



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

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

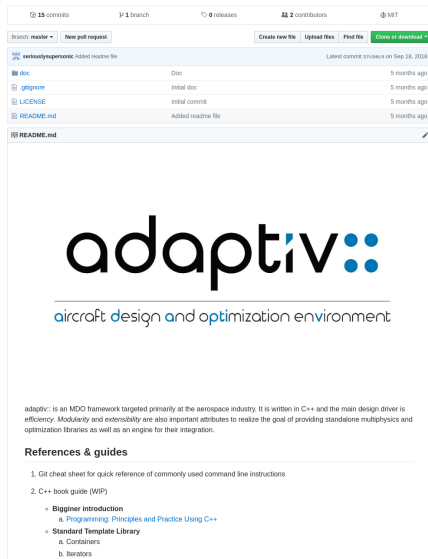


Why git?

- 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, there are buttons for 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The file list shows 'README.md' as the selected file, with other files like 'doc', 'g4gnone', 'LICENSE', and 'README.md' listed below it. The 'README.md' content is displayed, featuring the 'adaptiv' logo and the tagline 'aircraft design and optimization environment'. Below the logo, there is a description of the framework and a list of references and guides.

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

seriouslysupersonic Added readme file | Latest commit: 7b1b1b on Sep 18, 2018

File	Commit	Time
doc	Doc	5 months ago
g4gnone	Initial doc	5 months ago
LICENSE	Initial commit	5 months ago
README.md	Added readme file	5 months ago

README.md

adaptiv
 aircraft design and optimization environment

adaptiv: is an MDO framework targeted primarily at the aerospace industry. It is written in C++ and the main design driver is efficiency. Modularity and extensibility are also important attributes to realize the goal of providing standalone multiphysics and optimization libraries as well as an engine for their integration.

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 on a i7-4770k_(224 GFLOPS)

$$C \leftarrow \alpha A^T B^T + \beta C$$

