Lecture 6: Classes and Templates

Johannes Gerstmayr and Markus Walzthöni

University of Innsbruck

November 25, 2023

More than just C++

Today's plan:

- clean code, code duplication, refactoring
- coding rules / styles, Google standards (what is it?)
- exceptions, macros
- Exudyn: basics, memory management (no `new`), testing, etc.
- Pybind11 (what is it?)
- Numerical receipes in C++ (what is it?)

Clean code

- A large amount of scientific code is dead (unused) code
- Code without coding rules: draft paper, notes (may contain ingenious ideas!)
- Code with coding rules: published paper, book (same ideas, but in readable form)

- Imagine clean desk vs. full desk
- Imagine clear road vs. road construction (**frustrating**)
- If you leave the clean code paradigm, your code will slowly die
- R. Martin, Clean code: A handbook of agile software craftsmanship, Prentice Hall, 2014.

Clean code

- A large amount of scientific code is dead (unused) code
- Code without coding rules: draft paper, notes (may contain ingenious ideas!)
- Code with coding rules: published paper, book (same ideas, but in readable form)

- Imagine clean desk vs. full desk
- Imagine clear road vs. road construction (**frustrating**)
- If you leave the clean code paradigm, your code will slowly die
- R. Martin, Clean code: A handbook of agile software craftsmanship, Prentice Hall, 2014.

Clean code

- A large amount of scientific code is dead (unused) code
- Code without coding rules: draft paper, notes (may contain ingenious ideas!)
- Code with coding rules: published paper, book (same ideas, but in readable form)

- Imagine clean desk vs. full desk
- Imagine clear road vs. road construction (**frustrating**)
- If you leave the clean code paradigm, your code will slowly die
- R. Martin, Clean code: A handbook of agile software craftsmanship, Prentice Hall, 2014.

Clean code

- A large amount of scientific code is dead (unused) code
- Code without coding rules: draft paper, notes (may contain ingenious ideas!)
- Code with coding rules: published paper, book (same ideas, but in readable form)

- Imagine clean desk vs. full desk
- Imagine clear road vs. road construction (**frustrating**)
- If you leave the clean code paradigm, your code will slowly die
- R. Martin, Clean code: A handbook of agile software craftsmanship, Prentice Hall, 2014.

Clean code

- A large amount of scientific code is dead (unused) code
- Code without coding rules: draft paper, notes (may contain ingenious ideas!)
- Code with coding rules: published paper, book (same ideas, but in readable form)

- Imagine clean desk vs. full desk
- Imagine clear road vs. road construction (**frustrating**)
- If you leave the clean code paradigm, your code will slowly die
- R. Martin, Clean code: A handbook of agile software craftsmanship, Prentice Hall, 2014.

Clean code (2)

Practical aspects for clean code

- E.g., if you find inconsistencies "`NumberOf_Elemnts()`" → fix immediately!
  → The costs of not changing will be higher
- Do not postpone necessary changes (class structure; function interface; etc.); the longer you wait the more complicated it gets; you will never change it; **leads to frustration**
- If you detect a inconsistency in your coding style, immediately change it (even if 100 things to be changed)
- Use project-wide search/replace; Don't adapt only locally!
- There are tools: Microsoft Visual Studio allows to rename names incl. comments (very efficient and reliable)
- If you see some ways for simplification or code reduction, immediately apply it; it will pay off soon
- **Refuse to duplicate code** (one place to define something)

Clean code (2)

Practical aspects for clean code

- E.g., if you find inconsistencies "`NumberOf_Elemnts()`" → fix immediately!

  → The costs of not changing will be higher
- Do not postpone necessary changes (class structure; function interface; etc.); the longer you wait the more complicated it gets; you will never change it; **leads to frustration**
- If you detect a inconsistency in your coding style, immediately change it (even if 100 things to be changed)
- Use project-wide search/replace; Don't adapt only locally!
- There are tools: Microsoft Visual Studio allows to rename names incl. comments (very efficient and reliable)
- If you see some ways for simplification or code reduction, immediately apply it; it will pay off soon
- **Refuse to duplicate code** (one place to define something)

Clean code (2)

Practical aspects for clean code

- E.g., if you find inconsistencies "`NumberOf_Elemnts()`" → fix immediately!

  → The costs of not changing will be higher
- Do not postpone necessary changes (class structure; function interface; etc.); the longer you wait the more complicated it gets; you will never change it; **leads to frustration**
- If you detect a inconsistency in your coding style, immediately change it (even if 100 things to be changed)
- Use project-wide search/replace; Don't adapt only locally!
- There are tools: Microsoft Visual Studio allows to rename names incl. comments (very efficient and reliable)
- If you see some ways for simplification or code reduction, immediately apply it; it will pay off soon
- **Refuse to duplicate code** (one place to define something)

Clean code (2)

Practical aspects for clean code

- E.g., if you find inconsistencies "`NumberOf_Elemnts()`" → fix immediately!

  → The costs of not changing will be higher
- Do not postpone necessary changes (class structure; function interface; etc.); the longer you wait the more complicated it gets; you will never change it; **leads to frustration**
- If you detect a inconsistency in your coding style, immediately change it (even if 100 things to be changed)
- Use project-wide search/replace; Don't adapt only locally!
- There are tools: Microsoft Visual Studio allows to rename names incl. comments (very efficient and reliable)
- If you see some ways for simplification or code reduction, immediately apply it; it will pay off soon
- **Refuse to duplicate code** (one place to define something)

Clean code (2)

Practical aspects for clean code

- E.g., if you find inconsistencies "`NumberOf_Elemnts()`" → fix immediately!

  → The costs of not changing will be higher
- Do not postpone necessary changes (class structure; function interface; etc.); the longer you wait the more complicated it gets; you will never change it; **leads to frustration**
- If you detect a inconsistency in your coding style, immediately change it (even if 100 things to be changed)
- Use project-wide search/replace; Don't adapt only locally!
- There are tools: Microsoft Visual Studio allows to rename names incl. comments (very efficient and reliable)
- If you see some ways for simplification or code reduction, immediately apply it; it will pay off soon
- **Refuse to duplicate code** (one place to define something)

Clean code (2)

Practical aspects for clean code

- E.g., if you find inconsistencies "`NumberOf_Elemnts()`" → fix immediately!

  → The costs of not changing will be higher
- Do not postpone necessary changes (class structure; function interface; etc.); the longer you wait the more complicated it gets; you will never change it; **leads to frustration**
- If you detect a inconsistency in your coding style, immediately change it (even if 100 things to be changed)
- Use project-wide search/replace; Don't adapt only locally!
- There are tools: Microsoft Visual Studio allows to rename names incl. comments (very efficient and reliable)
- If you see some ways for simplification or code reduction, immediately apply it; it will pay off soon
- **Refuse to duplicate code** (one place to define something)

Clean code (2)

Practical aspects for clean code

- E.g., if you find inconsistencies "`NumberOf_Elemnts()`" → fix immediately!

  → The costs of not changing will be higher
- Do not postpone necessary changes (class structure; function interface; etc.); the longer you wait the more complicated it gets; you will never change it; **leads to frustration**
- If you detect a inconsistency in your coding style, immediately change it (even if 100 things to be changed)
- Use project-wide search/replace; Don't adapt only locally!
- There are tools: Microsoft Visual Studio allows to rename names incl. comments (very efficient and reliable)
- If you see some ways for simplification or code reduction, immediately apply it; it will pay off soon
- **Refuse to duplicate code** (one place to define something)

Clean code (3)

Test suite / unit tests

- Immediately install unit tests or larger tests
- This could be small tests designed for each class, or global tests for the project
- You either check for tests to run or compile, but mostly you try to check for outcome to be unchanged
- Tests can be built into local CI / CD tools or you run them manually; record results (incl. version)
- $\rightarrow$ approx. 80 larger tests in Exudyn; unfortunately results are not same on different platforms/compilers!!!

Benefits of tests?

- The biggest benefit is that you dare to change / revise code because you will see that it still runs correctly (at least regarding the tests)

Clean code (4)

Documentation

- Immediately add documentation (when writing code)
- Write class docu before starting to code: Class design
- Continuously revise documentation (others will have a hard times otherwise!)
- Add entry points (in addition to Sphinx, etc.): global readme.md and ReadTheDocs
- Add examples (these could be overlapping with tests!)

Wrong:

```
int i; //counter
//TODO: needs to be commented
for (i=1; i<100; ++i) {
    auto dx2 = Receive(i); //receive dx2
    dx2.TrpCRX(i);
    WriteToFile(dx2); //write to file
}
```

→ What is it all about? Why does it start with 1? what is dx2? what does TrpCRX? ...

Clean code (4)

Documentation

- Immediately add documentation (when writing code)
- Write class docu before starting to code: Class design
- Continuously revise documentation (others will have a hard times otherwise!)
- Add entry points (in addition to Sphinx, etc.): global readme.md and ReadTheDocs
- Add examples (these could be overlapping with tests!)

Wrong:

```
int i; //counter
//TODO: needs to be commented
for (i=1; i<100; ++i) {
    auto dx2 = Receive(i); //receive dx2
    dx2.TrpCRX(i);
    WriteToFile(dx2); //write to file
}
```

→ What is it all about? Why does it start with 1? what is dx2? what does TrpCRX? ...

Coding rules / style guide

- You have to define codeing style / standards for your project
- There are simple ones / incomplete
- Google C++ Style Guide:
  `https://google.github.io/styleguide/cppguide.html`

Wrong:

```
class myspecialclass {
  int I;
  double real_number;
  void DoSOMEOPERATIONS() {...}
  void get() const {...}
  void Set(double x) {...}
}
```

Coding rules / style guide

- You have to define codeing style / standards for your project
- There are simple ones / incomplete
- Google C++ Style Guide:
  `https://google.github.io/styleguide/cppguide.html`

Wrong:

```
class myspecialclass {
  int I;
  double real_number;
  void DoSOMEOPERATIONS() {...}
  void get() const {...}
  void Set(double x) {...}
}
```

Exceptions (1)

Sometimes we get to a point where no further computations make sense

```
double Inv(double x) {
    if (x == 0) {
        std::cout << "function f failed: x=0\n";
        //... what to do now?
        throw std::runtime_error("Inv: division by zero");
        }
    return 1./x;
}
```

Exceptions allow us to terminate locally:

- put `try /* */ catch (...) /* */` block in outer loops
- local codes my `throw` an exception; add useful information (string)
- know: you can debug codes, but exceptions will show errors much faster
- add more checks than necessary, could be removed in **fast** release

Exceptions (1)

Sometimes we get to a point where no further computations make sense

```
double Inv(double x) {
    if (x == 0) {
        std::cout << "function f failed: x=0\n";
        //... what to do now?
        throw std::runtime_error("Inv: division by zero");
        }
    return 1./x;
}
```

Exceptions allow us to terminate locally:

- put `try /* */ catch (...) /* */` block in outer loops
- local codes my `throw` an exception; add useful information (string)
- know: you can debug codes, but exceptions will show errors much faster
- add more checks than necessary, could be removed in **fast** release

Exceptions (2)

Example for try-catch block:

```
int main() {
    //print 1/x for x in range [-5,+5]
    for(int x=-5; x<=5; x++) {
        try {
            double invX = Inv(x); //may raise exception
            std::cout << "1/" << x << "=" << invX << "\n";
        }
        catch (const std::exception& e) {
            std::cout << "EXCEPTION raised: " << e.what() << '\n';
            std::cout << "1/" << x << "=NaN\n";
        }
    }
}
```

→ here, we could also use other mechanisms (check for `inf` in `invX`) !

C MACROS

- try to avoid!
- global (compiler) flags (global switches for code)
- unified, reusable macros ("functions")
- sometimes very helpful; but use (...) for functions!!!

```
//define constant macro (don't use!); but compiler can use it:
#define NUMBER 100
#define SQUARE(x) ((x) * (x)) //define macro function
#define DEBUG_ON              //switch

int main() {
    printf("Square of \%d is \%d\n", NUMBER, SQUARE(number));
    #ifdef DEBUG_ON
        std::cout << "Debugging is enabled\n";
    #endif
```

C MACROS

- try to avoid!
- global (compiler) flags (global switches for code)
- unified, reusable macros ("functions")
- sometimes very helpful; but use (...) for functions!!!

```
//define constant macro (don't use!); but compiler can use it:
#define NUMBER 100
#define SQUARE(x) ((x) * (x)) //define macro function
#define DEBUG_ON              //switch

int main() {
    printf("Square of \%d is \%d\n", NUMBER, SQUARE(number));
    #ifdef DEBUG_ON
        std::cout << "Debugging is enabled\n";
    #endif
```

Exudyn (1)

Some key features:

- **Multibody system code**: solve small and large systems
- Modeling in Python, 5.000.000 steps/second (wouldn't work in Python!)
- has approx. **550 classes**, 387 header files, 130 .cpp files
- 160,000 lines of code (**short!!**); 64% in C++, 13% Python,
- 11.5% definition and code generation, and 11.5% tests;
- 43% of C++ code is automatically generated.
- automated building of ≈15 **Python wheels** on Windows, MacOS and Linux
- highly structured naming of classes
- ≈250 Examples and test models, automated testing when building
- LLMs like it a lot because of highly structured code/examples: even WizardCoder-Python-7B-V1.0 knows it

Exudyn (1)

Some key features:

- **Multibody system code**: solve small and large systems
- Modeling in Python, 5.000.000 steps/second (wouldn't work in Python!)
- has approx. **550 classes**, 387 header files, 130 .cpp files
- 160,000 lines of code (**short!!**); 64% in C++, 13% Python,
- 11.5% definition and code generation, and 11.5% tests;
- 43% of C++ code is automatically generated.
- automated building of ≈15 **Python wheels** on Windows, MacOS and Linux
- highly structured naming of classes
- ≈250 Examples and test models, automated testing when building
- LLMs like it a lot because of highly structured code/examples: even WizardCoder-Python-7B-V1.0 knows it

Exudyn (1)

Some key features:

- **Multibody system code**: solve small and large systems
- Modeling in Python, 5.000.000 steps/second (wouldn't work in Python!)
- has approx. **550 classes**, 387 header files, 130 .cpp files
- 160,000 lines of code (**short!!**); 64% in C++, 13% Python,
- 11.5% definition and code generation, and 11.5% tests;
- 43% of C++ code is automatically generated.
- automated building of ≈15 **Python wheels** on Windows, MacOS and Linux
- highly structured naming of classes
- ≈250 Examples and test models, automated testing when building
- LLMs like it a lot because of highly structured code/examples: even WizardCoder-Python-7B-V1.0 knows it

Exudyn (1)

Some key features:

- **Multibody system code**: solve small and large systems
- Modeling in Python, 5.000.000 steps/second (wouldn't work in Python!)
- has approx. **550 classes**, 387 header files, 130 .cpp files
- 160,000 lines of code (**short!!**); 64% in C++, 13% Python,
- 11.5% definition and code generation, and 11.5% tests;
- 43% of C++ code is automatically generated.
- automated building of ≈15 **Python wheels** on Windows, MacOS and Linux
- highly structured naming of classes
- ≈250 Examples and test models, automated testing when building
- LLMs like it a lot because of highly structured code/examples: even WizardCoder-Python-7B-V1.0 knows it

Exudyn (2)

Some key features:

- Python – C++ coupling: **Pybind11** (but not everything exported / safety!)
- automatic code / documentation generation to keep Python – C++ interface consistent!
- highly optimized C++ code, multi-threaded; future: **GPU extension** (automated ...)
- Separate **memory management**:
  - small vectors / matrices with pre-allocated memory
  - large vectors / matrices with extensibility
  - code designed such that **no memory allocation** typically occurs after first computation step
  - specific design to store limited amount of data / results (2 million rigid bodies produce 192MB / step)
- see: `https://github.com/jgerstmayr/EXUDYN`

Exudyn (2)

Some key features:

- Python – C++ coupling: **Pybind11** (but not everything exported / safety!)
- automatic code / documentation generation to keep Python – C++ interface consistent!
- highly optimized C++ code, multi-threaded; future: **GPU extension** (automated ...)
- Separate **memory management**:
  - small vectors / matrices with pre-allocated memory
  - large vectors / matrices with extensibility
  - code designed such that **no memory allocation** typically occurs after first computation step
  - specific design to store limited amount of data / results (2 million rigid bodies produce 192MB / step)
- see: `https://github.com/jgerstmayr/EXUDYN`

Pybind11

Why?

- C++ coding is expensive, error prone, ...
- → performance-critical code in C++
- modeling, pre- and postprocessing, parameter variation, optimization, eigenvalues, ... in Python
- Python: drivers for you code, settings / input data, results

Why?

- semi-automatically link C++ data/methods to Python
- Pybind11 is a C++ header library, heavily templated (compilation "costs")
- Creates bindings for C++ types incl. NumPy "connectivity"
- Classes may be fully exposed from C++ to Python

Pybind11

Why?

- C++ coding is expensive, error prone, ...
- $\rightarrow$ performance-critical code in C++
- modeling, pre- and postprocessing, parameter variation, optimization, eigenvalues, ... in Python
- Python: drivers for you code, settings / input data, results

Why?

- semi-automatically link C++ data/methods to Python
- Pybind11 is a C++ header library, heavily templated (compilation "costs")
- Creates bindings for C++ types incl. NumPy "connectivity"
- Classes may be fully exposed from C++ to Python

Pybind11

C++:

```cpp
#include <pybind11/pybind11.h>
//regular C / C++ function:
int add(int i, int j)
  { return i + j; }

//define the Python module:
PYBIND11_MODULE(addModule, m) {
  m.def("add", &add,
        "adds two numbers");
  }
```

Python:

```python
from addModule import * #import C++ module

print(add(3,4)) #prints 7
```

Pybind11

C++:

```
#include <pybind11/pybind11.h>
//regular C / C++ function:
int add(int i, int j)
  { return i + j; }

//define the Python module:
PYBIND11_MODULE(addModule, m) {
  m.def("add", &add,
        "adds two numbers");
  }
```

Python:

```
from addModule import * #import C++ module

print(add(3,4)) #prints 7
```

Numerical receipes in C++ (1)

Why?

- most times you can just use Lapack, Eigen, etc. → but you will not find the underlying algorithms
- if you need to understand / rewrite basic numerical algorithms ...

Unfortunately it is not free / not open source!

- Numerical Recipes in C++. The Art of Scientific Computing, 2nd Edition, 2002, ISBN 0-521-75033-4.
- free 1992 C-version: `https://s3.amazonaws.com/nrbook.com/book_C210.html`
- online book: `http://nrbook.com/book.html`

Numerical receipes in C++ (1)

Why?

- most times you can just use Lapack, Eigen, etc. $\rightarrow$ but you will not find the underlying algorithms
- if you need to understand / rewrite basic numerical algorithms ...

Unfortunately it is not free / not open source!

- Numerical Recipes in C++. The Art of Scientific Computing, 2nd Edition, 2002, ISBN 0-521-75033-4.
- free 1992 C-version: `https://s3.amazonaws.com/nrbook.com/book_C210.html`
- online book: `http://nrbook.com/book.html`

Numerical receipes in C++ (2)

Example for Gaussian integration:

```
double qgaus(T &func, const double a, const double b) {
    static const double x[]={0.1488743389816312,0.4333953941292472,
        0.6794095682990244,0.8650633666889845,0.9739065285171717};
    static const double w[]={0.2955242247147529,0.2692667193099963,
        0.2190863625159821,0.1494513491505806,0.0666713443086881};
    double xm=0.5*(b+a);
    double xr=0.5*(b-a);
    double s=0;
    for (int j=0;j<5;j++) {
        double dx=xr*x[j];
        s += w[j]*(func(xm+dx)+func(xm-dx));
    }
    return s *= xr; //Scale the answer to the range of integration.
}
```

→ NR is a provider for good ideas, without searching large GitHub repos!!!

Outlook

This is it from my side

- Keep track, try some codeing (the forgetting curve is very steep!)
- With the help of AI-tools, you can also use it only for small portions of code!

Thanks a lot for participating!

Outlook

This is it from my side

- Keep track, try some codeing (the forgetting curve is very steep!)
- With the help of AI-tools, you can also use it only for small portions of code!

Thanks a lot for participating!