FAST. FLEXIBLE. FREE.
GROMACS

Everything around coding

Berk Hess (KTH)

A historical overview

- A short history of GROMACS development to illustrate professionalisation of code development
- Communication
- Strategy
- APIs
- Challenges

GROMACS in 1995

- I joined the GROMACS team in Groningen (NL) in 1995
- I shared a room with the three other main developers: short lines
- At that time GROMACS was one of the first scientific codes using C
- We used **version control**: CVS (Concurrent Versions System)
- We tested the code on several different platforms
- No code review
- Already then we had a very good user manual, but no development docs

“Management” easy in early times

- Basically no management at all:
  - Every PhD-student had their own project and goals
  - No overall goals at all, Herman Berendsen left everyone free
  - No coding standards (at some point someone rewrote a tool in C++ which resulted in obfuscating everything)
    - Still code was generally of good quality
  - Relatively good testing led to reliable output

Growing development team

- Over the years we attracted external contributors, e.g. Erik Lindahl, likely because GROMACS was fully open source
  - Other MD codes had communities mainly consisting of scientific collaboration and former PhD-students of the code owner
- For several years I have been the gatekeeper of GROMACS:
  - judging if we want certain functionality at all
  - judging the reliability of the output of the code
  - judging the code quality
- This became too much as more developers joined and my career progressed

Management of scientific codes

- Management of scientific codes is often bad:
  - no clear goals
  - no common standards
  - new members often need to figure out everything by themselves or a PhD-student needs to explain everything
  - scientific codes tend to grow with features needed for every project: this quickly leads to an unmaintainable mess of code

Solutions for managing scientific codes

- Sell the code to a company ⇨ slow death
- Only allow new code when THE PI approves ⇨ slow death
- Embrace the community
    - this requires the right tools to make a (distributed) community work

Tools: automate everything that can be automated

- Code formatting: clang-format
- Automated unit/module testing: google-test framework
- Regression testing: an old perl script
- Static analyser to check code logic, memory/address sanitizers
- All this is checked/tested automatically for every change uploaded to GitLab

- Missing: automated validation testing (at scale)

Coding style

- Correct code can be written and organised in very many different ways

- We have style guides at:
  https://manual.gromacs.org/current/dev-manual/style.html

- Style is not checked automatically, but hasn't caused much issues

Code organisation

- More complex features require more complex code
- Often many classes are needed
  - How the the classes be organised?
  - How should the classes interact?
  - Often existing code is affected. Should that be refactored?
- Design discussions are needed here

Code reorganisation

- GROMACS is a combination of messy legacy C code and newer, better organised C++ code
  - In particular the main MD-loop has become too unwieldy
- When to ask someone contributing new code to refactor existing code?
- Who should design this refactoring?
  - Authors of existing code are often no longer active in the project

Communication channels

- GitLab
- The GROMACS/BioExcel developer forum: https://gromacs.bioexcel.eu/c/gromacs-developers/
- The bi-weekly GROMACS Zoom call, announced on the developer forum, topics are requested by the participants
- Slack

Communication is important!

- Communication channels for technical discussions
- But probably even more important: personal interaction
  - An external developer said that his changes went through much easiest after he had visited Stockholm
  - On the other hand, we have had large remote contributions (Mark Abraham, Roland Schulz, …)

Training of use of GROMACS

- Tutorials are the things most new user do
    - GROMACS tutorials on many topics available from many groups world wide
    - The past few years: coordinated effort within BioExcel for basic tutorials and new more advanced features
- GROMACS workshops, beginner or on specific topics requested by the organisers; nowadays often coordinated and/or sponsored by BioExcel

Training of coding of GROMACS

- We have had a few developer meetings
- Now the first workshop on learning to code in GROMACS

Direction of GROMACS?

Ideal strategy

- Long term goals guiding overall development directions
- Medium term goals, e.g. for next yearly release
  - Sets short term goals for features to get into the release
- This might actual work in a company (with sufficient resources)

(Lack of) strategy in science

- Scientists might have long plans, but no (stable) funding for them
- Medium term funding is distributed over many projects
- **People** doing the actual work on those projects **come and go**
- If science is involved, progress can vary a lot and is not guaranteed
- If it software engineering it is difficult to fund it

⇨ hard to plan and even harder to execute plans

GROMACS strategy

- Separate releases from feature planning
    - Timed, yearly releases: what is ready goes in

- Try to generate synergies between tasks in different projects

Current GROMACS funding

- BioExcel-3: code maintenance, user-driven support, training
- Several other EU projects: task focussed, but often produce code
- National grants: some contain algorithm development
- Some universities outside Sweden have people working on GROMACS
- NVIDIA: 2-3 people working at NVIDIA
- Intel: 1 person at Intel, 1 person at KTH
- AMD: soon one person at AMD

Wishes vs what we can achieve

- In academia we always like to achieve many more things that we can achieve
- In addition things can turn out to be more difficult and issues can arise
- GROMACS has quite some resources, but we are always “understaffed”
  - In addition it is very difficult to find suitable candidates for positions
- We can make little promises on allocating resources to external projects
  - We need dedicate some (review) resources to contributions from hardware vendors

GROMACS external contributions

- SIMD non-bonded kernel (Erik Lindahl)
- PME (Erik Lindahl)
- PME MPMD parallelisation (Carsten Kutzner, Göttingen)
- Parallel improvements (Roland Schultz, USA)
- Selection and analysis framework (Teemu Murtola, Finland)
- Enforced Rotation (Carsten Kutzner, Göttingen)
- Computational Electrophysiology (Carsten Kutzner, Göttingen)
- Advanced alchemical features (Michael Shirts, USA)
- Modular integrator (Pascal Merz, Michael Shirts, USA)
- QM-MM interfaces (Gerrit Groenhof, Finland)
- CUDA acceleration & parallelisation (NVIDIA)
- OpenCL GPU code, targeting AMD (contractor of AMD)
- SYCL for Intel and AMD GPUs (Intel)
- Python API (Erik Irrgang, funded by grants of Peter Kasson, USA)
- Constant-pH code, not in yet (Gerrit Groenhof, Berk Hess)
- Many analysis tools (many contributors)
- …

Code is a liability!

- GROMACS is about 750 000 of (non-external) code
- We read somewhere that one needs one person to support 75000 lines
  - So we would need 10 people only for code maintenance!
- In academia, people contributing code often disappear after a few years
- We want features and performance, not code!
- This is why full unit + module + regression test coverage is important

Code quality standards

- Nearly all quality aspects that can be checked automatically are checked automatically
- Our coding guidelines limit the number of C++ features allowed
- But there are still many, in particular organisational, aspects of the code that can be handled in different, better or worse, ways
  - We strive for high code quality
  - But we should not have not too high requirements, especially for new contributors

APIs: solution to everything!?

User facing API(s)

- For users it can be very beneficial to have access to a Python API for setting up simulation workflows
    - No more bash scripts needed
- If done well, can be much more efficient by keeping things in memory

For developers: lower level APIs

An API:

- Clearly separates responsibilities
- Standardises interactions of modules with the rest of the engine
- Should not be changed, can be extended
  - No more porting of external features to newer GROMACS versions
  - No issues internally when the engine is refactored
- External contributions can more easily be managed externally

APIs can separate responsibilities

- Instead of you asking: where do I need to put functionality in GROMACS?
- Does the API support the needs of my functionality?
  - Maybe the API needs to be extended
- Functionality can be in (external) modules and maintained separately from main GROMACS
  - No increasing burden on the main GROMACS team (apart from API support)

Challenges with APIs

- Where to start?
    - We currently do not have an API expert
- Old GROMACS code often needs to be refactored to enable a simple API
    - Currently we can not return to high up in mdrun, modify something and continue the run
    - In general: we need to make sure that the engine does not have memory of the old state of the system/parameters after they get changed through the API

Accelerating effects of APIs

- With some basic API(s) present:
  - Users & developers can play around
  - New needs become clear
  - Needs will be more specific (as opposed to asking where in the GROMACS codebase do I need to hack in my change)
  - You can contribute to extending the API

GROMACS specific challenges

- GROMACS has (tens of) thousands of users world wide
  - Impossible to keep track of
    - We use polls to get an idea of their needs, useful, but limited in coverage and depth
- GROMACS probably has hundreds of developers world wide
  - We know a few and interact more or less with then
  - Most we likely don’t know about and we don’t know their needs