



## The Adaptiv Framework

Nuno Alves de Sousa

Instituto Superior Técnico  
Área Científica de Mecânica Aplicada e Aeroespacial

February 17, 2019

- 1 Best coding practices
- 2 Concepts Library
- 3 Linear Algebra Library

## 1 Best coding practices

Software quality

Prerequisites

Code development

## 2 Concepts Library

## 3 Linear Algebra Library

# Attributes of good software

*“Software and cathedrals are much the same – first we build them,  
then we pray.”*

*(Anonymous)*

# Attributes of good software

Not *what* the program does, but *how well* it does it:

**Maintainability** reduce/reverse “code entropy”  
cheaper/safer to change than to rewrite

**Dependability** availability, reliability, safety, integrity

**Efficiency** algorithmic efficiency  
storage efficiency

**Usability** “consumer” effectiveness and efficiency  
elegance and clarity perceived by the user

# Outline

## 1 Best coding practices

Software quality

Prerequisites

Code development

## 2 Concepts Library

## 3 Linear Algebra Library

# Where to start?

*“What happens before one gets to the coding stage is often of crucial importance to the success of the project.”*

*(Meek & Heath - Guide to Good Programming Practice)*

Higher-level prerequisites to provide a solid foundation for coding:

- Coding standards
- Choice of programming language
- Life cycle, architecture, design
- **Requirements**

# Coding standards

Coding conventions are particularly important in collaborative projects:

- Much easier to read someone else's code
- Uniform style (e.g. naming conventions for filenames, variables, etc)
- Deal with undereducated programmers
- Avoid insufficient library use
- Portability
- Commenting conventions:
  - Speed up knowledge transfer
  - Comment only what code expresses poorly (intent)
  - Comments lie, code never lies
  - Do not comment code modifications (use a [version control system](#))



# Version control

Source code is the most valuable asset of any software project

## Version control systems (VCS)

- Management of changes to all non-binary files
- Complete retrace of all versions of each file
- History of the authors of such changes

## Critical advantages

- Rollback of all tracked changes
- Work in an isolated fashion
- Seamless team collaboration
- Efficient and flexible scalability

# git - the world's leading version control system

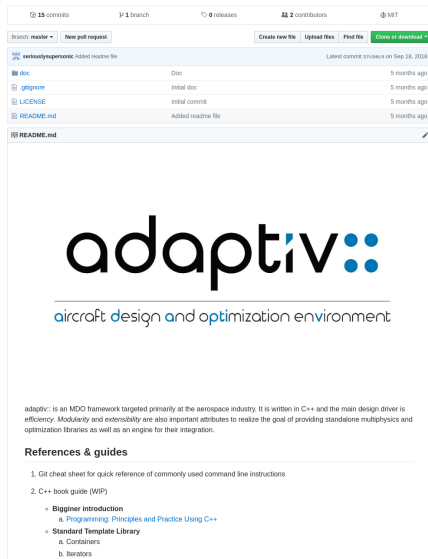


## Why git?

- Free and open-source
- Small and fast
- Encourages branching
- Distributed
- Built-in IDE support

## As a service:

- Source code hosting
- Code sharing platform
- GitHub, GitLab, etc.



The screenshot shows the GitHub repository for the Adaptive Framework. At the top, it displays 15 commits, 1 branch, 0 releases, 2 contributors, and MIT license. Below this is a table of files:

File	Commit	Time
doc	Doc	5 months ago
gignone	Initial doc	5 months ago
LICENSE	Initial commit	5 months ago
README.md	Added readme file	5 months ago

The README.md file content is displayed below the table. It features the Adaptive Framework logo and tagline: "aircraft design and optimization environment". The text describes the framework as an MDO framework for the aerospace industry, written in C++.

### References & guides

1. Git cheat sheet for quick reference of commonly used command line instructions
2. C++ book guide (WIP)
  - **Beginner Introduction**
    - a. [Programming: Principles and Practice Using C++](#)
  - **Standard Template Library**
    - a. Containers
    - b. Iterators

# Choice of programming language

C++ (hard, lack of knowledge, modern features)  
Use good well tested libraries (boost) - portability

Life cycle, architecture, design all depend on the **requirements**

## 1 Best coding practices

Software quality

Prerequisites

Code development

## 2 Concepts Library

## 3 Linear Algebra Library

## Build system

- Open-source, cross-platform set of tools to build, test and package software.
- Controls compilation process using platform and compiler independent config. files



*The defacto standard for building C++ projects*

### Advantages

- More time for coding
- Supported by most popular IDEs (e.g. VS, JetBrains, QtCreator)
- Support for multiple compilers (e.g. MSVC, GCC, Clang, Intel)
- Easy integration of 3rd party libraries

# Testing

content...

# Benchmarking

content...



- 1 Best coding practices
- 2 Concepts Library**
- 3 Linear Algebra Library

# Outline

- 1 Best coding practices
- 2 Concepts Library
- 3 Linear Algebra Library**

# Interface elegance vs code efficiency

Level 3 BLAS operations  $T(n) = O(n^3)$

Example:

$$C \leftarrow \alpha A^T B^T + \beta C \quad (1)$$

Desired interface:

```

1  #include<matrix.hpp>
2
3  using DMatrix = Matrix<double>;
4
5  const double alfa = 42;
6  const double beta = 1.618
7
8  void example() {
9      const std::size_t dim = 100;
10     auto a = DMatrix::random(dim); // Same for b and c...
11
12     c = alfa * a.transpose() * b.transpose() + beta * c;
13 }
```

# Interface elegance vs code efficiency

For an efficient implementation, the statement

```
12  c = alfa * a.transpose() * b.transpose() + beta * c;
```

should be translated into a call to the specialized CBLAS function:

```
12  cblas_dgemm(CblasColMajor, CblasTrans, CblasTrans  
13              ,dim ,dim, dim  
14              ,alpha  
15              ,a.data(), dim  
16              ,b.data(), dim  
17              ,beta  
18              ,c.data(), dim);
```

Overhead:

1 function call

0 temporaries

# Conventional operator overloading

Due to the normal order of evaluation of the C++ language,

```
12      c = alfa * a.transpose() * b.transpose() + beta * c;
```

actually generates:

```

9  void example() { // Assume proper initialization of a, b and, c
10      DMatrix temp1 = beta * c;           // call (A)
11      DMatrix temp2 = b.transpose();      // call (B)
12      DMatrix temp3 = a.transpose();      // call (B)
13      DMatrix temp4 = temp3 * temp2;      // call (C)
14      DMatrix temp5 = alfa * temp4;       // call (C)
15      DMatrix temp6 = temp5 + temp1;      // call (D)
16      c = temp6;                         // call (E)
17  }

18
19  DMatrix operator*(double d, const DMatrix& mat);           // (A)
20  void DMatrix::transpose();                                // (B)
21  DMatrix operator*(const DMatrix& m1, const DMatrix& m2); // (C)
22  DMatrix operator+(const DMatrix& m1, const DMatrix& m2); // (D)
23  DMatrix operator=(const DMatrix& m1, const DMatrix& m2); // (E)
```

# Conventional operator overloading

Overhead:

**7+** function calls  
**6** temporaries

## Performance comparison

$$C = \alpha A^T B^T + \beta C \quad (\text{Peak GFLOPS} = 224)$$

