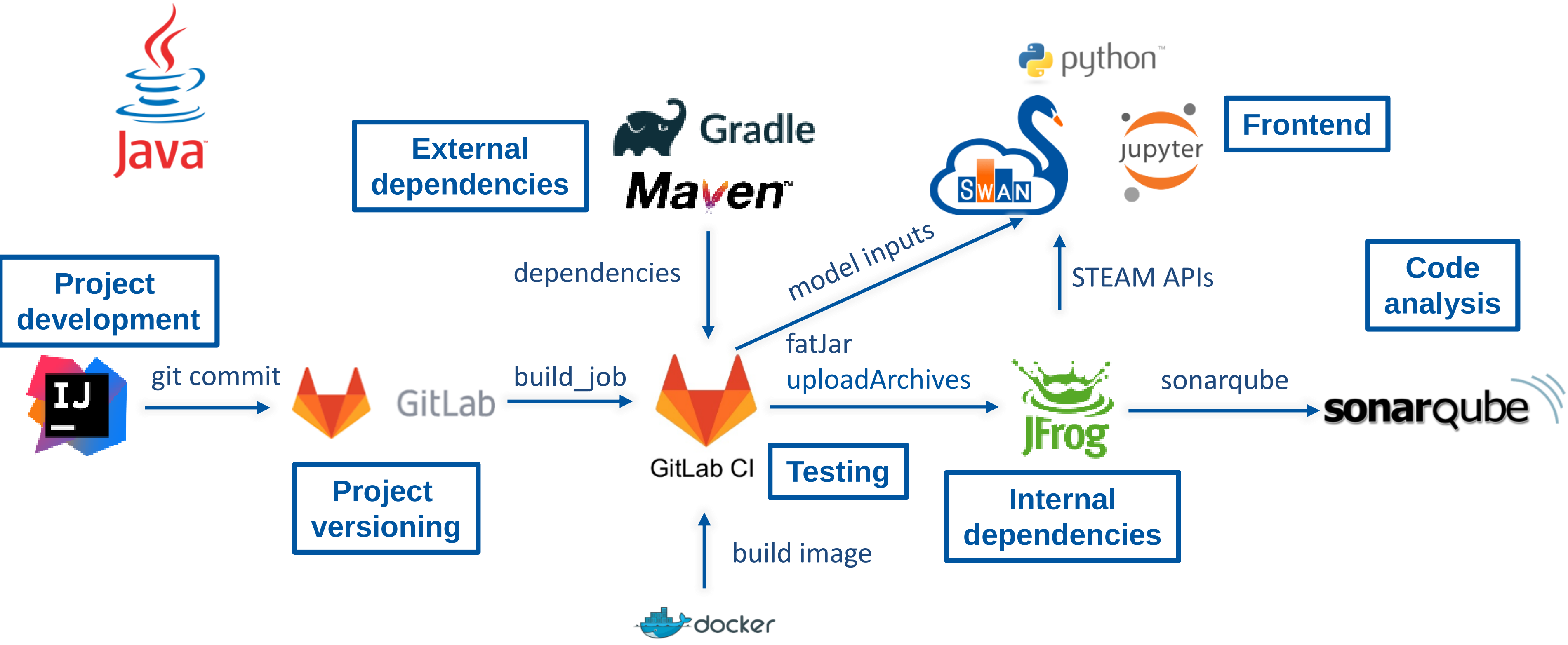
STEAM is a simulation framework to study transient effects in SC magnets.
It consists of several pillars:

1. Validated tools
   → standalone, co-simulation
2. Model generation API
3. Tool adapter API
4. Meta-Methods
   →co-simulation, optimization
5. Front-end to interact with APIs



Arjan's view
https://espace.cern.ch/steam/Shared%20Documents/Project%20Architecture/190206%20STEAM%20structure.pptx?Web=1

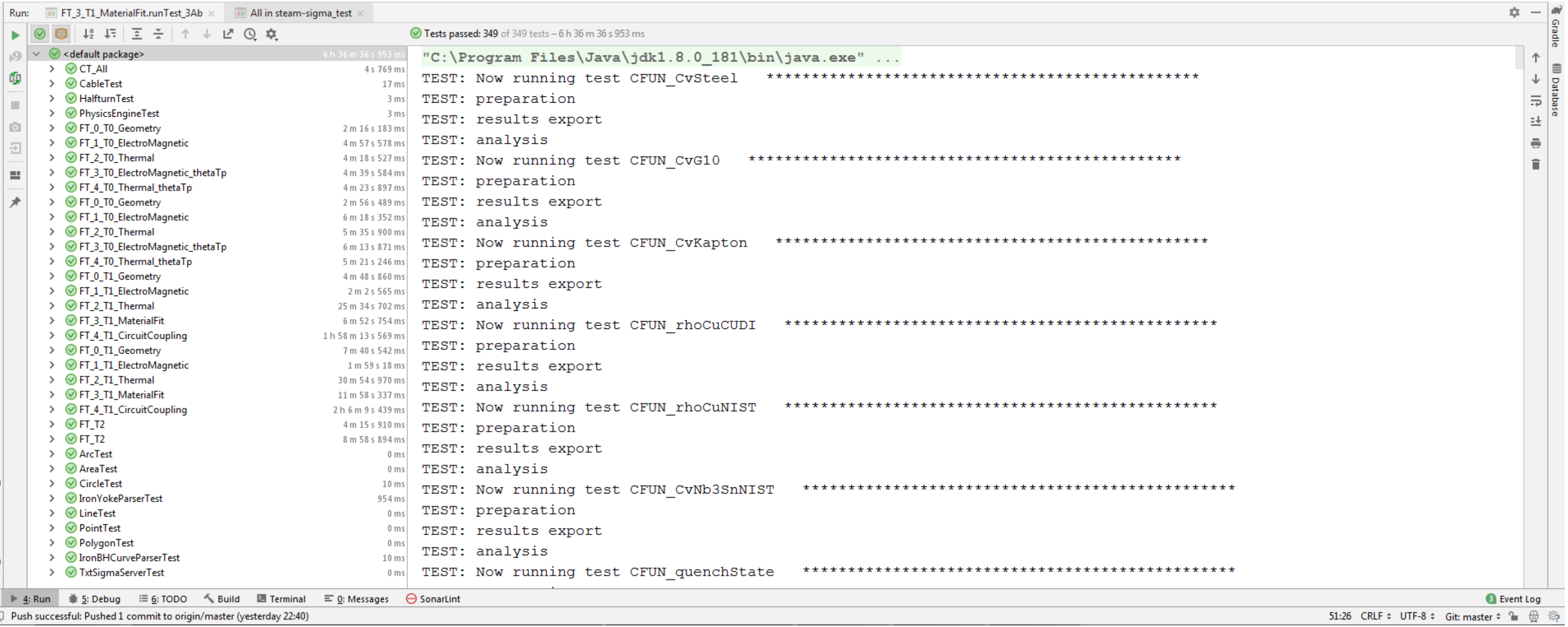
STEAM – Software Stack



The pipeline automatically executes project build, testing, sharing, and analysis.
This ensures the maintainability of the project (several 10k's of lines of code).

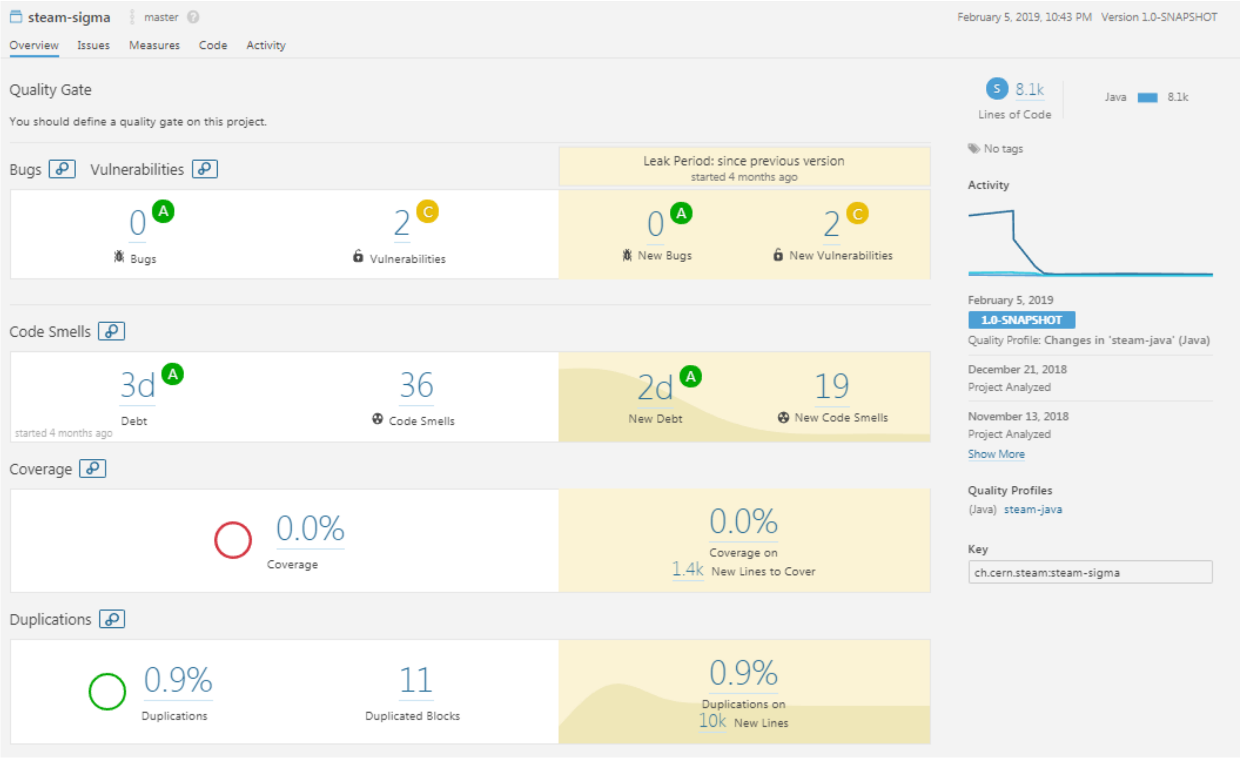
*Strong cooperation with MPE/MS (K. Król, J-C. Garnier)

STEAM-SIGMA – Unit & Integration Testing



Majority of tests are integration tests for the ANSYS UDE validation campaign by Lucas, Lorenzo and Edvard.

STEAM-SIGMA – Code Quality



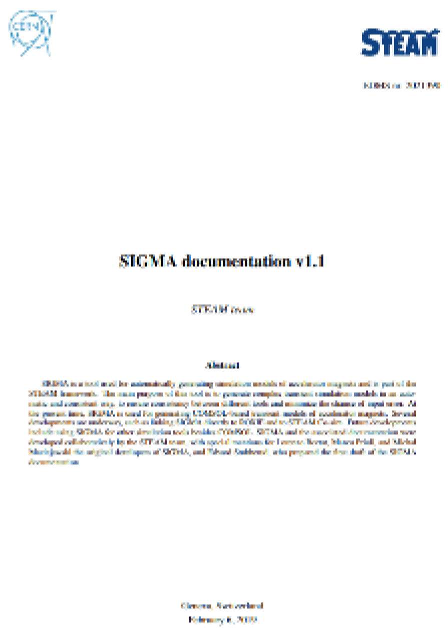
Feature	26.09.18	05.02.19
Bugs	117	0
Vulnerabilities	97	2
Code smells	**8.2 k**	36
Tech. debt	**85 d**	3 d
LoC (code+doc)	15 k	15 k

Most frequent issues:

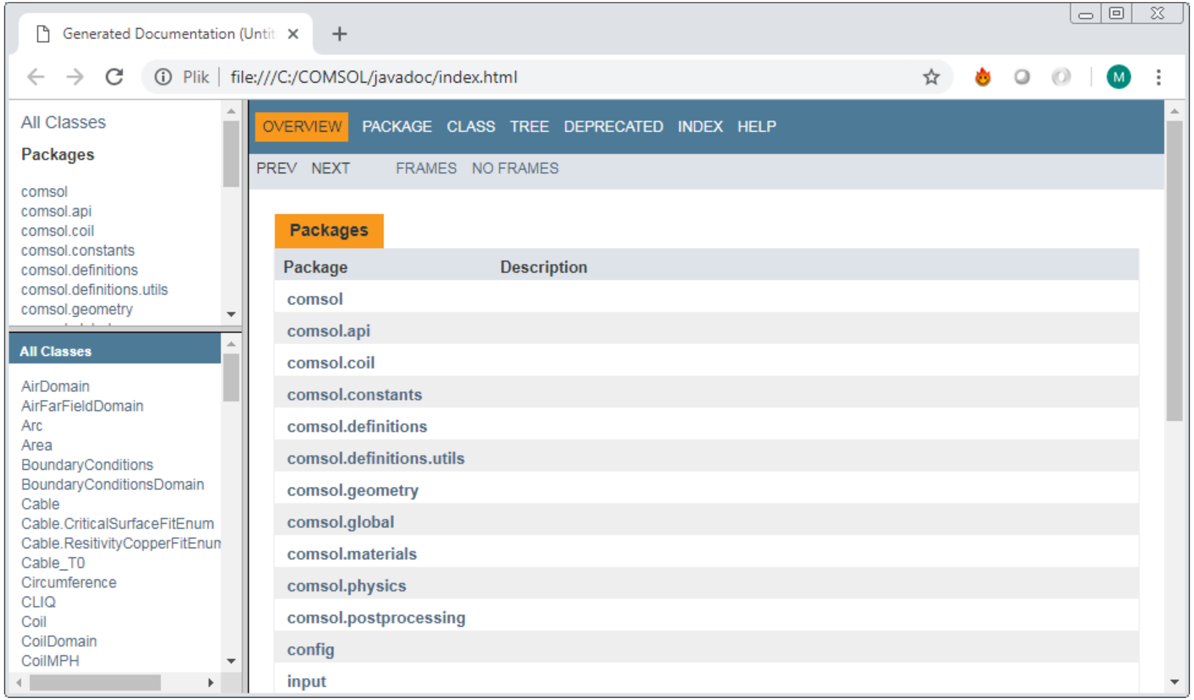
- No documentation
- Coding conventions violated
- Logic errors
- Code duplications
- Missing and failing tests

http://sonar.cern.ch/dashboard?id=ch.cern.steam%3Asteam-sigma

STEAM-SIGMA – Documentation



Minor updates



Documentation of the public API (demo)

LHC Signal Monitoring Project - Software Stack

Static code analysis
Test coverage

sonarqube

Interactive
notebooks

git clone

git push

GitLab

git clone

git push

Integrated
Development
Environment

Continuous
Integration

read

write

influxdb

Persistent
storage

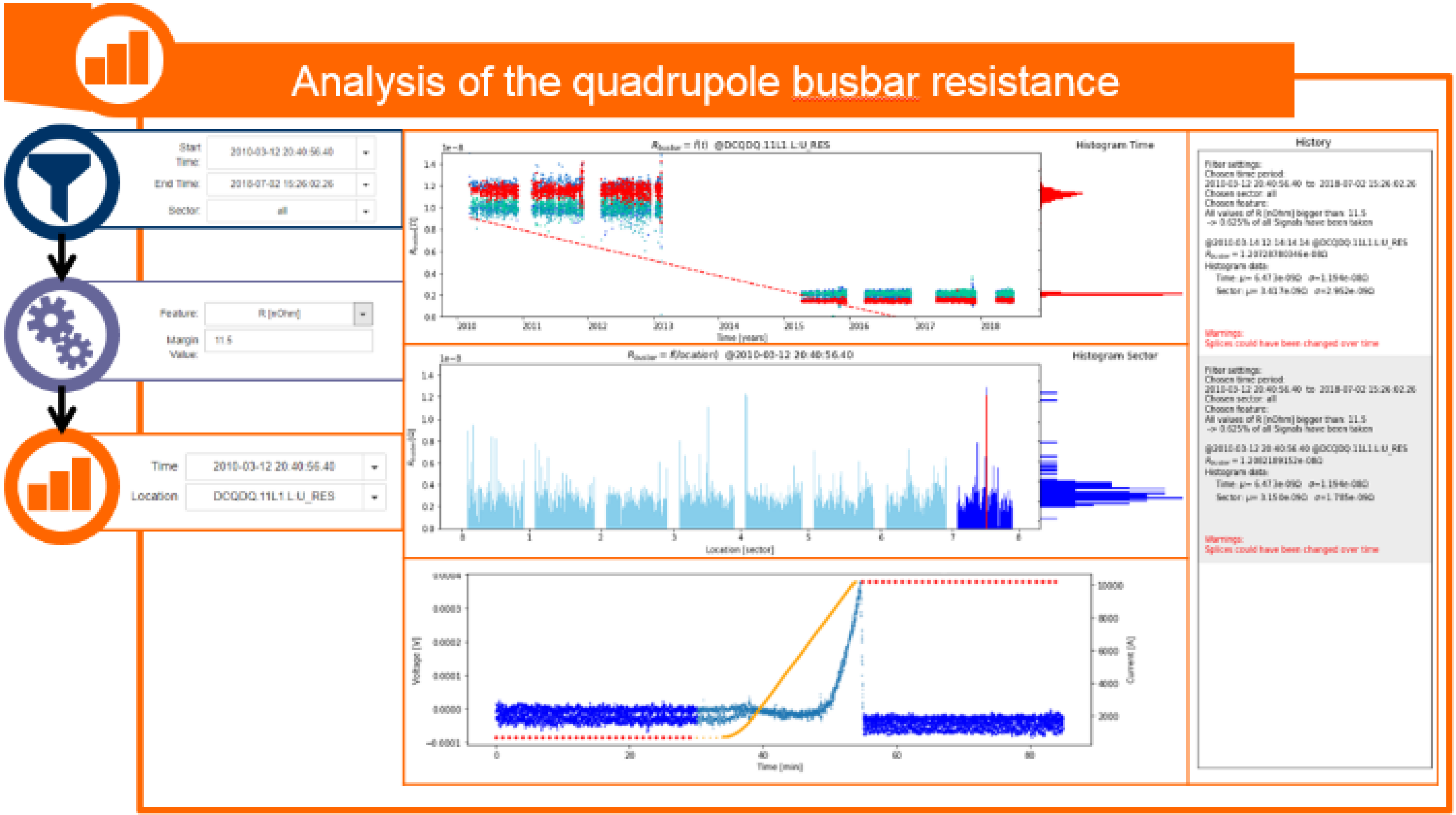
Strong cooperation with MPE-MS (Thibaud Buffet)

LHC Signal Monitoring Project – Python CI

```
class TestDFSignal(unittest.TestCase):

    def setUp(self):
        pass

    def tearDown(self):
        pass

    def test_validateKeyInKwargs_KeyNotInKwargs_returnsNone(self):
        """Test if the key passed isn't in kwargs then None is returned"""
        # arrange
        kwargs = {"string": "string", "int": 1, "float": 1.5, "date": datetime.datetime.now()}
        # act
        result = DFSignal.validateKeyInkwargs("key", str, **kwargs)
        # assert
        self.assertIsNone(result)
```

Arrange → Act → Assert

LHC Signal Monitoring Project – SWAN Dashboard



Simplicity is the art of maximizing the work not done

- KISS – Keep It Simple, Stupid!
- Recognition and use of design patterns
- Code review with experts (TE-MPE-MS, EN-ACE-EDM)
- Search for canonical, math-based problem representation
- Internal code refactoring
- Static code analysis with **sonar qube** (code duplications, code smells, complexity)
- **Humility in programming (complicated solution is not impressive...)**
  - http://labviewjournal.com/2013/05/humility-1/
  - http://labviewjournal.com/codereviews/Code%20Review%20Presentation.pdf
  - https://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html

Our Coding decalogue

1. **Naming conventions**: think twice if the name describes its purpose… and is short

2. **Use of functions**: break down the problem into small parts and solve one after another

3. **Representation**: failure at planning is planning a failure

4. **Exception handling**: a problem one expects and handles is not a problem anymore

5. **Architecture**: think about layers. For example Acquire → Analyse → Present

Our Coding decalogue

6. **Code testing**: no more fear while modifying thecode

7. **Parametrization**: stop commenting bits of code to execute different parts

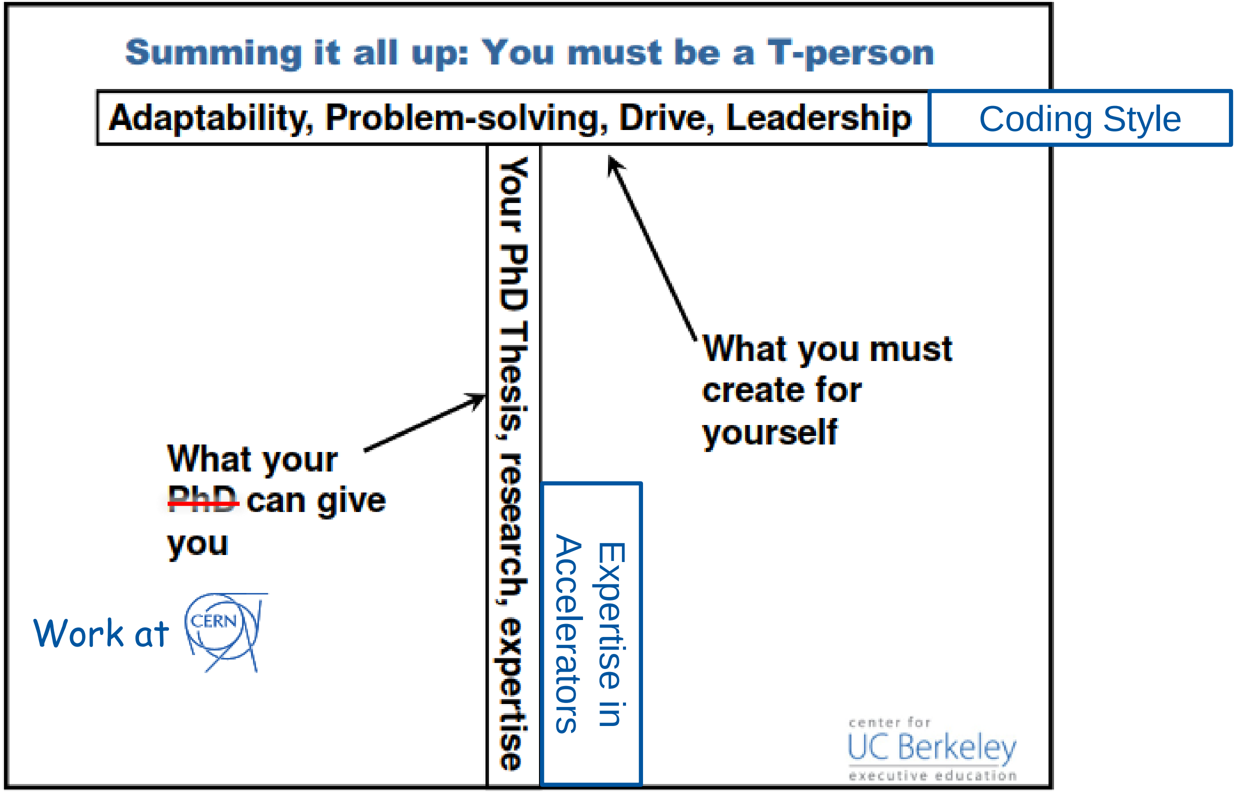
8. **Code versioning**: you no longer need *script_v1.m*, *script_v2.m*, *script_v3a.m*

9. **Pair programming**: two heads are better than one

10. **Code reviewing**: everyone knows what's happening around – expertise continuity

Point 10. extremely important while finishing contracts!!!

Be a T-shaped person



Thank you!