# Methodology for development of scientific software and test frameworks in function of precision of the expected results

TOMASZ PRZEDZIŃSKI

# Foreword

I have worked for around 10 years on number of scientific projects, including projects that have history of more than 30 years now.

As computer scientist I tried to identify the key factors of the development process, identify what factors into the success of these projects and try to understand how this knowledge can be used to improve the development of future projects.

This seminar is summary of my experience, which is described in more details in my PhD thesis.

# Agenda

- Few words about me
- Physics background
- State of the art
- Estimating the effort
- Managing scientific software development process
- Scientific software development cycle
- Quantitative evaluation of software complexity and precision
- Taxonomy of software tests and scientific software testing
- Summary

# Goals

Our goals are to:

- Present the physics background required to understand the common concepts of software development for physics experiments
- Analyse the scientific software development process
- Describe the methodology that emerged during the development of several tools
- Present how this methodology relates to development of several tools
- Show quantitative evaluation of scientific software complexity
- Address scientific software testing process
- Summarize our experience related to scientific software development process

# Few words about me

# Few words about me

**1986** – born in Skarżysko – Kamienna

**2005** – started MsC studies in Jagiellonian University in Krakow, Faculty of Physics Astronomy and Applied Computing Science

**2008** – from that year I work continuously for software aspects of some HEP projects

**2010** – defended MsC thesis: „Software for physics of tau lepton decay in LHC experiments” (available online, arXiv:1009.3877)

**2010** – started PhD studies in applied computing science

**2015** – started work in R&D company producing medical equipment

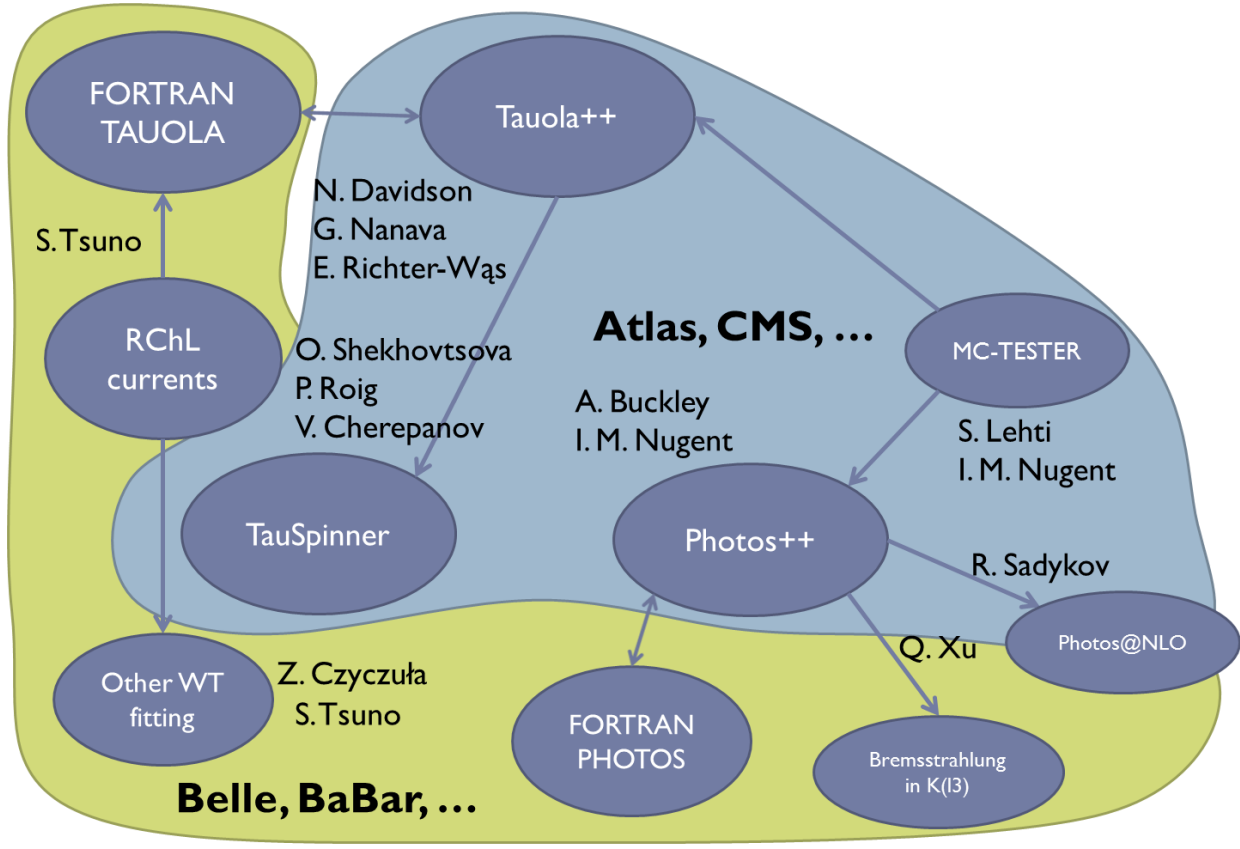
**2017–now** – worked for Parasoft (co-founded by Adam Kolawa, who got his master degree both on AGH and UJ in Krakow), later for several other companies, mostly in embedded R&D teams

# Main projects that I participated in

Since 2008 I have participated in a number of projects related to theoretical or experimental physics

Most of them in collaboration with:

- Institute of Nuclear Physics PAN, Krakow, Poland
- PH-TH group, CERN, Switzerland
- ATLAS experiment, CERN, Switzerland
- Belle experiment, KEK, Japan
- BaBar experiment, SLAC, USA
- RWTH Aachen University, Germany
- SFT group, CERN, Switzerland




Green colour corresponds to Low Energy projects, blue to High Energy projects

# Main publications

- Universal Interface of TAUOLA Technical and Physics Documentation
  By N. Davidson, G. Nanava, T. Przedzinski, E. Richter-Was, Z. Was.
  10.1016/j.cpc.2011.12.009. Comput.Phys.Commun. 183 (2012) 821-843. **Cited by 222 records**

- PHOTOS interface in C++: Technical and Physics Documentation
  By N. Davidson, T. Przedzinski, Z. Was.
  10.1016/j.cpc.2015.09.013. Comput.Phys.Commun. 199 (2016) 86-101. **Cited by 196 records**

- TauSpinner Program for Studies on Spin Effect in tau Production at the LHC
  By Z. Czyczula, T. Przedzinski, Z. Was.
  10.1140/epjc/s10052-012-1988-z. Eur.Phys.J. C72 (2012) 1988. **Cited by 73 records**
  - Notably, used by ATLAS Collaboration in Measurements of $\tau$ polarization in $W \to \tau\nu$ decays, doi:10.1140/epjc/s10052-012-2062-6.

- **Latest, published January 2021:**
  The HepMC3 Event Record Library for Monte Carlo Event Generators
  By A. Buckley, P. Ilten, D. Konstantinov, L. Lönnblad, J. Monk, W. Porkorski, T. Przedzinski, A. Verbytskyi.
  10.1016/j.cpc.2020.107310 Comput.Phys.Commun. 260 (2021) 107310

# Main publications

- Resonance chiral Lagrangian currents and tau decay Monte Carlo
  By O. Shekhovtsova, T. Przedzinski, P. Roig, Z. Was.
  10.1103/PhysRevD.86.113008. Phys.Rev. D86 (2012) 113008. **Cited by 73 records**

- Resonance chiral Lagrangian currents and experimental data for $\tau \to \pi^-\pi^-\pi^+\nu_\tau$
  By I.M. Nugent, T. Przedzinski, P. Roig, O. Shekhovtsova, Z. Was.
  10.1103/PhysRevD.88.093012. Phys.Rev. D88 (2013) 093012. **Cited by 66 records**

- Ascertaining the spin for new resonances decaying into $\tau^+\tau^-$ at Hadron Colliders
  By S. Banerjee, J. Kalinowski, W. Kotlarski, T. Przedzinski, Z. Was.
  10.1140/epjc/s10052-013-2313-1. Eur.Phys.J. C73 (2013) no.2, 2313. **Cited by 36 records**

- MC-TESTER v. 1.23: A Universal tool for comparisons of Monte Carlo predictions for particle decays in high energy physics
  By N. Davidson, P. Golonka, T. Przedzinski, Z. Was.
  10.1016/j.cpc.2010.11.023. Comput.Phys.Commun. 182 (2011) 779-789. **Cited by 26 records**

# Main publications

Part of the results presented here has also been published in
**Computing in Science and Engineering:**

- Software Development Strategies for High-Energy Physics Simulations Based on Quantum Field Theory. By T. Przedzinski, M. Malawski, Z. Wąs. 10.1109/MCSE.2019.2947017., Comput.Sci.Eng. 22 (2019) no.4, 86-98.

# Physics background

- SIMULATIONS IN HIGH ENERGY PHYSICS
- SOFTWARE IN PHYSICS EXPERIMENTS

# Simulation

High Energy Physics simulation – a Monte Carlo (MC) simulation of the effects of a collision of two particles. Such collision is called **an Event** and its data are stored in **Event Record**

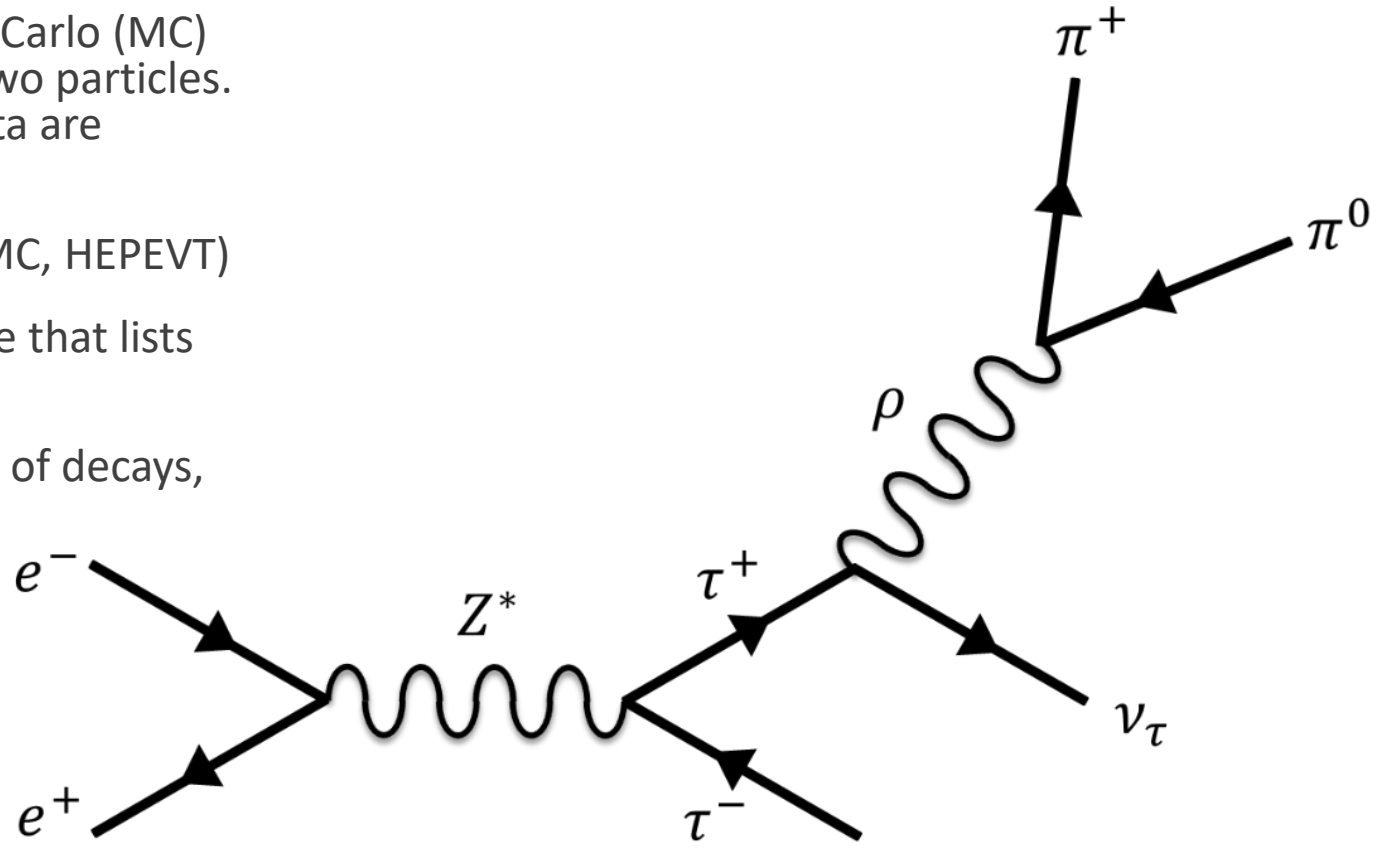
Few Event Record formats exist (e.g. HepMC, HEPEVT)

Essentially, Event Record is a data structure that lists **particles** and maps their **decay products**

Usually, this structure is a tree – a cascade of decays, but there are exceptions

Basic properties of a particle are:

- **Four-vector** and **mass** (virtuality)
- **ID** (particle identifier)
- **Status** (decaying, stable, etc.)



# Simulation

High Energy Physics simulation – a Monte Carlo (MC) simulation of the effects of a collision of two particles. Such collision is called **an Event** and its data are stored in **Event Record**

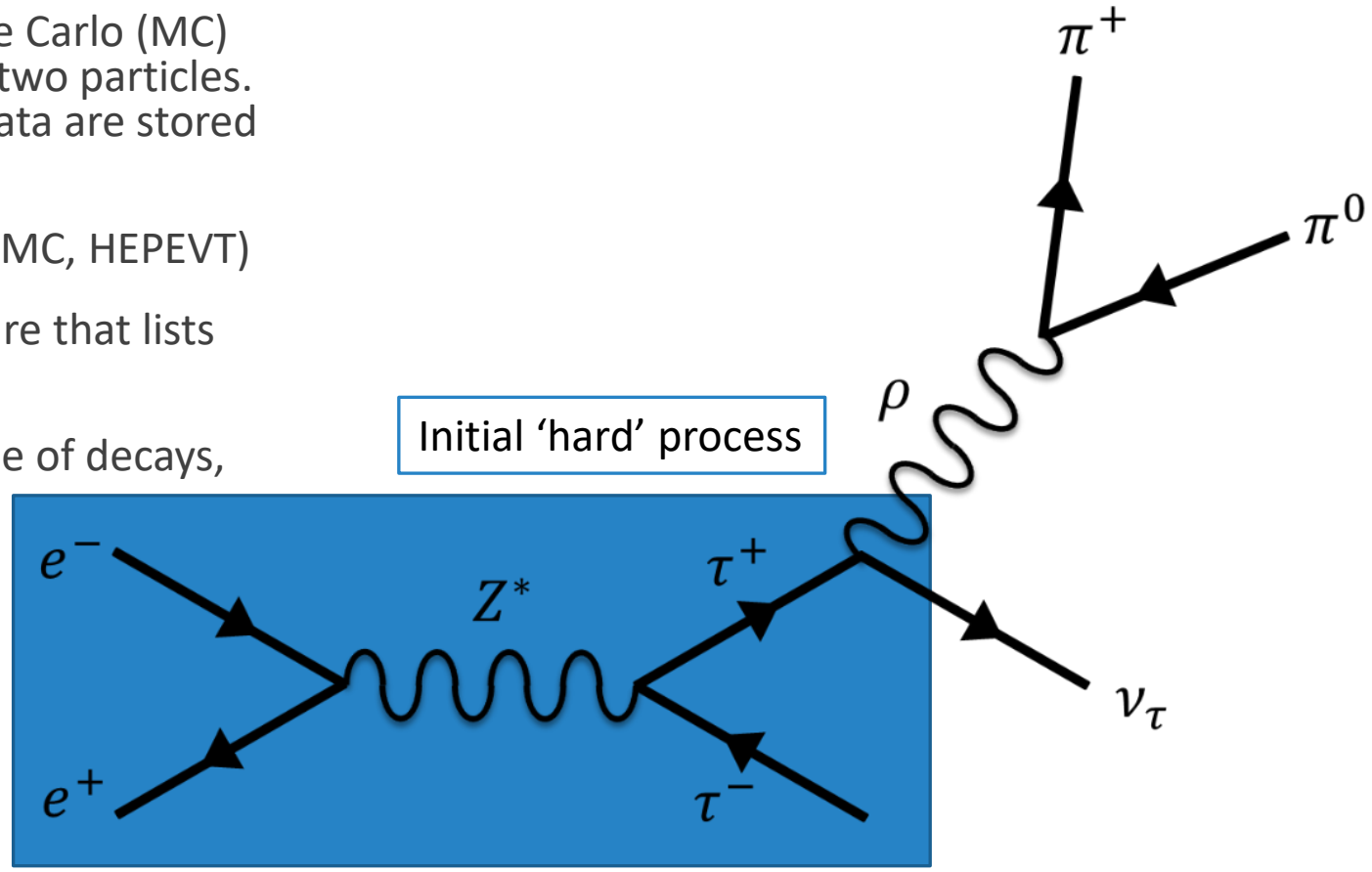
Few Event Record formats exist (e.g. HepMC, HEPEVT)

Essentially, Event Record is a data structure that lists **particles** and maps their **decay products**

Usually, this structure is a tree – a cascade of decays, but there are exceptions

Basic properties of a particle are:

- **Four-vector** and **mass** (virtuality)
- **ID** (particle identifier)
- **Status** (decaying, stable, etc.)



# Simulation

High Energy Physics simulation – a Monte Carlo (MC) simulation of the effects of a collision of two particles. Such collision is called **an Event** and its data are stored in **Event Record**

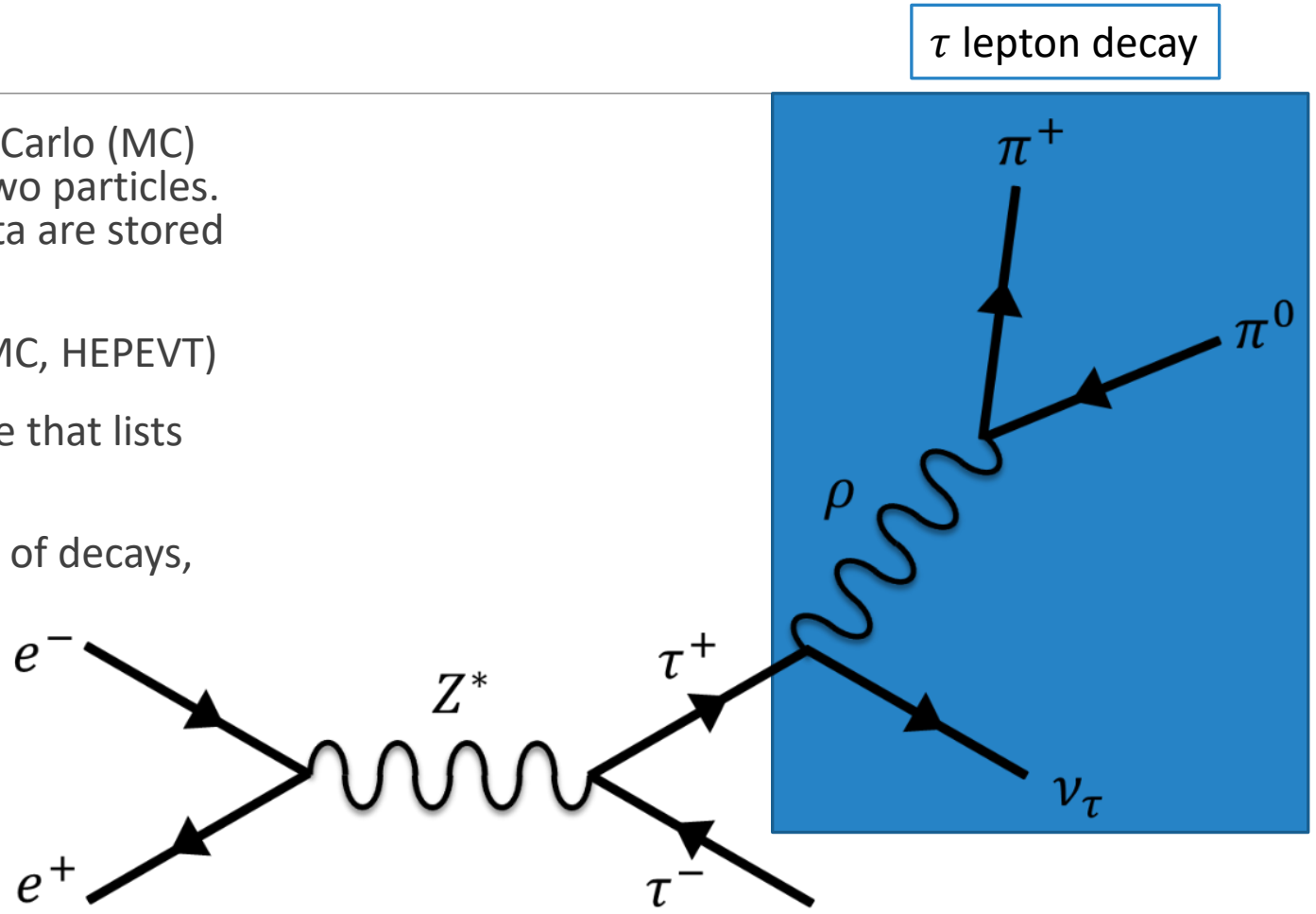
Few Event Record formats exist (e.g. HepMC, HEPEVT)

Essentially, Event Record is a data structure that lists **particles** and maps their **decay products**

Usually, this structure is a tree – a cascade of decays, but there are exceptions

Basic properties of a particle are:

- **Four-vector** and **mass** (virtuality)
- **ID** (particle identifier)
- **Status** (decaying, stable, etc.)



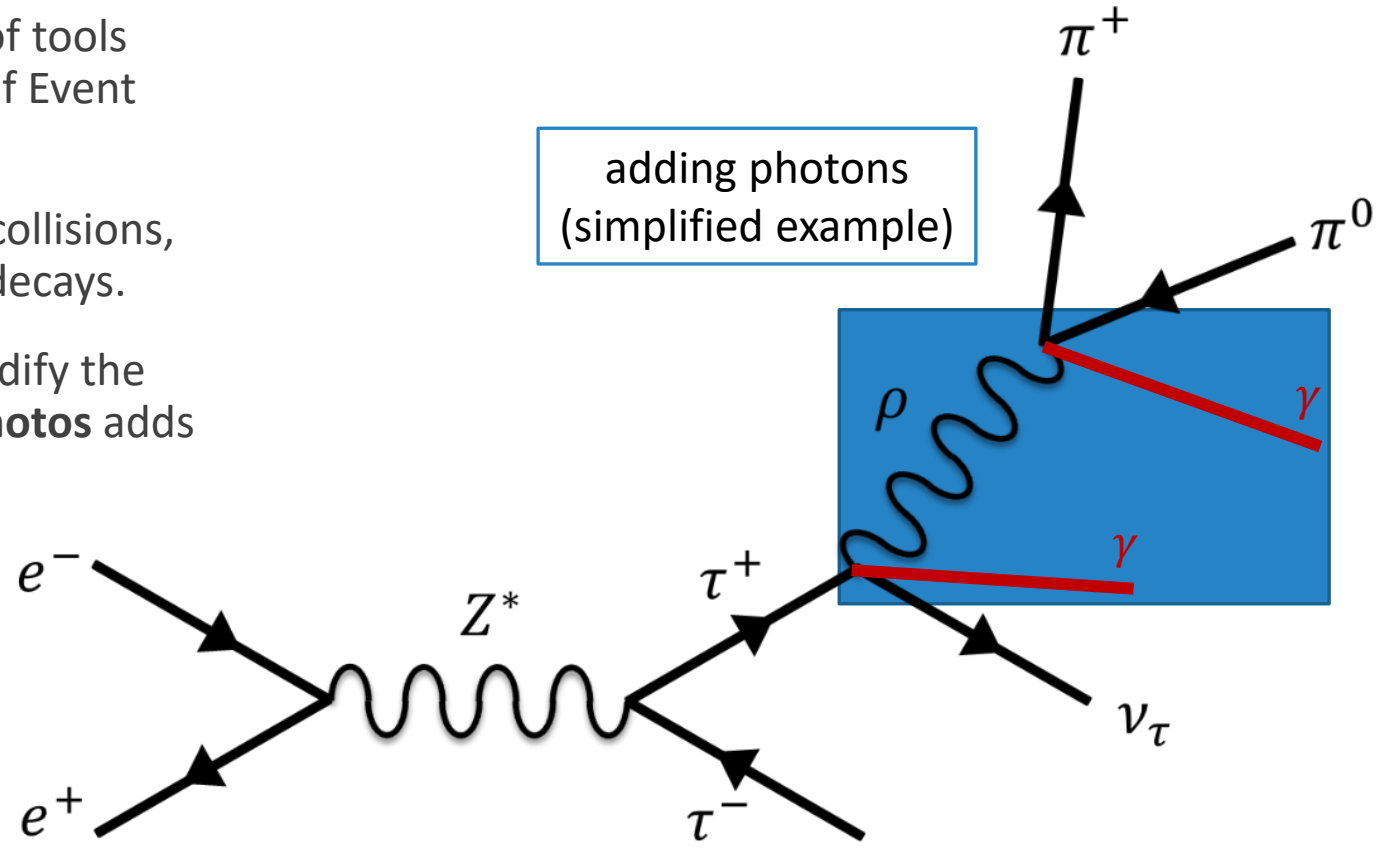
# Simulation

A simulation can be though of as a chain of tools that produce, modify or analyze content of Event Record.

**Generators** – calculate products of initial collisions, e.g. generator **Tauola** generates $\tau$ lepton decays.

**Post-processing tools** (afterburners) – modify the decay tree according to their roles, e.g. **Photos** adds additional photons to the decay tree

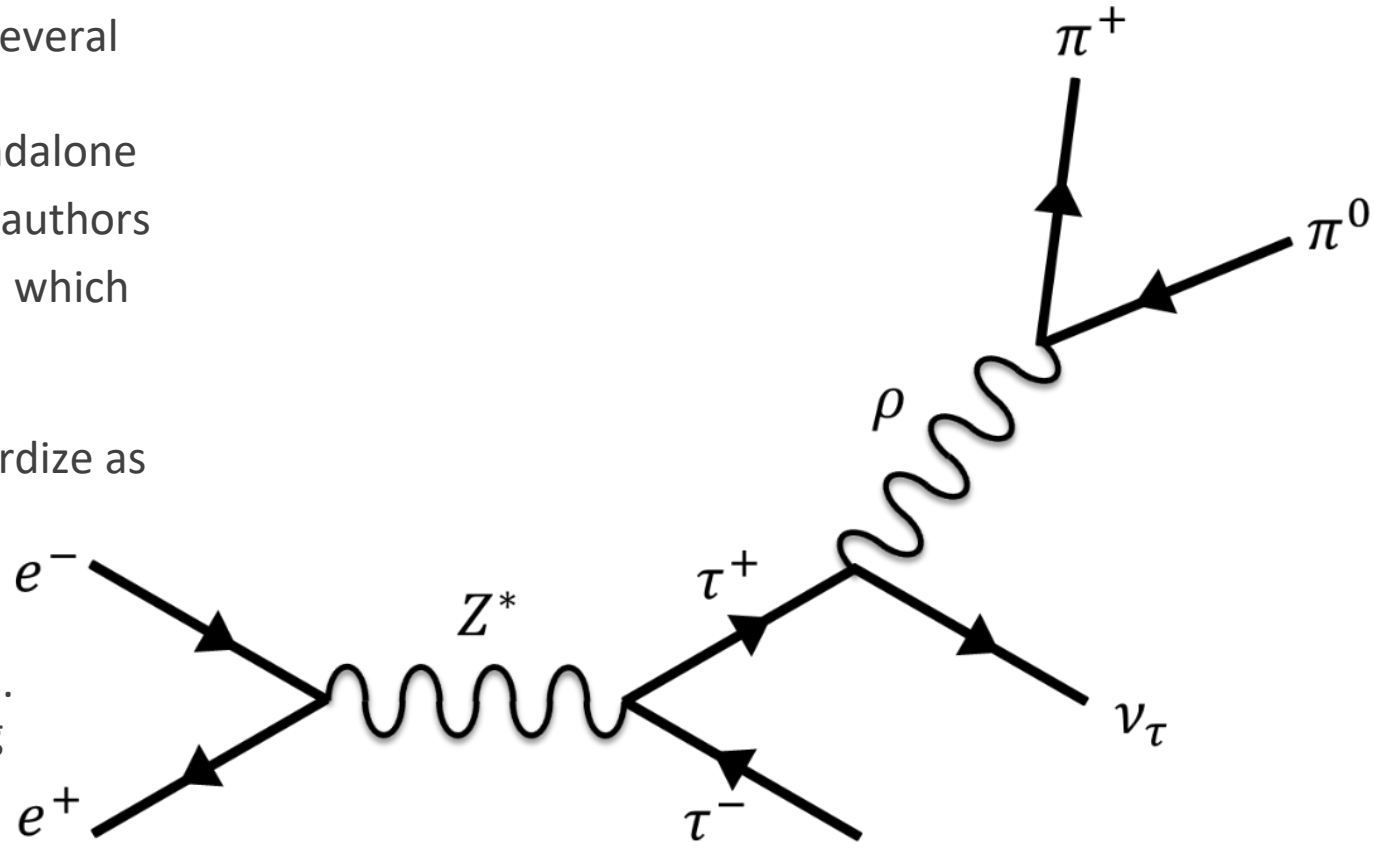
**Analysis tools** – does not modify the Event Record but extracts information encoded in it, e.g. **MC-Tester** creates a set of histograms based stable decay products of particle with given ID.



# Simulation

The most common source of problems – several tools operate on the same Event Record

- Tools are usually designed to work standalone
- In most cases, each tool have different authors
- Their authors follow ‘some’ standard in which they encode and decode Event Record information
- A huge effort has been made to standardize as much of it as possible but standards evolve and not everyone are willing to (or have need to) adapt
- Event Record itself can pose limitations. Authors overcome them by introducing a non-standard solutions



# Physics precision

Physics precision is a key concept of this seminar, but before we can address it we have to remember to take care of:

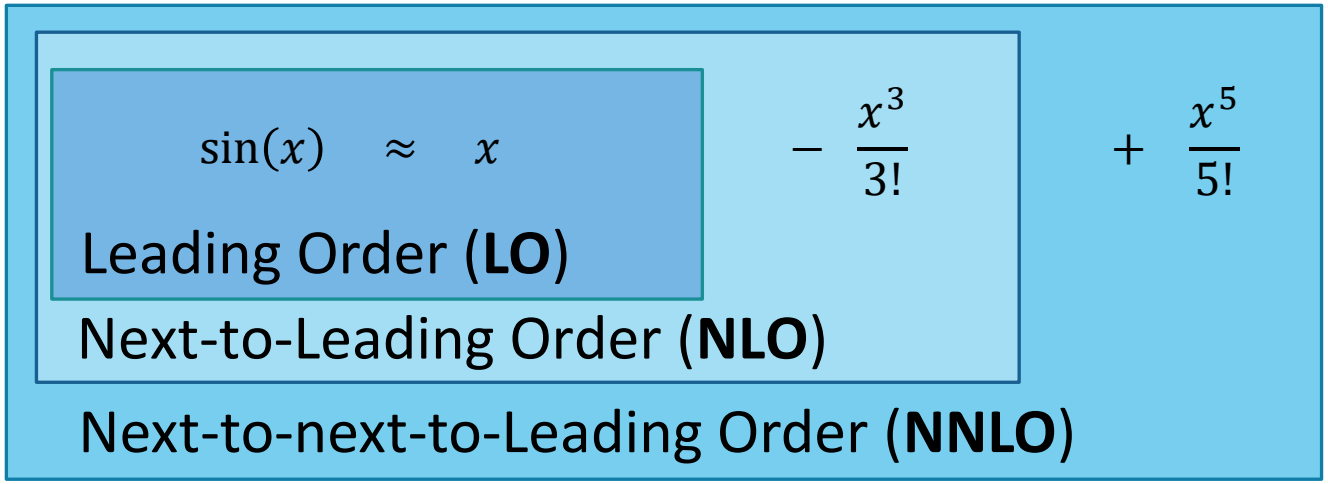
- **Statistical precision** – precision related to generated data sample size. Statistical error should be at least 3 times smaller than error that comes from the physics model
- **Technical Precision** – all practical consequences of numerical calculations (e.g. length of a computer word, operation ordering, precision of numerical integration, bugs, etc.). Technical precision should also be at least three times better than statistical precision

These aspects require a lot of attention especially if the goal of the project is to increase physics precision, since this may require improvements of technical or statistical precision.

# Physics precision

Physics precision can be thought of as comparison of different levels of approximation of a theoretical model.

An increase in precision is like taking into account one more term in the Taylor expansion.




W-boson mass measurement

- **1997 – the L3 detector at LEP:** $M_W = 80.75^{+0.26}_{-0.27} \pm 0.03 GeV$
- **2009 – measurements at CDF:** $M_W = 80.399 \pm 0.025 GeV$
- **2018 – LHC:** $\mathrm{M_W} = 80.370 \pm 0.019 GeV$

# Scale

The sense of the scale of the simulation helps to understand complexity of the tools

- A single generated proton-proton collision can produce an Event Record with more than 10 000 particles
- To observe an NLO effect large data set is needed. E.g. when testing Tauola MC Generator the most common data sample is: 10 million events
- Final state radiation process in Photos requires samples of: 100 million events
- Validating NNLO effects in Photos required samples of: 10 000 million events

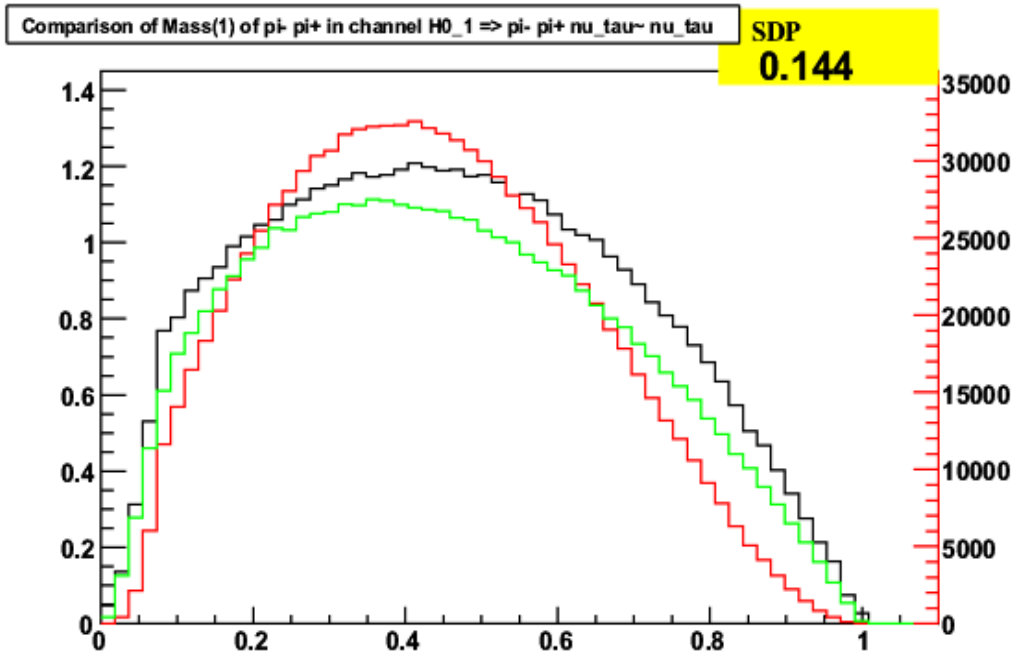
How does it looks in terms of output:

- A very small event for Tauola**:** 2 kB
  10 million events validation sample: 22 GB
- Typical event with 10 000 particles: 3 MB
  Validation sample for Photos NNLO effect: >30 PB
- This is obviously not feasible. Instead, events are generated for each test and dropped after analysis, which greatly increases the overall analysis time

# Scale

Note that most of the data is filtered out during analysis

- Given an event with 10 000 particles we are bound to find, say, around 100 interesting ones
- To observe a single property of these decays we can reduce the output data to as little as a single one-dimensional histogram
- The result of the analysis of 30 petabytes of data can be stored in a picture that takes few kilobytes




This is, of course, a simplification which is true only for the most basic observables. Usually a larger set of histograms is needed, augmented by many statistics gathered from the data sample.

# Estimating the effort

# Development effort

## Estimating the development effort

- Effort, typically expressed in man-days (MD), is the key factor used for planning and budgeting a project
- Usually estimated by experts based on historical data and experience
- Standardized estimation metrics (e.g. COCOMO) are strongly related to functional requirements

## However, in scientific software

- Each project is unique and has very little common with the others
- Software mostly implements non-functional requirements (NFR) such as improving precision of the results
- Code-based metrics are ill-fitted as they do not assess the effort required to implement algorithms

# Development effort

## Example code-based metrics

- High complexity – usually an indication of need to refactor but not in case of scientific software
- High amount of comment lines – a lot of details regarding the algorithms and possible future extensions is placed in the comments

| C++ project | Total lines | Comment lines | Code lines | Max indent | Total cyclomatic complexity |
|---|---|---|---|---|---|
| MC-TESTER | 8091 | 1716 | 6130 | 7 | 1181 |
| TauSpinner | 3064 | 781 | 2363 | 8 | 719 |
| Tauola++ | 4781 | 1151 | 3566 | 6 | 559 |
| Photos++ | 8584 | 2881 | 5697 | 8 | 973 |

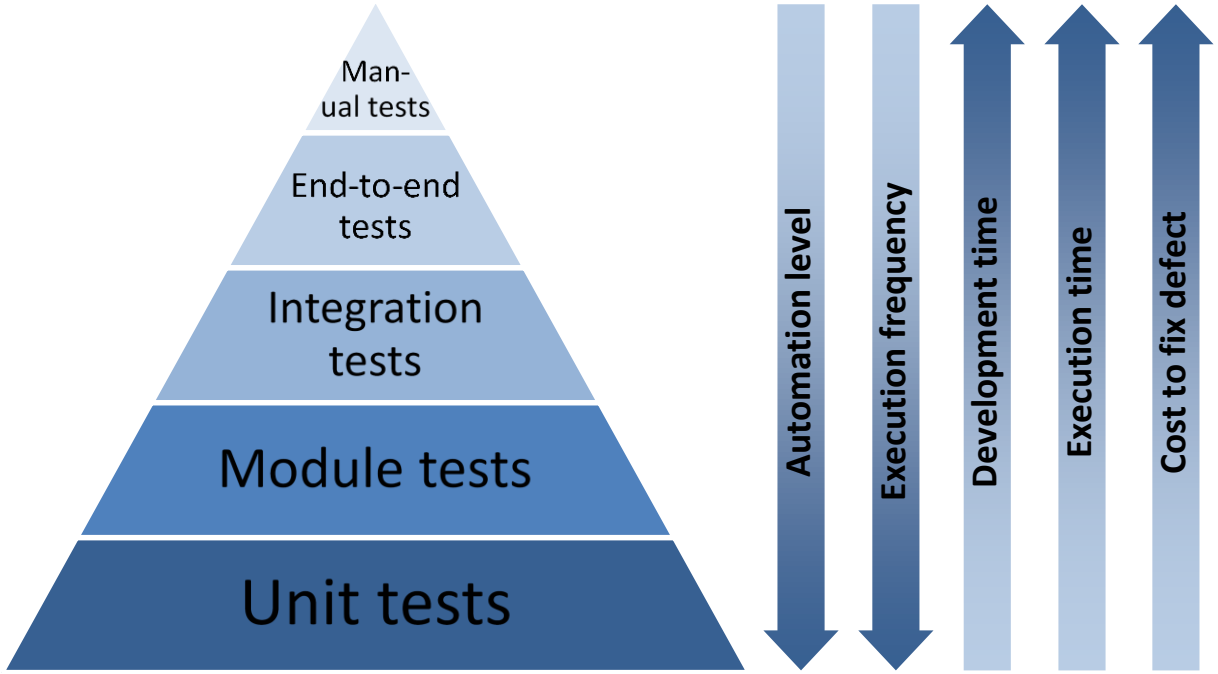
| Fortran project | Total lines | Comment lines | Code lines | Max call depth | Total cyclomatic complexity |
|---|---|---|---|---|---|
| Photos Fortran | 12612 | 3959 | 8237 | 7 | 566 |
| Tauola Fortran | 28088 | 2967 | 22456 | 22 | 2086 |

Code-based metrics for several C++ and Fortran projects described later. Metrics has been computed using Metrix++ (C++) and FPT (Fortran) based on source code without tests

# Testing effort

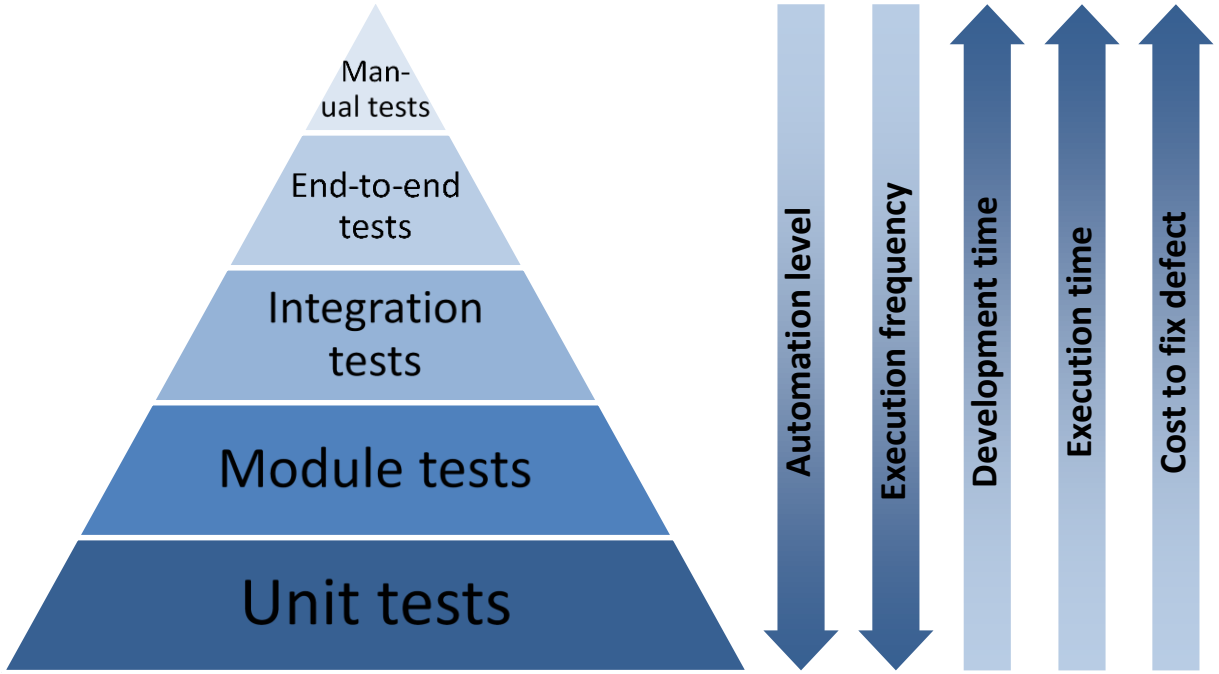
## Estimating the testing effort

- Typical cost of quality (COQ) estimations usually only consider functional testing
- Precision is a NFR and should follow NFR-related testing strategies
- Estimating NFR testing effort is as hard as estimating NFR implementation effort
- NFR testing is often neglected or delayed, as often are NFRs themselves
- NFR and NFR tests are usually first candidates for cost optimization



# Quality in scientific software

- Scientific software cannot neglect NFR testing
- Verification and validation are key elements of the development strategy
- Defects greatly harm team and software credibility
  - E.g. version 1.1.1 of Tauola++ had a defect in TauSpinner
  - It took 3 days for it to be spotted by community and fixed by us, yet the erroneous version remained in LCG library




Cost of fixing defects after deployment can be enormous. Risk of skimming on QA is too high

# State of the art

# Scientific software development

It is known that most of scientists are not familiar with the software development methods [1] but the awareness of tools and good practices is growing. This is often because majority of the effort is put into solving non-programming challenges.

In [2] the authors emphasize the need for testing not only the correctness of software in terms of agreement with theory, but also to test the implementation, when the project gains a wider user base.

There were attempts to formalize scientific development model based on knowledge acquisition [3]. They highlight differences in approach to the software quality from the point of view of a software engineer and scientist. Two key findings emerged from this study:

1. Scientists must be considered as part of the software system
2. Their testing is not product-based but knowledge-based

(references after last slide)

# Precision of scientific software

One of the early studies from 1994 [4] showed disturbing non-random numerical disagreement between the results of different packages. Estimated growth of this disagreement was around 1% per 4000 lines of Fortran code.

This data was used as feedback for the developers. It was shown that the results of this feedback can improve the quality of software.

There are also examples of solutions to problems related to detection of software numerical instability. One of them, presented in [5], combines stochastic and infinite-precision testing. Similar techniques were used to test software presented today.

(references after last slide)

# Scientific software testing

Number of issues unique to scientific software has been listed in the literature overview on testing scientific software [6]. These descriptions are quite accurate when related to software for physics communities.

The authors of this overview state that none of the primary studies reported the complexity of the software in terms of measurable unit such as coupling, cohesion, or cyclomatic complexity.

This is connected to the fact that the software engineering techniques are often not efficient in scientific environments. Code-based metrics have no meaning in scientific projects.

(references after last slide)

# Managing scientific software development process

- CHALLENGES
- METHODOLOGY

# Scientific software development process

Scientific project consists of:

- A set of libraries
- Source code
- Examples of the product usage
- Validation framework
- Scientific paper
- Technical documentation

The development team consists of:

- Theoretical physicists
- Experimental physicists
- Computing scientists

Additionally, the development process is aided by:

- Potential users
- Experts from different fields of physics

# Challenges – multi-layered structure

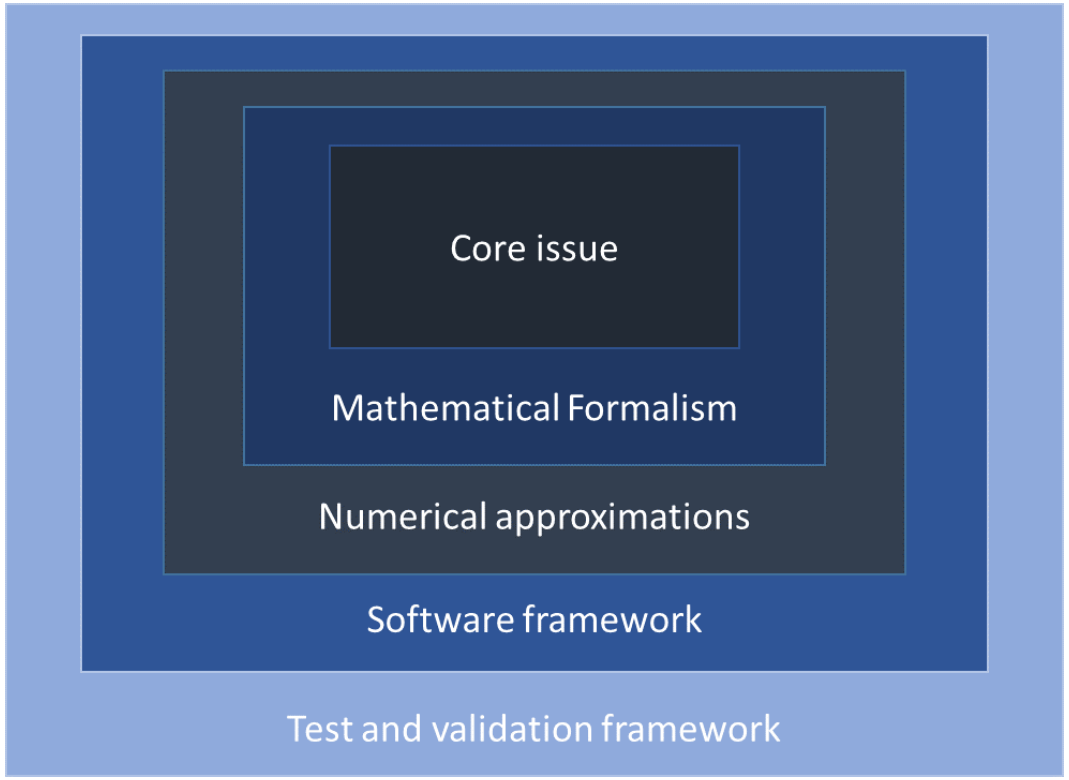
**Core** – e.g. modelled process

**Mathematical formalism** – how we can present it

**Numerical approximations** – how we can actually compute it

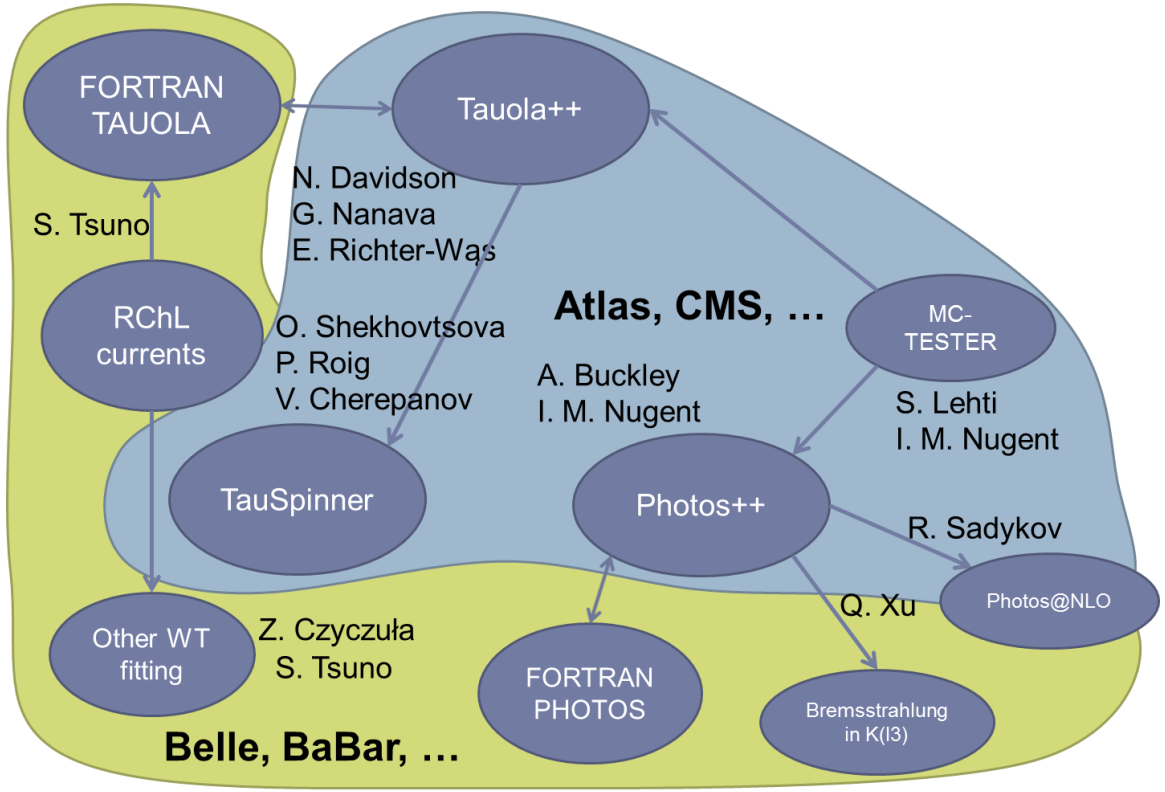
**Software framework** – how we organize the code, how users can use our code

**Test and validation framework** – how we can test that our implementation is valid and how we can prove or disprove our theory



# Challenges – human resource management

- Most complex projects require cooperation of huge amount of people ATLAS collaboration has more than a thousand researchers
- Even in smaller projects, new members come and go
- Specialists are forced to divide their time between research and training of new members
- Some specialists are available only for limited time. This time must be well spent
- Managing the knowledge spread around the team becomes an important task



# Challenges – non-standard software structure

- In many cases software engineer has to compromise software architecture to allow specialists to easily contribute to the project
- Project may require incorporation of other modules, usually done through a layer of wrappers
- Project must be adapted to different Event Records knowing, that some Event Records carry information that others don't.

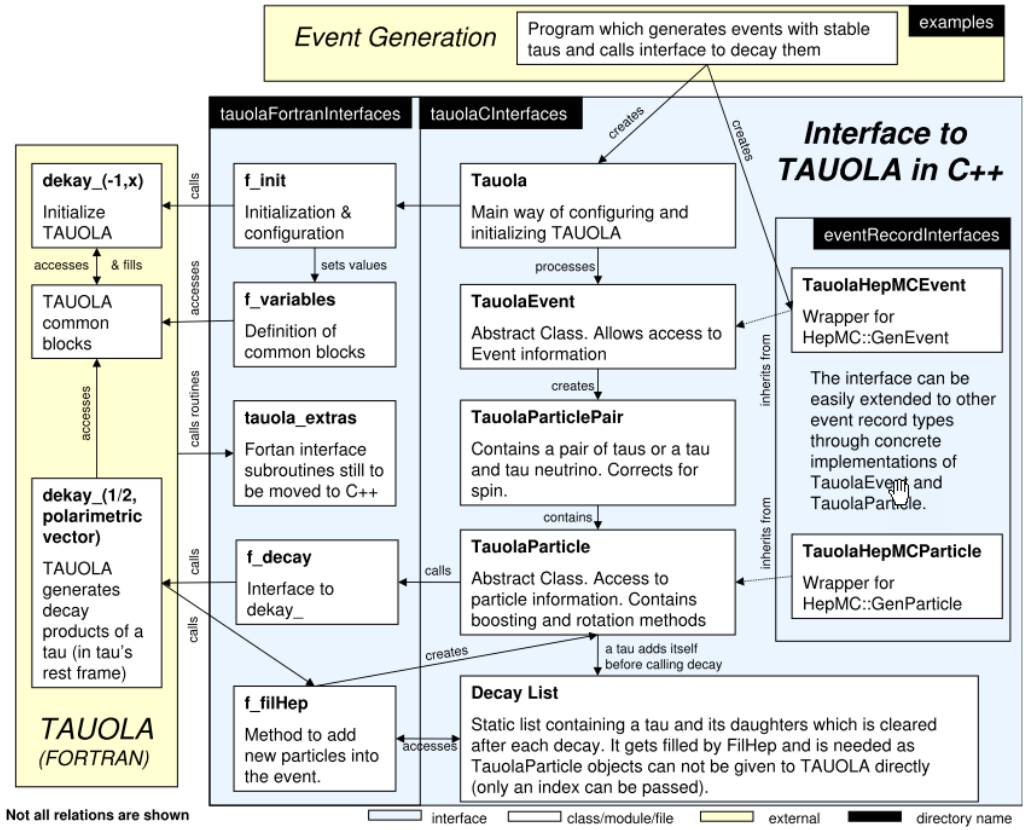
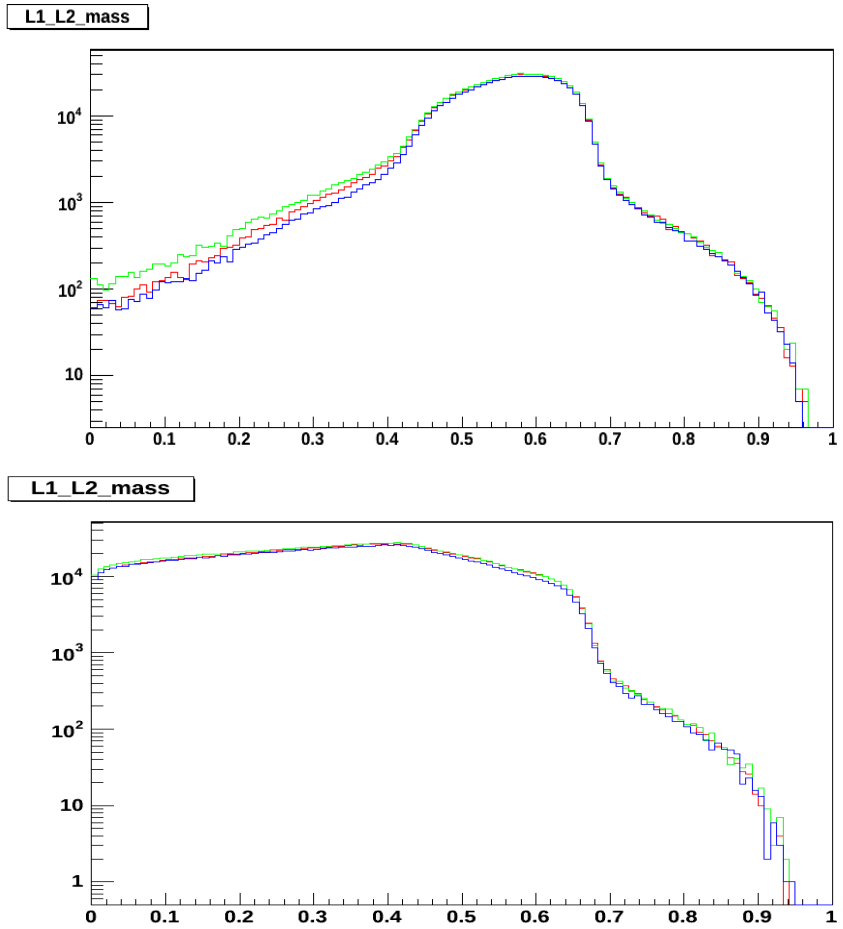



Figure 1: `TAUOLA C++ Interface` class relation diagram

# Challenges – complicated testing process

- It’s hard for computer scientists to understand the results. They should be validated by a specialist
- Some phenomena apply to very specific use-case scenarios and are near-invisible in other
- But even worse than that are technical issues. Numerical errors can manifest as similar to statistical fluctuation.
- Sometimes two overlapping issues cover each other making it hard to observe them during development
- Other technical issues (e.g. use of new compilers, rounding errors due to wrong casts, etc.) require very good testing approach and good validation framework that can find issues by looking past statistical fluctuations



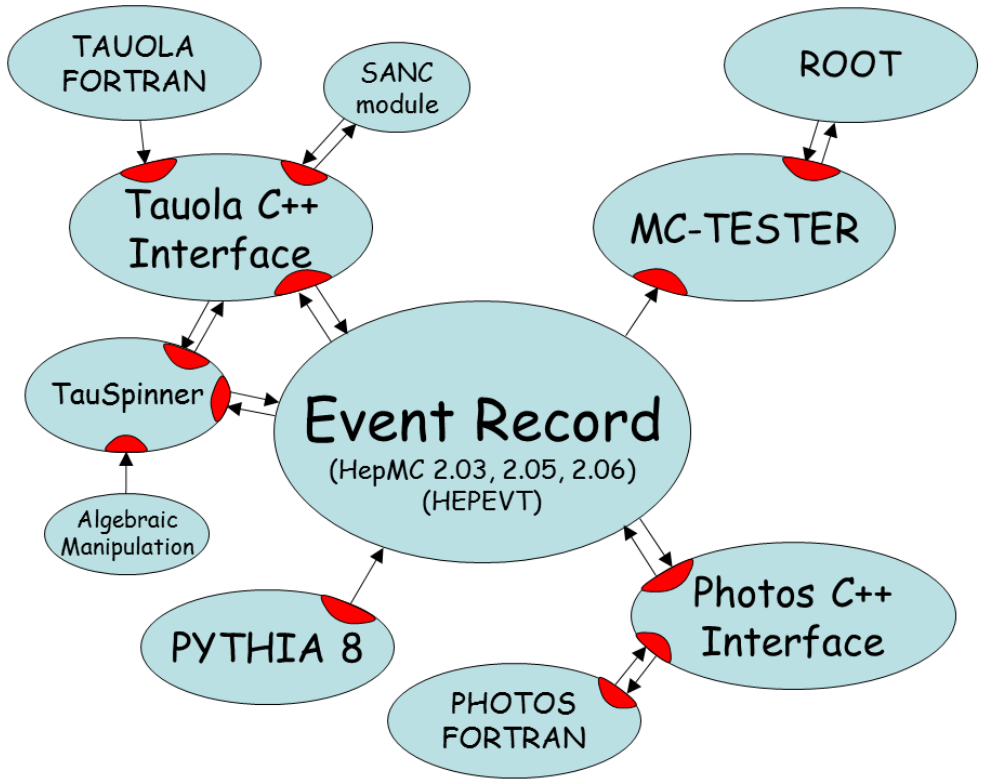
# Challenges – scarce user feedback

- Most of the users spare no time to read the documentation of an external tool that they are using
- User can ignore warnings, non-critical errors and other information as long as the result *seems* correct
- User starts reading documentation only when the results seems not to correspond to what they expect

- In case of scientific software, even with wrong setup the result may seem correct most of the time. Developers introduce warnings and errors to signal potential issues. These are, of course, often ignored
- Users can willingly risk use of the tool in an improper way as long as there is a chance it will handle these use-cases properly

# Challenges – high impact of project dependencies

- The tool must adapt to as many use case scenarios as possible
- The result of other tools can impact the result of our
- The medium used for communication (Event Record) can also impact the results or limit functionality
- Significant changes in other projects impact our project as well




Relations between different projects in example analysis chain.
Red segments symbolize interfaces used for the communication

# Challenges - I/O interface design issues

- Chain of dependencies creates multitude of possible variants of inputs

| | Generation step | Tool used for this step | Output |
|---|---|---|---|
| 1 | Hard process | General-purpose MC Generator (GMC) or specific hard-process generator or input file | 5-10 particles describing the hard process |
| 2 | Parton showers | GMC | From few to 1000 particles added to the event |
| 3 | Initial state radiation | GMC or separate tool (afterburner) | Photons and lepton pairs added to the event |
| 4 | Heavy particles decay | GMC with separate tools for specific processes | From few to 10000 particles added to the event |
| 5 | Final state radiation | GMC or afterburner | Photons and lepton pairs added to the event |
| 6 | Calculating the influence of the interference in radiation | Afterburner | Modification of existing particles, additional photon emissions |
| 7 | Applying electroweak corrections | Afterburner | Modification of existing particles in the event |
| 8 | Saving the event | Analysis software | Event data saved to file |

# Challenges - I/O interface design issues

- Chain of dependencies creates multitude of possible variants of inputs
- Some data may not be available so partial results based on partial data should be provided when possible
- User should be informed if input for the tool was imprecise (e.g. four-momentum non-conservation that has to be corrected)
- User should be informed that result of the tool may be imprecise due to input data quality

| Information needed for single $\tau$ decay | Mandatory or optional data | Result |
|---|---|---|
| Charge and four-momentum of the decaying $\tau$ | Mandatory | Decay products |
| Information about process that generated $\tau$ (or $\tau$ pair) | Optional | Corrections based on the initial process |
| Information about $\tau$ neutrino $\nu_\tau$ (or second $\tau$ In case of $\tau$ pair decay) | Optional | Corrections due to correlation between $\tau$ and $\tau$ neutrino ($\nu_\tau$) (or two $\tau$'s in case of $\tau$ pair decay) |
| Polarization vector of the intermediate boson | Optional | Longitudinal polarization |
| Information about "grandmothers" (incoming beams, particles from which the simulation has started) | Optional | Electroweak corrections |

# Methodology

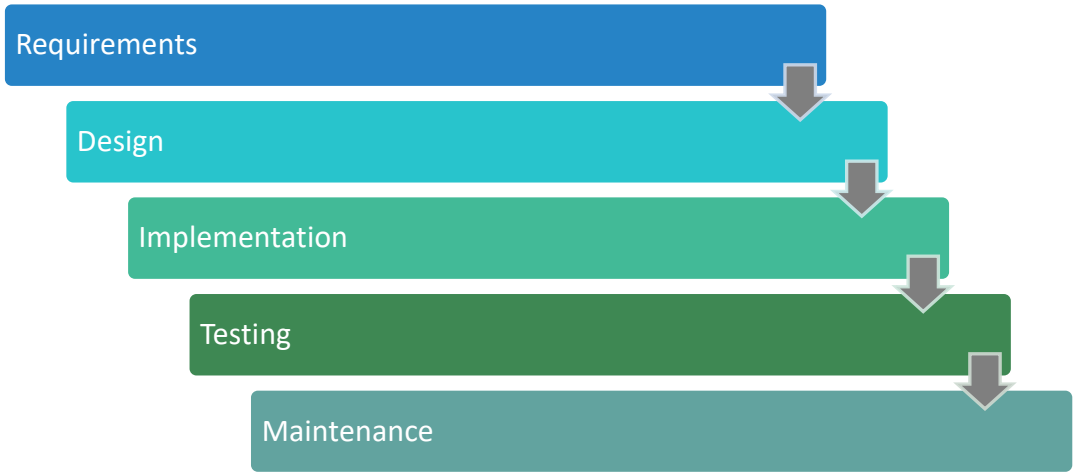
How common methodologies relate to these challenges

- The waterfall approach
- The agile methodologies
- Rational Unified Process (RUP)
- PRINCE2
- The automotive Spice (ASPICE)

# Methodology

How common methodologies relate to these challenges

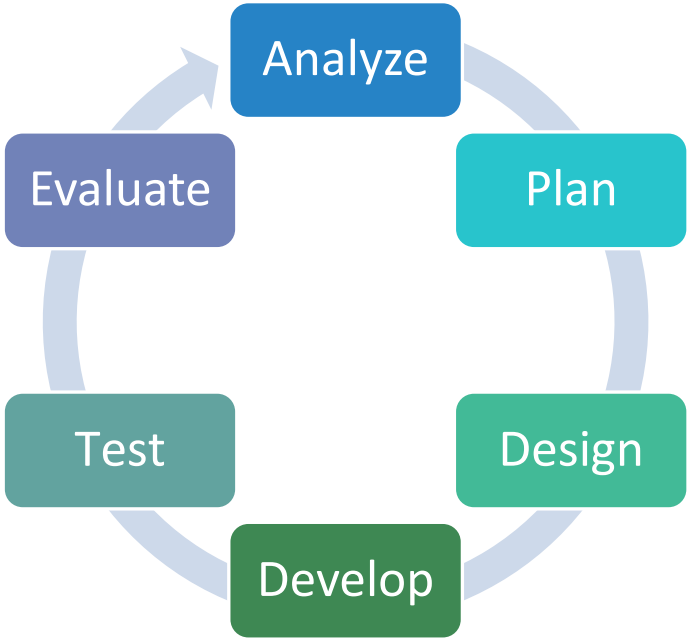
- **The waterfall approach**
- The agile methodologies
- Rational Unified Process (RUP)
- PRINCE2
- The automotive Spice (ASPICE)



# Methodology

How common methodologies relate to these challenges

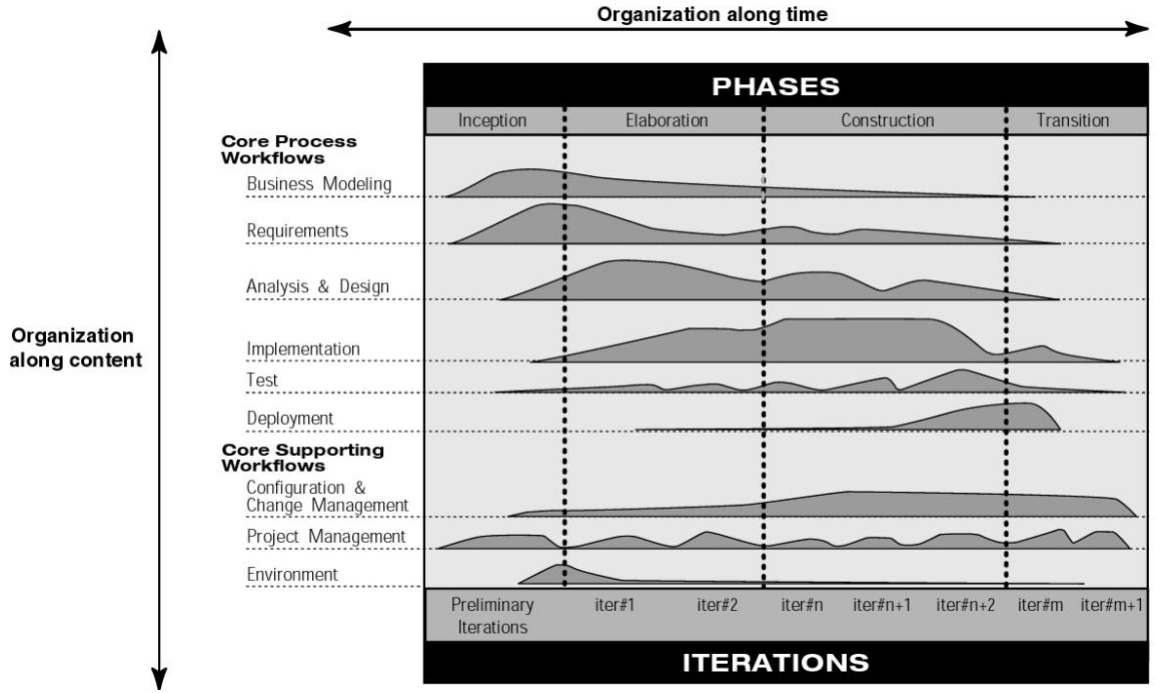
- The waterfall approach
- **The agile methodologies**
- Rational Unified Process (RUP)
- PRINCE2
- The automotive Spice (ASPICE)



# Methodology

How common methodologies relate to these challenges

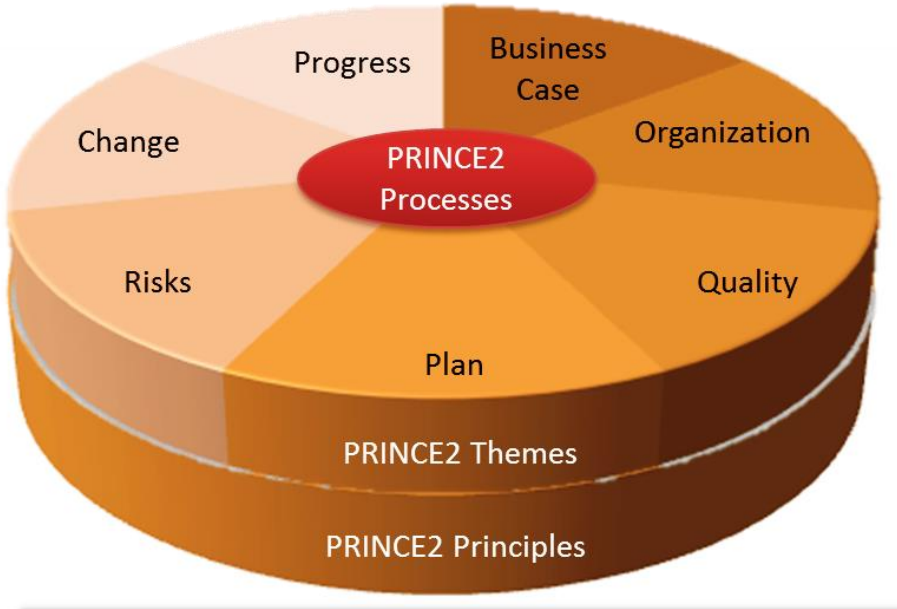
- The waterfall approach
- The agile methodologies
- **Rational Unified Process (RUP)**
- PRINCE2
- The automotive Spice (ASPICE)



# Methodology

How common methodologies relate to these challenges

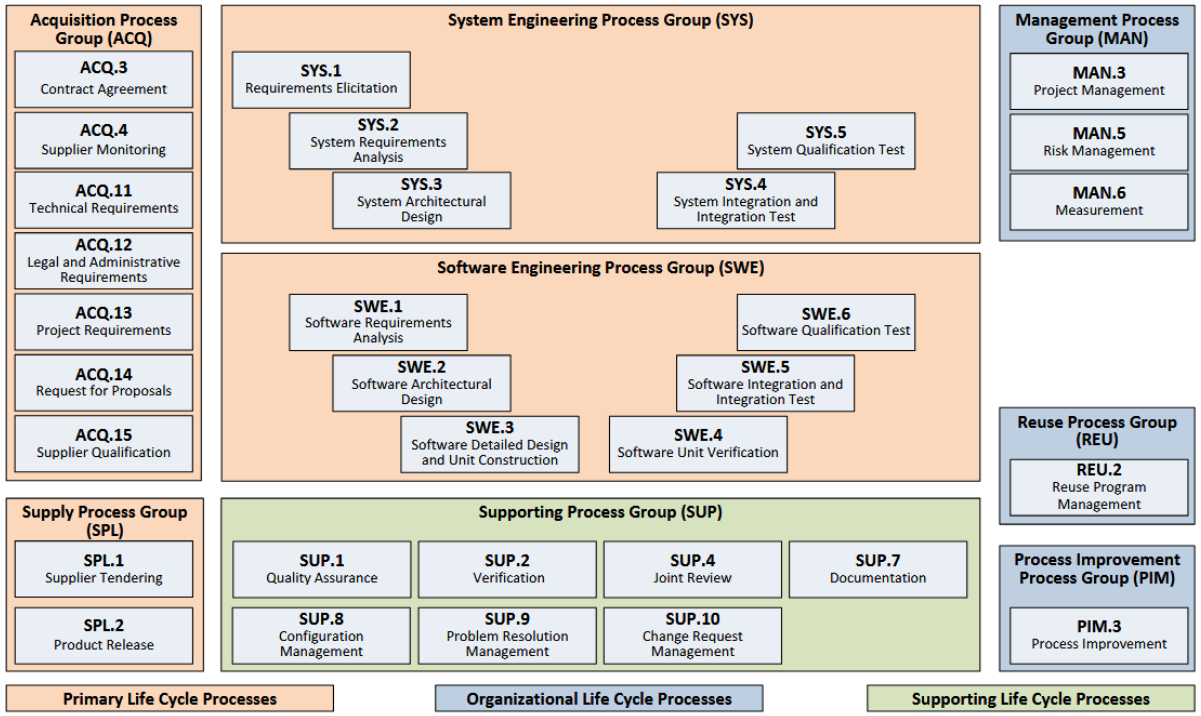
- The waterfall approach
- The agile methodologies
- Rational Unified Process (RUP)
- **PRINCE2**
- The automotive Spice (ASPICE)



# Methodology

How common methodologies relate to these challenges

- The waterfall approach
- The agile methodologies
- Rational Unified Process (RUP)
- PRINCE2
- **The automotive Spice (ASPICE)**



# Development cycle

- ELEMENTS OF THE SCIENTIFIC SOFTWARE DEVELOPMENT CYCLE
- EXAMPLES BASED ON HISTORY OF FEW TOOLS

# Thesis

Often scientific software development focuses on increasing the precision of the results. This process follows the cycle presented on the next slides.

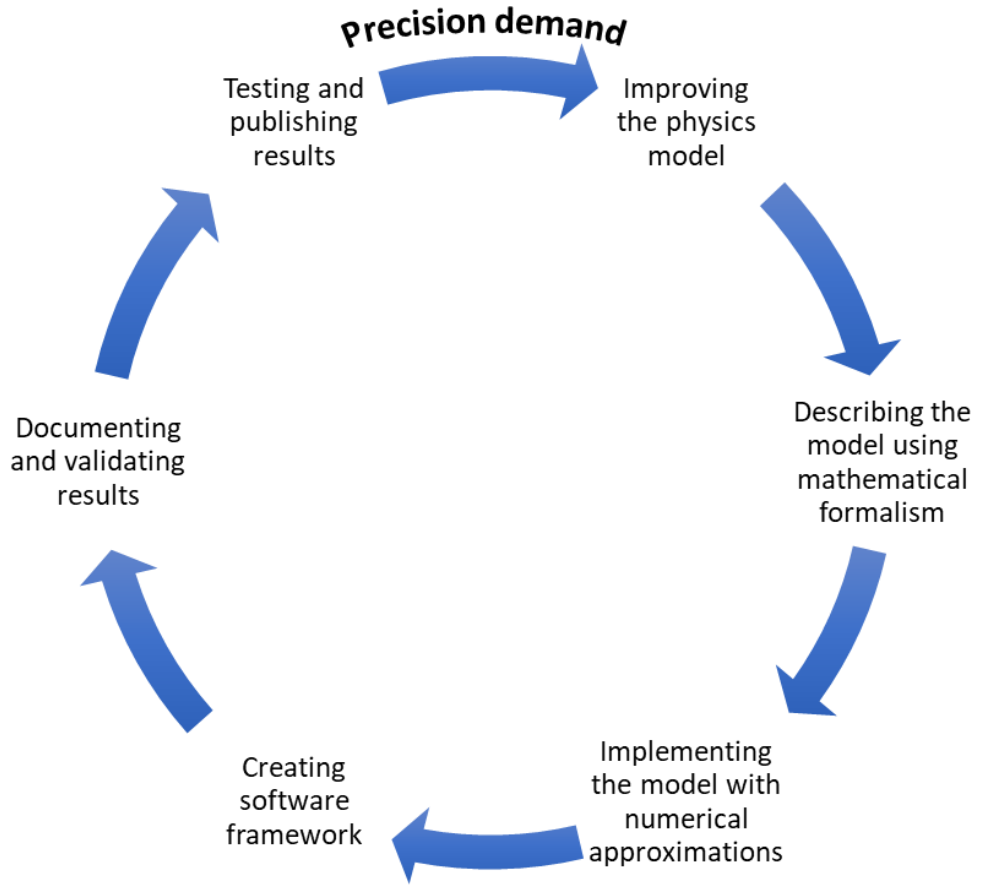
The effort put into testing and into building proper validation framework rises proportionally with the effort required to achieve results with increased precision.

# Development cycle

Following process emerged during the development of the tools described later. It evolved over time, took many forms and iterations.

It presents the scientific software development as a cycle leading to the improved scientific model which ultimately results in higher precision of the results.

Creating first version of the tool follows the same cycle. The difference is the motivation – the need to solve a problem instead of the precision demand.



# Development cycle

**1. Creating (or improving) the physics model**

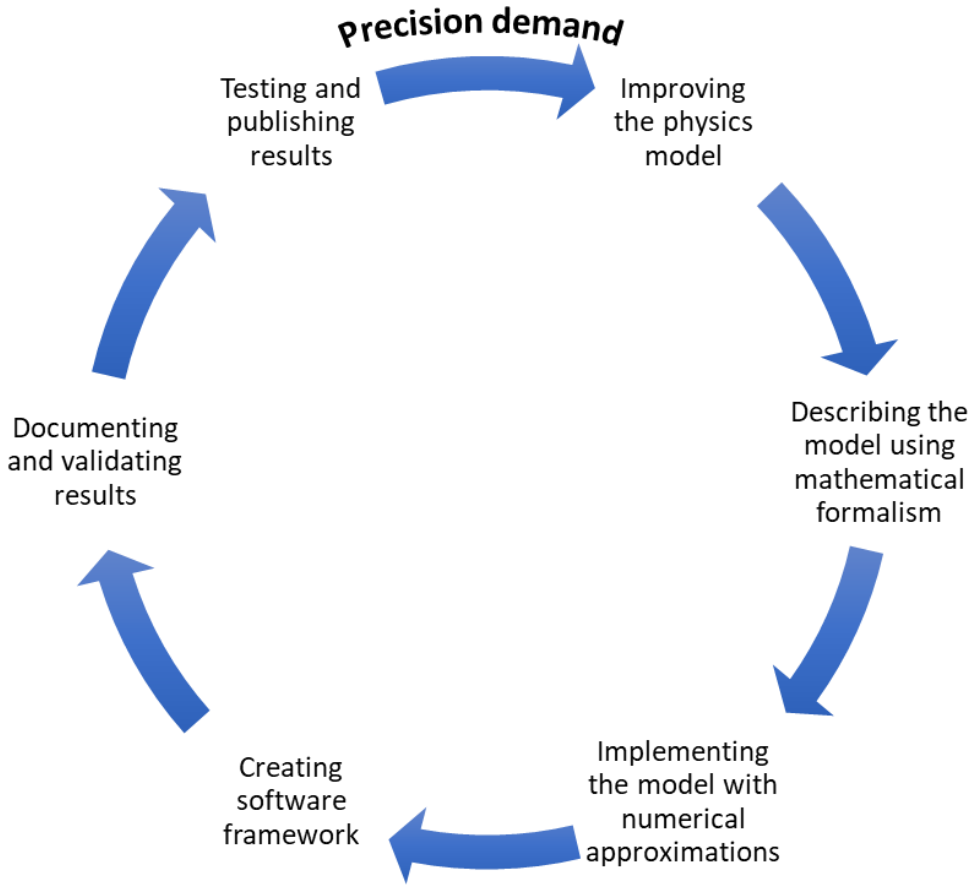
Fundamental content comes from scientists driven by the demand of scientific community

**2. Describing the model using mathematical formalism**

Mathematical formalism allows construction of a numerical algorithms and the analysis of its stability. It can show feasibility of the implementation or its potential to accumulate errors

**3. Implementing the model with numerical approximations**

This often requires solving the numerical restraints of the algorithms that have to be implemented.



# Development cycle

**4. Creating the software framework**

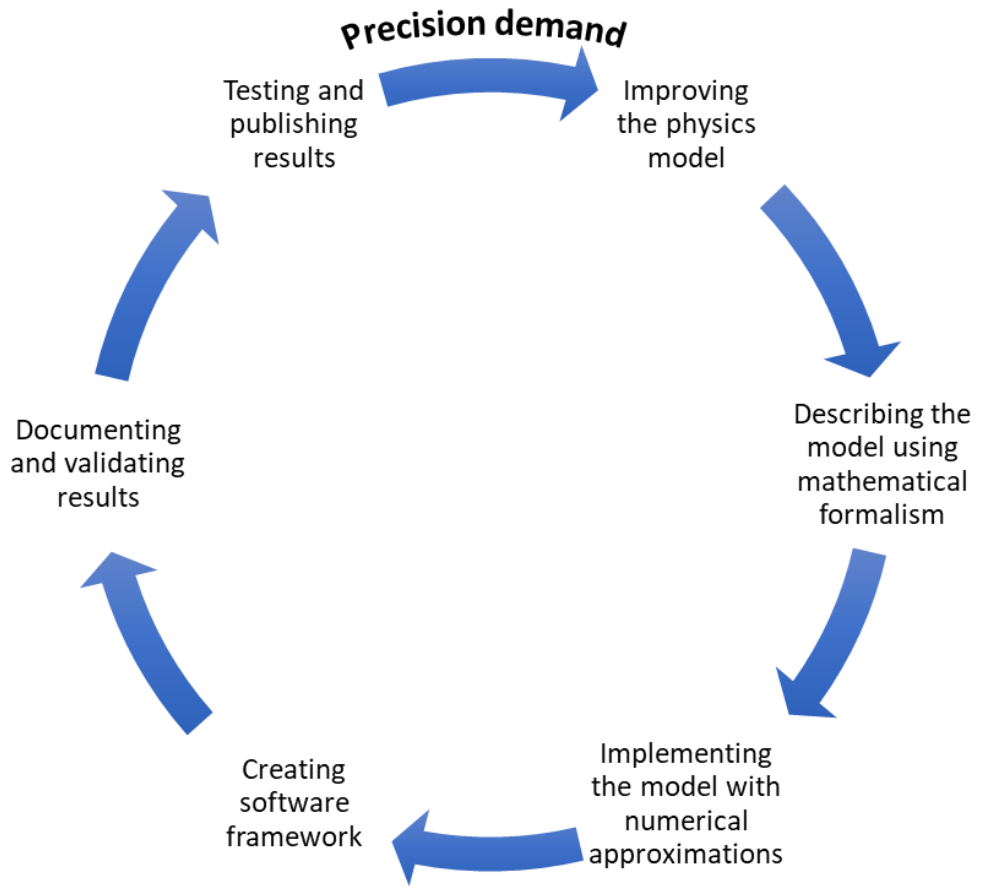
At the beginning software is usually built ad-hoc without any planning or architectural consideration. Only later it is refactored and put into some usable framework

**5. Documenting and validating higher precision results**

This is the crucial step that determines the worth of the algorithm itself. It cannot be automated and requires expert knowledge of the subject matter

**6. Testing and publishing the results**

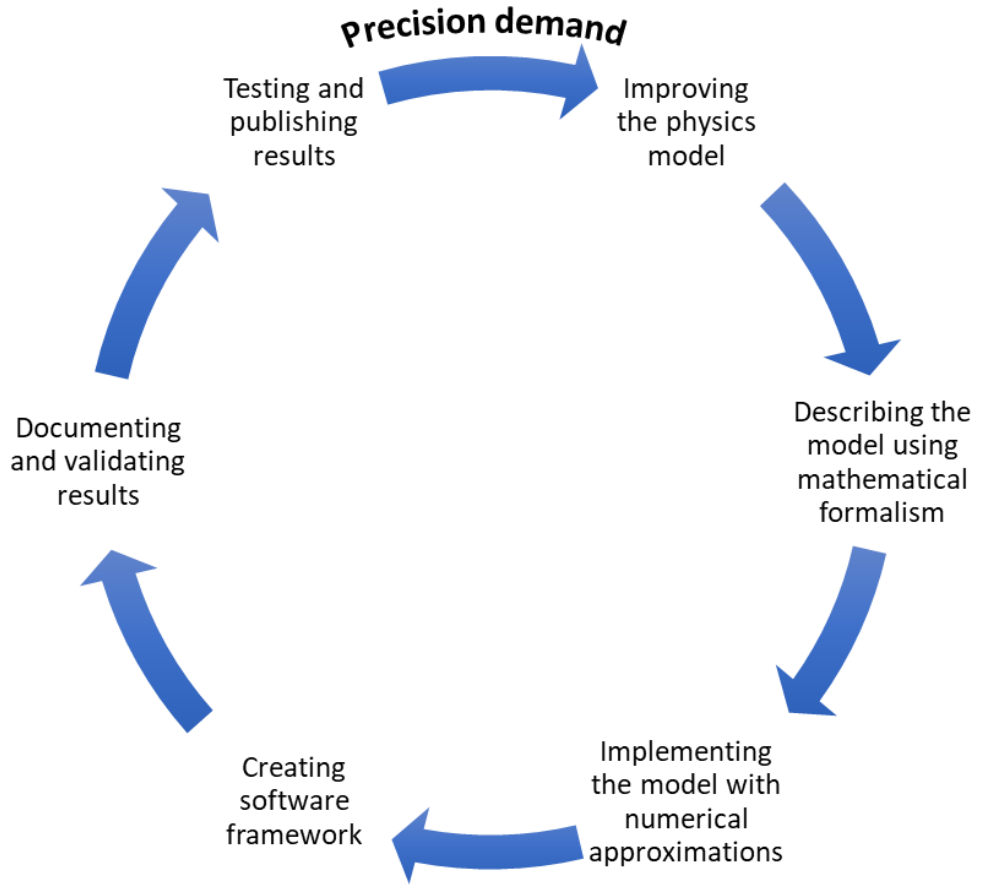
Testing possible use case scenarios is pushed to the very end as they are far less important than the validation



# Development cycle

At a first glance, this cycle can be compared to the development cycle in agile methodologies. The key difference is time. This cycle takes months to years to complete.

This time barrier is hard to overcome and it breaks the core concept of quick adaptation to changes that stands behind agile methodologies.



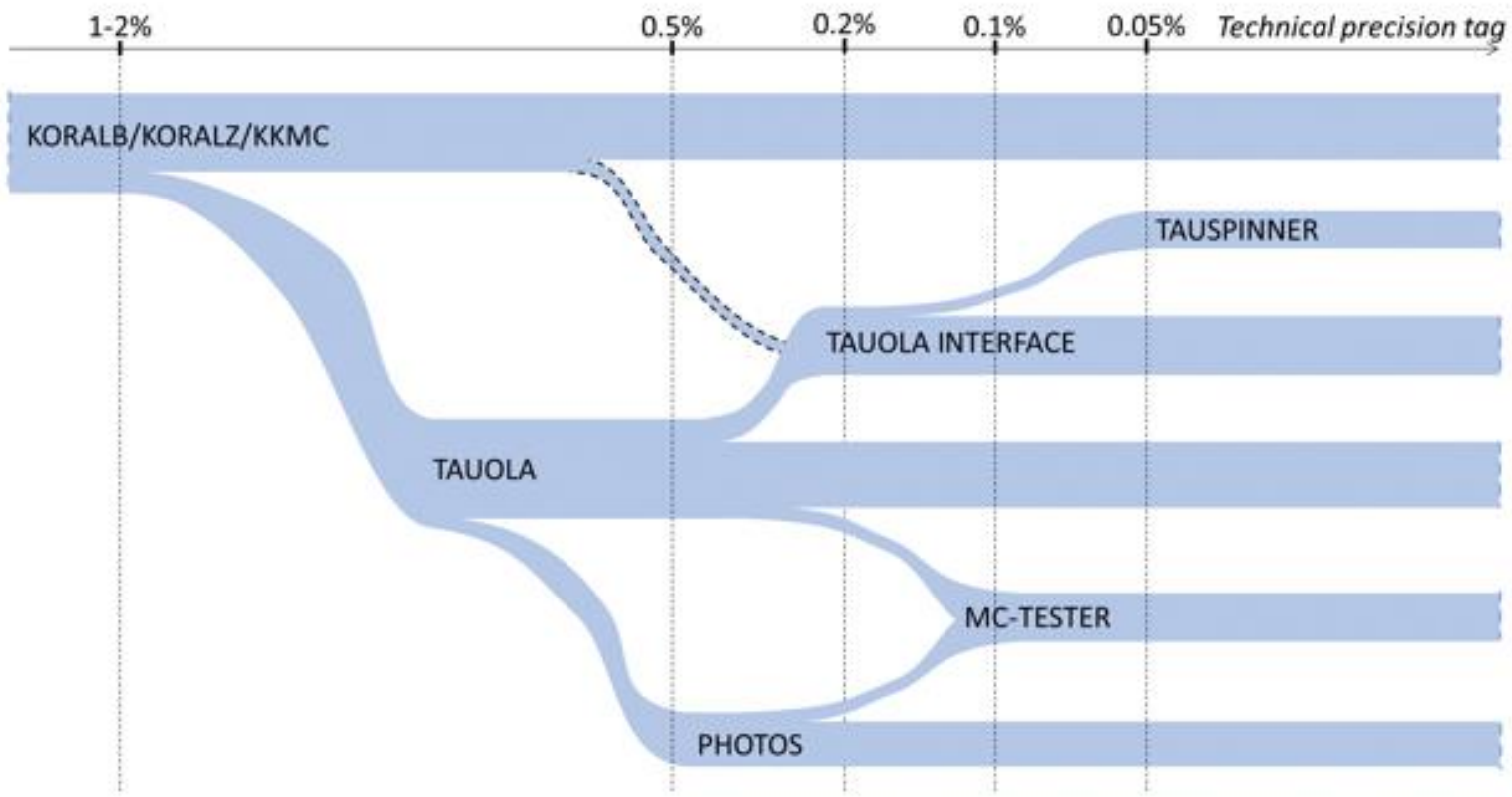
# Examples

# Examples

Figure on the right shows relations between the projects as function of time (precision).

Thin band represents preliminary step in project development, in particular its establishment.

Broad band represents evolution of mature project where applicability region is defined and dominant activity is for precision improvement

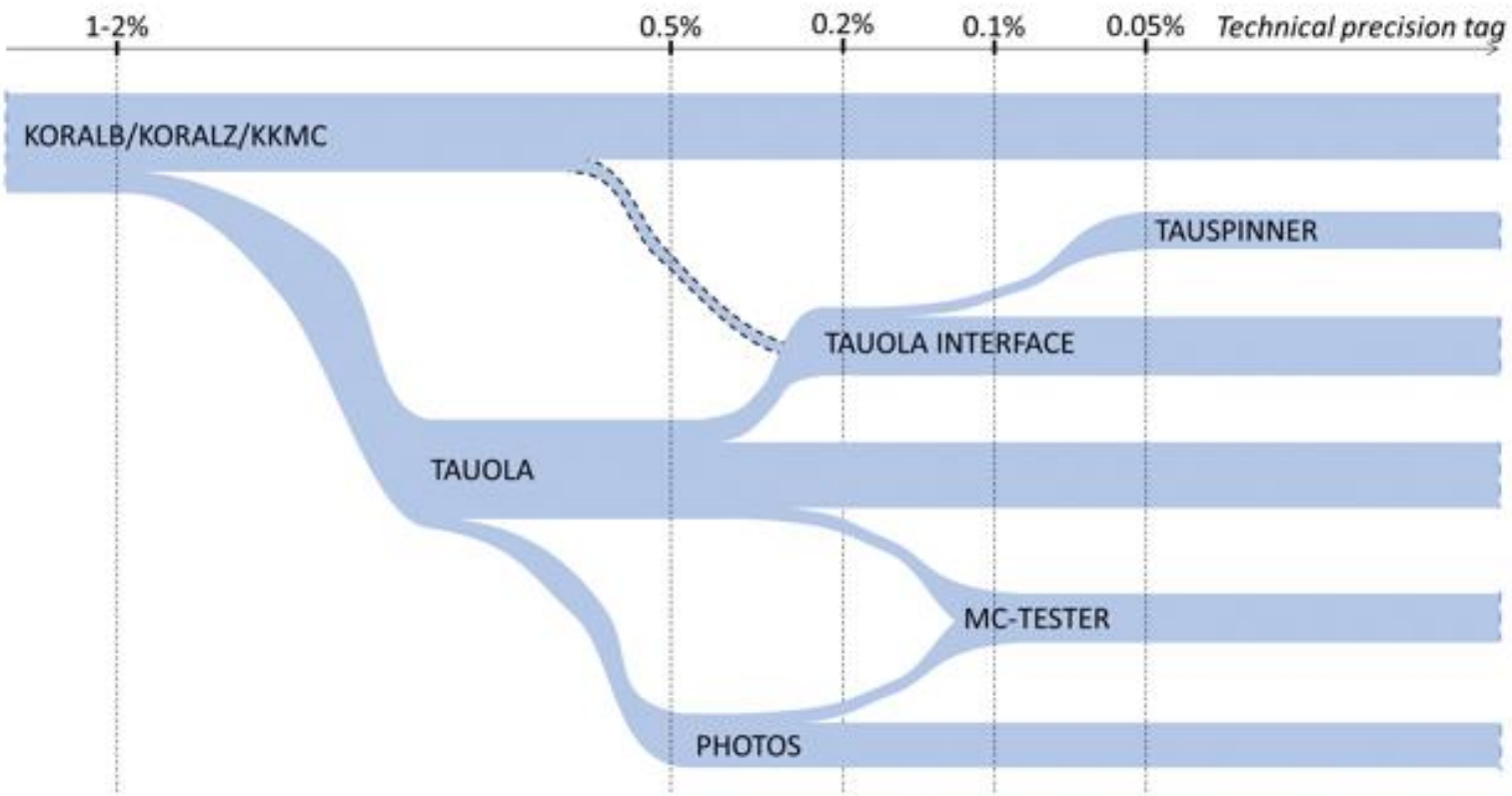


# Examples

Here, I will focus on example of Photos.

Few other examples are attached in Extra slides.

More examples are described in my PhD thesis.



# Improving the physics model

- Photos started as a branch of Tauola and was considered just an addition
- Once the demand for higher precision results raised, it became a widespread tool
- Over time its development has become more important than the development of Tauola
- The tool has been separated from Tauola
- Further updates incrementally introduced new extensions

# Describing the model using mathematical formalism

- The first implementation of Photos assumed single photon emission
- Equation for this model was simple and was easily translated into mathematical formalism
- When precision demand required the tool to take into account photons with increasingly smaller energies, an algorithm generating multiple photons had to be introduced
- This approach could not have been used for more than several photons. Calculations with three or four photons introduced large errors. A new mathematical description was needed to solve this problem
- This resulted in a new algorithm used to these days

# Implementing the model with numerical approximations

- Photos had hart time maintaining precision back, when single-precision floating-point operations were used
- Special algorithm for energy conservation correction had to be introduced
- This is even more visible now, when required precision is so high that even double-precision arithmetic introduces significant errors. For example, we sometimes get events where sum of the decay products appears to have negative mass.

# Creating the software framework

- As mentioned before, Photos started as a single file and was a part of Tauola
- It evolved into standalone tool with extensive tests and validation frameworks
- It was adapted to C++ environments by introducing the C++ interface
- Current version has been completely rewritten to C++

# Documenting and validating higher precision results

- Validating Photos started by comparing the results of the tool to results of similar tools
- This is often hard to perform as it is hard to match the exact condition the other tool was used in and create exactly the same projection as the authors of the other tool used to describe their results
- The comparison was performed as close to the other tool's environment as possible and the differences were carefully examined from the physics standpoint
- Several documents and sites with detailed summary of these comparisons were created. See e.g.:
  http://mc-tester.web.cern.ch/MC-TESTER/PHOTOS-MCTESTER/AtNLO/
  https://annapurna.ifj.edu.pl/~wasm/photosKKtest.html
  https://annapurna.ifj.edu.pl/~wasm/photosNLOtest.html

# Testing and publishing the results

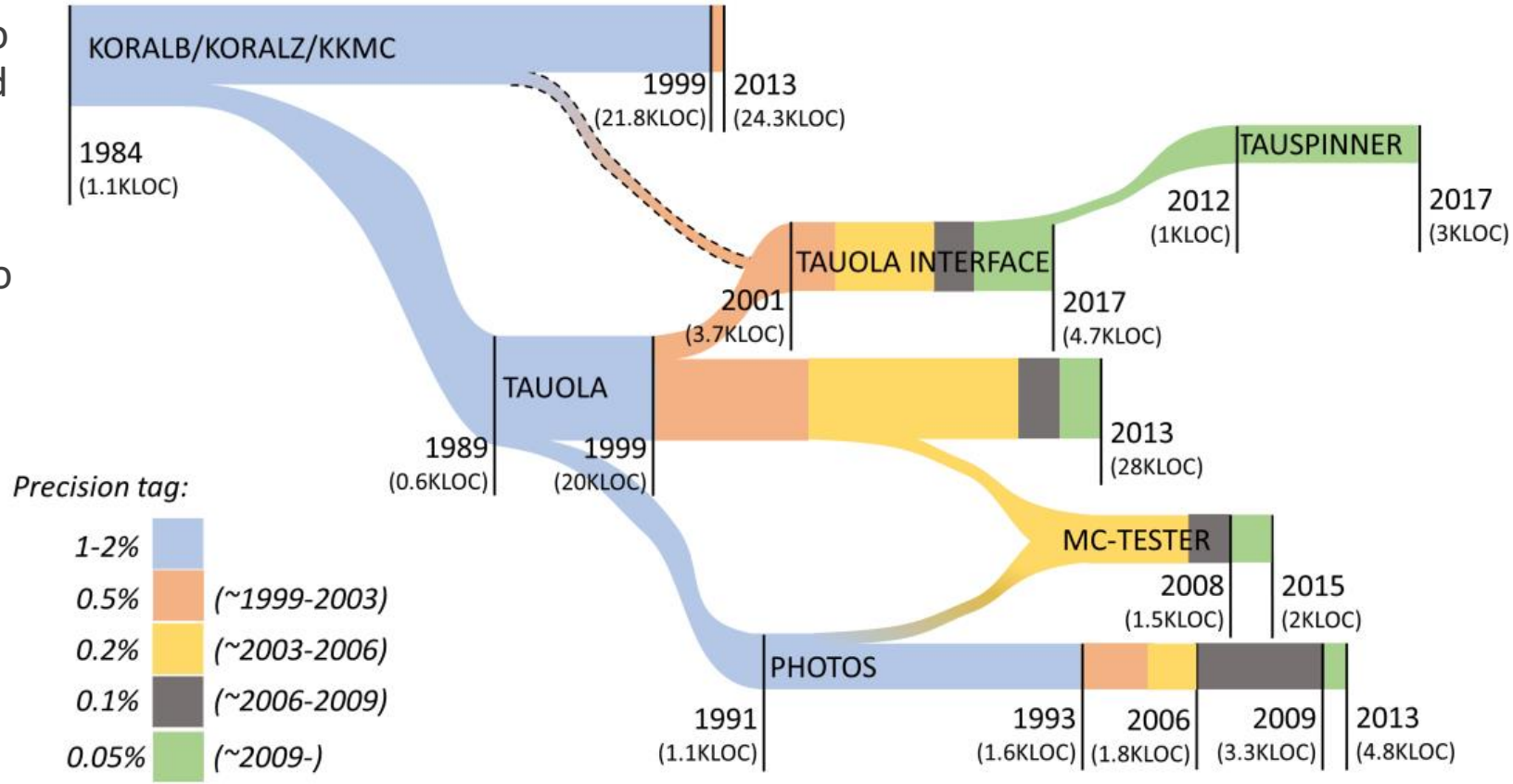
- The amount of tests needed to validate the a tool creates necessity to automate this process
- We've created a separate tool, MC-TESTER, which was used to validate Photos and other tools
- We have built a setup that generates test events used to check if the tool correctly handles various use cases
- We've automated the process of generation of large data samples for different sub-processes and comparing them against benchmark distributions
- Samples had to be carefully analysed manually before they were accepted as benchmarks
- We used this setup for each iteration to test for regression and expanded it with each iteration adding new cases and variants of the existing cases to show the result of different options

- Once results were validated, new software version was published. Usually this was accompanied by a publication that described the new features including validation results

# Quantitative evaluation

# Quantitative evaluation

Project code-base size (in Kilo Lines Of Code, KLOC) overlaid on the previous map of relations between the projects and scaled to show how code-base size related to precision.



# Quantitative evaluation

Evolution of PHOTOS Monte Carlo generator

- Typical experimental sample provided for reference to emphasize how the need for precision follows quality and quantity of data samples
- Result file size provided as indicative metric to effort required to validate the tool, as there is no systematic approach to quantify it

| Software name | Precision tag | Code size | Event record interface size | Test code size | Results file size | Typical experimental sample |
|---|---|---|---|---|---|---|
| **RADCOR** | 1-2% | 200 | 2 | - | 100kb | 10k events |
| **v1.0** | 1-2% | 1162 | $\leq$ 100 | $\leq$ 100 | 550kb | 30k events |
| **v2.0** | 0.5-1% | 1600 | 800 | $\leq$ 100 | 2.5MB | 100k events |
| **v2.15** | 0.2% | 1800 | 800 | 1271 | 2.5MB | 300k events |
| **v2.15 (new)** | 0.1-0.2% | 3315 | 1390 | 1271 | 19.5MB | 1M events |
| **v3.00** | 0.1% | 4747 | 3623 | 1573 | 20MB | 5M events |
| **v3.60** | 0.05% | 4747 | 3623 | 1573 | 21MB | 5M events |

# Quantitative evaluation

Evolution of TAUOLA Monte Carlo generator

| Software name | Precision tag | Code size | Test code size | Results file size |
|---|---|---|---|---|
| (unnamed) | 1-2% | 200 | 50 | ≤ 0.5MB |
| v1.5 | 1-2% | 1193 | 2215 | ≤ 1MB |
| v2.4 | 0.5-1% | 4016 | 8383 | ≤ 1MB |
| Tauola-Photos-F | 0.5% | 19695 | 8383 | 25.2MB |
| Tauola RChL 2012 | 0.05% | 28084 | 8383 | 35.6MB |
| Tauola RChL | 0.05% | 20194 | 9135 | 43.5MB |

Evolution of TauSpinner tool (evolution driven by new functionality, not precision requirements)

| Software name | Code size | Test code size | Results file size |
|---|---|---|---|
| v1.0.0 | 1010 | 147 | 840kb |
| v1.2.0 | 1181 | 214 | 1.3MB |
| v1.4.0 | 1694 | 2944 | 1.6MB |
| v2.0.0 | 2836 | 9661 | 8.0MB |
| v2.0.3 | 3064 | 10188 | 8.4MB |

# Quantitative evaluation

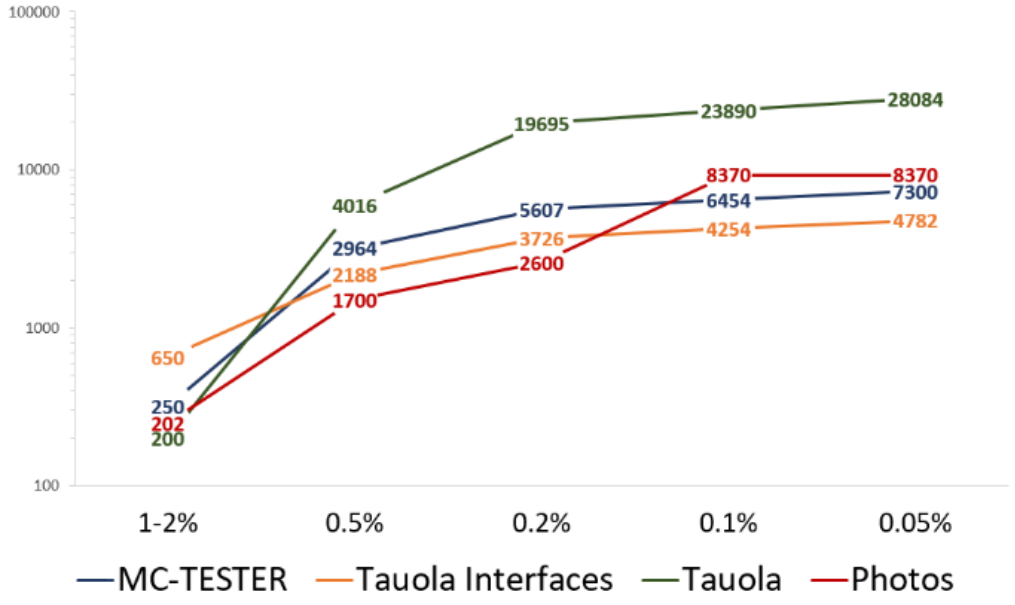
Evolution of MC-TESTER

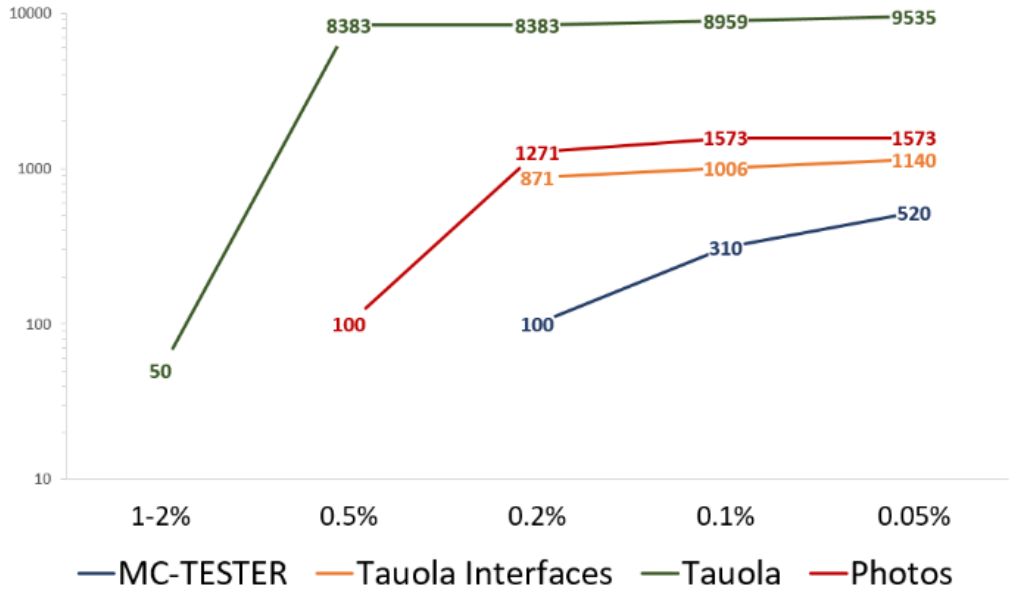
- Strongly correlated with evolution of other tools as this tool was used to test them
- Initial versions were duplicated and tailored for tests of specific tools

| Software name | Precision tag | Code size | Event record interface size |
|---|---|---|---|
| Tests of Tauola | 1-2% | 200 | 50 |
| Further tests of Tauola | 0.5-1% | 4000 | 100 |
| Tests of Photos v2 | 0.5-1% | 100 | 100 |
| Photos+ tests | 0.5-1% | 526 | 4370 |
| MC-TESTER v1.0 | 0.2-0.5% | 1271 | 4336 |
| MC-TESTER v1.23 | 0.05% | 1573 | 5541 |
| MC-TESTER v1.25 | 0.05% | 1963 | 5337 |

# Quantitative evaluation

Summary of code-base size and test size in function of precision of the results



# Testing

- TAXONOMY OF SOFTWARE TESTS
- TESTS USED IN SCIENTIFIC SOFTWARE

# Taxonomy of common software tests

Many taxonomies exist, depending on the approach and quantities that the taxonomy focuses on.

I present one that I have compiled while searching through several sources. Some tests have similar definition while having different naming conventions.

Extra slides contain taxonomy of these tests in relation to:

- Scope of the test
- Level of automation
- Whether given test is functional or non-functional

| | |
|---|---|
| Functionality tests | Smoke tests |
| Interface tests | Integration tests |
| Regression tests | Compliance tests |
| Unit tests | Localization tests |
| Compatibility tests | Installation tests |
| Maintainability tests | Configuration tests |
| System tests | Performance tests |
| Security tests | Usability tests |
| Stress tests | Reliability tests |
| Scalability tests | Acceptance tests |

# Tests used in scientific software

Automated tests are crucial, but least valuable for the scientific project

The most important tests are tests that verify if program can be used in particular condition. Such tests extend beyond computing aspects

The nature of the algorithm and the problem solved by the program are the driving forces for building the testing framework

Since testing is not the key focus of the seminar I have placed taxonomy of these tests in Extra Slides.

- Numerical technical tests
- Data flow tests
- Output metadata tests
- Core algorithms validation
- Systematics studies
- Multi-dimensional comparisons
- Testing with experimental cuts
- Random scan of parameter space
- Parameter impact test

# Useful testing techniques

Over the years we have also developed (or learned) several useful testing techniques that can be used to validate scientific software:

- Probing results at different steps of the computations
- Comparing results to semi-analytic calculations
- Testing against fake data
- Function switches
- Plain text output analysis
- Using static analysis tools
- Saving random number generator state

# Summary

# Summary

We have briefly overviewed:

- Components of Monte Carlo simulations in physics experiments
- Different type of software developed in physics community
- How physics precision is measured
- State of the art on scientific software development, precision and testing of scientific software

We have described:

- Challenges faced during scientific software development, with short examples
- Where existing methodologies fall short when dealing with scientific software
- An emergent methodology – development cycle that leads to software with higher precision of the results
- Example of this cycle based on a history of one of the tools
- Quantitative analysis of the development effort in function of the precision of the results
- Scientific software tests and how they fit into software testing taxonomy

# Summary

We have presented the following thesis:

> Often scientific software development focuses on increasing the precision of the results. This process follows the cycle presented during this seminar.
>
> The effort put into testing and into building proper validation framework rises proportionally with the effort required to achieve results with increased precision.

We have presented several examples of tools implemented and introduced to physics community that confirm this thesis.

We have presented several metrics that show the rise in effort proportional to increase of precision of these tools that back up this thesis.

# Summary

Main contribution of this work to software engineering:

- An attempt to present a systematic methodology that can be used to develop scientific software based on several examples
- Quantitative analysis of code-based metrics of several tools in function of precision of the results. In my opinion, this is a valuable data point for future or related work, especially that similar work is either hard to find or non-existent
- Quantitative analysis of relation between increased precision and increased testing effort

This contribution also includes subjects mentioned, but not elaborated here:

- Analysis of scientific software testing process
- Analysis and taxonomy of tests specific to scientific software
- Presentation of few testing techniques that can be useful during this development

# Summary

During my work for scientific community I contributed to 20 publications. Table on the right shows summary of citations of these papers. High amount of users of these tools over the years highlights their success and shows that the approach used to develop them was successful.

My contribution to these publication regards the Computer Science aspects of these projects.

I took part in development of the software related to these projects as well as contributed to the decision-making process regarding the software architecture, building the testing framework, software organization and development of automated testing processes and support tools.

This work has been supported in part from funds of Polish National Science Centre under decision DEC-2017/27/B/ST2/01391

| Cite summary | Citable papers | Published only |
|---|---|---|
| **Total number of papers analyzed:** | 20 | 13 |
| **Total citing count:** | 1,060 | 999 |
| **Average citing per paper:** | 53.0 | 76.8 |
| **Articles based on cite number:** | | |
| **Renowned papers (500+)** | 0 | 0 |
| **Famous papers (250-499)** | 1 | 1 |
| **Very well-known papers (100-249)** | 2 | 2 |
| **Well-known papers (50-99)** | 3 | 3 |
| **Known papers (10-49)** | 8 | 5 |
| **Less known papers (1-9)** | 4 | 1 |
| **Unknown papers (0)** | 2 | 1 |
| **$h_{HEP}$ index** | 13 | 11 |

Cite summary of papers co-authored by me, provided by INSPIRE community (https://inspirehep.net/authors/1066144).

This summary includes my MSc Thesis listed as unknown, unpublished paper.

17 of these publications are also available on Web Of Science

# Thank you for listening!

# Extra slides

# REFERENCES

# References


[1] S. M. Baxter, S. W. Day, J. S. Fetrow and S. J. Reisinger, **Scientific Software Development Is Not an Oxymoron**, PLoS Computational Biology 2 (sep, 2006) e87.

[2] R. Sanders and D. Kelly, Dealing with Risk in Scientific Software Development, IEEE Software 25 (jul, 2008) 21–28.

[3] D. Kelly, **Scientific software development viewed as knowledge acquisition: Towards understanding the development of risk-averse scientific software**, Journal of Systems and Software 109 (nov, 2015) 50–61.

[4] L. Hatton and A. Roberts, **How accurate is scientific software?**, IEEE Transactions on Software Engineering 20 (1994) 785–797.

[5] E. Tang, X. Zhang, N. T. Muller, Z. Chen and X. Li, **Software Numerical Instability Detection and Diagnosis by Combining Stochastic and Infinite-Precision Testing**, IEEE Transactions on Software Engineering. 2017, Vol. 43.

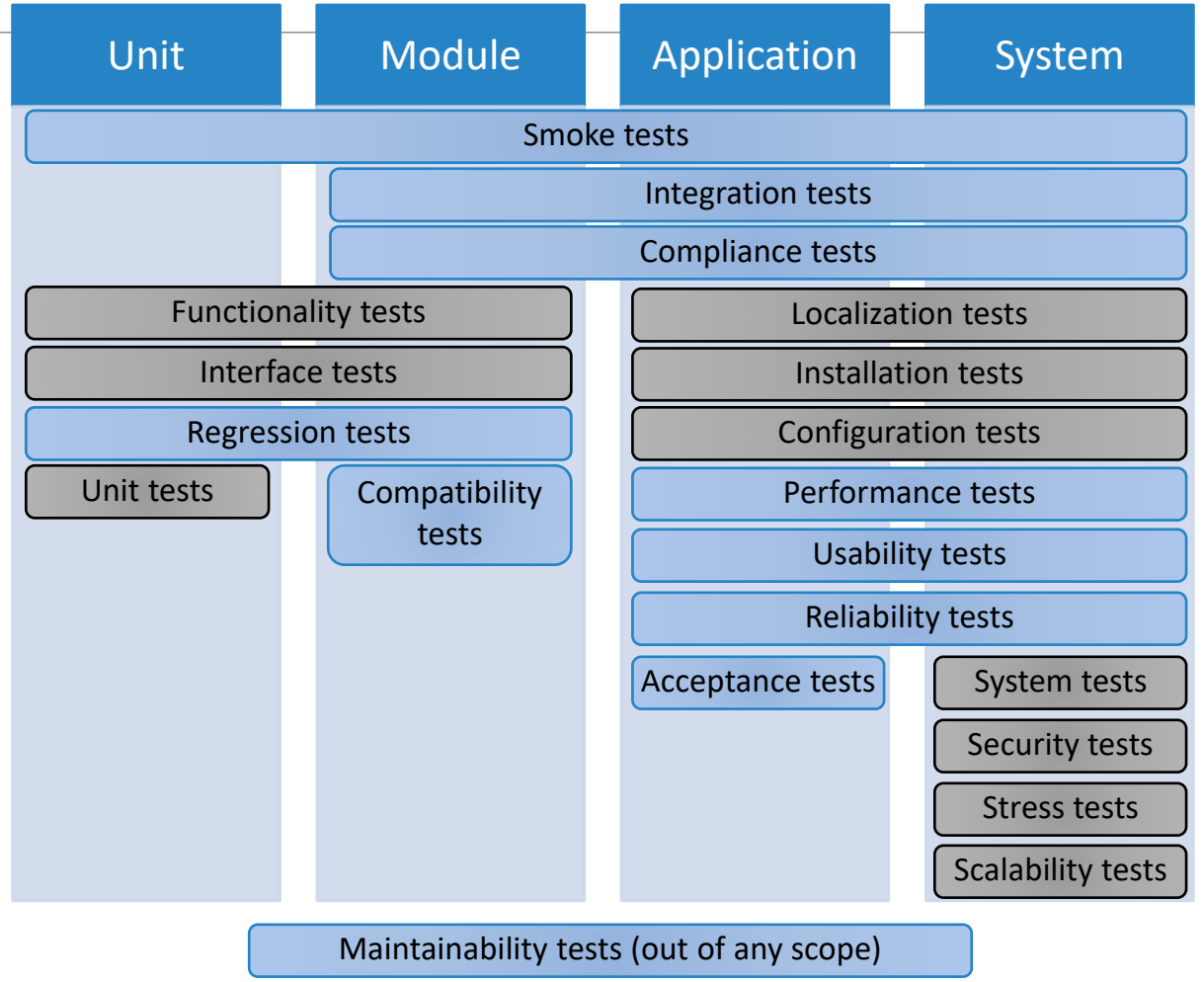
[6] U. Kanewala and J. M. Bieman, **Testing scientific software: A systematic literature review**, Information and Software Technology. 2014, Vol. 56.

# TAXONOMY OF SOFTWARE TESTS

# Taxonomy of common software tests

Scope of the test

- Scope of the grayed-out tests should not change
- Scope of the blue tests can vary



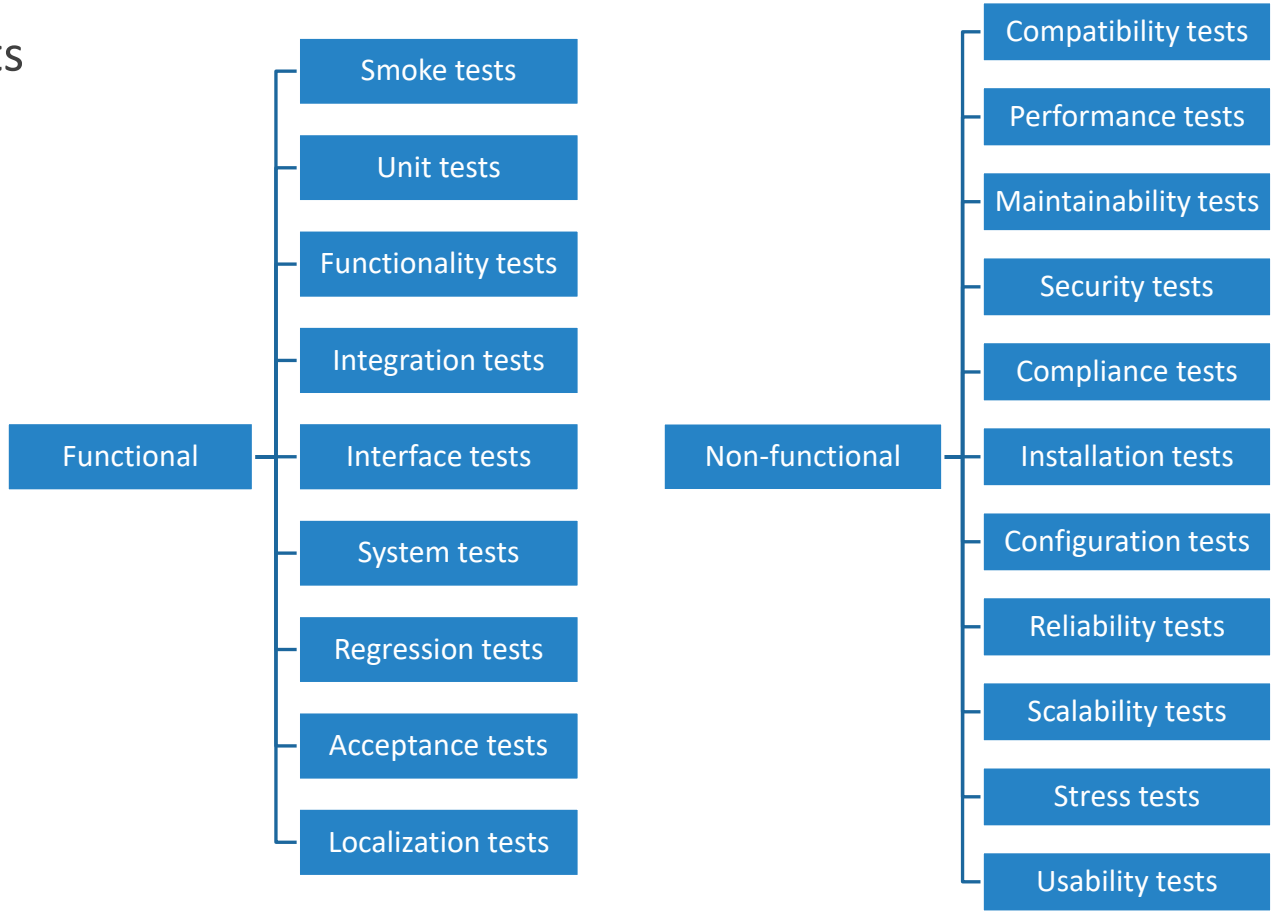
# Taxonomy of common software tests

Level of automation

| Automated | Automated and manual | Manual |
| --- | --- | --- |
| • Smoke tests<br>• Unit tests<br>• Functionality tests<br>• Integration tests<br>• Interface tests<br>• Compatibility tests<br>• Scalability tests<br>• Stress tests<br>• Performance tests | • Maintainability tests<br>• Compliance tests<br>• Reliability tests<br>• Regression tests<br>• Installation tests<br>• Configuration tests<br>• System tests | • Localization tests<br>• Security tests<br>• Usability tests<br>• Acceptance tests |

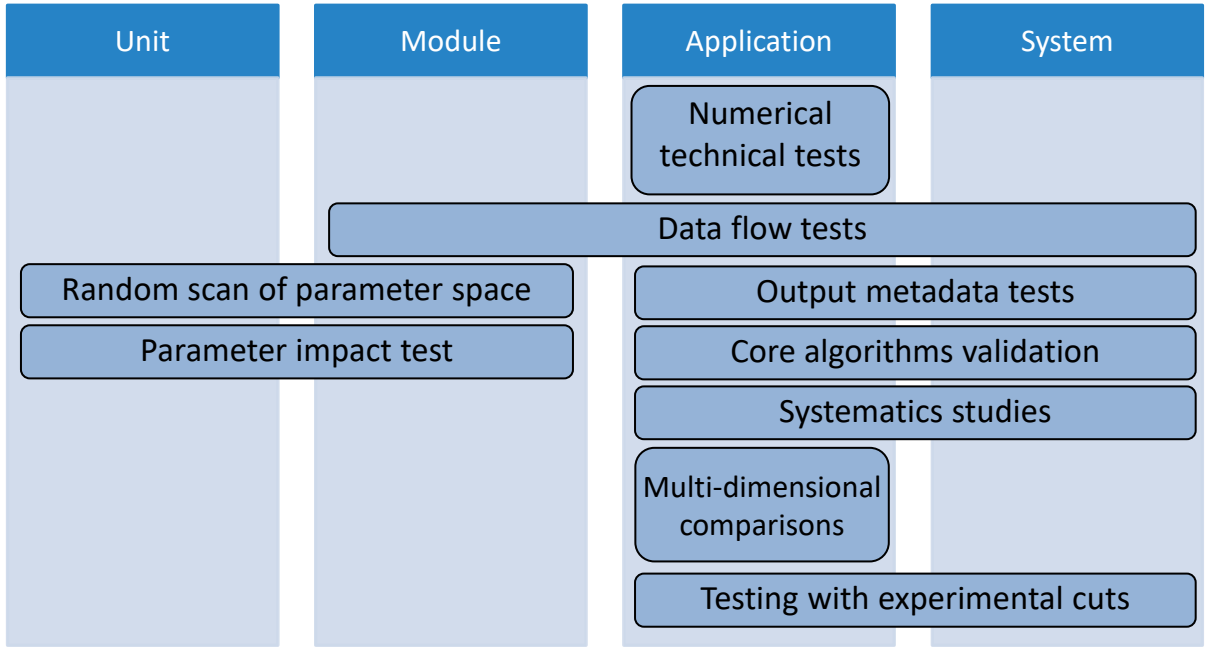
# Taxonomy of common software tests

Functional and non-functional tests



# Taxonomy of tests used in scientific software

Scope of the test



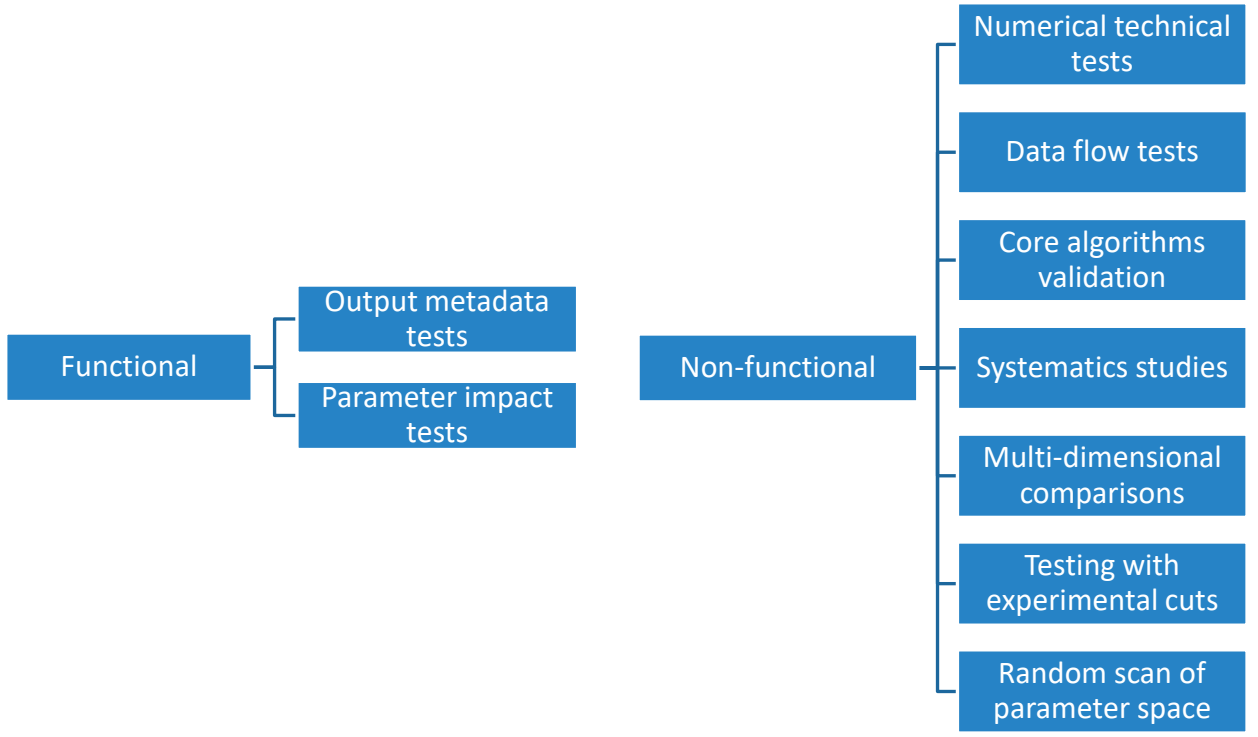
# Taxonomy of tests used in scientific software

Level of automation

| Automated | Automated and manual | Manual |
| --- | --- | --- |
| • Numerical technical tests<br>• Data flow tests<br>• Multi-dimensional comparisons<br>• Testing with experimental cuts<br>• Random scan of parameter space<br>• Parameter impact tests | • Output metadata tests | • Core algorithms validation<br>• Systematics studies |

# Taxonomy of tests used in scientific software

Functional and non-functional tests



# OTHER EXAMPLES OF THE DEVELOPMENT CYCLE

# Tauola

**Tauola** Monte Carlo tool for simulating τ decays started from the need of a model that would describe the τ decay process as part of other tools, KORALB and KORALZ

At that time (around year 2000 and earlier), technical model precision was at 1%.

The year 2001 witnessed the introduction of technical precision of 3‰ which drove the authors of the tool to improve the precision of the model. This happened a few times already; each iteration adding more and more details to the model.

The framework of this tool has evolved over time. The most significant change was transition to Tauola++, in which a C++ interface has been introduced that allowed the use of Tauola within C++ environments and with C++ event records and creating a way to add new extensions (such as electroweak corrections), which has been done over next years.

# Resonance Chiral Lagrangian (RChL)

An extensive analysis of the mathematical approach was needed when designing the framework for fitting the multi-dimensional distributions used in research regarding RChL currents. Initial fitting strategy resulted in lengthy computations. The analysis of the mathematical model helped us find the approximations that did not impact the fitting strategy while allowed us to produce preliminary results much faster. Without it a lot of time would have been wasted waiting on the results instead of improving the model.

When fitting RChL currents, algorithm internally calculates derivatives of the distributions with respect to all model parameters. We used variable step size 16-point gaussian integration to integrate over these parameters. However, function that used variable step size can create artificial discontinuities in the fitting function at points where step size changes. This was problematic and impacted algorithm convergence. We had to change the integration function lowering its precision. To offset this cost, and further improve convergence, we smoothened the integrant by changing the integration function to a function commonly used to describe resonances in particle physics.

# TauSpinner

TauSpinner started off as a single file fully dependent on Tauola++ project, including its framework and event processing algorithm. However, its use cases are different than that of Tauola. In next iterations TauSpinner became a separate tool with its own version number, source code, documentation, tests and examples. Its interface has been updated and the code that provides data to TauSpinner abstracted so that data can be provided from any source, including HepMC events and files.

When studying the applications of TauSpinner tool the algorithm was stripped apart and analysed by switching off and on number of options available within the tool. The results were compared against theoretical predictions and with each other to observe if the effect with and without specific option matches the expectation. The tool was also compared against existing tools and it has also been used on data with experimental cuts to emphasize the benefits of event re-weighting algorithm.