Software development for AM

- Alternative title: *Why MATLAB sucks (and Fortran too)*

Version Control System VCS

- *Everyone* that is writing text files should be using version control for everything.
- "But I . . .
  - have a Dropbox to store my files"
  - saved a copy on the common disc"
  - just email the files to my coworkers"
  - have my own system where i regularly save zips with a date in the filename"
  - don't need that because I'm a solo author"
- **No!**

Difference between backup and version control.

- Purpose of backups is mostly to restore the **current state** of your files in case of for example a hardware failure.
- OK for static resources, like pictures, movies, email archives etc.
- Dropbox, box, Google drive, external hard drives etc.
- Purpose of VCS is to be able to access **any previous state** of your tracked files and easily see the differences between two **arbitrary states** of files.
- Good for dynamic resources, like computer code.
- "This thing worked 2 weeks ago, it doesn't work now. What are the exact changes that has been done to my files between that point in time and now?"
- **Git**, SVN, Perforce, Mercurial etc.

Examples

Commit log example

Collaboration example

Collaboration example

Testing

Regression tests

- Did you break anything with that commit? What change caused the breakage?
- There should be tests for each element, each material model
- Not feasible to have for single developer/user codes
- Project should have infrastructure for adding tests as simple as possible, otherwise people won't do it.

Automation

- If it's not automatic then people are going to forget it.

Continuous integration

Collaboration

- Combined efforts leads to high quality code: Might spend a bit more time on your 1 component, but you get other components for free.
- The code will be large: VCS and testing absolutely *essential*. There is NO option.
- Projects with

Does programming language matter?

- Performance is far from everything.
- Many good options:
  - C++, Java, Python, Julia, D, Rust,
- Large existing code? Use whatever language they use.

MATLAB

- (+/-) Interpreted: Easy debugging! But compilation errors also often reveal mistakes immediately (static analysis).
- (+) Decent standard library of functions (but you should use a decent library/module in other languages anyway)
- (-) Slow and there is nothing you can do about it.
- (-) Punishes good software design (heavily penalized for calling functions)
- (-) Crude/nonexistent module/library support.
- (-) Actually supports classes, but noone uses those, therefore no data abstraction. *Everything* is just chunks of numbers.
- Only really good for doing linear algebra, and that's only 1% of the code.

Raw data

- x is a list of lists of lists of floats.

```
x = [[[1.,2.],[3.,4.],[0.,2.]],
     [[3.,4.],[2.,0.],[3.,4.]],
     [[5.,6.],[1.,1.],[3.,3.]],
     ...
    ]
```

Data abstraction

- x is a list of triangles

```
x = [Triangle(Point(1., 2.), Point(1., 2.), Point(0., 2.)),
     Triangle(Point(3., 4.), Point(2., 0.), Point(3., 4.)),
     Triangle(Point(5., 6.), Point(1., 1.), Point(3., 3.)),
    ...
    ]
```

Why?

- Data abstractions are absolute necessity when program size grows
- Classes connect functions and data
- Object-oriented programming is an excellent fit for FEM.
  - Plug and play elements/materials: Prime example of the need for polymnorphism.
- MATLAB does basic data abstraction, but most likely you are working with blocks of numbers

FORTRAN and C are not good options.

- Languages must allow you to build custom data types. No language will have thought of everything.
- Manual memory management leads to many unneccessary bugs. Typical Fortran:

```
Allocate(Ysimp(1,mtrl_par_type%NP))
...
deallocate(Ysimp)
```

- Does not enable (encourage) abstractions: You are stuck in low-level programming hell.

```
call dgemm('N','N', 9, 9, 9, 1.d0,a, 9, b, 9, 0.d0, c, 9)
```

What would you change to multiple $a * b$ where a is 6x24?

C++ is actually quite good

- C++ was actually invented to *build abstraction*
- C++ *without* using any libraries would be awful
- Compilers have good warnings that helps to fix bug before even running the code.
  - Memory errors (segmentation faults, leaks) are really really easy to find.
- Code can look as simple as MATLAB. Typical C++ code:

```
c.beProductOf(a, b);
c = a * b; # We can overload operators if we want
s = {s[0], s[1], 0, 0, 0, s[3]}; # Plane stress -> full
```

- You basically never do any memory management.
- Many other languages are also quite good