Are you writing software for science?

Center of Excellence for Computational Biomolecular Research

Do you need to write correct software for your science?
Will your software need to change over time?
Do you need to make your software faster, or more scalable?
How will your users learn how to use your software?
How will you convince people to invest in your software?
What will happen when the original author moves to a new project?

All software projects share these concerns, and in BioExcel (https://bioexcel.eu/) we are tackling them head-on. Our users need automated workflows for biomolecular simulations that can be shown to work correctly when run on exascale resources. This requires high quality software, but even the most outstanding projects will quickly degrade in quality as they grow more complex – unless you also have a high-quality *software-development processes*. The BioExcel consortium develops and maintains several software packages, including GROMACS, HADDOCK, pmx, and CPMD QM/MM. These codes address different problems, are written in different languages by different sub-teams, delivered to users in different ways, and all have unique challenges in identifying good process and seeking to improve it. They are neither perfect nor necessarily the best examples of software development, but they are real-world examples of major complex scientific software packages that have adopted more or less advanced formal software development processes.

Your software project will benefit from reflecting on your process and developing a plan to improve it. Some areas will be easier, or more urgent, to address than others. Accordingly, in this white paper, we record the current state of the software-development processes within BioExcel, along with plans. These "worked examples" will give you real insight into state-of-the-art scientific software development activities and guide you to make good choices for your team. We hope that these ideas will help you develop correct software faster and cheaper, be portable, attract users, and demonstrate to funders that your software is worth supporting.

Your plan will need to be customized. Your software must reflect your needs, the needs of your other users, and the available resources. But some aspects are common to almost all software projects, so we will discuss these first and suggest external resources for grappling with them. Then we will briefly describe the four software packages whose processes are described herein, so you can then focus on the one that best fits your needs. Finally, the details of the process for each package will be given.

Common elements of software development processes

Most scientific software projects should have a plan for all the following aspects. There is often useful “low-hanging fruit” to consider for improvements.

- Agree on a **license** model for your software, remembering that programmers, their employers, and their funding agencies may have expectations that need to be negotiated. Mention the license in your documentation and provide it in full alongside the software. Licenses are tools, so pick a license that will help the project achieve its goals.
- Make frequent formal **releases**, so that your users can benefit from your new work, and you have an accomplishment to report. Don’t wait until it’s “ready” – software never can be until real users have used it for real work.
- Use **version control** during development, so that experiments can be rolled back, bugs more easily found and tracked, and your science can be reproduced.
- Document the **dependencies** – what versions of tools, compilers, libraries and operating systems are required? What is supported? What is tested? Which other version, libraries and/or tool might work but aren’t tested? How do you decide on what new dependencies can be introduced?
- Plan code changes before rushing in – **gather requirements** from your users first, so that you build a thing that is useful while avoiding designs that block future changes. You won’t be able to do everything that they want, but you should listen for ideas and opportunities and prioritize them.
- Use unit tests, and **design for testability**. Before starting implementations, separate features into small methods that can be tested exhaustively to guarantee correctness. Write the tests first – the implementation is finished when it passes the unit tests.
- Design **performance tests** relevant to current real-world usage, and use these to guide profiling experiments, plan optimizations, check for regressions, and support announcement of performance improvements. Keep a balance between the needs of researchers now, and the direction in which HPC platforms are evolving.
- Document the **stability of features** so that users can know what is experimental, and what is reliable.
- While features are assets, **all code is a liability**. No matter how amazing new code is, it will have to be maintained, tested, updated, and documented – which takes time. Is the new feature absolutely worth this time – and who will maintain it next year?
- **Remove less used features** to reduce the complexity of code and simplify maintenance. All members of a team need to share the responsibility for cleaning and maintenance, or the ones doing it will eventually leave the project.
- **Announce deprecation** in advance, before removing support for a feature, so that users are aware that their needs are considered, and perhaps others will contribute to the project to keep something that they need.
- Publish a place to **record issues with the software** – this can be as simple as a shared Google document or wiki page. Users might want to help understand if there is an issue and how it might be resolved.

- Use **formal code review** so all changes are checked and approved by somebody else in the team. This might initially slow down your commit rates, but it will reduce the number of bugs drastically and increase your overall development pace.
- Consider **adopting automated review & integration tools** even for small projects. There are many cloud services that provide this at low cost.
- Adopt a common **coding style**, so that your developers can read and maintain different parts of the code easily, and your code reviews can focus on the substance of the change, and not where tabs and punctuation should go.
- Consider adopting formal user-experience (UX) design processes, e.g. as documented in the User Experience for Life Sciences (UXLS) toolkit,[1] so that your product will best fit the actual needs of your users.

There are several excellent publications and knowledge bases for scientific software development. The UK-based Software Sustainability Institute (https://www.software.ac.uk/) has one of the best and most up-to-date collections of guides you will find useful (see https://www.software.ac.uk/resources/guides).

Software development process and readiness level

There is no unique "correct" level of software, but it is important to be aware of the extremely broad range and differences between early scientific concepts, test implementations, software that can be used internally with tight feedback loops with the developers, limited external usage, and widespread deployment in external production environments with adequate support, training and quality assurance in place.

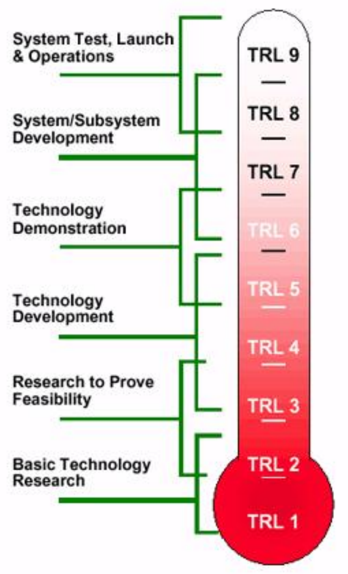
System Test, Launch & Operations
System/Subsystem Development
Technology Demonstration
Technology Development
Research to Prove Feasibility
Basic Technology Research
TRL 9
TRL 8
TRL 7
TRL 6
TRL 5
TRL 4
TRL 3
TRL 2
TRL 1

Figure 1 NASA illustration of Technology Readiness Levels

Both the US Department of Defense, NASA, and the European Commission have formulated "Technology Readiness Levels" (TRL).[2] The scale ranges from TRL1 ("basic principles observed and reported") to TRL9 ("actually proven through successful mission operations"). Although not specifically developed for software or scientific applications, it is a good idea to consider what level your software project aims to achieve. The higher levels are difficult to reach without extensive investments in documented development processes. Simple processes suit projects whose scope is small, or lifetime short.

[1] UX Toolkit for Life Sciences https://www.uxls.org/

[2] http://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf Technology readiness levels (TRL), HORIZON 2020 – WORK PROGRAMME 2014-2015 General Annexes, Extract from Part 19 - Commission Decision C(2014)4995.

In future work, BioExcel will evolve guidelines for development process that will help achieve higher levels of readiness. Commercial and academic users should be able to understand the readiness of any technology proposed for use in a project, as part of their risk assessment. For example, deploying simulation software on a problem type different from the test suite accepts a higher risk of failure.

Brief description of the BioExcel software packages

These brief descriptions will help you identify which of the detailed descriptions is likely to have useful process ideas that you might adopt, because they come from a software team working on similar software, languages, libraries, or delivery process.

- **GROMACS** (http://www.gromacs.org) combines a high-performance classical molecular dynamics simulation engine with powerful tools for preparation and analysis. It is released under a liberal business-friendly open source license (LGPL 2.1) and comprises 2 million lines of C++11 targeting all HPC platforms. Development is led by a core team at KTH in Stockholm and enjoys active contributions from a wide range of other academic and industrial partners. A formal code-review process is accompanied by pre-submit continuous integration testing on several different OS, libraries, and compilers.
- **HADDOCK** (http://haddock.science.uu.nl) implements integrative modelling of biomolecular complexes based on the CNS engine together with a set of extension scripts that incorporate information from a wide range of experimental sources. The code consists mainly of a Python layer orchestrating all the computations performed with CNS. A large fraction of the docking protocol itself is written in the CNS scripting language. Users typically access HADDOCK via its grid-enabled web server, supported by several EGI partners in The Netherlands, Europe, and other suppliers worldwide. The team at Utrecht University handles development of both the core software and the web server.
- **CPMD QM/MM** (http://www.cpmd.org) effort within BioExcel is a new combination of CPMD (written in Fortran 90) with GROMACS, implemented at the Juelich Research Center. The new QM/MM interface and its communication library designed to manage the data exchange between the two codes, is being written in Fortran2008 and C++11, respectively. This interface will allow one to couple CPMD to GROMACS (and later other classical molecular dynamics codes) with a minimal core of changes so that both codes can run on different MPI processes on supercomputers.
- **pmx** (http://pmx.mpibpc.mpg.de) is a preprocessor for molecular dynamics software that builds efficient molecular topologies for alchemical free energy calculations to predict stabilities and affinities. It is tightly bound to GROMACS development, facilitating massively parallel free energy calculations. pmx can be run both standalone as well as from a webserver. The core and infrastructure are maintained by the Max Planck Institute for Biophysical Chemistry in Göttingen.

Development process details for GROMACS

GROMACS[3] is a molecular dynamics simulation engine that can simulate the Newtonian equations of motion for systems with hundreds to millions of particles. It is primarily designed for simulations of biochemical molecules like proteins, lipids and nucleic acids. Its chief virtue is that it is extremely fast at calculating the expensive non-bonded interactions necessary for such systems. It is a state-of-the-art best-in-class implementation of molecular dynamics, with high-performance implementations for a wide range of commodity CPUs and GPUs, including all current and several anticipated future HPC platforms. It is in use by thousands of scientists, leading to citations of associated scientific articles currently numbering more than 2000 per year.

Figure 2 depicts the current state of GROMACS code development process. Having made appropriate plans before writing code, developers upload proposed changes for review on our Gerrit code-review server, triggering automated continuous-integration testing via Jenkins, and awaiting review from other developers. Automatic cross-references to the Redmine bug reports and feature requests are made. Once code is accepted into the master branch of the repository, it is automatically pushed to our Github backup repository.

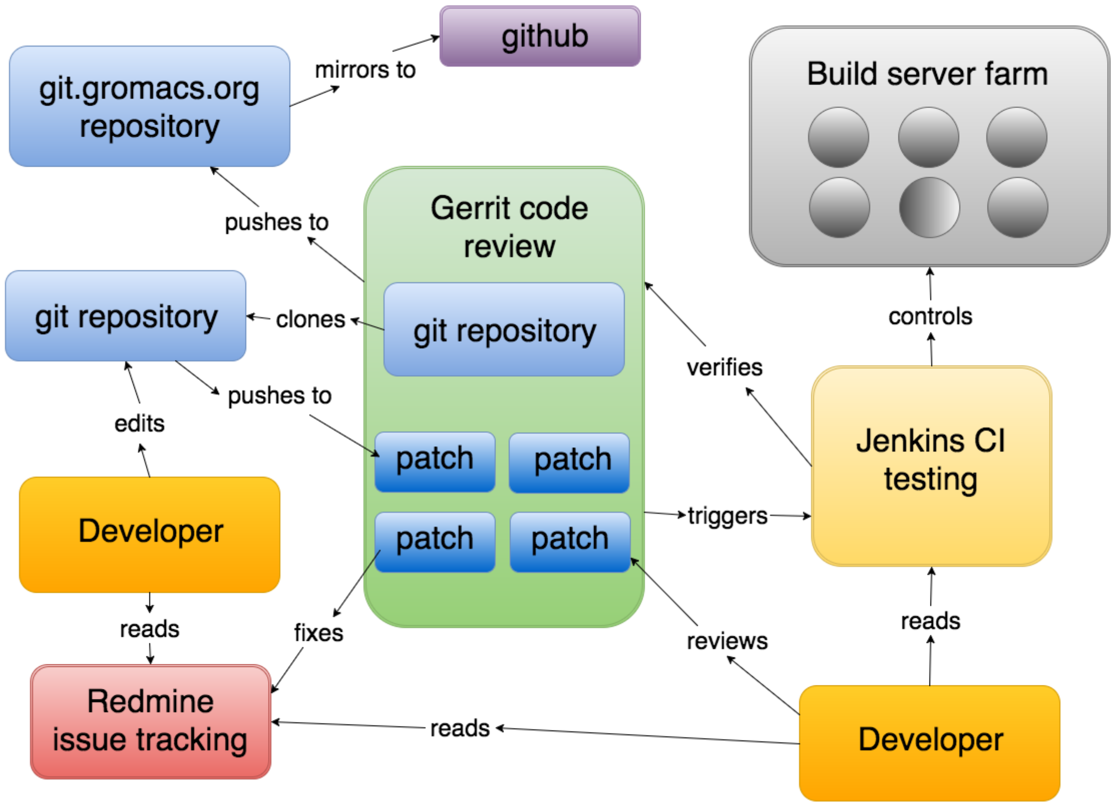

Figure 2 Current GROMACS development workflow description

[3] Abraham, M.J., et al., *GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers.* SoftwareX, 2015. **1–2**: p. 19-25.

Copyright and License

GROMACS is released under the Lesser GNU General Public License (LGPL) v2.1. This is an intentional choice to balance our requirements for all extensions to the main codebase to be freely available (which is important for our academic impact), while still enabling commercial applications to link with and use GROMACS as a library. Each source code file has a copyright statement, which is automatically updated for each applicable year that it changes, which is sometimes recommended by legal experts to ensure the license is effective. Source tarballs contain the appropriate COPYING file, that additionally notes the origin of code bundled along with GROMACS, and the license under which it is redistributed.

Architecture

GROMACS includes a high-performance simulation engine mdrun, several independent tools for preparing simulations, and a suite of around 50 post-simulation analysis packages. Currently all are run on a command-line terminal from a common gmx wrapper binary by naming the required sub-tool. This provides a small surface area for potential conflicts with other software on a user's system, and many convenient features including the ability to deprecate or rename tools while still retaining a way for users to learn whether and where the old functionality is still available. Having a single entry point to the library will also help our future transition to provide runtime dispatch with several different dynamic libraries optimized for different types of hardware. Much of the approximately 2 million lines of code is specific to mdrun, and much is shared across all the tools, and the long-term future of the package will require extensive reorganization to reflect which components support which functionality and thus require appropriate focus when developing and maintaining the software. However, no substantial changes to the user interface are planned, although simplifying the performance-oriented interface to mdrun would offer substantial improvements to user results and experience.

Feature request handling

Current: Anyone, including regular developers, may search and open issues at https://redmine.gromacs.org to seek interest, find common ground and identify strategies. Usually proceeding to actual development will require someone actively finding a source of funding and a suitably experienced developer, and enough time and enthusiasm in the rest of the developer community for testing, code review – and a long-term plan for how the feature will be maintained.

Future: In future phases of BioExcel, provide opportunity for clients to discuss e.g. development of new features on a contract basis. Given the complexity and size of the code, we will also start requiring that all new major features are initiated as a discussion in a Redmine feature-thread, so it is possible to request significant changes to the architecture before development starts.

Bug tracking

Our Redmine server https://redmine.gromacs.org also allows any user to register an account and file a bug report, along with uploaded input files. Developers regularly browse for relevant issues and engage in productive discussions with the reporter(s) and each other. When fixes are uploaded for code review, they can cross-reference the bug report, and automatic HTML cross links are created. Release notes can then note that the bug was fixed and provide links to more detail as required.

Version policy

Formerly: GROMACS used a major.minor.patch numbering system (e.g. 4.6.3). Patch releases could only fix things that didn't work. Ad hoc decisions led to a new version being described with a major or minor version increase.

Current: GROMACS numbers each release as year.patch (e.g. 2016.1) so that it is clear to users how old their version is, and that the release was made because it was time to do the annual release. Since software development can generally produce only two out of three of high quality, on time, or with specified functionality, we focus on the former two so that users can benefit from the new functionality that has been completed, without risk of open-ended waiting for other functionality. This provides the wider development team with confidence that an accepted feature will get into the hands of users without having to wait for completion of some other feature intended to go into the next release. However, it also means it is the responsibility of each developer to make sure their feature is not only ready on time but has gone through code review and been accepted by the team before the release deadline.

Future: No changes are planned. Current resources do not permit more than one annual major release, combined with patch releases every few months.

Version control

Git is used for version control. All code development should start from the official git repository (git://git.gromacs.org) or one of its mirrors (including https://github.com/gromacs). Developers modifying GROMACS will benefit from being able to use git tools such as grep and rebase to work with the code. Code development should not start from a release tarball, but if this has happened then it can be copied onto a git repository with that version checked out.

Branching

A stable master branch will exist and is generally open for both refactoring and functionality changes, including adding and removing features. Stability will be assured through code review and continuous integration testing. No features can be committed until they pass code review, provide full documentation both of source code and user-facing features, and pass all integration tests – the master branch is not allowed to decay between releases. When it is time for a new annual release, a named branch is made to reflect this. No changes to the scope of functionality are expected after this time, while any issues of module integration are resolved before the actual release. After the release, fixing may continue on the release branch according to policy (no new features are allowed in any release branch, and only critical fixes in old release branches), and such fixes will be merged into any more recent release branches, and then into the master branch.

Modularization

Current: GROMACS is making a transition from C89 to a modular, unit-tested modern C++11 library. This transition is slow and costly, but expected to deliver long-term benefits from lower maintenance, better extensibility and higher portability. The classes within these modules encapsulate behavior alongside the data necessary to implement it, and require full Doxygen[4] documentation of files, classes, members and methods. The modules follow a layered design that is enforced by checking scripts.

[4] http://www.doxygen.org

Future: Draft a target module hierarchy for the wider development community to use as a mental model during the refactoring process. All new modules will be required to have 100% unit test coverage before they can be committed.

Development process details

Current: Developers are generally autonomous academics who propose ideas on the gmx-developers mailing list, or on Redmine, and interested other developers contribute ideas. These evolve on an ad hoc basis. Developers progress to develop working code which is then uploaded for peer review on our Gerrit review server at https://gerrit.gromacs.org. Developers can propose code changes that are tagged “request for comment” or “work in progress” so that reviewers know that high-level feedback is needed now. An ongoing challenge is to motivate others in the developer community to contribute to design and review stages in a timely fashion, because most developers are academics with multiple responsibilities and GROMACS development is only a small fraction of their output (and not always clearly recognized for career progression). However, since all developers are subject to the same review and testing process, all do already recognize that nobody can progress in isolation.

Developers require active encouragement to remember to *separate changes that refactor the code from those that change the functionality*, because the time of code reviewers is a particularly scarce resource, and such separation maximizes the value delivered by reviewer time.

Future: BioExcel developers will lead the way in producing a written plan, in particular for how the user interface will evolve alongside features.

Review Process

Current: All code changes go through two-person review at https://gerrit.gromacs.org, one core developer, and one other developer. Preferably at least one of those has previously contributed to code in this area, so they are able to make an informed review reasonably efficiently. Review is difficult for both parties, so feedback and responses need to be considered carefully and should be technical and impersonal. Proposed changes do not have to be perfect to pass review, but they should represent a clear improvement in at least some aspect without undue compromise on other aspects. Complete Doxygen documentation is a strict requirement for all new code. Authors may declare a reviewer suggestion out of their intended scope, in which case the discussion should move to a Redmine issue for future consideration. Authors may negotiate that someone else takes over responsibility for the patch.

Future: The most important quality of scientific code is its correctness at implementing the method claimed. Only a subset of kinds of proposed changes can compromise this with GROMACS. It is inefficient to require the same level of developer scrutiny on all changes, particularly when the number of developer hours available to the project is strictly limited. In concert with improved testing, identify areas of the code for which greater risk is acceptable, e.g. refactoring of modules already under high quality unit tests, improvements to user or developer documentation. For major changes to older modules, we will gradually start requiring that it is combined with rewriting the code as proper C++11, with complete unit test coverage.

Integration approach

Current: No long-term development branches exist, so proposed code changes are normally based off a recent master-branch commit, and thus can be readily rebased for integration with the current HEAD commit of the master branch. It is the responsibility of each developer of a feature to track and integrate this feature as the stable master branch evolves in parallel with his/her development.

Future: Some larger development efforts may benefit from a long-running feature branch, which would then need to be integrated back into the master branch. A git merge is perfect from a technical point of view but does not meet the needs of a policy where code review must occur before acceptance into the master branch. Given the limited developer effort available for code review, code needs to be presented for review in small pieces that each passes its own unit tests, and any available integration or end-to-end test. Any move to implement a long-running feature branch must present and justify and process that would lead to effective review.

Testing requirements

Current: All proposed code changes are tested automatically by our Jenkins continuous-integration server at http://jenkins.gromacs.org. It builds the code and runs the tests on multiple compilers, operating systems, standard libraries, accelerators and CPU architectures. This caters directly to ensuring long-term portability of the code base to the anticipated features of the exascale landscape. Several static analysis tools are also run. All tests must pass before a change can be acceptable. The range of tests and tools are reviewed and updated continuously[5]. Release versions, e.g. source tarballs, are tested on a range of configurations before being made available to users.

Future: Since the range of testing needs to expand, we wish to use more than one tier of testing, to limit the number of machines required for doing the builds. Fast-running tests will run for each proposed code change on important platforms, and longer running tests will also run on a wider range of platforms once the proposed code has been accepted.

Specific activities to be achieved over the course of BioExcel are:

- expanding test coverage (particularly including rerun, multi-simulation, and replica-exchange functionality),
- replacing old testing Perl driver script with GoogleTest C++ driver,
- deploying automated performance regression testing,
- deploying Docker-based multi-tier CI builds to streamline the implementation, giving developers faster feedback,
- continuing to add support for warning-free builds for up-to-date compilers, code analyzers, and libraries,
- deploying pre-release testing suite that verifies correctness of ensembles produced, perhaps using the Copernicus workflow engine, and
- adding support for the Memory, Leak and UndefinedBehaviour Sanitizer code-analysis tools to the CI configuration,
- report annual increases in test coverage.

[5] http://jenkins.gromacs.org/job/Documentation_Nightly_master/javadoc/dev-manual/jenkins.html

Release Process

Current: Formal releases will only be made from release branches and will follow the versioning scheme in use at the time that branch forked from the master branch. Generally, at least one release candidate is made available for the community for around a month of informal testing before the final release. During this time users and developers are encouraged to attempt to validate that their simulations of interest work at least as well (or better) than previously and test new features. Debian and Fedora projects already test GROMACS on a wide range of platforms, and they will be invited to help with this effort. The eventual source and tests tarballs for the release are built by Jenkins from a commit in the repository that will later be tagged with the release number. That tarball is automatically tested by the Jenkins continuous-integration server on a range of installation configurations, to verify that the build works and the tests pass. The documentation that matches the release is automatically generated from the tarball and prepared for easy deployment to web servers. The stages of the release process are shared with the wider community through emails to the users, developers and announcement mailing lists, posts on social media outlets (Facebook, Google+, Twitter), and both the GROMACS and BioExcel websites.

Future: Finalize automation of final details of an automatic release build, including construction and deployment of the release notes, and embedding tarball checksums in the appropriate locations.

Deprecation

When a feature is identified as superseded, is no longer functioning, or is lacking developer support for maintenance then after due consultation with the user and developer community, it will be deprecated. This means it will be announced as deprecated in the next annual release, so that users who run the code have warning that they are at risk of depending on code that will not be maintained in future. Thereafter it may be removed in the master branch at the discretion of the developer community, which means it will not be part of the major release another year later. When code is known to have been broken for multiple years already, and thus cannot be in active use in a maintained branch, it might be removed without a deprecation notice in the software, but this will be acknowledged in the release notes of the next annual release. Support for older hardware platforms is also removed gradually, to make effective use of developer time porting to new hardware; older versions of the code still run on the older hardware for those who cannot upgrade usefully.

Retirement

Current: Each (annual) major release is supported until the next major release (i.e., typically one year) on a best-effort basis by developers to fix any shortcoming in the implementation of functionality intended to be supported in that release. That includes code correctness, accuracy and presence of documentation, functioning of build system, and simulation performance, whether reported by users or developers. The stability of related code paths will form part of the decision about whether, where and how to fix a bug. If a bug fix is not feasible, then we will alter the code to prevent future releases from running that wrong code, and report this to the user in subsequent bug-fix releases. After one year, there will be a further year of best effort to fix any severe issue that would produce scientifically incorrect results (whether in mdrun or tools).

The developers reserve the option to fix a bug only in the master branch if that seems the most practical course. Because large scale refactoring of the code base is underway, it is impractical to remember all the details of several implementations that were used in past years, current implementations, and the proposed future implementations, and bugs and developer conflicts have been introduced because of such lapses in understanding.

Future: It may become feasible to support release branches for longer periods. We have identified several improvements that will make this feasible; the transition to a modular C++11 library needs to be substantially complete, the test infrastructure must be contained in the source-code repository, an automated suite of performance tests must be available and automated, and a wide range of simulation quality tests must be available and automated. When a GROMACS library API is developed (which is a key outcome of a funded NIH project), then we will consider whether the point of long term support becomes the version (or level) of the API, rather than the release version, and again the existence of automated testing and the stability of the GROMACS source-code infrastructure will be key components in this decision.

Code commenting

Code is commented with a view to explaining what the code is doing, rather than how. If it is not obvious how the code is working, it should be split into smaller methods with explicit long names for variables and methods, use better control flow, and have examples with working tests. Care should be applied to document the interfaces of methods separately from internal implementation comments. New and newly refactored methods and source files must have Doxygen-style comments that are automatically built into the developer guide[6]. The Jenkins continuous integration environment enforces these requirements, and code that does not follow it cannot be committed.

Coding standards and styles

Current: The specification for code style[7] is part of the codebase, and proposed changes will undergo the same type of review as source code changes. All simple requirements (spacing, indentation, etc.) is checked automatically during continuous integration. The GROMACS style is loosely based upon the Google C++ Style Guide[8], with the notable exception that we permit the use of C++ exceptions. In practice, we will avoid writing code that might throw exceptions in our performance-sensitive kernels because we have great experience in writing these kernels such that there will be no checkable error conditions.

Future: Update these to account for new recommendations in the CppCoreGuidelines[9] following the best practice and wisdom of the larger modern C++ developer community, who largely share our interests in correctness and high performance, by default and by construction.

[6] http://jenkins.gromacs.org/job/Documentation_Nightly_master/javadoc/#documentation-for-developers

[7] http://jenkins.gromacs.org/job/Documentation_Nightly_master/javadoc/dev-manual/style.html

[8] https://google.github.io/styleguide/cppguide.html

[9] https://github.com/isocpp/CppCoreGuidelines

Development process details for HADDOCK

HADDOCK[10] is an integrative, information-driven docking approach that supports a large variety of data from biochemical, biophysical and bioinformatics methods (e.g. mutagenesis data, NMR chemical shift perturbations, cross-links from MS, EPR-derived distances, cryo-EM densities, and bioinformatics co-evolution predictions). The software is made available through a user-friendly web interface[11,12] which has attracted a large user community worldwide (10000+ users) and resulted in over 120 deposited structures of complexes in the PDB. HADDOCK has demonstrated a strong performance in the blind docking experiment CAPRI, belonging to the best performing approaches and is currently the most cited software in the protein-protein docking field. The HADDOCK software is available both for download and as a web-server. The stable release of the software and web server is HADDOCK 2.2.

Internally, HADDOCK is separated in two components, whose interdependence is depicted in Figure 3. A high-level Python layer manages job submission and pre- and post-processing steps. The webserver implementation adds several pre- and post-processing steps to this Python layer that are currently not available in the standalone version of the software. The performance-critical molecular mechanics computations and structural and energetics analyses are performed using the CNS (Crystallography & NMR System) engine (http://cns-online.org), which includes its own scripting language. This high-level language means that in most of the cases, development does not require coding in the original language of CNS (FORTRAN77). Some FORTRAN77 routines specific to HADDOCK are provided with the standalone software and require re-compiling the CNS executable. The CNS software comes with its own suite of tests that should be run after recompilation. As such, we trust that if those tests are successfully passed, CNS will provide reliable results.

[10] C. Dominguez, R. Boelens and A.M.J.J. Bonvin HADDOCK: A protein-protein docking approach based on biochemical or biophysical information. J. Am. Chem. Soc., 125, 1731-1737 (2003).

[11] S.J. de Vries, M. van Dijk and A.M.J.J. Bonvin The HADDOCK web server for data-driven biomolecular docking. Nature Protocols, 5, 883-897 (2010)

[12] G.C.P van Zundert, J.P.G.L.M. Rodrigues, M. Trellet, C. Schmitz, P.L. Kastritis, E. Karaca, A.S.J. Melquiond, M. van Dijk, S.J. de Vries and A.M.J.J. Bonvin. The HADDOCK2.2 webserver: User-friendly integrative modeling of biomolecular complexes. J. Mol. Biol., 428, 720-725 (2016).

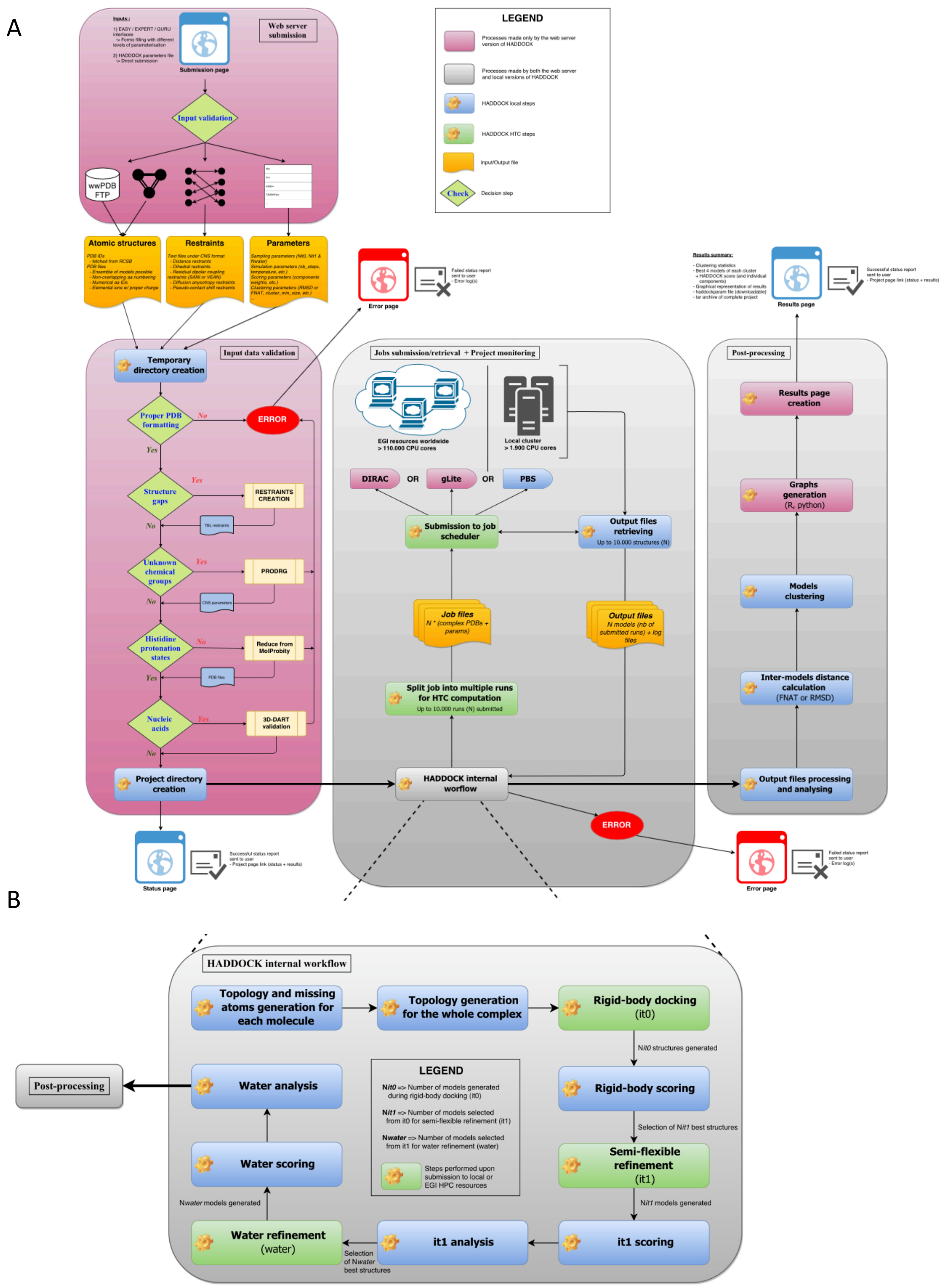
A
Web server submission
Submission page
Input validation
wwPDB FTP
Atomic structures
Restraints
Parameters
LEGEND
Processes made only by the web server version of HADDOCK
Processes made by both the web server and local versions of HADDOCK
HADDOCK local steps
HADDOCK HTC steps
Input/Output file
Check
Decision step
Error page
Results page
Input data validation
Temporary directory creation
Proper PDB formatting
No
ERROR
Yes
Structure gaps
RESTRAINTS CREATION
TBL restraints
Unknown chemical groups
PRODRG
CNS parameters
Histidine protonation states
Reduce from MolProbity
PDB files
Nucleic acids
3D-DART validation
Project directory creation
Status page
Jobs submission/retrieval + Project monitoring
EGI resources worldwide > 110.000 CPU cores
Local cluster > 1.900 CPU cores
DIRAC
OR
gLite
PBS
Submission to job scheduler
Output files retrieving
Up to 10.000 structures (N)
Job files
Output files
Split job into multiple runs for HTC computation
Up to 10.000 runs (N) submitted
HADDOCK internal worflow
Post-processing
Results page creation
Graphs generation (R, python)
Models clustering
Inter-models distance calculation (FNAT or RMSD)
Output files processing and analysing
B
HADDOCK internal workflow
Topology and missing atoms generation for each molecule
Topology generation for the whole complex
Rigid-body docking (it0)
Nit0 structures generated
Post-processing
Water analysis
LEGEND
Nit0 => Number of models generated during rigid-body docking (it0)
Nit1 => Number of models selected from it0 for semi-flexible refinement (it1)
Nwater => Number of models selected from it1 for water refinement (water)
Steps performed upon submission to local or EGI HPC resources
Rigid-body scoring
Selection of Nit1 best structures
Water scoring
Semi-flexible refinement (it1)
Nwater models generated
Nit1 models generated
Water refinement (water)
it1 analysis
it1 scoring
Selection of Nwater best structures

Figure 3 Workflow description of (A) the HADDOCK web portal, and (B) the internal workflow of HADDOCK itself.

Development workflow

The workflow used by the HADDOCK developers when modifying the code is depicted in Figure 4. Most of the development in HADDOCK takes place at the CNS scripting level, such as the recent implementation of a cryo-EM restrained docking protocol. The parsing of data, structures, and the actual docking calculations and analysis are coded in roughly 80,000 lines of CNS scripts. A description of the various stages of the docking protocol, with reference to the CNS scripts called, is provided in the online manual[13]. Changes to those scripts require running a suite of tests to ensure that the code does not produce wrong results for pre-existing scenarios (see testing requirements below). Changes at the Python level are less frequent and mostly required to accommodate new types of experimental data or changes to the general workflow (e.g. number of molecules supported), as was recently the case[14].

Most of the developments in HADDOCK over the years have been triggered by scientific questions posed by users and by the core team when working on collaborative projects. Smaller developments occur more often at the level of the force field and associated topologies, namely adding support for new molecules and amino-acids post-translational modifications. These updates are usually triggered by user requests and in the current development process, these will not typically trigger a new software or web server release. Any new addition to the supported molecules and modifications thereof is added to the server page providing the list of supported modified amino acids[15].

[13] http://www.bonvinlab.org/software/haddock2.2/docking/

[14] E. Karaca, J.P.G.L.M. Rodrigues, A. Graziadei, A.M.J.J. Bonvin and T. Carlomagno. An Integrative Framework for Structure Determination of Molecular Machines. Nature Methods 14, 897-902 (2017).

[15] http://haddock.science.uu.nl/services/HADDOCK2.2/library.html

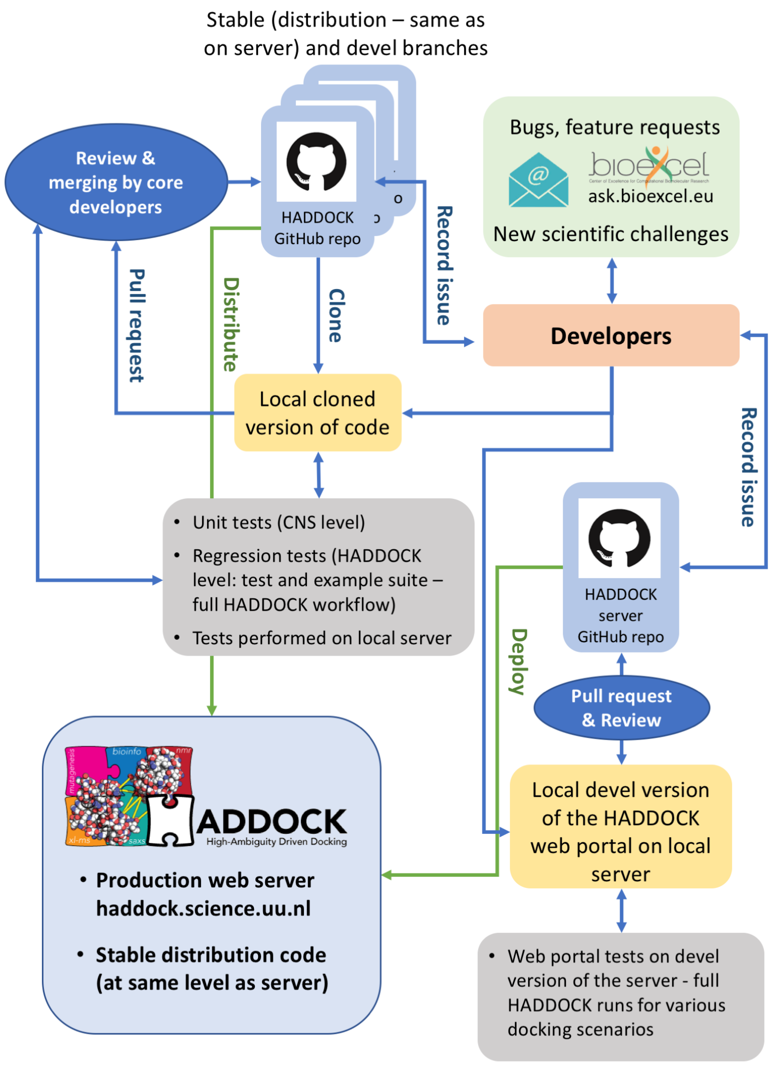
Stable (distribution – same as on server) and devel branches
Review & merging by core developers
HADDOCK GitHub repo
Bugs, feature requests
bioexcel
ask.bioexcel.eu
New scientific challenges
Pull request
Distribute
Clone
Record issue
Developers
Local cloned version of code
Record issue
Unit tests (CNS level)
Regression tests (HADDOCK level: test and example suite – full HADDOCK workflow)
Tests performed on local server
HADDOCK server GitHub repo
Deploy
Pull request & Review
HADDOCK High-Ambiguity Driven Docking
Production web server haddock.science.uu.nl
Stable distribution code (at same level as server)
Local devel version of the HADDOCK web portal on local server
Web portal tests on devel version of the server - full HADDOCK runs for various docking scenarios

Figure 4 HADDOCK current development workflow

Retirement

Current: Historically, the lifetime of a release is undefined. While the development version of HADDOCK is constantly updated as new features are added, the officially distributed standalone version is kept as tied to the web server version, since this allows for a much smoother user support by pointing users to the web portal in case of local installation problems or setup issues.

We use a major.minor version numbering system, where changes in the major version do not imply backwards compatibility. The current version is 2.2. The previous version, 2.1, was discontinued in 2017 after a two-year deprecation period. Users and third-party developers making use of the HADDOCK portal were given a final six-month notice before the removal of the version from our web portals.

Future: Our participation to BioExcel has made us aware of the need for a clear policy in supporting software and web portal versions. For future releases of the web portal and associated standalone version, we will implement a retirement policy that will support and maintain a previous version of the software and web portal for a maximum of two and one years, respectively, after the release of a new version of those. While one year might seem rather long for the web portal, our web portals are not updated that often (except for minor fixes), and this is also needed since the duration of research projects might be quite long and it is important to allow users to complete their research work using one and the same version of the software for consistency.

Deprecation

Current: No policies in place for deprecation. Users with older versions of the software requesting support are encouraged to upgrade to the most current version.

Future: All references and entry points to the various web portals will be updated to point to the latest official release and a note will be added to the older versions clearly stating the end date of the service and support (at least 6 months before the planned retirement). Third party software developers relying on the web portal will be notified directly after the release of a new version of the portal.

Release Process

New features are implemented in local development versions of the software and then added to the web frontend. The translation of local to web portal is the limiting factor in our release cycle, since the framework used to build the portal is not scalable, custom-made, and poorly documented. We are currently rewriting the web portal to 1) make use of the Flask framework[16] and 2) automatically update the web forms from the standalone software. Together, these changes should allow much faster implementation of new features in the web portal.

All components of the HADDOCK software are version-controlled using Git and GitHub. Each major release is stored under its own branch. Development of new features usually requires forking and creating new feature branches (e.g. cryo-EM support), while small bug fixes and improvements are directly committed to the main branch after testing. Each developer works on a fork of the main repository and issues pull requests when his changes are ready to be merged. Rights to commit changes is granted to a small number of core developers. Acceptance of a pull request requires the approval of at least one other core developer, although minor bug fixes and changes are often accepted and committed without this revision process. We require all changes to be properly documented, in terms of commit messages, and self-contained, to ease making release notes and reverting changes in case of problems.

[16] http://flask.pocoo.org

Version control

The old web portal machinery was recently put under version control, mainly to facilitate deployment on other servers. The new Flask server is also being developed on a new repository in GitHub. For development purposes, we run a separate version of the web portal that is public and can be accessed to test new features. Any incremental bug fixing alongside minor feature development and topology/force field extensions are first implemented and tested on this development version before pushing any changes into the production web portal and standalone version. This rolling release development model does not result in a new version every time a minor change is added, since it would mean continuously opening and operating new web portals. Additionally, since computations are typically distributed on a grid of heterogeneous servers distributed around Europe and the world, numerical reproducibility of docking results cannot be guaranteed. Users that wish such reproducibility will have to run a local version of HADDOCK. This does mean more manual intervention to perform the computations since the local version does not include all the validation, pre- and post-processing/analysis steps performed by the web portal. Some users have however commented that they prefer to use the web portal since they know that in that way they always have access to the latest version of HADDOCK.

Review Process

Several core HADDOCK developers can submit pull requests in Github. Rights to accept commits will be given to a few selected core reviewers as well as the main reviewer (Alexandre Bonvin). The versioning system ensures that, even if a faulty commit would have been accepted, the code can be reverted to a previous commit.

Any new feature added to HADDOCK (e.g. the support for a new type of information) should be accompanied with an associated test and example runs. Further, the complete test and example suite should be run and demonstrated to lead to similar results.

For bug fixes and minor features, successful execution of test suite should be demonstrated (see testing process below).

Feature request handling

Current: New features are often added as result of new challenges and scientific questions. In most cases users directly contact the developers via email (either directly via our university emails, or through our user support email: haddock.support@gmail.com. The http://ask.bioexcel.eu HADDOCK forum provides another feedback/requirement mechanism. Considering the limited funding and the absence of core and permanent software developer, new feature requests are prioritized based on the scientific interest and expected impact on the user community.

Future: Under BioExcel, we started making use of the “issues” mechanism offered by GitHub to add and track internal feature requests. GitHub allows to classify issues in various categories, including “enhancement,” which is best used for feature requests. This will allow referring to them when committing software in the HADDOCK GitHub repository that addresses a feature request.

Bug tracking

Current: Bug tracking has been done via internal documents shared between core developers. This information is not available to users. The input is typically collected via direct emailing (see Feature request handling) and via the BioExcel support forum.

Future: Under BioExcel we started making us of the "issues" mechanism offered by GitHub to add and track bugs. GitHub allows to classify issues in various categories, including "bug". This allows referring to them when committing software in the HADDOCK GitHub repository that fixes the bug.

Testing requirements

Any bug fixes should be validated by running the test suite in the local version and demonstrating that similar results are obtained.

Any new feature should be accompanied with a well-documented test and example with all required input data and an example output file.

Any feature/bug fix pushed to the web portal should be first tested and validated on the development version of the web portal, demonstrating that the full workflow is running properly, and similar results are obtained. For new features, a single, self-contained HADDOCK parameter (haddockparam) file should be provided to allow simple testing via the "file upload" interface of the server.

Testing process

HADDOCK development currently uses a private repository on GitHub. Test modules are separated from the code on their own GitHub repository. These run the full workflow for the local version of the software with reduced settings to make sure that everything is working properly, allowing to execute the complete test suite in less than one hour on a laptop. Examples for various scenarios are separated from the code on their own GitHub repository. These run the full workflow with standard or optimal setting for the various scenarios to make sure that all the molecule types supported, and the various combination of input data are properly working.

Next to testing the entire HADDOCK workflow, unit tests are defined for the CNS software used as the computational engine interpreting and executing the HADDOCK scripts. CNS come with its own test suite[17] that checks that the compiled executable properly works, testing all various commands, modules, and energy functions.

Any update pushed to the web portal, should first be successfully tested on the development version of the web portal, ensuring that the various scenarios provided in the example set are all successfully running on the web portal, including the pre- and post-processing steps not offered by the standalone version of the software. For this, version-specific, self-contained single haddockparameter files should be used via the "file upload" interface of the web server.

[17] See the Installation / Test Suite section in http://cns-online.org/v1.3

Those files will be added to the test suite on GitHub. Note that this is a computationally demanding process since full runs should be performed at this stage, which will typically take days to over one week to complete depending on the number of example cases and the load on our server.

Development process details

Any new development should be linked to an identified feature documented as an issue in GitHub. This allows for discussion, planning and review as the work on a feature progresses. The following steps will be followed:

1. Identify and describe a new feature/development by creating an issue in the haddock GitHub repository
2. Write a development/implementation plan and solicit feedback from the HADDOCK developers (eventually, if needed from other BioExcel experts).
3. Plan tests, implementing either regression tests in case of changes at the CNS script level in HADDOCK (i.e. new tests/examples running the complete HADDOCK workflow), or unit tests if new functionalities are added to CNS (i.e. self-contained CNS test scripts testing the specific new feature implemented).
4. Attempt implementation and testing by creating a new branch of the current version of HADDOCK
5. Finish code, add test and example to the test/example suite
6. Run the full test/example suite for validation
7. Submit to main developer for review
8. Merge code upon approval
9. Run the entire test/example suite in the merged version for final validation

Any developments affecting the web portal will follow the same mechanism, with the additional requirement that single, self-contained haddockparameter files be generated to test the new feature via the “file upload” interface of the web portal. Validation using all test cases are first performed on the devel version of the web portal. Depending on the impact of a new feature and the number of new features implemented, either a minor update of the software/portal or a major release will be rolled out.

Integration approach

The HADDOCK developers will work on their own separate branch of the code using standard GitHub mechanisms for this. Once all tests and validation have been successfully performed on the local branch a pull request is done in GitHub to merge the changes in the main branch. GitHub automatically checks if an automated merge can be performed, which needs to be approved by the main reviewer. If this is not the case, manual merging will be required, which will involve the developer putting in the merge request and the main/core reviewer(s). This effectively means merging the current official branch and the developer’s branch into a new branch that will require another round of tests to make sure the integration was successful. Only then will it be merged into the main branch.

Code commenting/standards/styles

Code is commented with a view to explaining what the code is doing and why a specific part was added, rather than how. If it is not obvious how the code is working, then the naming of variables and methods should be improved.

HADDOCK is consisting mainly of Python code and CNS scripts, with some additional C code and a collection of various scripts (csh, sh, awk, perl). For the Python part we aim at following the Google Python style guide recommendations[18].

For the CNS scripts, proper indenting, commenting and clear variable naming should be followed.

Architecture

Quite static for the HADDOCK web server: The server machinery drives the local work, starting the HADDOCK python process which drives the CNS-based computations by submission to a local batch system or to distributed grid/cloud resources. For the local version, in order to move to exascale and allow for efficient, large scale computations (i.e. running thousands of parallel docking runs on various systems), we will have to decouple the pre- and post-processing steps from the core of the computations and also consider a model in which the entire HADDOCK process is submitted to a batch system, rather than the HADDOCK python process sending thousands of individual jobs to it.

Design

The web portal design has since the start been geared toward user-friendliness, providing foldable menus to hide forms when not used. Furthermore, several access levels to the portal are offered that only expose a limited or more extended sets of input options depending on the complexity of the scenario. Usability and scenario-specific web forms is something that will remain central in any future development. The new Flask framework currently under development will allow for much more flexibility in that respect and a faster incorporation of new features into the server (something which is currently delayed).

[18] https://google.github.io/styleguide/pyguide.html

Development process details for CPMD QM/MM

CPMD (http://cpmd.org/) is an *ab initio* (or quantum) molecular dynamics software package developed and maintained by IBM Research. It employs the density functional theory to solve the many-electron problem and a plane-wave basis set to expand the wave function. To perform the time evolution of the quantum system it implements both the Born-Oppenheimer and the Car-Parrinello molecular dynamics approaches.[19] It was reported to be able to scale up to several million threads on a 16.32 PFLOPS supercomputer with almost 100% efficiency.[20]

However, even such highly-parallel *ab initio* code can treat relatively small systems (few thousands of atoms), while many applications, in particular biological ones, require processing systems containing hundreds of thousands and millions of atoms. Such simulations are not feasible using full quantum methods and multiscale approaches, where different parts of the systems are treated at different levels of accuracy and resolution, are employed. The QM/MM methods are those two-layer multiscale approaches where the (small) region that requires the most accurate description (QM part) is treated at quantum level, while the rest of the system (MM part) is described with a classical force field.

CPMD has a built-in QM/MM interface[21] designed and developed by the group of Prof. Ursula Röthlisberger at EPFL (Lausanne, Switzerland) that couples CPMD with the old GROMOS96 (van Gunsteren et al. 1996) classical molecular dynamics routines to perform a QM/MM molecular dynamics simulation. However, this implementation suffers from a set of drawbacks. First, the force fields that can be used to describe the MM part are only GROMOS96 and AMBER99. Then, the scalability of the MM part is limited because the version of GROMOS96 employed has only OpenMP parallelization without MPI part. Therefore, only one node can be used for classical part which leads to scaling issues in case of large MM parts on massively parallel architectures (like IBM Blue Gene). Finally, GROMOS96 has a commercial license, which requires a user to buy it before using the QM/MM interface in CPMD.

To address those issues a novel QM/MM interface is being developed. The designed workflow of a QM/MM-enabled simulation based on CPMD is shown in Figure 5.

[19] Car, R. and M. Parrinello, *Unified Approach for Molecular Dynamics and Density-Functional Theory.* Physical Review Letters, 1985. **55**(22): p. 2471-2474.

[20] Weber, V., et al., *Shedding Light on Lithium/Air Batteries Using Millions of Threads on the BG/Q Supercomputer*, in *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*. 2014, IEEE Computer Society. p. 735-744.

[21] Laio, A., J. VandeVondele, and U. Rothlisberger, A Hamiltonian electrostatic coupling scheme for hybrid Car–Parrinello molecular dynamics simulations. The Journal of chemical physics, 2002. 116(16): p. 6941-6947.

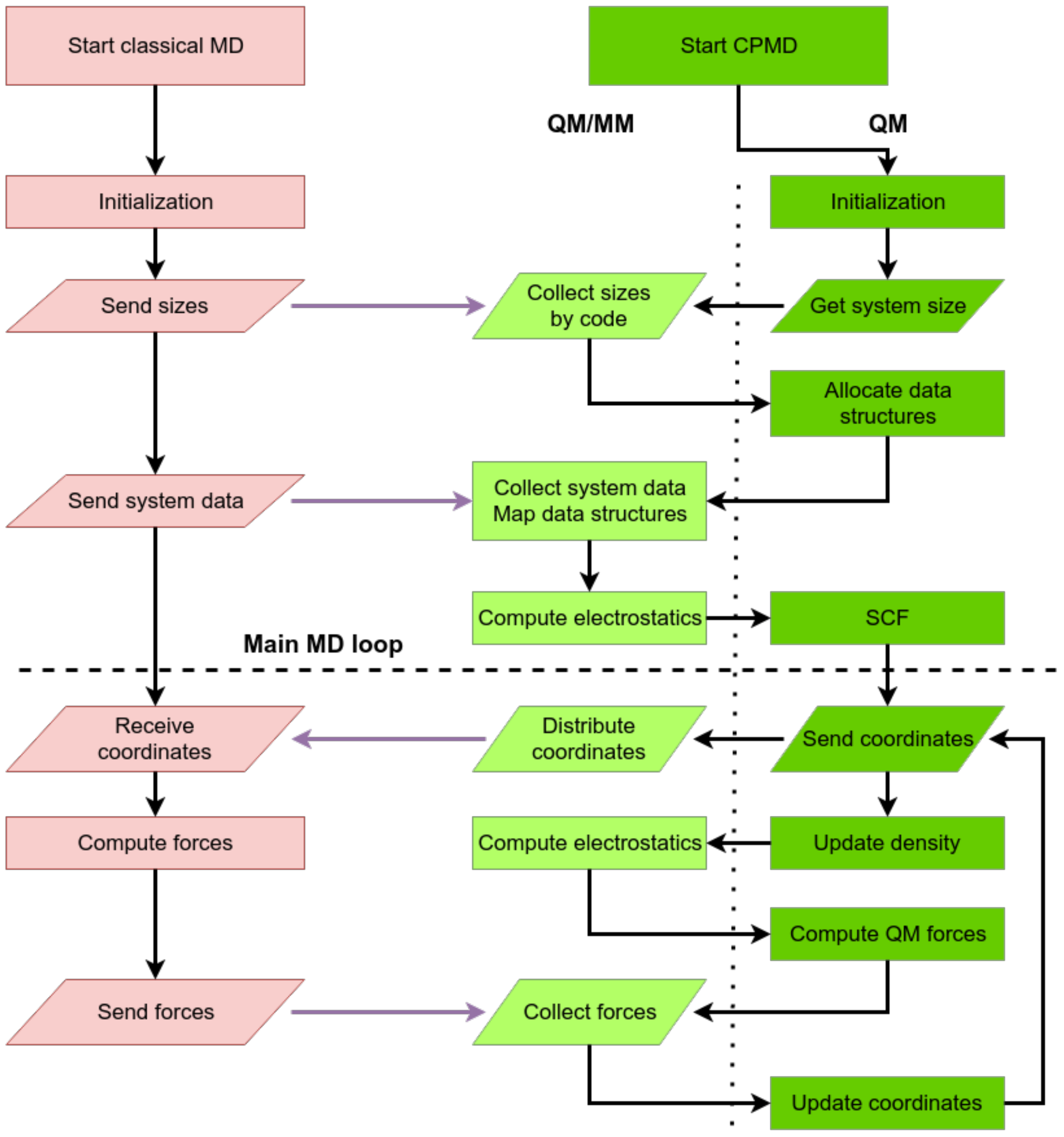

Figure 5 CPMD QM/MM-enabled simulation workflow

Copyright and License

After some discussion within the developer team, it has decided to release the new QM/MM interface under the Lesser GNU General Public License (LGPL) v2.1. In fact, this choice prevents the new QM/MM interface to be distributed inside the tarball of the CPMD source code that can be downloaded from the CPMD website (www.cpmd.org), due to the restrictions of the IBM general license under which CPMD is made available. Different websites, including the BioExcel one, has been selected for releasing the patches that allow introducing the needed changes in the code, and full information about the new QM/MM interface, including how to enable the new QM/MM interface of the code will be inserted in the CPMD manual which is the main source of information about CPMD in the CPMD community.

Version control

Git is used for version control. For the initial stage of the development the merge-based Github workflow is employed. A stable master branch will be used to create builds. All changes are done on a separate branch. Feature branches are merged into master branch through merge requests. Before the review process an automatic build and testing procedure is triggered. A three-level testing procedure is used. The first level of testing is the unit testing checking the validity of code changes introduced in separate modules. After the successful passing of unit tests, a set of integration tests is triggered. Finally, the test coverage analysis is performed using gcov[22]. The testing procedure is handled by the Gitlab continuous integration system. An approval of at least two members of the development team is needed to fulfill the merge request.

Feature request handling and bug tracking

Future: An issue tracking system will be used for reported bugs and feature requests when the program will be released for the user test phase. At the moment a Gitlab issues tracker is planned to be used, with a possible fallback to Redmine in case if the Gitlab functionality proves to be not sufficient.

Testing requirements

A PFUnit unit-testing framework is used to write automated tests. All functional parts of the code (i.e. not constructors, accessors, etc.) must be covered by unit tests. Moreover, proposed changes should not lower the test coverage percentage. Failing to comply with these requirements results in the rejection of the proposed changes. A feature incorporating multiple code units should have a sensible integration test provided. Lack of an integration test without a valid reason stated will also result in the code rejection. Tests are run automatically using Gitlab continuous integration system on several build-servers, running on different hardware and software to assure the portability of the code.

Development process details

Current: The team of developers is currently a small number of autonomous academics in different Institutions that coordinate the general development lines through short face-to-face or Skype meetings.

The code development process is organized following the Github workflow, based on a Gitlab service. After the code changes are committed to the repository an automatic build is triggered, running automated tests following the build. If tests are successful a code review process is started. If the proposed changes are approved by two other members of the team then these changes are merged into a master branch. This workflow is depicted in Figure 6.

Future: We are planning to build up a new dedicated website for the QM/MM interface with the release of the next stable and optimized version of code within the end of the BioExcel project this year. This website is intended to be the main reference for the users of the new

[22] https://gcc.gnu.org/onlinedocs/gcc/Gcov.html

QM/MM interface. In particular, it will contain and update the list of the new features and bug reports from users collected by the developers (through the CPMD mailing list, ask.bioexcel.eu forum, private emails), together the release version of the code where the feature is implemented or the bug fixed. If the number of users will increase considerably, a webpage where the users can directly report issues will be implemented as well. The website will be also the primary source of information for external developers who want to contribute to the development: since we expect only a small number of external contributors, they will be fully integrated in the current development process.

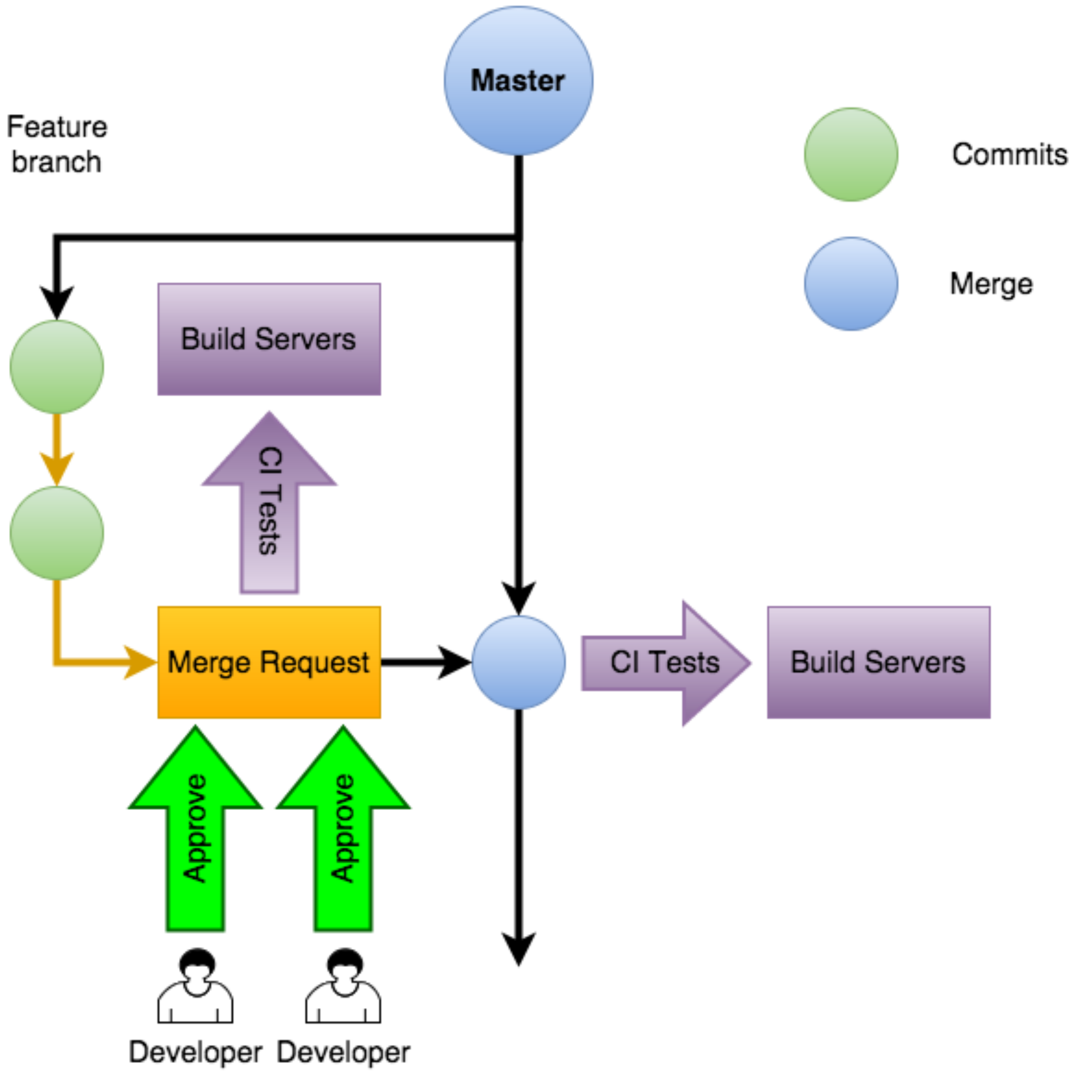

Figure 6 QM/MM development workflow during initial MiMiC development phase

Code commenting

Every type and subroutine (except constructors and accessor methods) has Doxygen-format comments stating its purpose (i.e. explaining what code is doing).

Coding standards and styles

The part of the code integrated in CPMD is being developed to comply with Fortran2008 standard. Coding style is loosely based on the Fortran90 Best Practices[23]. Execution flow can only be controlled using if and do statements - usage of goto statements is strictly forbidden. On top of that a usage of OOP techniques is encouraged (though sometimes is limited due to the language issues or performance considerations) to make the maintenance of the code easier.

The communication library is being developed in C++, with style loosely based upon the Google C++ Style Guide[24].

Finally, the minor changes in GROMACS are implemented by following the GROMACS style described in the corresponding section above.

Design

The design of the novel QM/MM interface has been devised having in mind the execution model based on the multiple program multiple data (MPMD) approach. In this model CPMD and the classical molecular dynamics code (e.g. GROMACS) run independently occasionally exchanging data. To establish the data connection an ad-hoc communication library is used. The advantages of this approach are i) the possibility to couple CPMD virtually with any classical molecular dynamics code (and consequently to employ any force field in the MM part) with minimal code intervention; ii) benefitting from the parallelization scheme of both the classical molecular dynamics code and CPMD; iii) overcoming possible licensing conflicts between the CPMD's proprietary license and the one of the classical molecular dynamics code.

[23] http://www.fortran90.org/src/best-practices.html
[24] https://google.github.io/styleguide/cppguide.html

Development process details for pmx

pmx is a Python-based software package[25] [26] that provides utilities for handling biomolecular structure and topology files directly compatible with GROMACS. The particular strength of the pmx libraries lies in their application to the hybrid structure and topology generation for alchemical single topology based free energy simulations. The automated pmx based procedures readily allow the calculation of free energy changes due to amino acid mutations in several contemporary molecular mechanics force fields.[27] Recently, also support for nucleic acid mutations has been added.[28] The basic scheme underlying the workflow of the automated hybrid protein structure/topology generation is depicted in Figure 7. Preparation of the system for alchemical molecular dynamics simulations following this scheme is available via a command line interface. A user-friendly web-based infrastructure implementing the outlined workflow has recently been launched,[29] and is available at http://pmx.mpibpc.mpg.de.

[25] Seeliger, D. and B.L. de Groot, *Protein Thermostability Calculations Using Alchemical Free Energy Simulations.* Biophysical Journal, 2010. **98**(10): p. 2309-2316.
[26] Gapsys, V., et al., *pmx: Automated protein structure and topology generation for alchemical perturbations.* Journal of Computational Chemistry, 2015. **36**(5): p. 348-354.
[27] Gapsys, V., et al., *Accurate and Rigorous Prediction of the Changes in Protein Free Energies in a Large-Scale Mutation Scan.* Angewandte Chemie International Edition, 2016. **55**(26): p. 7364-7368.
[28] Vytautas Gapsys, Bert L. de Groot. Alchemical Free Energy Calculations for Nucleotide Mutations in Protein-DNA Complexes. JCTC 13: 6275-6289 (2017).
[29] Vytautas Gapsys, Bert L de Groot. pmx Webserver: A User Friendly Interface for Alchemistry. J. Chem. Inf. Model. 57: 109-114 (2017).

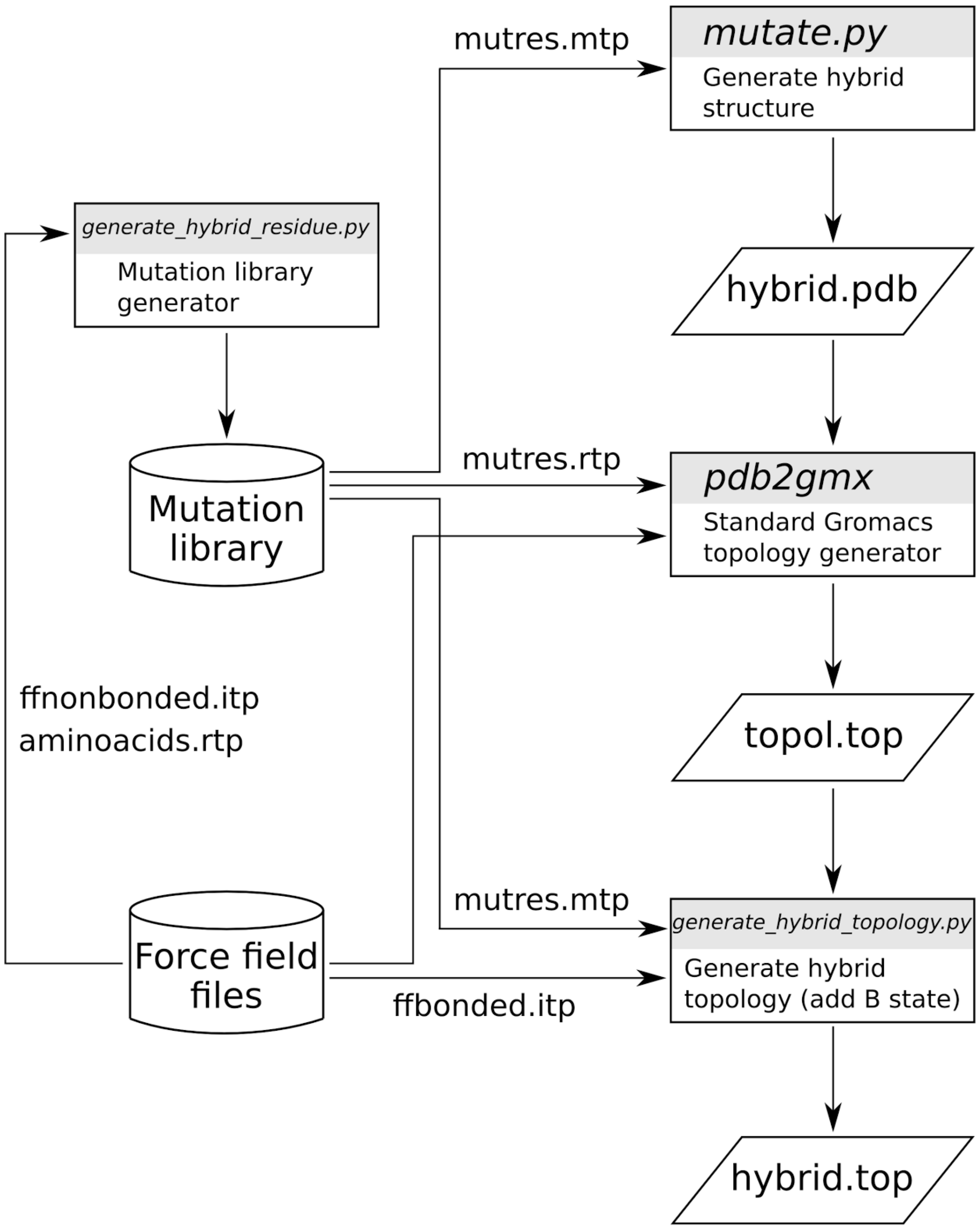
mutate.py
Generate hybrid structure
mutres.mtp
generate_hybrid_residue.py
Mutation library generator
hybrid.pdb
Mutation library
mutres.rtp
pdb2gmx
Standard Gromacs topology generator
ffnonbonded.itp
aminoacids.rtp
topol.top
mutres.mtp
generate_hybrid_topology.py
Generate hybrid topology (add B state)
Force field files
ffbonded.itp
hybrid.top

Figure 7 Workflow of the pmx-based hybrid structure/topology generation for an amino acid mutation

Retirement

pmx is dependent on the GROMACS infrastructure for handling structures and topologies. Hence, the retirement of outdated pmx versions is tightly linked to the changes in the GROMACS force field organization and the tools handling the structure/topology operations.

Following a substantial change in the GROMACS force field structure, the old pmx version is retired immediately once an update is released. Older versions can still be downloaded and used, but are no longer actively supported.

Deprecation

Features in pmx currently are not deprecated. When a newer version of a feature is implemented, the old version remains available to the users.

Release Process

The releases are guided by the implementation of new features and bug-fixes. Major releases are accompanied by major features, whereas minor releases focus on bugfixes and stability. The updated versions are immediately pushed onto GitHub. The version control and documentation of the changes is provided by the Git version control system.

Review Process

Current: pmx is actively developed by one person, thus no independent review process is currently implemented.
Future: a code review process will be added, as soon as resources permit.

Feature request handling

Addition of new features is stimulated by the scientific questions to be solved. Also, updates are considered upon request from the user side. These requests come in both electronically as well as during workshops and tutorials. The requests can be sent by email directly to the developers or posted on GitHub. A major novel feature that was recently added was support for nucleic acids. In the near future full support for arbitrary modifications in ligands will follow. Prioritization of feature requests is based on the fraction of the user base that is anticipated to benefit from the new features.

Bug tracking

Users can report bugs as GitHub issues at https://github.com/dseeliger/pmx/issues that pmx developers can act upon, however the testing process (described below) serves as the main method for preventing bugs before users find them.

Testing requirements

The **first level check** for a newly generated mutation library: the hybrid topology needs to contain all the bond/angle/dihedral parameters of the non-hybrid topologies representing physical states.
Second level check: test cases have been set up that must yield zero as computed free energy. The majority of errors that can occur will lead to deviations from zero and therefore these calculations provide an excellent quality control: these thermodynamic cycles must converge to zero for every new mutation library and software version.

Testing process

Current: a set of standardized regression tests is performed internally after generating new mutation libraries and software releases. A selected set of mutations is carried out and checked against previous, validated versions for deviations beyond the statistical uncertainty. This is carried out via customized scripts. These tests are carried out for each pmx release as well as each supported major GROMACS release. The main development platform is Linux.
Future: The standardized scripts required for the testing procedure are planned to be shipped together with the pmx package for the user to test the installation as well as offer the possibility to utilize these tests to check newly implemented user-features.

Development process details

There are two main forces driving the pmx development. Firstly, the development is guided by scientific questions which dictate the direction for the new features to be implemented. Secondly, major changes in the GROMACS force field organization and structure/topology handling utilities require adjustments to the pmx code. These are discussed in detail between the pmx and GROMACS developers.

Integration approach

Current: pmx has only one active developer, therefore the bug-fixes and new features are integrated into the master branch as soon as they pass the testing phase.
Future: introduce code review when resources permit. The core development team receives active support from advanced users who are stimulated to participate in code review.

Code commenting/standards/styles

pmx is written in Python 2. The code does not adhere to a specific external standard but uses a coherent internal standard that has matured over the years. The pmx source code contains concise comments, and maintains an understandable class, method and variable naming convention for optimal readability. This has proven a robust and extendable framework since the first release in 2010.

Architecture

pmx implements several classes for the structure and topology handling (see Figure 8). The setup for the alchemical single topology simulations of the amino acid, DNA or ligand modifications comprises several scripts (see Figure 7) that utilize the data structures in Figure 8. All the utilities provided by pmx are available via command-line interface. The amino acid mutation setup is also available as an online web-server. Online access to the other features is planned for the future development.

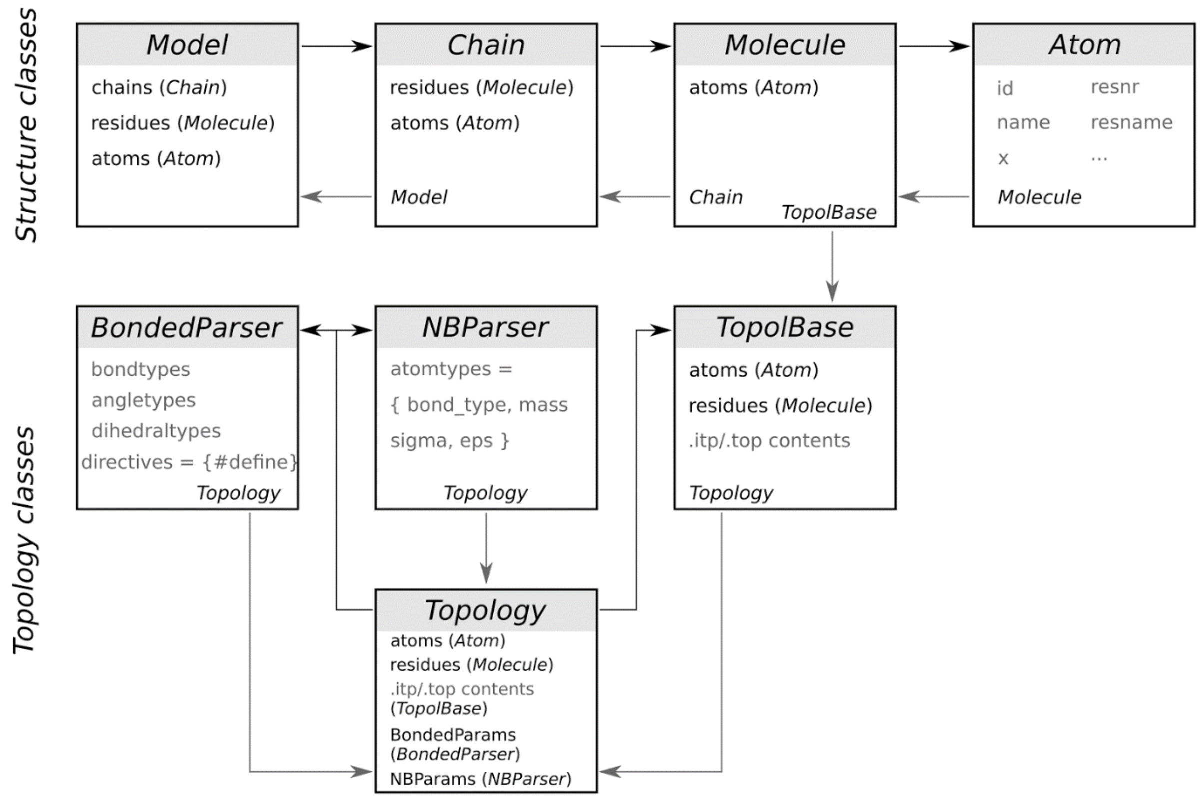

Figure 8 The main classes underlying the pmx architecture

Design

A command-line interface is designed as a convenient medium for a power user who needs to perform a large scale amino acid/DNA/ligand modification scan, as it allows for convenient scripting also on architectures without a graphical interface, such as remote supercomputer clusters. The pmx scripts can be executed on a local machine or on a cluster in an automated manner. pmx can be incorporated into a workflow management system for a fully automated high throughput computational mutation screening.

An online web-server provides a user-friendly means to obtaining the amino acid mutation structures and topologies without the need to have a local installation of pmx. The web-server also provides a feature of generating files required for the mutational scans of a full protein by a user selected amino acid.

Concluding Remarks

We hope that these snapshots of the development processes for the BioExcel codes have been useful. These real-world projects are using tools and methods that work, but they have evolved over time and will continue to change to suit the individual projects' needs. Ongoing incremental investment into code development process is needed for any project to provide software that continues to meet its users' needs, while remaining correct, portable, and performant.

Author List

The contributing authors from the BioExcel Centre of Excellence, along with their academic affiliations, were

Forschungszentrum Jülich (Jülich Research Centre), Germany – Emiliano Ippoliti

KTH (Royal Technical University), Sweden – Mark J. Abraham, Rossen Apostolov, Erwin Laure, Berk Hess, Erik Lindahl

Max Planck Institute, Germany – Bert L. de Groot, Vytautas Gapsys

Utrecht University, The Netherlands – Alexandre M.J.J. Bonvin, Mikael Trellet, Adrien S.J. Melquiond, João P.G.L.M. Rodrigues[30]

[30] Now at Stanford University School of Medicine, Palo Alto CA, USA