



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Bachelor's thesis

Evaluation of Analysis Frameworks in the ATLAS Experiment at CERN and Design of an Optimized Framework for the Search and Determination of Properties of New Particles

Georgiy Ivannikov

Department of Software Engineering
Supervisor: doc. Dr. André Sopczak

January 7, 2021



Acknowledgements

I would like to thank my supervisor, André Sopczak for providing guidance and feedback throughout this project.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work for non-profit purposes only, in any way that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on January 7, 2021

.....

Czech Technical University in Prague
Faculty of Information Technology

© 2021 Georgiy Ivannikov. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Ivannikov, Georgiy. *Evaluation of Analysis Frameworks in the ATLAS Experiment at CERN and Design of an Optimized Framework for the Search and Determination of Properties of New Particles*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Existuje několik experimentů, které berou data na Large Hadron Collider (LHC) v CERNu ve švýcarské Ženevě. Jedním z nich je experiment ATLAS. Detektor ATLAS se používá k měření vlastností Higgsova bosonu a k hledání nestandardní fyziky, tj. Temné hmoty. Detektor generuje velké množství nezpracovaných dat a poté je informace zpracovávána softwarovým systémem ATLAS. Podle údajů získaných z detektoru identifikuje spolupráce ATLAS fyzické objekty. Těmito fyzickými objekty jsou například elektrony, miony nebo jiné elementární částice. Objekty pak mohou být použity k opětovnému vytvoření reakcí vyvíjejících standardní částice nebo hypotetické částice. Tato studie poskytuje konkrétní příklad vývoje softwaru pro Higgsovy bosony s cílem poskytnout návrhy na jeho zlepšení. Kromě toho je v této práci navržena technika dokumentování rámců. Rovněž se diskutuje o nápadech, jak lze zlepšit monitorování pokroku.

Klíčová slova ATLAS, CERN, LHC, framework, Python, C++, dokumentace, testování, benchmark, GitLab, Jira, Doxygen

Abstract

There are several experiments that take data at the Large Hadron Collider (LHC) at CERN in Geneva, Switzerland. One of them is the ATLAS experiment. The ATLAS detector is used to measure the Higgs boson properties and to search for non-standard physics, i.e dark matter. A large amount of raw data is generated by the detector, after which the information is processed by the ATLAS software system. According to the data obtained from the detector, the ATLAS collaboration identifies physical objects. These physical objects are, for example, electrons, muons or other elementary particles. The objects can then be used to recreate the reactions evolving standard particles or hypothetical particles. This study provides a specific example on the software development for Higgs bosons with the aim of providing suggestions to improve it. Moreover, in this thesis, a technique of documenting frameworks is suggested. Ideas are also discussed on how progress monitoring can be enhanced.

Keywords ATLAS, CERN, LHC, framework, Python, C++, documentation, testing, benchmark, GitLab, Jira, Doxygen

Contents

Introduction	1
1 CERN	3
1.1 ATLAS	3
1.2 Large Hadron Collider	4
1.3 Higgs boson	4
2 Data flow	5
2.1 xAOD, DxAOD	5
2.2 Group Frameworks	5
2.3 FakeBkgTool	6
2.3.1 Fake Factor Method	6
2.3.2 Standard Matrix Method	7
2.3.3 Likelihood Matrix Method	7
2.4 Output	10
3 Standard Matrix Method	11
3.1 Installation and set up	11
3.2 Structure	12
3.3 Result visualization	13
3.3.1 TRExFitter	13
3.3.2 TRExFitter configuration file	14
3.3.3 Result	15
4 Framework evaluation	17
4.1 Usability	17
4.1.1 Group framework 2	17
4.1.2 FakeBkgTool	18
4.2 Performance	18
4.2.1 Group framework 2	18

4.2.2	FakeBkgTool	19
4.2.3	Testing performance	19
4.3	Flexibility	20
4.4	Data structures	21
4.4.1	Group framework 2	21
4.4.2	FakeBkgTool	21
5	New outlined framework	23
5.1	Requirements	23
5.2	Stability	24
5.2.1	Unit testing	24
5.2.2	Performance testing	25
5.2.3	Integration testing	25
5.3	Architecture	25
5.3.1	Current state	25
5.3.2	Microservice architecture	26
5.3.3	New framework architecture	26
5.4	Performance	26
5.5	Configurability	27
5.6	Automatization	27
5.7	New framework evaluation	28
5.7.1	Pros	28
5.7.2	Cons	29
6	Documentation and development tracking	31
6.1	Documentation	31
6.1.1	API documentation	31
6.1.2	Software documentation for its installation and applica- tion	32
6.2	Development tracking	32
6.2.1	Jira	33
6.2.2	Improvement of development tracking	33
6.3	Communication	38
	Conclusion	39
	Bibliography	41
A	Acronyms	45

List of Figures

2.1	Comparison of the results of estimating a dataset with 5 loose dilepton events for Standard and Likelihood MM. The four plots show the effect of changing the number of tests.	9
2.2	Comparison of the results of estimating a dataset with 1000 loose dilepton events for Standard and Likelihood MM. The four plots show the effect of changing the number of tests.	10
3.1	The dotted line represents the actual form of the events found in the used sample. After smoothing, the strong red line indicates the form of the systematic uncertainty. The blue line shows the symmetrical effect of moving the system in the opposite direction. The blue hatched area is the statistical uncertainty.	16
4.1	Class that contains information about an event	22
4.2	Illogical use of data structures	22
6.1	Integrations tab in Jira	35
6.2	Form for filling in data about Jira	36
6.3	Jira connected	37
6.4	Merge request title with issue id	37
6.5	Jira page with link to merge request in comments	38

List of tables

3.1 FakeBkgTool tool structure	13
--	----

Introduction

ATLAS collaboration has to process, store and analyze a large amount of data coming from the ATLAS detector operating in the LHC tunnel. The amount of data will grow rapidly because the LHC is constantly being modernized, and, thus, ATLAS records more and more data, therefore the software used for analysis of this data is an integral part of the work. For this purpose, the members of the ATLAS collaboration have developed several frameworks. The developers and maintainers of the ATLAS software are typical high energy particle physicists, therefore the software and the processes to develop this software, according to software engineering standards can be improved in several aspects.

The aim of the work

The aim of this bachelor's work is to analyze existing frameworks, in order to make suggestions for their improvement. It is also an aim to make suggestions that will help improve the routine associated with working with software in the ATLAS collaboration.

The first chapter provides the theoretical part of this work so that it is possible to imagine what the ATLAS organization is and how the software plays a role in their activity.

The second chapter describes the data flow in ATLAS. A brief description is given of the data processing, starting from receiving them from the detector to the final analysis result.

The third chapter describes the main ATLAS framework for data analysis called Athena. The process of installing the framework, the format of input data and output data come up as well as the use will be described.

The fourth chapter will reflect the practical part of this thesis. This will show how the framework is evaluated according to the software engineering criteria.

The fifth chapter will describe the concept of a new framework and describe the pros and cons of switching to it from the current one.

The sixth chapter will describe the software documentation strategy and provide suggestions for improving the development tracking.

The last chapter contains the conclusion.

CERN

This chapter aims to acquaint the reader with CERN, ATLAS and the Large Hadron Collider. Also, special attention will be paid to the ATLAS detector and how the data stream from the detector is processed.

CERN is a project financed by more than twenty European states, a dozen more countries and international organizations have observer or associate member status in it. The most important and famous CERN project is the Large Hadron Collider - an elementary particle accelerator. The LHC has two general purpose detectors, ATLAS and CMS [1]. It explores from the Standard Model to extra dimensions and particles that can be dark matter. The difference between the detectors lies in their technical designs. The experiments performed on these detectors are referred to as the detectors name [1].

1.1 ATLAS

ATLAS is designed to investigate different types of physics that might be detected in energetic collisions at the Large Hadron Collider. Some of this research is about confirming or improving measurements of the parameters of the Standard Model, while many others are looking for new physics. ATLAS uses an advanced decision making system (trigger system) to tell the detector which events to record and which to ignore [2].

The detector produces a huge amount of raw data - about 25 MB per event (in its original raw form, suppressing zeros reduces it to 1.6 MB) for each of 40 million beam intersections per second in the center of the detector, which gives a total of 1 PB per second raw data [3]. For all recorded events, an offline reconstruction is performed that converts the signals from the detectors into physical objects such as hadronic jets, photons and leptons. For the reconstruction of events, grid computing (LHC Computing Grid) is intensively used, which allows parallel use of computer networks of universities and laboratories around the world for the resource-intensive (in terms

of using processor time) task of reducing large amounts of initial data to a form suitable for physical analysis. The software for these tasks have been developing for more than twenty five years and continues to improve as the experiment progresses [4].

1.2 Large Hadron Collider

The Large Hadron Collider is a colliding particle accelerator designed to accelerate protons and heavy ions (lead ions) and study the products of their collisions. The center of mass energy of the collisions is 13 TeV. The main task of the Large Hadron Collider is to reliably detect at least some deviations from the Standard Model - a set of theories that make up the modern understanding of fundamental particles and interactions. Although it has many advantages, it also possesses many difficulties. For example, it does not describe the gravitational interaction whilst also not explaining the existence of dark matter or dark energy. The Large Hadron Collider should help answer questions that are not resolved in the Standard Model [5].

1.3 Higgs boson

The Higgs boson is an elementary particle, the quantum of the Higgs field, which necessarily arises in the Standard Model of particle physics due to the Higgs mechanism of spontaneous breaking of electroweak symmetry. It was postulated by British physicist Peter Higgs in his fundamental papers published in 1964. After several decades of searches, on July 4, 2012 [2], as a result of research at the Large Hadron Collider, a candidate for its role was discovered - a new particle with a mass of about 125-126 GeV / c^2 [6]. There is good reason to believe that this particle is the Higgs boson. In March 2013, there were reports from individual CERN researchers that the particle found six months earlier was indeed the Higgs boson [7].

Data flow

The ATLAS Computing System processes the data produced by the ATLAS detector. Computing software are used to store, process and evaluate large amounts of collision data. Data from the ATLAS detector is calibrated, reconstructed and automatically distributed worldwide by the ATLAS Data Management system. The ATLAS Production System then filters through these events and selects the ones needed for a particular type of analysis. This shrinks the data set down to a manageable size in order for the scientists to analyze the data [8]. The data flow from recording the ATLAS detector data to the analysis is as follows: FakeBkgTool is a framework that was created by

| **Detector** | **xAOD** | **DxAOD** | **GW1** | **GW2** | **FakeBkgTool**

the ATLAS collaboration for the data analysis. Before the data reaches the framework for processing they are placed in containers through xAOD and DxAOD. In the subsequent analysis it is more convenient to use a data structure n-tuples called using Group Framework 1 (GFW1) and Group Framework 2 (GFW2).

2.1 xAOD, DxAOD

xAOD, DxAOD - tools for creating containers that will store event data received from the detector. The large amount of recording data is therefore structured in different formats. This step is necessary to separate the data according to specifications. The containers include physical reconstructed objects like muons, electrons and other elementary particles [8] [9].

2.2 Group Frameworks

There are two group Frameworks: Group Framework 1 and Group Framework 2. Both frameworks serve to represent the data in a data structure called

ntuples. The frameworks then store all the information in a way that can be handled by the scientists. The process goes in such a way that first GFW1 processes the events and wraps the data in ntuples, then GFW2 reads these ntuples, loop over all the entries and apply some event selections that slim the GFW1 ntuples in order to produce mini-ntuples. These mini-ntuples can be used for further analysis. GFW1 ntuples have a large size and can only be stored on the world wide grid, therefore GFW2 which is a slipped version is created such that this data can be stored for fast access on disks. Disks space is about 10 TB. Group framework 1 is recreated several times a year them, which takes about one week each time. [10].

2.3 FakeBkgTool

The fakeBkgTool is a framework written in the C++ programming language created by the ATLAS collaboration as a general analysis tools for the scientists. The framework contains several methods for analysis: Standard Matrix Method, Fake Factor Method and Likelihood Matrix Method. Each method differs in the speed of processing the data. These 3 methods are used to estimate non-prompt and fake lepton backgrounds directly from the data and do not rely on the simulation [11].

2.3.1 Fake Factor Method

The fake factor method is a data-driven procedure for modeling background arising from misidentification. The method provides a measurement of the yield and the kinematic distributions of fake background. It is a general technique, applicable to any physics analysis in which particle-level selection criteria is used to suppress background. The fake factor method can be used with any number of final state particles and it is independent of the event selection. In the following it is presented in the context of modeling the background to misidentified electrons and muons, referred to as “leptons”, but the general discussion and techniques described are applicable to the background modeling of any particle with identification criteria: photons, hadronic-taus, heavy-flavor jets, or more exotic objects such as lepton-jets [12]. The fundamental idea of the fake factor method is simple: select a control sample of events enriched in the background being estimated, and then use an extrapolation factor to relate these events to the background in the signal region. The method is data-driven provided the control sample is selected in data, and the extrapolation factor is measured with data. For background arising from particle misidentification, the extrapolation is done in particle identification space. The control sample is defined using alternative particle selection criteria that are chosen such that the rate of misidentification is increased. [13] The extrapolation factor relates background misidentified with this criteria to background misidentified as passing the full particle selection of the signal

region. The extrapolation factor is referred to as the “fake factor”. The fake factor is measured and applied under the assumption that it is a local property of the particles being misidentified and is independent of the event-level quantities. The fact that the extrapolation is done in an abstract particle identification space can be conceptually challenging, but the underlying procedure is straightforward [14].

2.3.2 Standard Matrix Method

This technique is based on a linear relationship established between the unknown prompt or non-prompt nature of the leptons and their observed isolation and identification quality. Using the loose lepton definition as baseline, the identification and isolation quality for a given lepton can take two exclusive values: tight or not-tight [15]. The matrix method is expressed analytically:

$$\begin{pmatrix} N_{TT} \\ N_{T\bar{T}} \\ N_{\bar{T}T} \\ N_{\bar{T}\bar{T}} \end{pmatrix} = \begin{pmatrix} rr & rf & fr & ff \\ r(1-r) & r(1-f) & f(1-r) & f(1-f) \\ (1-r)r & (1-r)f & (1-f)r & (1-f)f \\ (1-r)(1-r) & (1-r)(1-f) & (1-f)(1-r) & (1-f)(1-f) \end{pmatrix} \begin{pmatrix} N_{RR}^{TT}/rr \\ N_{RF}^{TT}/rf \\ N_{FR}^{TT}/fr \\ N_{FF}^{TT}/ff \end{pmatrix}.$$

$$NF = NTT + NTT + NTT \quad (2.1)$$

where

- N – number of events
- F – non-prompt leptons
- R – prompt leptons
- T – tight leptons
- r(f) – efficiencies and represent the probability of a loose prompt (non-prompt) lepton to be tight
- RF FR FF can be obtained by matrix inversion

2.3.3 Likelihood Matrix Method

The Standard Matrix Method is restricted in two key ways that are particularly important at low statistics:

- the estimate for the number of fakes can be negative
- the uncertainty is the quadrature sum of the fake estimates from each event, which may not be appropriate

This method uses the likelihood function from mathematical statistics and thus solves these problems.

The likelihood function in mathematical statistics is the joint distribution of a sample from a parametric distribution, viewed as a function of a parameter. This uses the joint density function (in the case of a sample from a continuous distribution) or the joint probability (in the case of a sample from a discrete distribution) calculated for the sampled data [15] [16].

The likelihood approach offers several advantages:

1. Avoiding nonphysical answers
2. Having better statistical precision than the standard MM
3. Remains robust even when the real and fake efficiencies are not so different

A comparison of the analysis results between likelihood MM and standard MM is shown in the Figures 2.1 and 2.2.

1. Entries – number of tests
2. Measured N fake – measured number of fake events
3. Blue dash line – exact number of fake events
4. Red area – standard MM result
5. White area – likelihood MM result

It follows from these graphs that even with a small number of tests, the likelihood MM result is close to real. The advantage of standard MM is that the method was able to find the exact number of fake events with fewer tests, but up to this point the results were far from optimal.

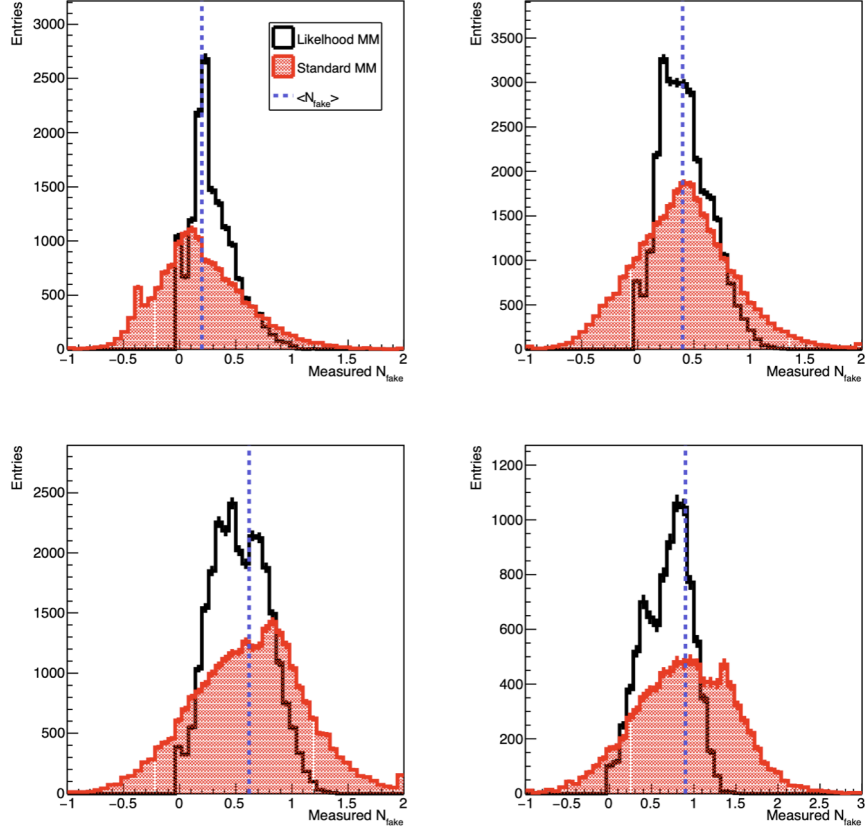


Figure 2.1: Comparison of the results of estimating a dataset with 5 loose dilepton events for Standard and Likelihood MM. The four plots show the effect of changing the number of tests.

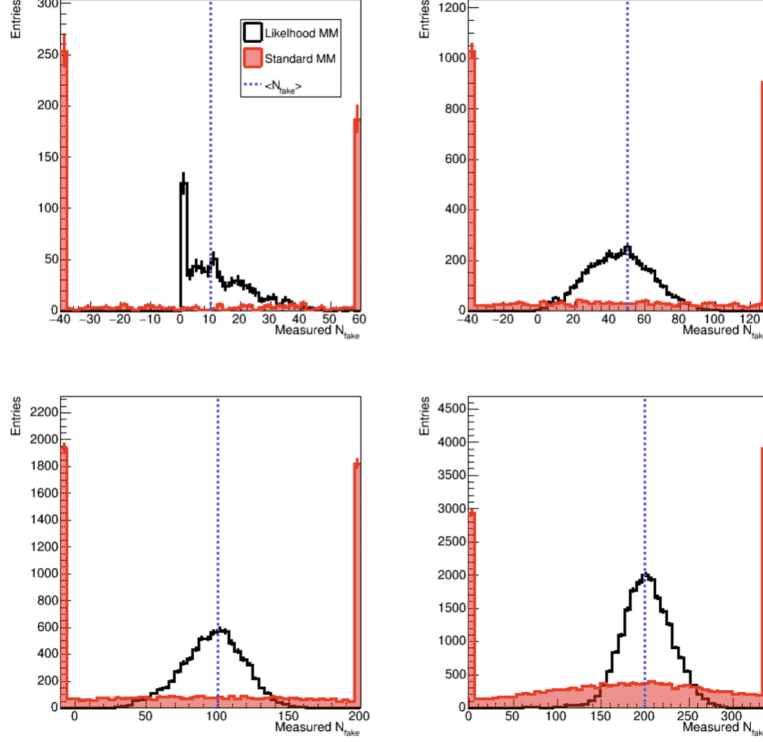


Figure 2.2: Comparison of the results of estimating a dataset with 1000 loose dilepton events for Standard and Likelihood MM. The four plots show the effect of changing the number of tests.

2.4 Output

The conclusion of data analysis is presented as root files. ROOT is the standard ATLAS/CERN tool for histogramming and storing data. ROOT enables statistically sound scientific analyses and visualization of large amounts of data: today, more than one exabyte (1,000,000,000 gigabyte) are stored in ROOT files.

Standard Matrix Method

The full workflow for data analysis using the Matrix Method is outlined in this chapter. It will also identify the method of visualizing the effects of the analysis. A concrete example of a tool which is used to illustrate the complexity of the existing software provided in this chapter.

3.1 Installation and set up

Until recently, in order to use any method for data analysis, it was necessary to have the whole FakeBkgTool framework installed. This approach was criticised as storing the entire framework took up an enormous amount of memory and the installation period was very time consuming. Therefore, recently at one of the meetings of the ATLAS team, which develops and maintains this framework, a great emphasis was placed on the sole purpose of finding a solution. The Standard Matrix Method was put forward due to the fact that this tool can run code in standalone mode. This tool contains the minimal set of FakeBkgTool packages needed to run, in sparse checkout mode. The project has the following directories [17]:

- **run** – directory containing a file (input.txt) listing the root files with data sets for further analysis. Also, at the end of the analysis, files with the analysis results will be saved in this directory.
- **source** – contains C++ code for using the Matrix Method, which is part of the FakeBkgTool framework.
- **build** – directory that contains all the necessary files and the Makefile for building the project. To build the project, one needs to run the command

```
cmake ../source
```

3. STANDARD MATRIX METHOD

- **util** – contains a script “run.sh” that starts data analysis. To run the script, one needs to run the command

```
./run.sh ./run/input.txt true
```

In the case of data analysis from Monte Carlo, one writes ‘true’ as third argument, otherwise ‘false’.

- **data** – contains a file with efficiency for muons. It also contains an XML table with the characteristics of input leptons. Each table block has the following format:

```
<ROOT source={file with efficiency path}>
  <{lepton name} type={real/fake efficiency}
  input=\central value" stat={stat}>
  <TH1 X={momentum} > mu_pt_eta_ge2j1b </TH1>
  </{lepton name}>
</ROOT>
```

Before running the script, one needs to run the commands

```
setupATLAS
lsetup "root 6.14.04-x86_64-slc6-gcc62-opt"
```

to use aliases (command abbreviation) of the ATLAS and root commands.

3.2 Structure

All the tools in the FakeBkgTool framework consist of the same structure, input and output data format. They contain a constructor and three functions: initialize, execute, finalize. Each method is described in Table 3.1

Constructor	Initialization of all FakeBkgTool variables, the declaration of properties, the instruction of all the pointers, etc.
Initialize	Histogram and trees for output are created.
Execute	Here FakeBkgTool framework work with each individual event. Incoming data in the form of variables, analyze them and then filled the histograms.
Finalize	All post-processing actions are in this function. At this stage, framework analyzes all the events, fills in the histograms and records them in files if necessary.

Table 3.1: FakeBkgTool tool structure

3.3 Result visualization

After the end of the script, root files are produced with the analysis result in the ‘run’ directory. At the next point, to simplify working with them it was important to illustrate this data. For this function, ATLAS utilizes the TRExFitter tool.

3.3.1 TRExFitter

TRExFitter is a tool to create graphs or charts to allow the researcher to statistically display their results. The idea behind it is to build upon existing code and macros and link them with a series of new classes in order to provide additional convenient features [18].

The framework has the following features:

- read efficiencies
- read input histograms
- read input ntuples
- calculate exclusion limit
- calculate significance
- draw pre-fit plots

3. STANDARD MATRIX METHOD

- and many others informative plots

Input:

- histograms or ntuples
- configuration file

Output:

- Pre-fit and post-fit data/MC plots and tables
- Various plots for checks: pull plot, correlation matrix, systematics before and after smoothing etc.

3.3.2 TRexFitter configuration file

Based on the configuration file, the TRexFitter framework will produce plots.

Parametres:

- **NtuplePaths** – path to the inputs to use (in our case, the path to root files in the run directory)
- **MCweight** – weights to apply to Monte Carlo data
- **Selection** – pre-selection to apply
- **Fit** – type of fit to perform
- **Limit** – settings to obtain a limit
- **MCweight** – weights to apply to Monte Carlo data
- **Region** – These blocks specify which regions enter the fit. Note that while there is only one instances of all the previous blocks, there can be multiple regions, and each has a unique name
- **Type** – can be CONTROL / SIGNAL / VALIDATION.
- **Sample** – Just as for regions, there can be many samples, which will be used to build the fit model.
- **SampleType1** – One has one sample for data with the corresponding type, one SIGNAL and a few BACKGROUND type samples. All of these will enter in every region, unless we specify otherwise.
- **NtupleFile** – name of the .root file for visualization.
- **Systematic** – the blocks here specify systematic effects to be considered in the fit. There can also be many, each with a unique name.

- **Samples** – specifies the Sample as given systematic acts on.
- **Smoothing** – applies smoothing to the shapes of the systematic effects via this option.

3.3.3 Result

All plots, graphs and charts produced by TRExFitter for the Matrix Method are in the `ttWChecks_MM` folder. In the case of Monte Carlo, in the `ttWChecks_MM.mc_comporasion` folder. Folders contains the following content:

- **Plots** – contains one plot showing data and MC for each defined region, as well as summary plots.
- **Tables** – contains various tables in text or .tex format, showing for example the yields per sample and per region.
- **PieChart.png** and **SignalRegions.png** – show background composition and fraction of signal in the defined regions.

3. STANDARD MATRIX METHOD

An example of a TTBar plot showing the effect of the ttbar shower systematic in the signal region is show in Figure 3.1.

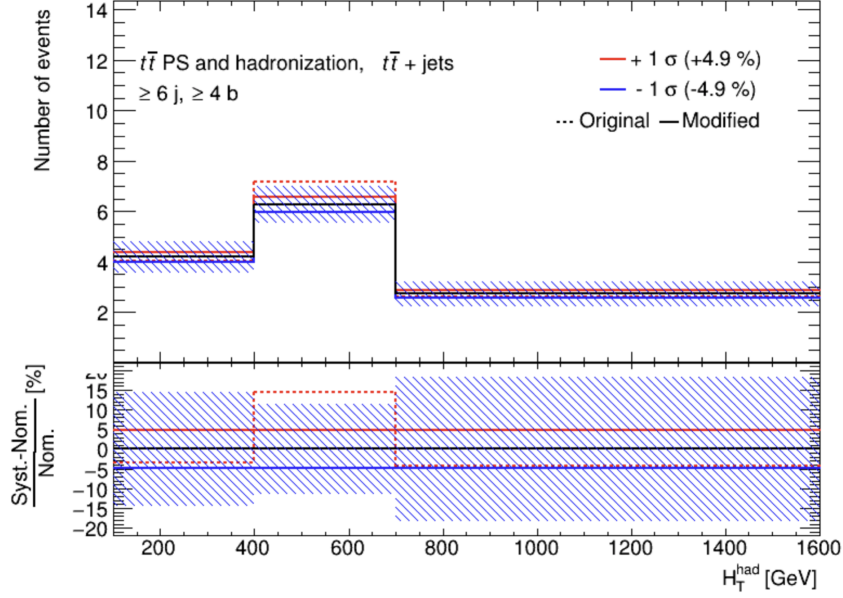


Figure 3.1: The dotted line represents the actual form of the events found in the used sample. After smoothing, the strong red line indicates the form of the systematic uncertainty. The blue line shows the symmetrical effect of moving the system in the opposite direction. The blue hatched area is the statistical uncertainty.

Framework evaluation

This chapter will evaluate the main frameworks expressed throughout this thesis: GFW2 and FakeBkgTool.

4.1 Usability

There are 3 different ways to use frameworks: locally, lxplus (remote server) and on the Grid. Each method differs in the parameters used at running and in the initial setup. The initial setup is the sequential launch of commands that allow researchers to use ATLAS tools. The usability is evaluated based on the launch of the tool themselves, because there is no difficulty to set up ATLAS.

4.1.1 Group framework 2

The setup procedure is simple. First one needs to clone the package and then source the setup script for the first time. If one wants to submit jobs to the grid later on one needs to configure the proxy. To start the framework, one needs to run the script. To run the script, one must pass next parameters: list of input root files, name of the output root file and sum of the events weights used for the normalization. In the case of launching on the Grid, one needs to additionally pass the user id and submission id and do not need to pass the sum of events. A simple configuration and a small number of parameters make the framework usage as simple as possible. Despite this, when using it, one needs to periodically change the source code. Typical cases which require changes in the GFW2 framework are [19]:

- the change of tight lepton or jet selections
- missing variables already stored in GFW1 ntuples
- recalculation of variables based on information from GFW1 ntuples

This makes using the framework not so convenient and easy, because changing the source code can pose difficulties for a scientist who is not directly involved in the GFW2 development.

4.1.2 FakeBkgTool

A full description of using the FakeBkgTool framework is outlined in the previous chapter, Standard Matrix Method. The usage is identical for all other methods of the framework, with the exception that the Fake Factor Method does not need a file with efficiencies. The problem is that in addition to this, it is necessary to change the source code of the framework for each specific case. As described above, each method has the same structure and basic functions: constructor, initialize, execute, finalize. Every time one uses the FakeBkgTool, one needs to change the initialize function, which is responsible for where the result will be saved. In addition, during sessions with ATLAS scientists, it was noted that it is often necessary to change the source code. As described in the previous chapter, the constructor in the FakeBkgTool framework methods is used to initialize variables, properties, etc. The constructor contains the following properties that need to be changed:

- `inputFiles` – location of input files with efficiencies(in case of using Fake Factor Method, this property is not used).
- `m_convertWhenMissing` – bool where set 'true' mean to compute fake factors from fake efficiencies only if the latter are provided in the config files.
- `m_tightDecoNameAndType` – name (and type) of the decoration used to indicate whether leptons pass the 'tight' requirements or not.
- `m_progressFileName` – name of ntuple file with results from a subset of the data.

Frequent changes to the source code coupled with insufficient documentation makes the use of the framework difficult and confusing.

4.2 Performance

ATLAS frameworks are huge and powerful computing systems. Each of these systems requires high performance for reasonable runtime.

4.2.1 Group framework 2

The GFW2 is written in Python. Python is a high-level general-purpose programming language focused on improving developer productivity and code

readability. A broad number of useful functions are included in the standard library for dealing with arrays and for performing data structure operations with a minimum amount of written code. These advantages make Python an ideal choice for the ATLAS framework. In its arsenal, Python also has such libraries as Pandas and Numpy. These libraries are used in Data Science and Machine Learning to efficiently work with arrays. ATLAS does not use them at the moment, but the code written in GFW2 makes it easy to incorporate the data from the library and thus boost the performance of algorithms that convert the data arrays to ntuples[19].

4.2.2 FakeBkgTool

The framework where computation time plays an important role is the FakeBkgTool. Since this framework is used for data analysis and it uses broad complexities, it is necessary to have the maximum speed of operations. The FakeBkgTool framework is written in C++. C++ is a general-purpose compiled, statically typed programming language. C++ is the same C language, only with OOP (object-oriented programming) support and many libraries. This implies that, compared to higher-level languages, it is possible to work with low-level memory as well as faster code execution. This maximizes the productivity in data analysis. However, the C++ language has its drawbacks, such as a high probability of errors, because it is necessary to control memory and the compiler will not be able to tell if the code has errors and may crash. At the moment, the FakeBkgTool framework rarely undergoes any global changes, so using the C++ language is the best solution and makes the framework work as efficient as possible.

4.2.3 Testing performance

Since the FakeBkgTool framework occupies the main difficulty in working in terms of time, in this section the performance of this particular framework will be tested. Benchmarks are one way to test a performance. Benchmarks provide an opportunity to obtain information about the number of cycles and the complexity of the algorithm, in addition to the time and memory consumption. While working on a project, testing with benchmarks seems impossible due to the lack of resources to run projects locally. The next way to test software is to measure CPU time. The performance of the Standard Matrix Method will be measured on a 505 MB data sets.

1. For testing code located in the "util" project directory in the file `ttW_applyMM.origcxx` will be used, because all the mathematical operations for data analysis are used here.
2. The second step is to import the library "ctime":

4. FRAMEWORK EVALUATION

```
#include <ctime>
```

3. The next step is to wrap the code that reads data from data sets and the code that performs calculations with the following commands:

```
std::clock_t c_start = std::clock();  
// code  
std::clock_t c_end = std::clock();
```

4. At the end of the program, one needs to measure the result in seconds and displays it on the screen:

```
long_double time_elapsed_ms = 1000.0 * (c_end-c_start) / CLOCKS_PER_SEC;  
std::cout << "CPU time used: " << time_elapsed_ms / 1000.0 << " s\n";
```

5. After finishing the set up, one needs to run the script to analyze the data from the data sets
6. The obtained results are:

```
CPU time used: 566 s - time taken to read data  
CPU time used: 10194 s - time spent on computation
```

This indicates that most off the time is used for actual computations. Further studies would be needed to determine the CPU time use in individual functions.

4.3 Flexibility

In this case, it is worth evaluating how the frameworks are suitable for 2 cases: single top-Higgs and axion-like-particle, because the top-top-Higgs case is analyzed by default.

No changes are required from GFW2 to cover these cases. It is enough to start the framework and generate ntuples. For single top-Higgs case, all changes must be made in the FakeBkgTool framework. Not much change needs to be done because single top-Higgs also needs to estimate non-prompt and fake lepton backgrounds. Using the Standard Matrix Method as an example, one needs to make the following updates:

- `ttWapplyMMorig.cxx`. For the single top-Higgs case, the "selection cuts" expression must be changed.
- `efficiencies.xml`. In the file with efficiencies, one needs to change the selection type depend on the case. For the Fake Factor Method, nothing needs to be changed, because this method does not use efficiencies.

In the case of axion-like-particle, not everything is so trivial. Unlike the top-top-Higgs and single top-Higgs, photons are analyzed in axion-like-particle. FakeBkgTool can be adapted for this case, but requires a significant rewriting of the code and it is easier to create a new framework that specializes in axion-like-particle.

Concluding that the framework is flexible enough for small changes, but they are not ready for more global updates. It is also worth noting that the source code must be changed for each case, which is bad practice.

4.4 Data structures

The type of data structures that were used in the software plays an important role, because the possibility to extend the code depends on it. Data structures are also selected for a specific task in order to achieve the best performance.

4.4.1 Group framework 2

GFW2 does not use complex data structures and uses only arrays and standard data types such as bool, int etc. The purpose of this framework is to group data into analysis ntuples. ATLAS collaboration members use this data structure to analyze data.

A tuple is a fixed length ordered set. One example of a data set that GFW2 handles looks like this:

```
mc16_13TeV.364174.Sherpa_221_NNPDF30NNLO_Wenu_MAXHTPTV70_\
140_CFilterBVeto.deriv.DAOD_HIGG8D1.e5340_s3126_r9364_p4133
```

This data set contains complete information about the recorded event registered in the ATLAS detector from LHC proton collision. For example mc16_13TeV means simulations protons of 13 TeV collision energy.

To keep information about events, the ATLAS has 2 types of data structures: ntuples and full root trees, which are used for storing C++ objects. They use the same basic format, however, the ntuple is simpler, therefore, when using root trees, the processing time of the input data in the FakeBkgTool can increase. The ntuple is tabular where each event consists of a fixed length row of data. Ntuple allow mixed data types, where each element could be an integer, float, double, or a fixed length array. This makes the given data structure optimal for the required task [20].

4.4.2 FakeBkgTool

The FakeBkgTool framework is written in C++, which supports the use of classes and structures, which makes it possible to use more complex data structures to increase the efficiency and readability of the code. The ATLAS

4. FRAMEWORK EVALUATION

developers have taken advantage of this opportunity and sufficiently represent the data in a more understandable form. One example is a class that contains information about an event, which is shown in Figure 4.1

```
class EventInfo {
public:
    static EventInfo* Instance();           ///<> get singleton instance
    void clear();                          ///<> clear all data members
    const std::string dump() const;         ///<> dump all data members to string

    unsigned int runNumber() const;         ///<> get run number
    void setRunNumber( unsigned int value ); ///<> set run number

    unsigned int eventNumber() const;       ///<> get event number
    void setEventNumber( unsigned int value ); ///<> set event number

    unsigned int eventType() const;         ///<> get LVL1 event type
    void setEventType( unsigned int type );  ///<> set LVL1 event type

    unsigned int triggerType() const;       ///<> get LVL1 trigger type
    void setTriggerType( unsigned int type ); ///<> set LVL1 trigger type

    unsigned int extendedLID() const;       ///<> get extended LVL1 ID
    void setExtendedLID( unsigned int lid ); ///<> set extended LVL1 ID

    unsigned int bcid() const;              ///<> get bunch crossing ID
    void setBCID( unsigned int bcid );      ///<> set bunch crossing ID

    unsigned int time() const;              ///<> get time in sec
    void setTime( unsigned int time );      ///<> set time in sec

    unsigned int timeNs() const;            ///<> get time offset in ns
    void setTimeNs( unsigned int time );    ///<> set time offset in ns

protected:
    /// default constructor
    EventInfo();

private:
    static std::auto_ptr<EventInfo> m_instance;

    unsigned int m_runNumber;              ///<> run number
    unsigned int m_eventNumber;            ///<> event number
    unsigned int m_eventType;              ///<> LVL1 event type
    unsigned int m_triggerType;            ///<> LVL1 trigger type
    unsigned int m_lid;                    ///<> extended LVL1 ID
    unsigned int m_bcid;                   ///<> bunch crossing ID
    unsigned int m_time;                   ///<> universal time in sec
    unsigned int m_timeOffset;             ///<> time offset in ns
}; // class EventInfo
} // namespace LVL1CTP
```

Figure 4.1: Class that contains information about an event

At the same time, there are places where the use of data structures can be improved. As can be seen in the Figure 4.2, instead of using an array of structures that would act as a DTO define and store information about the name and type, this code uses a separate array for names and for types. These parameters have the same index in array, are related to each other and they could be combined into some kind of data structure. Also, there are many code sections where variable names are not self-explanatory, therefore this makes the code difficult for understand for developers who has recently joined the project.

```
/// default constructor of a TH1 tool
HistogramUtils::HistogramToolTH1::HistogramToolTH1(const std::string& type, const std::string& name, const IInterface* parent) :
    AthHistogramTool(type,name,parent),
    m_parserX(0)
#ifdef XAOD_ANALYSIS
    , m_trigDecisionTool("Trig::TrigDecisionTool/TrigDecisionTool")
#endif
{
    declareInterface<IAthHistogramTool>( this );

    DECLARE_HIST_PROPERTIES();
    DECLARE_AXIS_PROPERTIES(X);
}
```

Figure 4.2: Illogical use of data structures

New outlined framework

This chapter describes a new framework for analyzing large data sets recorded by the ATLAS detector. This framework partially solves the problems that currently exist in the current framework, and will highlight its strengths.

5.1 Requirements

Before starting to develop a framework, there is need to define the requirements. Requirements will be divided into 2 types: requirements that apply to all frameworks in general and requirements for an analysis framework for processing large data sets.

Since the framework will be used in the future when developing other software, the first requirement will be a good architecture. The difference between a framework and a library is that a library can be used in a software product simply as a set of subroutines of similar functionality, without affecting the architecture of the software product and without imposing any restrictions. At the same time, the framework dictates the rules for building the architecture of the application, setting the default behavior at the initial stage of development. The next requirement is a full test coverage. If the work of the framework is unstable, then this will cause problems for all the software that uses it, therefore it is necessary to achieve the maximum possible stable operation and this can be done by writing tests. The next necessary requirement is the documentation of the code. A large number of people work with the framework who did not take part in development, so it is necessary to convey information on how to use the framework. This applies to both the documentation of the framework in general and the compilation of the API documentation.

In the case when one considers the framework as a tool for analyzing a large amount of data, then, first of all, the speed of running the code will be very important, since data in such an amount can be processed for hours and days. Also, there are several projects of data analysis (top-top-Higgs, single

top-Higgs, axion-like-particle), therefore, it is necessary that the framework be universal for all projects and the configuration would be as simple and understandable as possible. Also, given that the configuration often changes, it is necessary, if possible, to automate the processes for changing it to optimize the development time. As a result, the following requirements have been highlighted:

- Documentation
- Stability
- Good architecture
- High performance
- Configurability
- Automatization

This chapter solves all requirements except documentation. For a more detailed analysis of the documentation, a separate chapter is further highlighted. It will analyze the documentation from the developer and user side. Improved tracking of development progress will also be suggested.

5.2 Stability

The stability of the software is increased with the help of various types of testing. Every software should be tested, because bugs can greatly delay development and block scientists who will use this framework. For a data analysis framework, the following types of testing can be distinguished:

- Unit testing
- Performance testing
- Integration testing

5.2.1 Unit testing

Unit testing - type of tests in programming that allows to check for correctness individual modules of the source code of a program, sets of one or more program modules along with the corresponding control data, procedures for use and processing. With the right software development process, testing comes before the implementation of the functionality. Also, during unit testing, it is necessary to test individual small functions, therefore, in order to fully cover these requirements, one needs to fulfill the requirements, which will be described in the section Architecture.

5.2.2 Performance testing

In the classical sense, this type of testing is used to check how the software withstands the load under certain conditions. For example, when testing a site, performance is checked by emulating user requests to the tested site at minimum, average, and maximum values. In the case of testing a data analysis framework, one needs to test productivity for different amounts of data and monitor so that at least it does not drop with new implementations. To test performance, one can use Google Benchmark, which was discussed in the chapter Framework evaluation in the section Performance.

5.2.3 Integration testing

Integration testing is one of the phases of software testing, in which individual software modules are combined and tested in a group. In our case, the models are frameworks (GFW1, GFW2, FakeBkgTool ...), and with each new implementation, especially if it concerns processing incoming or outgoing values, one needs to check that the framework remains compatible with the rest of the frameworks.

5.3 Architecture

This chapter describes the current state of the framework and suggests a new and improved architecture. The architecture described in this chapter is one of the suggestions for improving the discussed framework.

5.3.1 Current state

At the moment, the framework contains several significant flaws that affect the stability of the framework. It happened more than once that due to a lack of tests, a critical error occurred in the development of the framework, which was noticed only several weeks later. The main disadvantages of the current state:

- No tests
- Instability
- Difficulties in support
- Difficulty reading and understanding the code

To solve current problems, it is proposed to use a microservice architecture.

5.3.2 Microservice architecture

Microservice architecture is used when the individual tasks of the application can be easily divided into small functions - independent services. These services can be written in different programming languages, since they communicate with each other using the REST API. Thanks to this, it will be possible to combine GFW1, GFW2 and FakeBkgTool frameworks that are written in different programming languages. Most often, microservices are launched in so-called containers. These containers are available over the network to other microservices and applications, and the orchestration system manages them all: examples are Kubernetes, Docker Swarm, etc. The microservices architecture makes it easy to scale applications. To implement a new function, it is enough to write a new service. If the function is no longer needed, the microservice can be disabled. Each microservice is a separate project, so work on them is easy to distribute between development teams.

5.3.3 New framework architecture

The framework will have a microservices-based architecture. The main services will be a service for receiving and processing data, which will later be transferred to the service for data analysis. Data analysis can also be split into several services. This decision can be made by the head of developers based on a more detailed analysis of parts of the code and identification of its interdependent parts. A service will also be introduced to interact and save information to the database, which, in its current state, has been replaced by manual saving of the set date to the data folder. The framework will be based on the OOP paradigm, since the current framework uses this paradigm and this will simplify the portability of the code, which will speed up the introduction of the new architecture into everyday life. The architecture and each microservice will be thoroughly tested with unit tests and integration tests in order to stabilize and determine the work of the framework [21].

5.4 Performance

In terms of performance, compared to the current framework, everything could be left unchanged, because, as it was found in the chapter "Framework evaluation", the framework has optimal performance thanks to the tool that was chosen for writing it and the lack of unnecessary complexity of algorithms. It is only tracking, that needs to be added, so that the performance does not decrease, which can be done thanks to the test described in the previous section.

5.5 Configurability

The future new framework should be configurable enough to increase its flexibility and there is no need to change the source code for each case. This can be achieved by using a configuration file. This is currently implemented in the TRexFitter framework. The config file will contains all the properties that the framework contains and their values. An example of how a config file can be put into use:

1. This process is described using the Standard Matrix Method as an example. The structure of the method is described in the chapter Standard Matrix Method.
2. In the project, open the file "ttW_applyMM.cxx", which is located in the "utils" directory. Instead of hard-coded initialization of properties, reading values from the console, add:

```
string inputfile;
cin >> inputfile;

bool m_convertWhenMissing;
cin >> m_convertWhenMissing;

string m_tightDecoNameAndType;
cin >> m_tightDecoNameAndType;

string m_progressFileName;
cin >> m_progressFileName;
```

3. After that, it is necessary to create a configuration file in which all properties will be stored as "key - value". It is created in the "input" directory.

```
inputfile - /afs/cern.ch/user/a/achomont/public/input_120320.txt
m_convertWhenMissing - false
m_tightDecoNameAndType - tight
m_progressFileName - result_120320.root
```

4. Next, one needs to parse this configuration file. The values that will be received will be used when running the "ttW_applyMM.cxx" file.

5. Then, run the script and the Standard Matrix Method will use the properties declared in the configuration file.

5.6 Automatization

In the modern world, everything strives for automation. CERN systems should be no exception. The automation will have 2 parts.

The first part is to automatically test the pushed code into the GitLab repository, which will provide instant response to bugs and failed tests. This can be done using the GitLab CI tool. Each launch requires an initial configuration, which is a series of commands that can be automated. Each command is a pipeline. Pipeline is a set of tasks organized into stages in which one can build, test, package the code, deploy the finished assembly to the cloud service, etc. Since the framework does not need to be deployed anywhere, it will be enough to leave only 3 pipelines: build, run and test. If at least one pipeline does not pass, it will not be possible to add a new change to the main branch of the project.

The second part is to create scripts for deploying each of the framework services, which will be automatically launched when the Docker container with the service source code is launched. This will allow the use of frameworks without the need of acknowledging of additional commands and environments. Docker is an open platform for developing, delivering, and operating applications. Docker is designed to deploy applications faster and run each independently on other systems and infrastructure. Each application is launched in the Docker container, which is a separate environment with its own configuration [22].

5.7 New framework evaluation

5.7.1 Pros

1. **System stability.** The new framework will be tested in detail. Based on this, the system will contain fewer errors than in the current one code, which will lead to greater system stability.
2. **Finding errors faster.** The framework has an architecture built on microservices, which allows developers to find bugs faster than with the current frameworks due to the system connectivity. This will allow to quickly respond to emergency problems and speed up the work.
3. **Possibility to save money.** Support, bug fixes and the addition of new features, thanks to the improved architecture, will be faster than in the current system, which will save developers and institutions resources.
4. **System scalability.** The improved architecture is highly scalable as adding a new service to the system only requires adding a new service (benefits of microservice architecture) resources.
5. **Division of responsibilities.** Thanks to the new architecture, each developer can focus on working on his specific service (a separate project), without having to understand how the complex system works. This will increase the efficiency of work on each individual function and the system as a whole.

5.7.2 Cons

1. **Time-consuming.** Developing a new framework requires operations such as detailed analysis, development and testing. Each action takes time and resources of specialists. It will also be necessary to maintain the current system until the new framework comes into use, which will require even more experts time and may slow down work on the new system.
2. **Requires detailed testing.** A framework of this type requires much resources for detailed testing, since accuracy, error-freeness and stability are important.
3. **High cost.** The framework requires large resources. All resources will cost a certain amount, which can be very large due to the abundance of requirements. There is also additional overhead for passing messages between microservices.

Documentation and development tracking

At the moment, ATLAS has a large amount of undocumented code, as well as software that lacks documentation for use. In addition, the documentation that currently exists is not systematized and it is not located in one specific place to make it easier to find. Part of the documentation is on the ATLAS Wiki pages, part is saved in presentations from meetings, and part has only the ReadMe form in the repositories. In addition, ATLAS does not have a system communication between team members and most issues are resolved via Skype. Part of the work is not tracked and it is difficult to monitor the state of development or any other activity. All of this results in slow development, difficulty of integrating new members and misunderstandings between specialists. This chapter is devoted to how ATLAS can eliminate these problems and improve the work processes. Also in this chapter a solution will be proposed on how to improve communication between ATLAS members.

6.1 Documentation

This section is divided into two subsections. The first subsection describes what needs to be done to document the existing code without spending much time. It will also document the existing software for its use and consolidate the documentation in one location to provide current and potential ATLAS collaboration members with more understandable access and a clear approach to familiarization.

6.1.1 API documentation

API documentation is a technical content deliverable, containing instructions how to effectively use and integrate with an API. This documentation is carried out in such a way that a tool is selected, which, based on comments in the

code that corresponds to the syntax specific for this tool, creates documentation. Most of the modern solutions provide an HTML file that contains this documentation in the form of a web site, in which there is convenient navigation, a visual representation of the class hierarchy, etc. The most popular tool for documenting C++ code is Doxygen. Doxygen generates documentation from a set of sources and can also be configured to extract program structure from undocumented sources. It is possible to draw up graphs of dependencies of software objects, class diagrams and source codes with hyperlinks. Doxygen is a console program in the spirit of classic Unix. It works like a compiler, parsing source code and generating documentation. The parameters for creating documentation are read from a configuration file in plain text format. Some parts of the ATLAS code already contain comments and it will not take long to change them to fit the Doxygen syntax. Even in the absence of comments, it is possible to generate simple documentation with graphs of dependencies of data structures, graphs of inheritance, etc. Even such simple documentation will simplify the integration of new developers in projects [23].

6.1.2 Software documentation for its installation and application

The software documentation for its installation and application is quite detailed and understandable in ATLAS. It is also worth noting that in each documentation there are contacts and the name of the developers who is responsible for it and in case of ambiguity, there is an opportunity to contact them and clarify all the questions. The main problem is that all the documentation is not concentrated in a certain place and it takes a lot of time to find it. IT firms often have their own Wiki page where one can find answers to frequently asked questions, read or watch a tutorial on any technology that is created or used in this company. ATLAS also has such a page, but it does not contain all the necessary documentation. The solution is to move all documentation to the Wiki page and use the Wiki page as the main portal for storing documentation in the future. Achieving this result will take a certain amount of time, but in the long term this will be compensated for, because there will be no need to spend extra time looking for answers to questions about the use of software.

6.2 Development tracking

Improving development tracking will improve the productivity of ATLAS teams, because in the absence of deadlines and control, development speed is greatly reduced.

It also partially cover gaps that the documentation does not cover, because at any moment the developer can find out when a particular function was

implemented, what it depends on, who was the contributor, etc. This section describes suggestions for improvement.

6.2.1 Jira

Jira is a commercial bug tracking system designed to organize interaction with users, although in some cases it is also used for project management. The main accounting element in the system is the task. The task contains the project name, theme, type, priority, components and content. The task can be extended with additional fields (also new custom fields can be defined), applications (for example - photos, screenshots) or comments. The task can be edited or simply change its status, for example, from “open” to “closed”. Which state transitions are possible is determined through a custom workflow. Any changes to the task are logged. Jira has a large number of configuration options: for each application, a separate task type can be defined with its own workflow, a set of states, one or several views. In addition, with the help of so-called “schemes”, one can define own access rights, behavior and visibility of fields for each individual Jira project, and much more [24].

6.2.2 Improvement of development tracking

ATLAS already uses Jira for its work, but not on all projects and does not do it effectively. Most of the time ATLAS uses this system to create an issue and assign it to the developer. Also, the ATLAS issue has only 3 states: “TODO”, “In development”, “Done”. The rest of Jira’s features are not used. In order to increase efficiency, one can add the following features:

- Add a new state (for example “Test Passed” or “Approved”). This is necessary in order to be sure that the task that should be solved in this ticket has really been solved. When creating a ticket, there is always a creator of this ticket. In this process, he takes on the role of the approver who can close the given ticket. If the ticket does not pass any criteria of the approver, he must add a description of the problem, why the ticket was not accepted in the comments and return it back to the “TODO” state, after which the developer will complete it. This cycle ends after the approver closes the ticket. Thanks to this, one can be sure that all tickets that were closed and removed from the board were resolved and moved development forward.
- Use the “Monitor Problem” function. In addition to the developer and the approver, other members may need to monitor the status of a task, therefore this function will be very useful. When the state of the ticket is changed or a comment is added, everyone who follows the problem would receive an email with a notification about the change in the state of the ticket. And all members interested in this problem would know

its current status and there would be no need to contact the developer every time.

- The next suggestion is to start using related issues. There are several types of related tickets and the most important ones are "depends on ..." or "is a blocker for ...". This should be used to avoid creating blockers for other developers so that development does not slow down.
- The main possible improvement in development tracking is Jira's connectivity and repository management system. ATLAS uses GitLab as the system for managing repositories. GitLab has for example many more features such as wikis, bug trackers, CI / CD pipelines. And also it has a function thanks to which one can configure the relationship between a ticket in Jira and a merge requests and commits in GitLab. In order to implement this system, one must follow the convention that the name of the commit and the name of the branch contains the ID of the ticket in Jira. Then, configure GitLab so that by the ticket ID, GitLab can create a link, which, when clicked, leads to a ticket in Jira.

The integration of Jira into GitLab is the most important suggestion for innovation in development tracking. Thus, in the following example of a test project in ATLAS GitLab will be described.

Purpose: When one commits to git, one registers a task from Jira by the name in the comment, after which two things happen:

- In GitLab, the issue name turns into an active link in Jira.
- In Jira, a comment is added to the task with links to the commit and the user who made it, as well as the mention text itself.

Set up:

1. One needs a GitLab user with administrator rights in each of the projects that will be connected. Integration is configured separately for each project.
2. Open the project in GitLab, go to Settings - Integrations. Scrolling down, one sees "Project services" with a long list of services that can be connected (Figure 6.1).

6.2. Development tracking

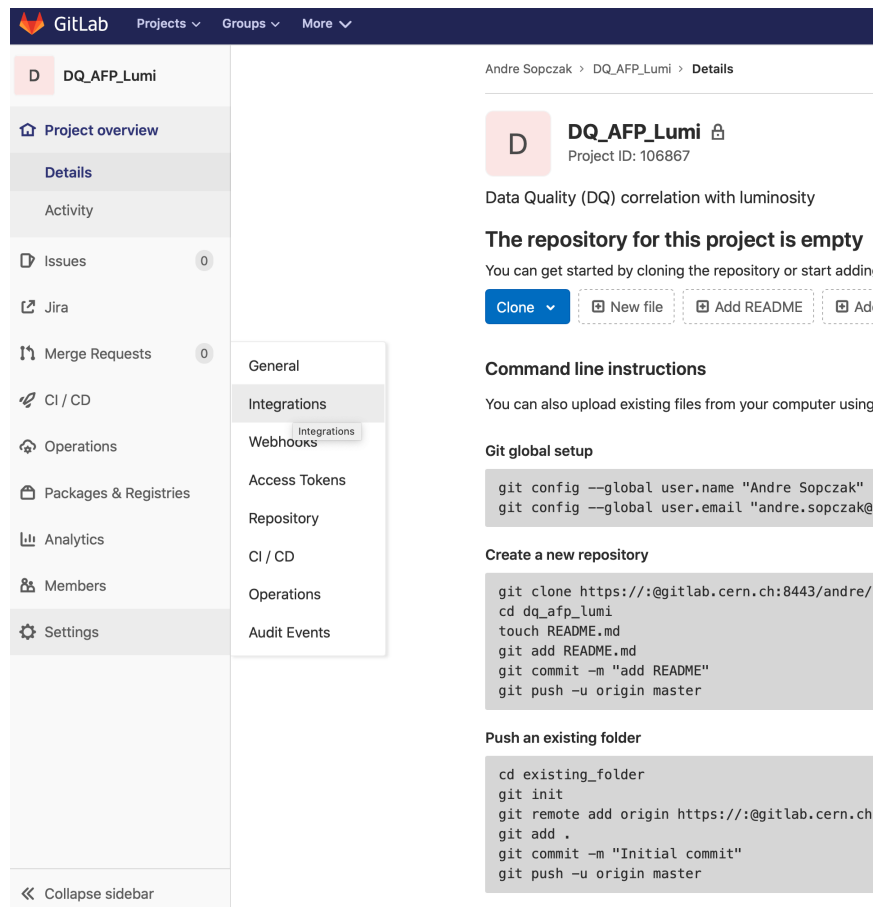


Figure 6.1: Integrations tab in Jira

3. In this list one finds Jira, the form appears (Figure 6.2)

6. DOCUMENTATION AND DEVELOPMENT TRACKING

You need to configure JIRA before enabling this service. For more details read the [JIRA service documentation](#).

Active ☐

Trigger ☒ **Commit**
JIRA comments will be created when an issue gets referenced in a commit.

☒ **Merge request**
JIRA comments will be created when an issue gets referenced in a merge request.

Web URL

JIRA API URL

Username or Email

Password or API token

Transition ID(s)

Figure 6.2: Form for filling in data about Jira

- Put a tick Active to activate the connection.
- Enter ATLAS's Web URL in Jira:

`https://its.cern.ch/jira/secure/Dashboard.jspa`

- The following fields are filled: Username, Email and Password, Token.
- In transition id field one needs to write the id of the transition to the closed state. This id is obtained by API:

`https://its.cern.ch/rest/api/2/issue/ISSUENAME-123/transitions`

where ISSUENAME-123 is the name of some task in the desired state. JSON with an array of transitions will be received, from which one can take the desired id.

- As a result, in GitLab in Settings - Integrations, Jira now has a green indicator (Figure 6.3)

	HipChat	Private group chat and IM	
	Irker (IRC gateway)	Send IRC messages, on update, to a list of recipients through an Irker gateway.	
	JIRA	Jira issue tracker	3 hours ago
	JetBrains TeamCity CI	A continuous integration and build server	
	Kubernetes	Kubernetes / OpenShift integration	
	Mattermost notifications	Receive event notifications in Mattermost	

Figure 6.3: Jira connected

- and the Jira item will appear in the project menu, which leads to the corresponding project in Jira.

Using:

When creating a Merge Request (or creating a commit), one needs to specify the issue id (Figure 6.4).

Title

ATLMCPROD-983 Test MR

Start the title with [Draft:](#) or [WIP:](#) to prevent a merge request that is a work in progress from being merged before it's ready.

Description

Choose a template

Write

Preview

Describe the goal of the changes and what reviewers should be aware of.

Markdown and quick actions are supported

Figure 6.4: Merge request title with issue id

As a result, a comment will be added to the corresponding task (Figure 6.5). And an active link will appear in the GitLab.

6. DOCUMENTATION AND DEVELOPMENT TRACKING

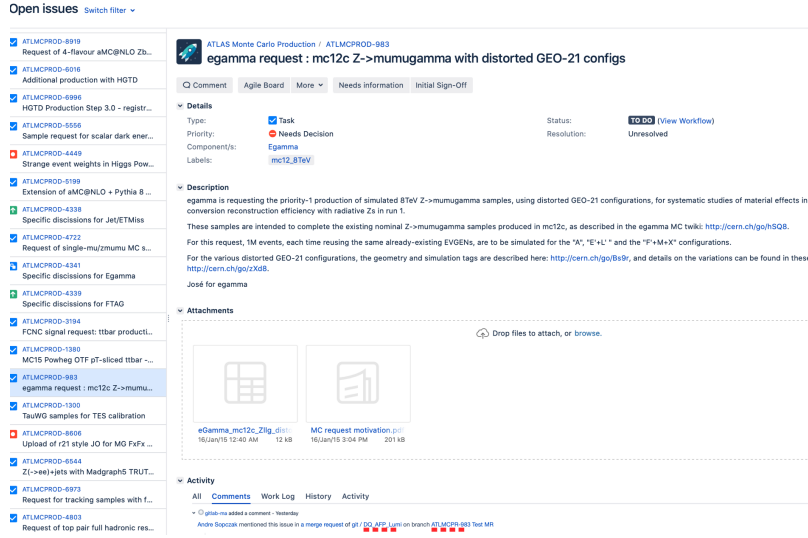


Figure 6.5: Jira page with link to merge request in comments

6.3 Communication

At the moment, ATLAS uses two main tools for communication: email and Skype. Skype has a list of disadvantages, such as:

- Many bugs
- No group video calls on Linux (many ATLAS members use Linux as their main operation system)
- Not a feasible alternative to professional team chat

These disadvantages can be ignored when used in everyday life, but as a mean of communication in a large collaboration like ATLAS, Skype is a quite bad choice. It is suggested to use Microsoft Teams as an alternative.

Microsoft Teams is an enterprise platform that brings chat, appointments, notes and attachments together in a workspace. This platform eliminates the disadvantages that Skype has, and also has the following advantages [25]:

- File storage for work
- It is possible to quickly organize an online conference
- Each participant groups in the course ongoing changes
- Integration of results group work

Conclusion

This thesis addresses improvements in the ATLAS software development. It contains a strategy for documentation and tracking the development progress. Specific suggestions for implementation are supported by practical examples.

The CERN chapter contains basic information about LHC and ATLAS, the projects and their goal.

The Data flow chapter explains the data flow in ATLAS from the moment it is recorded in the detector to final analysis result.

The Standard Matrix Method chapter gives an example of one of the tools of the analysis framework and gives specifics on the workflow and how it is possible to achieve an analysis result.

The Framework Evaluation chapter evaluates frameworks by software engineering criteria. This was done so that the pros and cons of the current frameworks could be highlighted for the next chapter. This chapter also includes the performance testing of the Standard Matrix Method tool.

In the chapter New Framework, the main points that a new framework should have are highlighted. Furthermore, solutions are proposed to improve the workflow. Testing strategy of the new framework is also described in detail. In addition, pros and cons of switching to a new framework from the existing one are described.

In the Documentation and development tracking chapter, a strategy is outlined for documentation as well as improvements in tracking the development progress. One of the suggestions for improving the tracking of development is the integration of Jira with GitLab and detailed instructions are given using the example of integrating a test project from GitLab ATLAS with one of the real boards in Jira.

This thesis serves as a detailed proposal for the implementation of future frameworks. It has been determined that the implementation of the suggestions, as well as ideas for documentation, will improve the workflows in ATLAS, as in particular for new software developers.

□

Bibliography

- [1] CERN. *CERN's mission*. [online], 2020, [cit. 2020-06-10]. Available from: <https://home.cern/about/who-we-are/our-mission>
- [2] CERN. *Higgs Boson*. [online], 2020, [cit. 2020-06-10]. Available from: <https://home.cern/science/physics/higgs-boson>
- [3] Rincon, P. *Large Hadron Collider turns on 'data tap'*. [online], 2020, [cit. 2020-07-21]. Available from: <https://www.bbc.com/news/science-environment-32976838>
- [4] CERN. *Experiments*. [online], 2020, [cit. 2020-06-10]. Available from: <https://home.cern/science/experiments>
- [5] CERN. *LHC*. [online], 2020, [cit. 2020-06-10]. Available from: <https://home.cern/science/accelerators/large-hadron-collider>
- [6] Particle Data Group. *Towards the discovery of the Higgs boson*. [online], 2020, [cit. 2020-09-11]. Available from: <https://pdglive.lbl.gov/Particle.action?node=S126&init=0>
- [7] Rubakov, V. *Towards the discovery of the Higgs boson*. [online], 2012, [cit. 2020-09-11]. Available from: https://elementy.ru/nauchno-populyarnaya_biblioteka/431961
- [8] CERN. *CERN. Software and Computing*. [online], 2020, [cit. 2020-06-10]. Available from: <https://atlas.cern/node?page=10>
- [9] ATLAS Collaboration. *ATLAS. Derivation Framework*. [online], 2020, [cit. 2020-08-21]. Available from: <https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/DerivationFramework>
- [10] CERN. *ATLAS. N-tuple and Event Collection facilities*. [online], 2020, [cit. 2020-08-28]. Available from: http://lhcb-comp.web.cern.ch/Frameworks/Gaudi/Gaudi_v9/GUG/Output/GDG_Ntuple.html

- [11] Gillam, T. P. S.; Lester, C. G. *Improving estimates of the number of ‘fake’ leptons and other mis-reconstructed objects in hadron collider events*. [online], 2020, [cit. 2020-08-28]. Available from: <https://arxiv.org/pdf/1407.5624.pdf>
- [12] ATLAS Collaboration. *Expected electron performance in the ATLAS experiment*. [online], 2011, [cit. 2020-11-12]. Available from: <https://cds.cern.ch/record/1345327>
- [13] ATLAS Collaboration. *Measurements of the electron and muon inclusive cross-sections in proton-proton collisions at $s = 7$ TeV with the ATLAS detector*. [online], 2012, [cit. 2020-11-12]. Available from: <https://cds.cern.ch/record/1379865>
- [14] Mashishi, L. *$W + jets$ background estimation using the fake factor method for the $H \rightarrow Sh \rightarrow e^\pm \mu^\mp + \geq 2jets$ search with the ATLAS detector*. [online], 2020, [cit. 2020-08-28]. Available from: https://indico.cern.ch/event/779076/contributions/3275609/attachments/1788448/2912704/Fake_factor_estimation_using__Z__jets_control_sample__Dilepton_Analysis__3.pdf
- [15] CERN. *Search for charged lepton-flavour violation in top-quark decays at the LHC with the ATLAS detector*. [online], 2018, [cit. 2020-09-30]. Available from: <https://cds.cern.ch/record/2638305/files/ATLAS-CONF-2018-044.pdf>
- [16] Bjørnstad, J. F. *On the Generalization of the Likelihood Function and the Likelihood Principle*. [online], 1996, [cit. 2020-09-29]. Available from: <https://www.jstor.org/stable/2291674?seq=1>
- [17] CERN. *Documentation for ttWMatrixMethod*. [online], 2020, [cit. 2020-09-01]. Available from: <https://gitlab.cern.ch/nbruscin/ttWMatrixMethod>
- [18] CERN. *Documentation for TRExFitter*. [online], 2020, [cit. 2020-09-15]. Available from: <https://trexfitter-docs.web.cern.ch/trexfitter-docs/>
- [19] CERN. *Documentation for ttHMultiGFW2*. [online], 2020, [cit. 2020-09-15]. Available from: <https://gitlab.cern.ch/atlasHTop/ttHMultiGFW2>
- [20] Stanford. *Ntuples vs. TTrees*. [online], 2005, [cit. 2020-09-15]. Available from: <https://www.slac.stanford.edu/exp/glast/wb/test/pages/rootPages/ntuplesVsTTrees.htm>

- [21] 1Cloud. *Briefly about types of software architectures*. [online], 2018, [cit. 2020-09-15]. Available from: <https://habr.com/ru/company/1cloud/blog/424911/>
- [22] Habr. *Understanding Docker*. [online], 2015, [cit. 2020-09-15]. Available from: <https://habr.com/ru/post/253877/>
- [23] Doxygen. *Doxygen*. [online], 2020, [cit. 2020-12-15]. Available from: <https://www.doxygen.nl/index.html>
- [24] Atlassian. *Jira*. [online], 2020, [cit. 2020-12-15]. Available from: <https://www.atlassian.com/ru/software/jira>
- [25] Microsoft. *Microsoft Teams*. [online], 2020, [cit. 2020-12-15]. Available from: <https://www.microsoft.com/en-us/microsoft-365/microsoft-teams/group-chat-software>
- [26] ATLAS Collaboration. *ATLAS. HSG8DerivationFramework*. [online], 2020, [cit. 2020-08-20]. Available from: <https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/HSG8DerivationFramework>
- [27] CERN. *Documentation for ttHMultiGFW1*. [online], 2020, [cit. 2020-09-15]. Available from: <https://gitlab.cern.ch/atlasHTop/ttHMultiGFW1>

Acronyms

CERN European Organization for Nuclear Research

GW1 Group Framework 1

GW2 Group Framework 2

MC Monte Carlo

DTO Data transfer object

MM Matrix Method

LHC Large Hadron Collider

XML eXtensible Markup Language

GB Gigabyte

MB Megabyte

PB Petabyte

FF Face Factor

ATLAS A Toroidal LHC Apparatus

CMS Compact Muon Solenoid

OOP Object Oriented Programming

CI Continious Integration

CD Continious Delivery

ID Identifier

JSON JavaScript Object Notation

A. ACRONYMS

API Application Programming Interface

IT Information Technology