Organizing the Collaborative Development of Scientific Code Base

Based on 1-year in the Computation Geosciences group

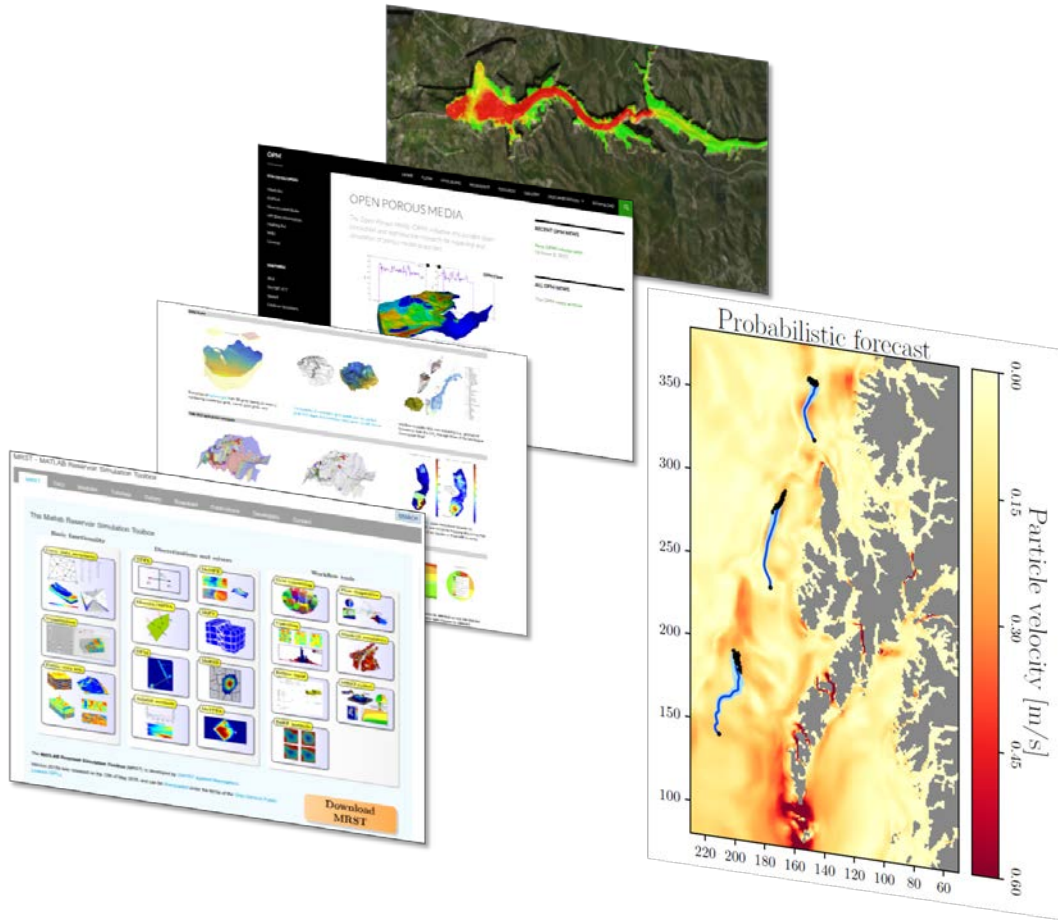
Florian Beiser

Coding Group 06.10.2021

Computational Geosciences Group

- Expertise:
  Cutting-edge numerical methods for PDE, history matching *(statistics!)*, and so on with applications in geoscience
- Open-source software:
  - https://www.sintef.no/mrst
  - https://github.com/metno/gpu-ocean
- MRST used in more than 400 publications
- 15 permanent researchers internally developing these software products (many more externally)

Organizing the collaboration is essential!



Central platform for version control and collaboration:

Before I start…

Preliminary comments

- Git is a helpful tool…
  …but I am not going to give a tutorial – just google the keywords and you will find a lot of help ;-)
- I do not have examples with R, since I will stay more on the surface, but it works the same…
  …Emma showed R-specific tricks already before! :-D
- The presentation is experience-based and not the one and only way…
  (I did a small survey in the CG group and even within us there are different habits)
  …please interrupt if you have comments, such that we all together here can find improved habits

The "Science" in Software Development

- Computational and applied research *(that is what most of us statisticians do!)* requires numerical simulations and practical implementation
- Research requires space for trying out, prototyping, discarding and expanding ideas – this transfers to scientific code bases
- … but the resulting code base should still be clean!

SINTEF

1. How to set the foundation?
    1. Kicking-off a new code development project
    2. Getting-in to an existing code base

2. How to organize the daily development?

Kicking off a code base

1. Initializing a new git repository
2. Making guidelines for collaborators clear from the beginning! It helps later on…

- README:
  - Requirements
  - Installation
- wiki:
  - Mathematical background
  - More detailed server set-up
  - Common APIs
  - Coding guidelines
  - Design patterns:
    - Nomenclature for classes, instances, functions, variables
    - E.g., use a paper for reference

Getting-in to an existing code base

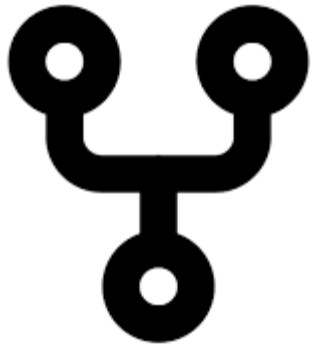
- **Read** the specific guidelines,
- **Install** the software.

Going further…

- **learn** from the tutorials,
- and **follow** the programming style.

Let's get started…

- **Fork** the repository
  (eventually, remember set different `origin` than `upstream`!) …
- … check out a new **branch** …
- … and you are good to go to the daily development!



SINTEF

1. How to organize the start?
    1. Setting-up a new code base
    2. Getting-in to an existing code base

2. How to organize the daily development?

Communication

Git issues or *Trello*, etc. for clear responsibilities,

- Identifying what new modules/features/etc should be
- Assigning tasks
  - In best case, working in separated classes or functions

Additionally, very important are informal tools for daily (better minutely) communication like *Slack,* etc.

- Exchange for questions
  - Enhances development efficiency a lot

Close communication is a key factor!

Know as much from you collaborators, that you know what you must do but also you must not do

Communication

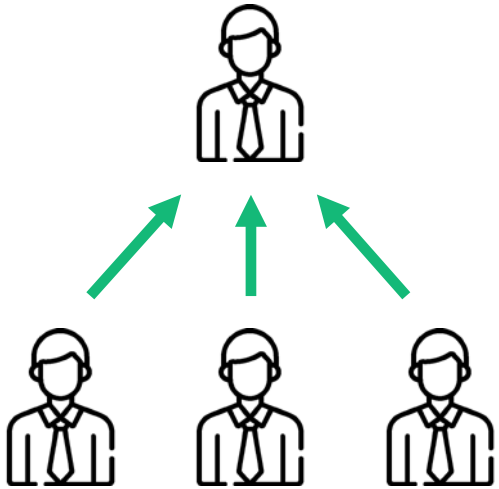
… I tried to add a screenshot from our Slack but it got unreadable small :-/

- Messages back and forth
- Code lines *(better link to git – you can create links to a specific line in a specific commit!)*
- References to new commits in different branches
- Immediate discussions about latest results
- Short calls
- (some jokes in between)

Potentially this makes Slack threads a mess, but we all know what is going on in the code

Pull Request

1. Individual or small group developments
2. **Pull requests:** one or a few are checking and allowing pull request



This is not a "big boss",
but just someone taking responsibility
for all checks

Pull Request: Requirements

- Short branches
  - The shorter, the less merge conflicts
- Explaining commit messages
  - Like a little news feed for co-developers
- clean code in the classes
  - *Always think you would write the code for someone else – it helps me the most to keep it clean*
- Tutorials
  - Notebooks to help new users to get started with every part of the code! (Documenting the way of use)
- Successful uni tests
  - Ensures workability of code base

It maybe sounds like unnecessary extra-work…
… but it will pay off when you have a bug

Clean Coding Principles

Should go without saying, but cannot be repeated often enough…
… especially since other people should be able to use the code, too!
(Or others can even help debugging then – can there be more incentive than that? :-P)

- Meaningful variable/function/class names
- Explanatory comments
- Refracturing
- Human readable structure
- …

(Clean Code – A Handbook of Agile Software Craftmanship, Robert Martin)

Final Comments

- Efficiency can also include the time needed for development among all researchers (make it easy for you colleagues, since we are stronger as team)
- Allowing for rapid prototyping makes many things easier to understand
- Open source code enables more collaborations…
  and can increase the efficiency and outreach of research

- Good start and good communication are keys!
- Successful software development is no marvel, if everyone follows the guidelines ;-)