

1. Block 4.2: CS6514: Software Architecture Design Portfolio

Merlin++

Art Ó Liathain

September 2025

2. Abstract

Merlin++ is a C++ charged-particle tracking library developed for the simulation and analysis of complex beam dynamics within high energy particle accelerators. Accurate simulation and analysis of particle dynamics is an essential part of the design of new particle accelerators, and for the optimization of existing ones. Merlin++ is a feature-full library with focus on long-term tracking studies. A user may simulate distributions of protons or electrons in either single particle or sliced macro-particle bunches. The tracking code includes both straight and curvilinear coordinate systems allowing for the simulation of either linear or circular accelerator lattice designs, and uses a fast and accurate explicit symplectic integrator. Physics processes for common design studies have been implemented, including RF cavity acceleration, synchrotron radiation damping, on-line physical aperture checks and collimation, proton scattering, wakefield simulation, and spin-tracking. Merlin++ was written using C++ object orientated design practices and has been optimized for speed using multicore processors. This article presents an account of the program, including its functionality and guidance for use.

3. Table of Contents

Contents

1. Block 4.2: CS6514: Software Architecture Design Portfolio	1
2. Abstract	2
3. Table of Contents	3
4. Tech Stack	4
5. Domain Model	5
6. Utility Tree	6
7. Use Case Diagram	7
8. 4 + 1 Diagram	8
8.1. Logical View	8
8.2. Physical View	8
8.3. Development View	8
8.4. Process View	9
9. Codescene	9
10. DV8	11
11. SonarQube	12
12. Understand	13

List of Figures

Figure 1 Tech Stack	4
Figure 2 Utility Tree	6
Figure 3 Use Case diagram	7
Figure 4 Codescene coupling graph	9
Figure 5 Codescene maintainability circle	10
Figure 6 Dv8 Dev tool analysis	11
Figure 7 Dv8 high level analysis	11
Figure 8 Technical debt cost	12
Figure 9 Duplicated code analysis	12
Figure 10 Function complexity understand	13
Figure 11 Coding language breakdown	13
Figure 12 Understand code dependency graph	13
Figure 13	14

4. Tech Stack

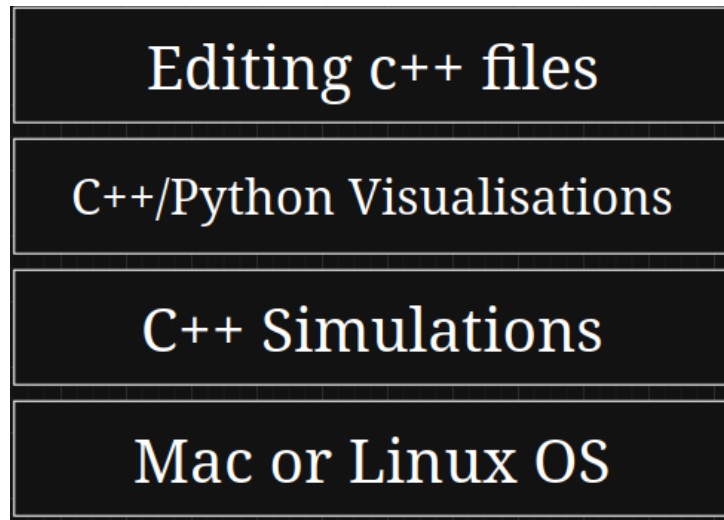
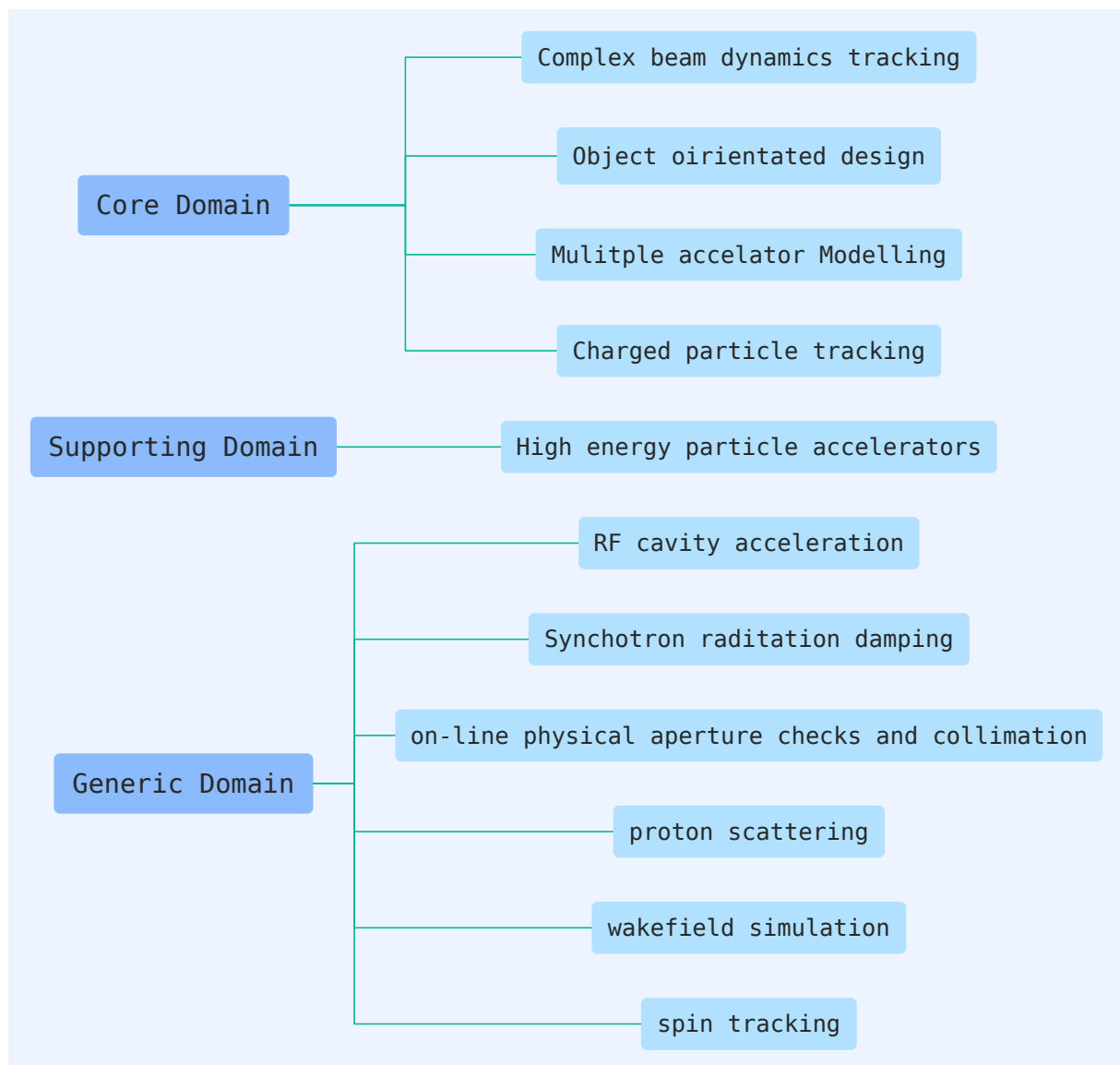


Figure 1: Tech Stack

The design philosophy of OOP that Merlin strives towards is seen by the tech stack. C++ was a pragmatic choice to allow for a high performance language that is grounded in OOP to be the core language. This performance was also pivotal as the simulations need to be performant. An issue in the codebase comes from the loose python integration, this is because its used for the tests even though cmake is being used and python is used lightly for the visualisations. This lack of cohesion adds to the complexity of adding to tests for no good reason. The entrypoint of editing a file with classes makes sense in the sense of ease of use for developers but this entrypoint makes the process much less user friendly and adds to the difficulty of non-technicals using the program.

5. Domain Model



The Merlin project is a tool focused on simulating complex particle collisions and acceleration. It incorporates a wealth of well known and studied formulas which contribute to the generic domain of the application. The feeds into the core use of the application of beam dynamics and particle tracking in differenct accelerator simulations. This flexibility of choice and algorithm application allows for Merlin to be a well suited tool for its use case. Arguably in the core domain, the application of OOP has a place. This is due to the fact that the code base is architected in a better style than other academic projects where architecture has been considered for the future maintainability and continuation of the project.

6. Utility Tree

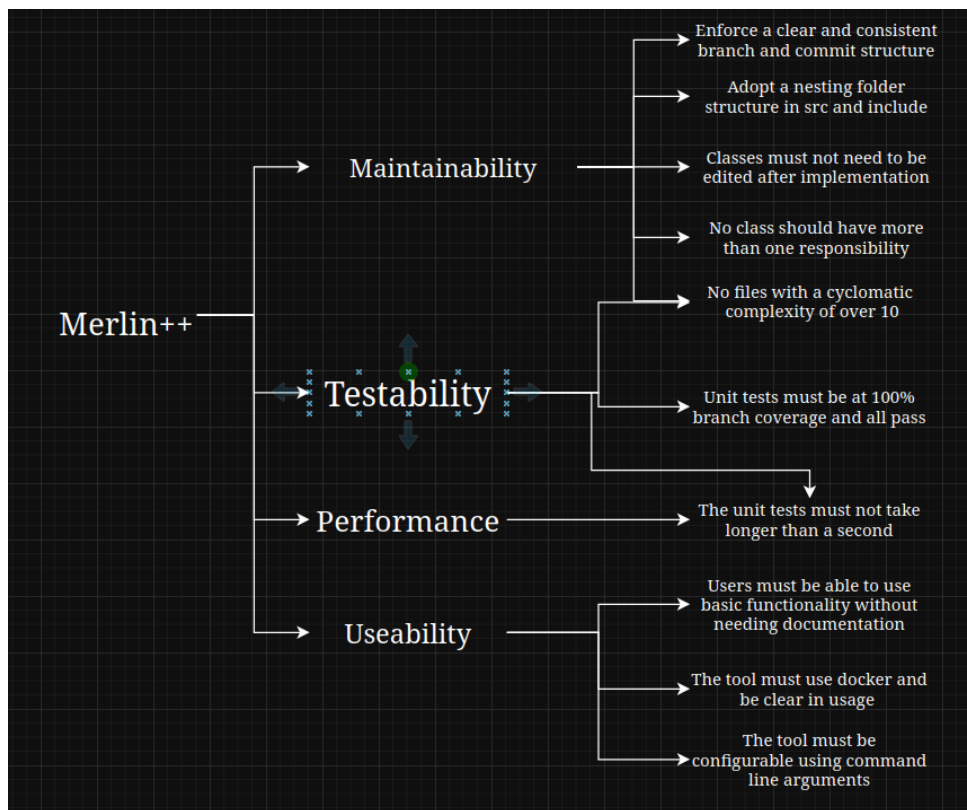


Figure 2: Utility Tree

This utility tree combines both what is present in the codebase and what it should aspire for with concrete quality requirements. These requirements are a baseline for a strong code project, things such as commit and file structure are a must to encourage collaboration and modularisation. Useability is important to allow for a wide range of user testing and user usage. The more useable the project is the better it is on both ends allowing developers to focus on more core problems and allows users to focus on using the tool rather than fighting it. Things like a clear configuration embedded in code as well as containerisation to allow for consistent builds are a must.

7. Use Case Diagram

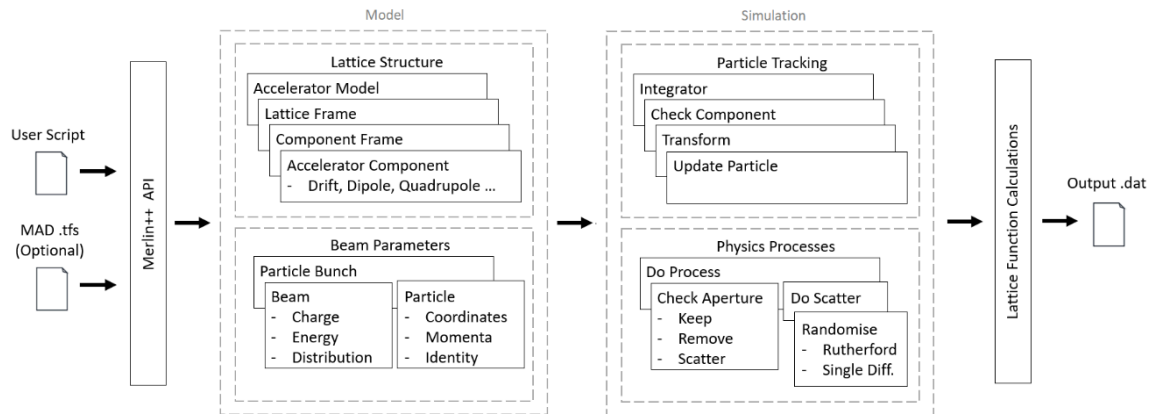
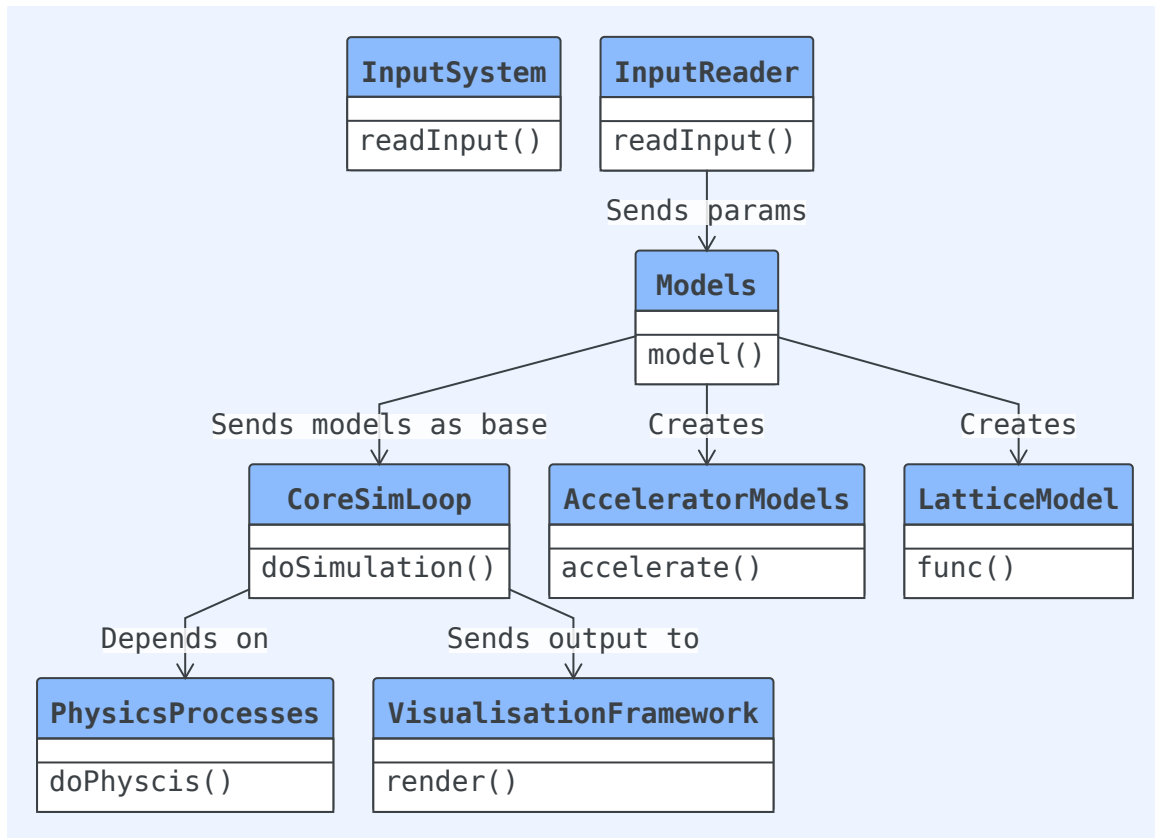


Figure 3: Use Case diagram

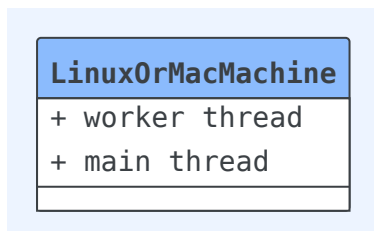
This use case diagram shows the flow directly from script creation to the output. This shows how the script once inputted into Merlin first models the structure of the simulations. This is then passed to the simulation section which then simulates according to the parameters outputting an output.dat. This is a well made system but the main gripe I have is with the entry point being a user made script, within my understanding even a config file would be preferable to make the system simpler to understand and remove the need for in depth tutorials for basic use.

8. 4 + 1 Diagram

8.1. Logical View



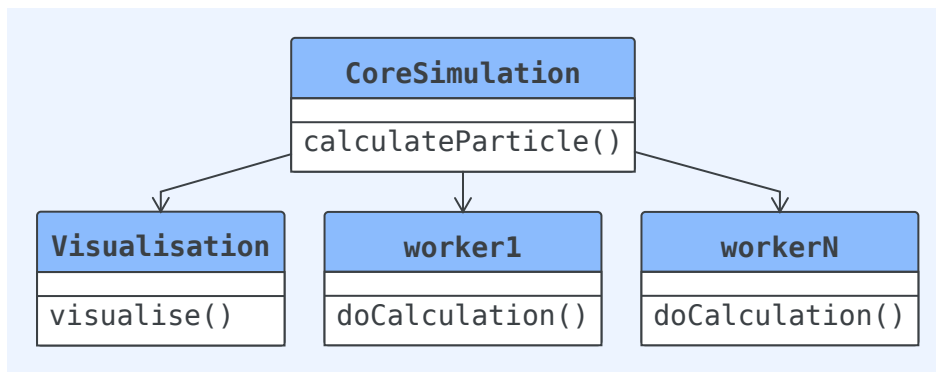
8.2. Physical View



8.3. Development View

- The structure of the code is flat there is no file or folder organisation
- There are larger multiclass components such as Particle tracking, Physics processes and lattice calculations but these are only grouped in code not in file structure
- There are no git rules for commits

8.4. Process View



Each node represents a thread in the process view where the Core simulation can spin up worker threads for calculations in a HPC environment

9. Codescene

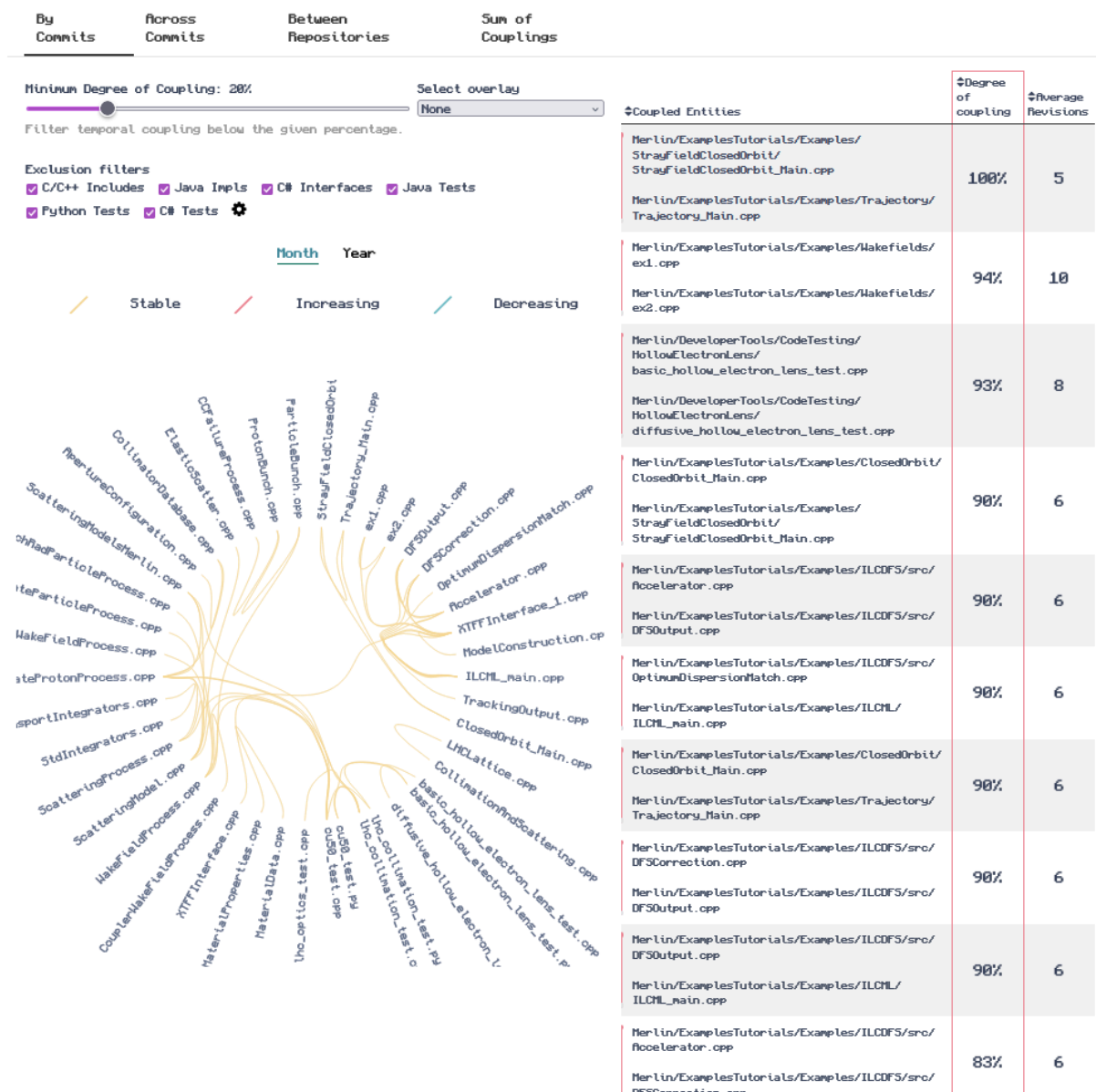


Figure 4: Codescene coupling graph

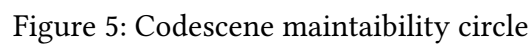


Figure 5: Codescene maintainability circle

10. DV8

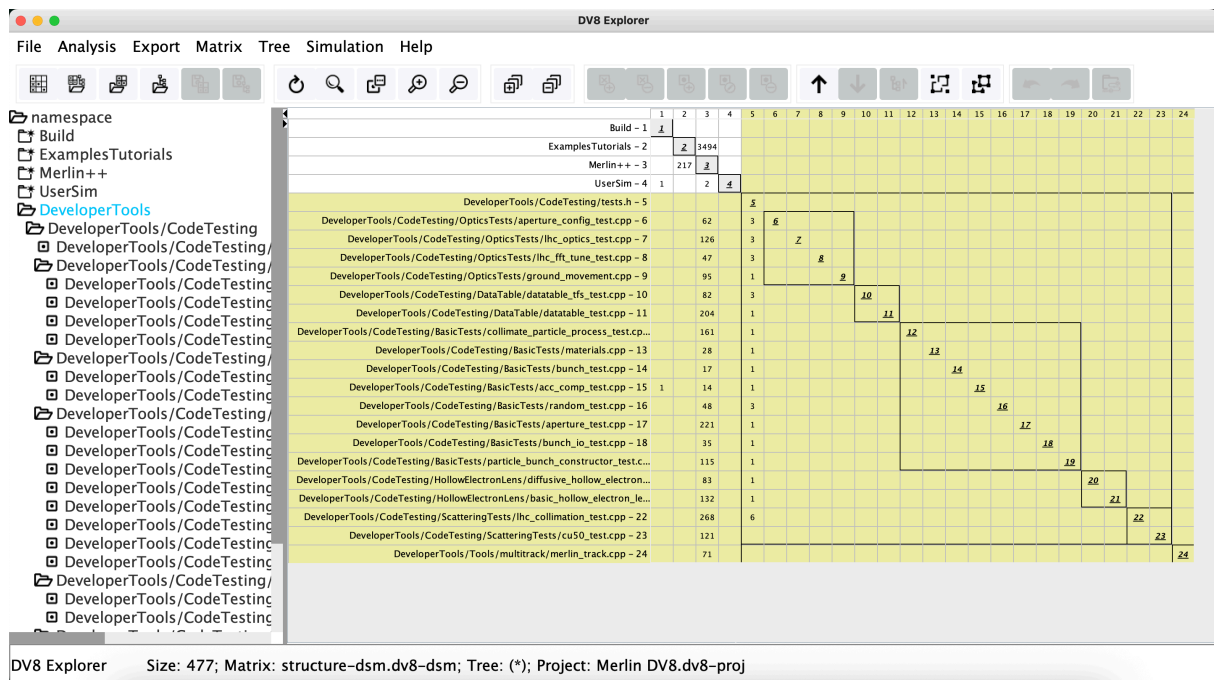


Figure 6: Dv8 Dev tool analysis

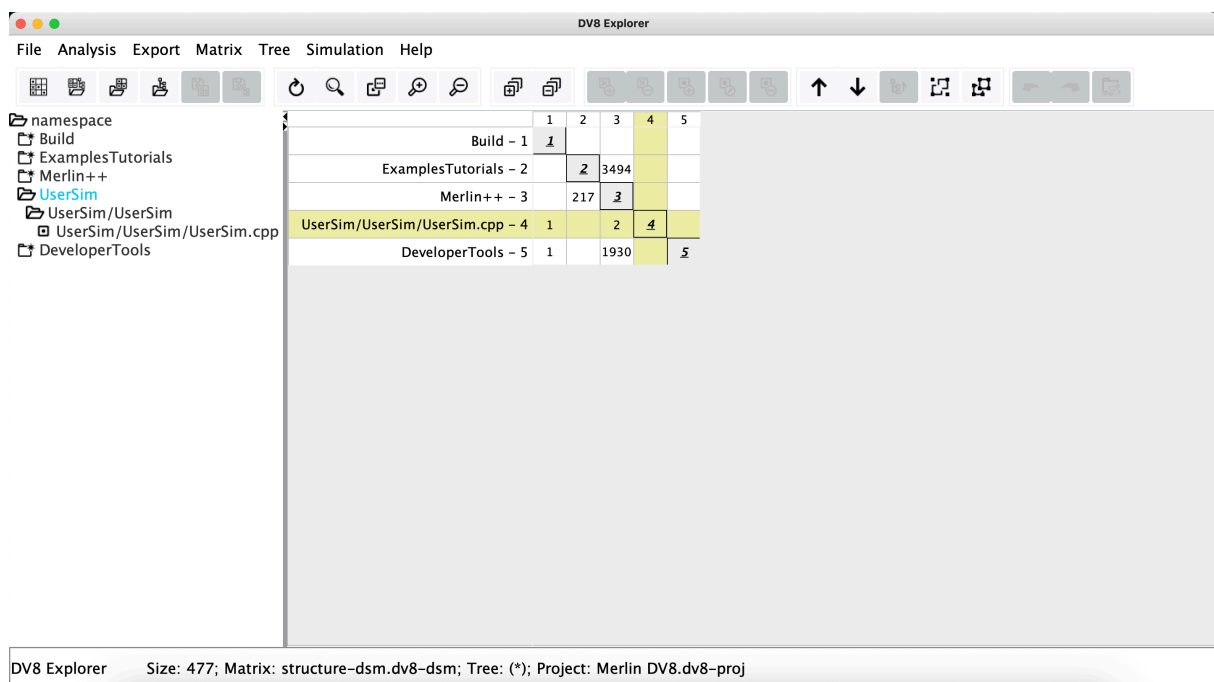
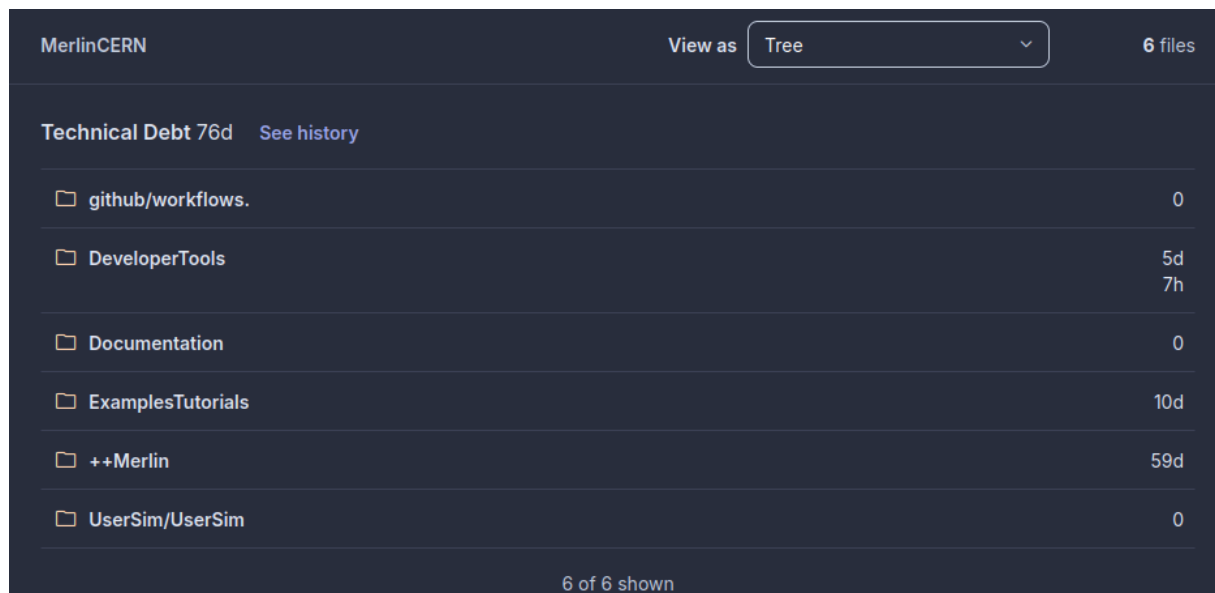


Figure 7: Dv8 high level analysis

11. SonarQube

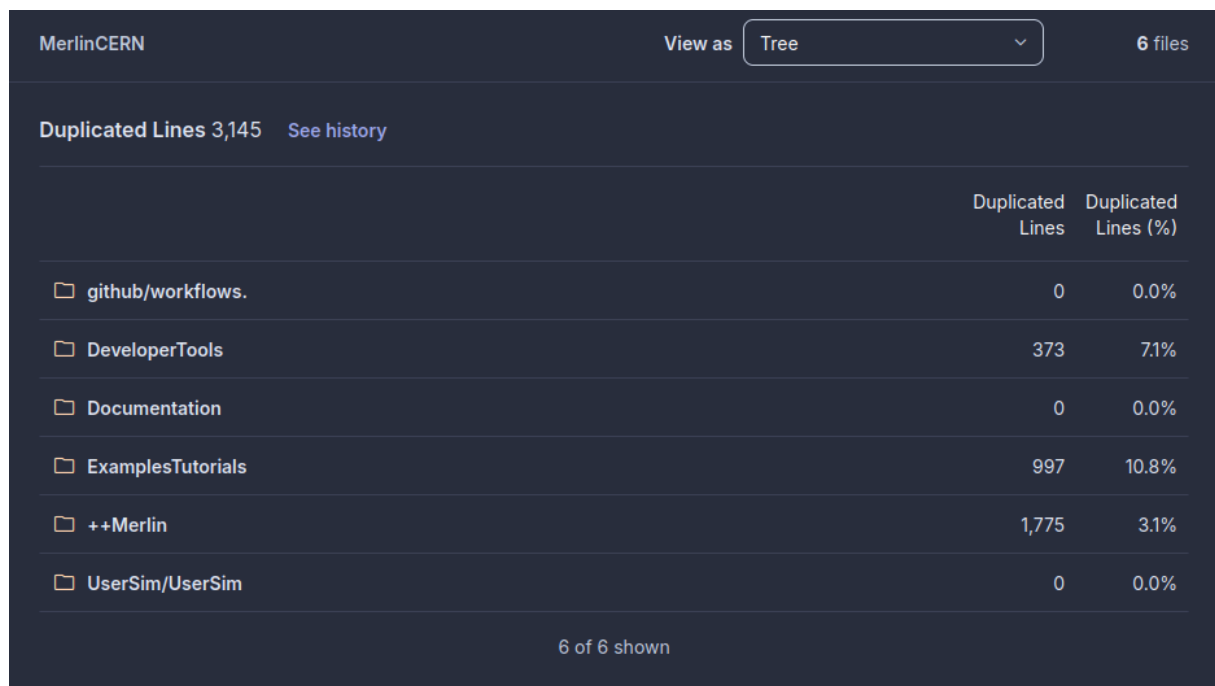


This screenshot shows the 'Technical Debt' section for the 'MerlinCERN' project in SonarQube. The view is set to 'Tree' and shows 6 files. The table lists the following items with their respective debt values:

Item	Debt
github/workflows.	0
DeveloperTools	5d 7h
Documentation	0
ExamplesTutorials	10d
++Merlin	59d
UserSim/UserSim	0

6 of 6 shown

Figure 8: Technical debt cost



This screenshot shows the 'Duplicated Lines' section for the 'MerlinCERN' project in SonarQube. The view is set to 'Tree' and shows 6 files. The table lists the following items with their duplicated lines and percentage:

Item	Duplicated Lines	Duplicated Lines (%)
github/workflows.	0	0.0%
DeveloperTools	373	7.1%
Documentation	0	0.0%
ExamplesTutorials	997	10.8%
++Merlin	1,775	3.1%
UserSim/UserSim	0	0.0%

6 of 6 shown

Figure 9: Duplicated code analysis

12. Understand

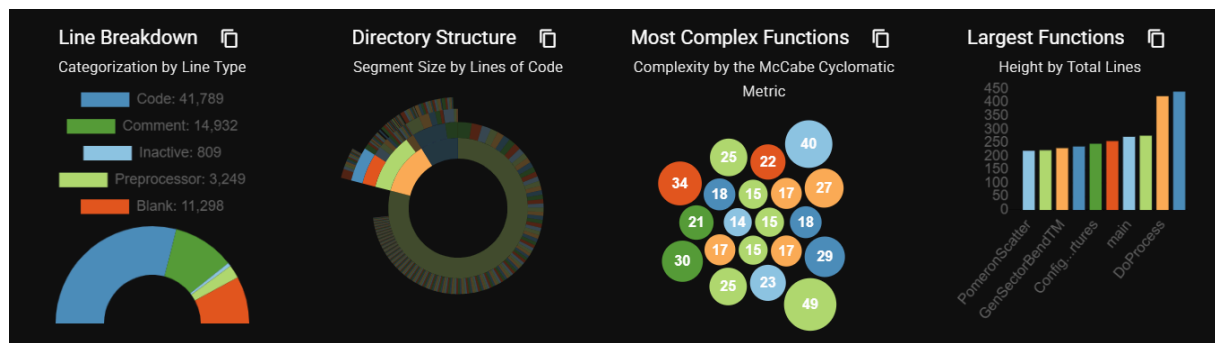


Figure 10: Function complexity understand



Figure 11: Coding language breakdown

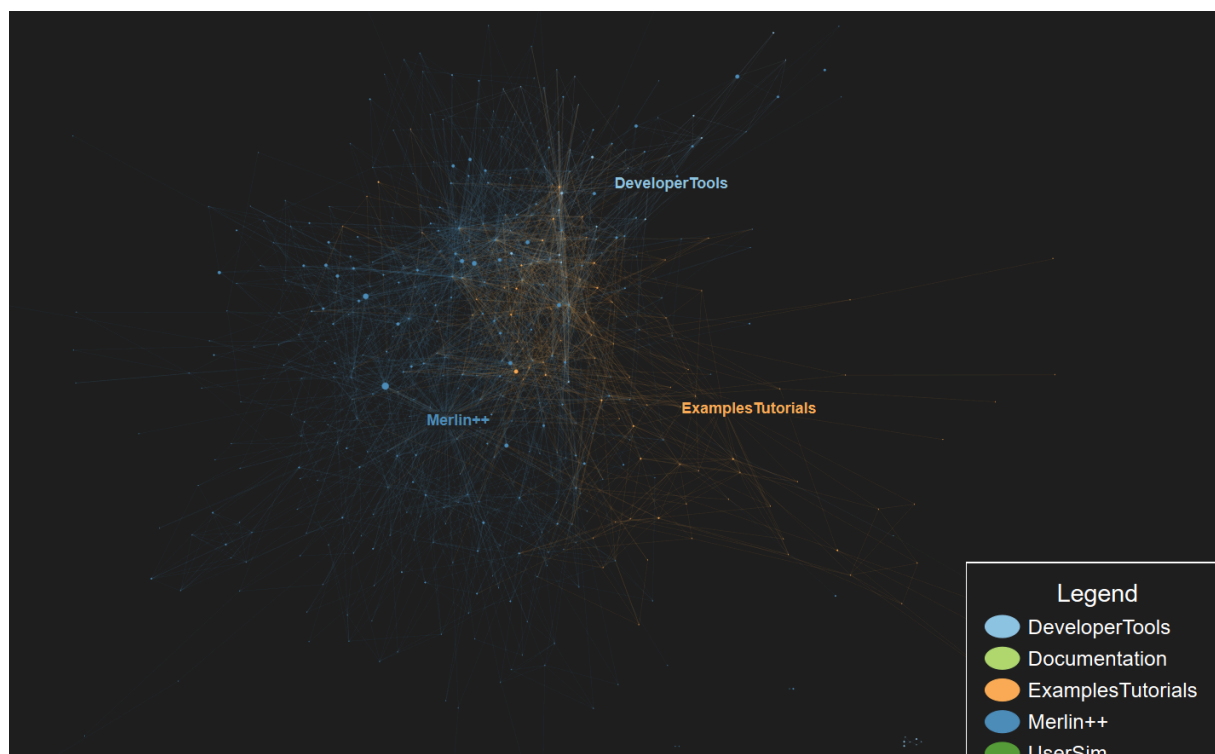


Figure 12: Understand code dependency graph

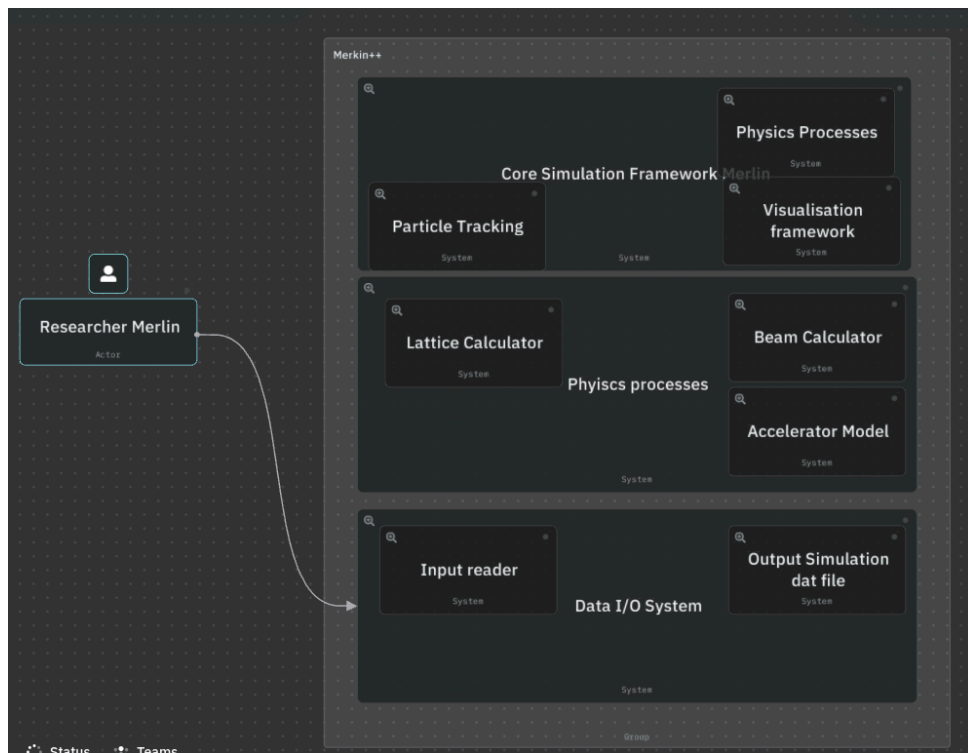


Figure 13: