H2020 EINFRA-5-2015

bioexcel

Center of Excellence for Computational Biomolecular Research

www.bioexcel.eu

Project Number 675728

D1.1 – Specification of software engineering, testing & QA

WP1: Software Scalability & Usability

The opinions of the authors expressed in this document do not necessarily reflect the official opinion of the BioExcel partners nor of the European Commission.

Document Information

Deliverable Number	D1.1
Deliverable Name	Specification of software engineering, testing & QA
Due Date	2016-12-31 (PM14)
Deliverable Lead	KTH
Authors	Mark Abraham (KTH), Alexandre Bonvin (UU), Adrien Melquiond (UU), Emiliano Ippoliti (Juelich), Bert de Groot (MPG)
Keywords	Project Management, software engineering
WP	WP1
Nature	Report
Dissemination Level	Public
Final Version Date	2016-12-28
Reviewed by	Ian Harrow (IHC), Erwin Laure (KTH), Rossen Apostolov (KTH), Stian Soiland-Reyes (UNIMAN), Carole Goble (UNIMAN)
MGT Board Approval	2016-12-28

Document History

Partner	Date	Comments	Version
KTH	2016-03-01	First draft	0.1
KTH	2016-03-28	Draft for final review	0.2
KTH	2016-03-31	Final	1.0
KTH	2016-12-12	Draft for final review	1.1
KTH	2016-12-22	Draft for PMB review	1.2
KTH	2016-12-23	Draft for PMB review	1.3
KTH	2016-12-28	Final version	1.4

Executive Summary

This document reports on the planning within BioExcel WP1 for software-engineering tasks. It describes the current state of testing and quality assurance for the pilot codes CPMD, GROMACS and HADDOCK, and the GROMACS input-preparation software pmx. It also specifies the plans within BioExcel for implementing established best practices for software engineering processes while extending code usability, functionality and performance. These plans are intended to provide high value to the biomolecular simulation community that BioExcel serves by delivering effective and usable software that will be sustainable and portable through the transition to the exascale era. Aspects of engineering process that are useful to unify across the development sub-teams will be discussed first, and then a detailed description for each sub-team will follow, describing the current status and plans for the future where we have identified priorities for improvements to our development process.

There is a large range of reasonable practice in software engineering, and the efficacy of many of them is still under debate. Further, that efficacy varies with the kind of software being developed, the funding model, the team doing the development, and the risks acceptable to its users. Accordingly, this document does not attempt an exhaustive description of possible approaches, nor to determine which practices are best. It does refer to existing work in this area, and specifies how the BioExcel software developers have chosen to develop, test and assure the quality of their software.

Contents

1 Introduction

The pilot applications in BioExcel (GROMACS, QM/MM CPMD, and HADDOCK) are committed to delivering high quality life-science software that meets its users' needs by following accepted best practice in software engineering, to the extent that available resources permit.

Much scientific software is written to serve the needs of a single researcher or a single short-term research project, and it can be appropriate to write such code without considering whether it will be re-used.[1-3] However, some of the pilot codes for BioExcel have matured to the point where they serve the needs of thousands of life scientists, and the success of the consortium depends on expanding the user base and range of quality software that serves it. Each code may be exposed to hardware and software changes during the transition from the peta- to exascale era. These concerns strongly suggest a more professional approach to justify the investment of further development resources, and to reduce the risk associated with widespread use. The software development within BioExcel will incorporate existing recommendations for best practice for scientific computing.[4] However, the different applications within BioExcel

- are written in different programming languages,
- have matured independently and to different extents,
- address different kinds of biomolecular simulation problem,
- are delivered to end users differently, and
- are implemented by very small numbers of software developers located at different institutions,

so few details of the software-engineering process can be usefully unified across BioExcel. This document will describe the processes that can be usefully unified across the consortium. It will also record the details of the state and plans for the sub-teams within the consortium. These vary between sub-teams, because the needs, priorities and capabilities do differ. This record will serve as a model for future improvements within the teams, for expansion to include new teams, and future unification of elements that become common to sub-teams.

Much has been written about how to implement best practice in software engineering. Best practice in this field is also evolving rapidly as the hardware, languages and tools evolve. One important work identified seven basic principles for successful large-scale commercial software delivery.[5] Many of these are generally applicable to the software engineering effort in BioExcel, and are consistent with published best practice for scientific software development.[4] A description of each follows, and the way in which the BioExcel software development will address them is described in *italics*. These approaches will be expanded upon in the following sections for the pilot codes.

1. Manage using a phased life-cycle plan, to specify what will be done, recognizing that effort in design phases leads to shorter coding, debugging and validation times.
   The Description of Action and Deliverable 1.1 forms part of that plan for the overall consortium. For each new feature, individual sub-teams within BioExcel will produce written plans, and consult with other teams on the

kinds of high-level aspects where independent perspective can lead to better development results.

2. Perform continuous validation after having agreed on expectations for the software, to review progress, and automatically test results.
Each BioExcel software sub-team will develop or expand automated techniques for testing their software for desirable properties such as correctness or performance. This is a key requirement for all kinds of software, but scientific user communities make particular demands for it, and our development processes will specifically address it.
3. Maintain disciplined product control, in particular to avoid opportunistic changes to the plan, or promising more than the available developer effort will permit.
Resources are limited and the scope of the work within the Description of Action is large, so the BioExcel software development efforts will need to choose manageable sub-projects with a view to being able to deliver quality software in suitable graduations over the lifetime of the project. This will permit the consortium to show value in the hands of users by the time of the final review, and ensures that some software has been delivered even if progress in some aspects has been slower than anticipated.
4. Use modern programming practices, in particular to use development techniques such as encapsulation, modularity, version control, issue tracking, code review, automated code analysis, and unit testing to make most efficient use of time in the long term.
These are well established and standardised techniques that BioExcel developers will follow because they provide demonstrated value in producing software that is correct, can be maintained and extended, and which provide resilience in the face of anticipated changes to hardware (e.g. new accelerators, processors and memory) and libraries as we approach the exascale era. Different approaches will be used by each sub-team to suit the needs of software delivered in different ways and written in different languages.
5. Maintain clear accountability for results, in particular that useful progressive milestones exist for individuals and teams
BioExcel software development naturally has high-level accountability in the agreed deliverables and their need to respond to user feedback. Each of the sub-teams structures their work plans with the local context and broad objectives in mind.
6. Use better and fewer people, to avoid communication overheads
This principle is generally inapplicable to academic software development, such as done in BioExcel, because the desired scientific domain knowledge and software-engineering expertise are normally incompatible with university salary structures and the lack of clear career path within academe where software is generally not recognized as a worthwhile output. BioExcel can only afford a few software developers, so this principle of software development is not a problem for BioExcel to manage.
7. Maintain a commitment to improve the process
BioExcel software developers are committed to improving their process, investing time in areas that have been shown to be deficient (e.g. because

> *problems have arisen), while recognizing that overall resources are very limited and must be shared across multiple endeavours.*

Preparing for the exascale era is of paramount importance for the development of the BioExcel codes. New hardware (CPUs and accelerators), middleware, libraries and programming paradigms will emerge, rapidly. All the BioExcel codes leverage established technologies that provide good prospects for long-term portability and performance. These technologies include git, Jenkins, Gerrit, GitHub, GitLab, CUDA, OpenCL, python, Flask, Grid engines, CNS, CPMD, GROMACS, FFTW, Fortran, C++ and CMake, alongside purpose-built components such as the GROMACS SIMD portability layer. The technology-watch components and deliverables of WP1 serve to prompt the BioExcel developers to review the suitability of these decisions as future directions become clear. Most of the software developed for the BioExcel pilot codes is not at risk when new hardware emerges, with the notable exception of GROMACS, whose development and testing process described below caters directly to mitigating this.

We have consulted with the UK-based Software Sustainability Institute (https://www.software.ac.uk/) to get valuable input from them into our software-engineering process. Representatives of BioExcel have attended some of their events, and our teams follow the excellent advice in many of their guides (https://www.software.ac.uk/resources/guides). We have completed their software evaluation process to get direct feedback on our process, and will continue to collaborate with them to ensure our knowledge stays current.

New software functionality will be made available in project deliverables after 17 and 36 months, and reported in scientific publications authored by BioExcel participants, but it is recognized that the true utility in scientific software is that it permits its users to innovate effectively.[2] Demonstrating the quality of the output of the software can be more important than new functionality, particularly if it permits users to adopt more recent versions of the software with confidence that old results are reproducible, or clearly corrected. In BioExcel, the quality of the software output will be assured through extensive testing, monitored via feedback from the BioExcel interest groups, and feedback used to guide future development efforts. New software and modules developed within BioExcel will be released under licenses consistent with CC-BY models.

2 Unified process for gathering requirements and planning developments

The major unifying thread across the BioExcel software development teams is that they include practising scientists with experience of using a range of scientific software to solve biomolecular problems, and developers of such software. It is widely agreed that effort spent in planning phases of software development projects delivers tangible benefits in the form of faster progression through coding, testing and acceptance phases.[6] The BioExcel development sub-teams will conduct feature development after a planning phase that includes

- consulting with prospective users about functionality changes,

- assessing that the available coding resources are adequate,
- deciding that potential risks to the usefulness of the software once complete are acceptable,
- drafting a user interface, and
- producing a written plan.

In particular, the draft user interfaces and development plans will be shared across the software-development teams for feedback on the proposed user interface, and development plan. This leverages the different experience and perspective of developers from other sub-teams to add useful value via this form of peer review. Prospective users will also be consulted to improve the chances that the implementation is fit for its purpose. It is expected that success of this model within BioExcel will motivate other developers within the broader application development teams to participate and perhaps provide a consultative service to the broader biomolecular simulation software development community. We will monitor and report on this process internally and at formal project reviews.

3 Development process details for GROMACS

GROMACS[7] 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, which have a lot of complicated bonded interactions. Its chief virtue is that it is extremely fast at calculating the 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 in excess of 2000 per year.

Figure 1 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.

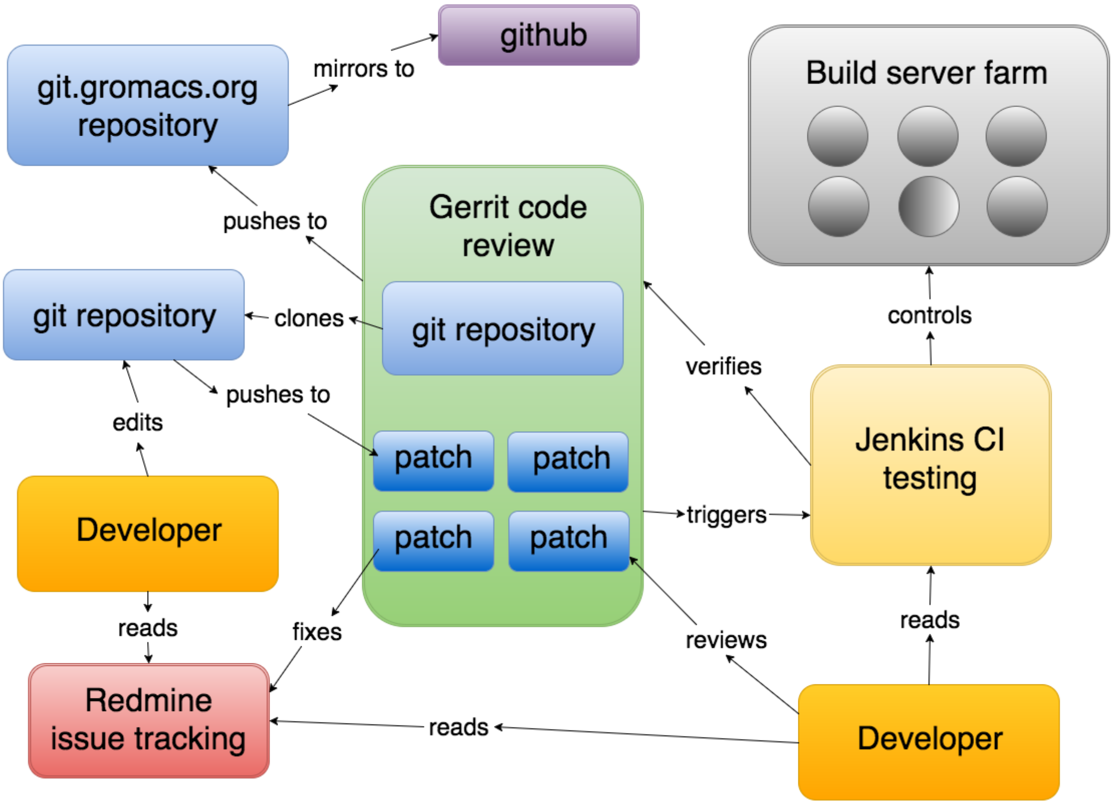

Figure 1 GROMACS development workflow description at the start of the BioExcel project

3.1 Copyright and License

GROMACS is released under the Lesser GNU General Public License (LGPL) v2.1. Each source code file has a copyright statement, which is automatically updated for each applicable year that it changes, which is necessary for the license to be 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.

3.2 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. 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, on which kinds of existing and emerging hardware architectures, and thus require particular 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.

3.3 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 and code review.

Future: In future phases of BioExcel, provide opportunity for clients to discuss e.g. development of new features on a contract basis.

3.4 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.

3.5 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.

Future: No changes are planned. Current resources do not permit more than one annual major release.

3.6 Version control

Git is used for version control. All code development should start from the official git repository (git.gromacs.org) or one of its mirrors (including 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.

3.7 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. 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, and such fixes will be merged into any more recent release branches, and then into the master branch.

3.8 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, easier support of new simulation methodology, better extensibility to emerging hardware platforms, and higher portability across them. The classes within these modules encapsulate behaviour alongside the data necessary to implement it, and require full Doxygen documentation of files, classes, members and methods. The modules follow a layered design that is enforced by checking scripts.
Future: Draft a target module hierarchy for the wider development community to use as a mental model during the refactoring process.

3.9 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. 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 maximises 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.

3.10 Review Process

Current: All code changes go through two-person review, 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. 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 have the ability to 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.

3.11 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.

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.

3.12 Testing requirements

Current: All proposed code changes are tested automatically by our Jenkins continuous-integration server. It builds the code and runs the tests on multiple compilers, operating systems, standard libraries, accelerators and CPU types. 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. See http://jenkins.gromacs.org/job/Documentation_Nightly_master/javadoc/dev-manual/jenkins.html for more details. Released versions, e.g. source tarballs are tested on a range of configurations before release.

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 single-node performance regression testing on the performance benchmark suite, perhaps deploying tools such as Allinea Performance Reports or Intel VTune Amplifier,
- manual benchmarks and testing on existing petascale resources (though this is currently of dubious value, because GROMACS is extremely

sensitive to network latency, and typically that latency is affected irreproducibly by every job that shares the network on the machine),
- 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.

3.13 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. 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.

3.14 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. Generally it will be 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 at the discretion of the developer community. 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.

3.15 Retirement

Current: Each (annual) major release has one year of best effort 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 issue that affects scientific correctness (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. In particular, 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.

3.16 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, then prefer to improve the naming of variables and methods, use better control flow, or show examples with working tests. New and newly refactored methods and source files must have Doxygen-style comments that are automatically built into the developer guide available at http://jenkins.gromacs.org/job/Documentation_Nightly_master/javadoc/#documentation-for-developers.

3.17 Coding standards and styles

Current: The GROMACS style is loosely based upon the Google C++ Style Guide (https://google.github.io/styleguide/cppguide.html), 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. See http://jenkins.gromacs.org/job/Documentation_Nightly_master/javadoc/dev-manual/style.html.

Future: Update these to account for new recommendations in the CppCoreGuidelines (https://github.com/isocpp/CppCoreGuidelines), 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.

4 Development process details for HADDOCK

HADDOCK[8, 9] is an integrative, information-driven docking approach which supports the incorporation of a large variety of data from NMR and other biophysical methods (e.g. cross-links from MS, EPR-derived distances, mutagenesis data, as well as the use of SAXS and IM-MS data to filter docking solutions). The software is made available through a user-friendly web interface,[10, 11] which has attracted a large user community worldwide (8000+ 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 its field. HADDOCK is currently available both as a software for local installation and as a web-server. The stable release of the software and web server is HADDOCK 2.2.[11]
Both the stand-alone version and the most commonly used (>90% of the user base, >8000 registered users) web portal version of HADDOCK implement a complex workflow managed at a higher level by a Python code components, with the web server implementation adding several pre- and post-processing steps not available in the current local version of the software. These worflows are depicted in Figure 2 and Figure 3.

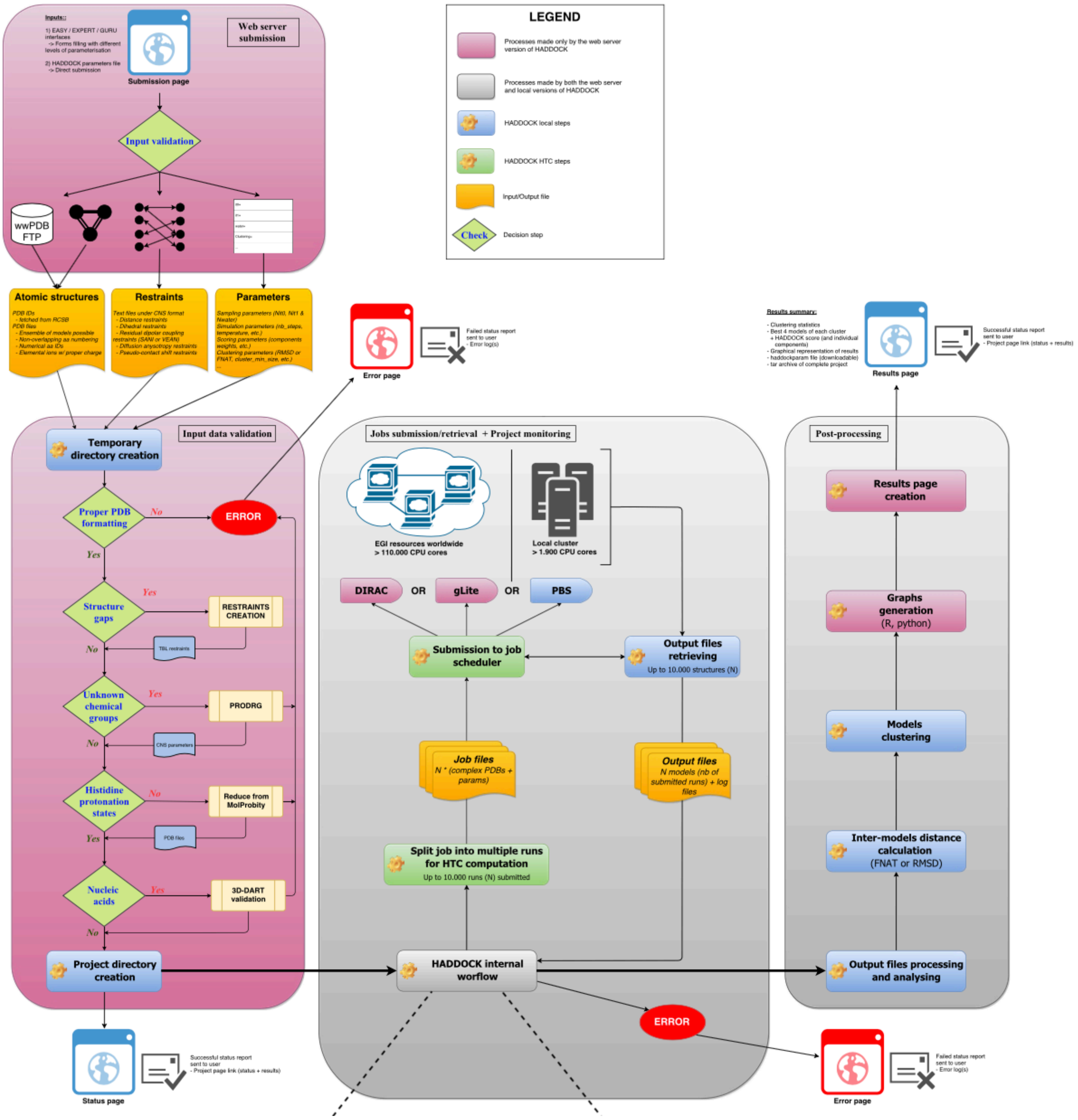
Inputs::
1) EASY / EXPERT / GURU interfaces
-> Forms filling with different levels of parameterisation
2) HADDOCK parameters file
-> Direct submission
Submission page
Web server submission
Input validation
wwPDB FTP
Atomic structures
PDB IDs
- fetched from RCSB
PDB files
- Ensemble of models possible
- Non-overlapping aa numbering
- Numerical aa IDs
- Elemental ions w/ proper charge
Restraints
Text files under CNS format
- Distance restraints
- Dihedral restraints
- Residual dipolar coupling restraints (SANI or VEAN)
- Diffusion anysotropy restraints
- Pseudo-contact shift restraints
Parameters
Sampling parameters (Nit0, Nit1 & Nwater)
Simulation parameters (nb_steps, temperature, etc.)
Scoring parameters (components weights, etc.)
Clustering parameters (RMSD or FNAT, cluster_min_size, etc.)
...
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
Failed status report sent to user
- Error log(s)
Results summary:
- Clustering statistics
- Best 4 models of each cluster + HADDOCK score (and individual components)
- Graphical representation of results
- haddockparam file (downloadable)
- tar archive of complete project
Results page
Successful status report sent to user
- Project page link (status + results)
Input data validation
Temporary directory creation
Proper PDB formatting
No
ERROR
Yes
Structure gaps
Yes
RESTRAINTS CREATION
TBL restraints
No
Unknown chemical groups
Yes
PRODRG
CNS parameters
No
Histidine protonation states
No
Reduce from MolProbity
PDB files
Yes
Nucleic acids
Yes
3D-DART validation
No
Project directory creation
Status page
Successful status report sent to user
- Project page link (status + results)
Jobs submission/retrieval + Project monitoring
EGI resources worldwide
> 110.000 CPU cores
Local cluster
> 1.900 CPU cores
DIRAC
OR
gLite
OR
PBS
Submission to job scheduler
Output files retrieving
Up to 10.000 structures (N)
Job files
N * (complex PDBs + params)
Output files
N models (nb of submitted runs) + log files
Split job into multiple runs for HTC computation
Up to 10.000 runs (N) submitted
HADDOCK internal worflow
ERROR
Error page
Failed status report sent to user
- Error log(s)
Post-processing
Results page creation
Graphs generation
(R, python)
Models clustering
Inter-models distance calculation
(FNAT or RMSD)
Output files processing and analysing

Figure 2 Workflow description of the HADDOCK web portal.

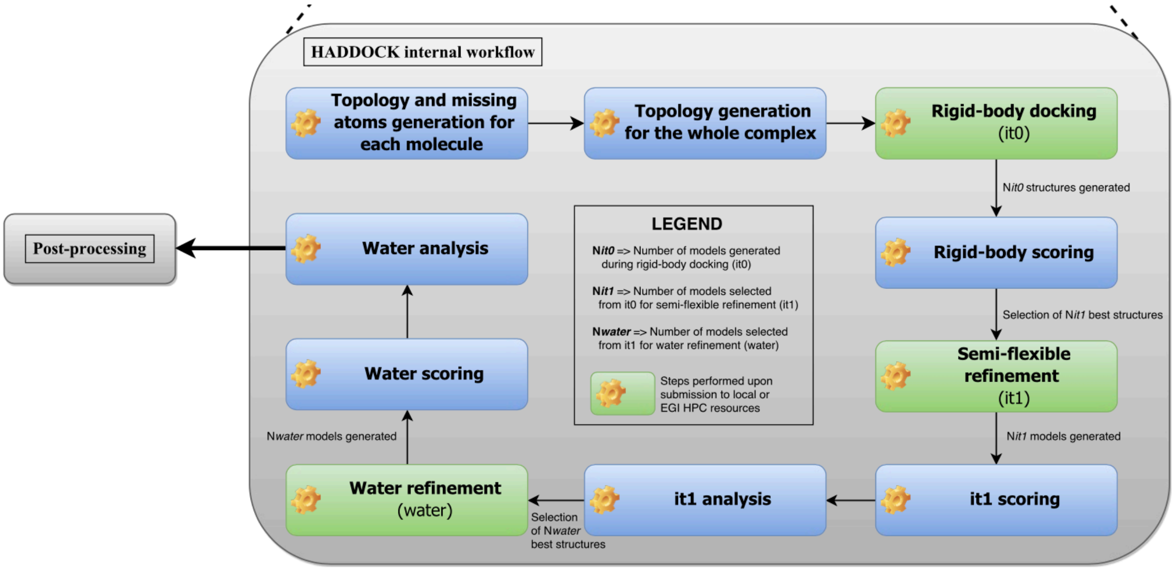
HADDOCK internal workflow
Topology and missing atoms generation for each molecule
Topology generation for the whole complex
Rigid-body docking (it0)
Nit0 structures generated
Rigid-body scoring
Selection of Nit1 best structures
Semi-flexible refinement (it1)
Nit1 models generated
it1 scoring
it1 analysis
Selection of Nwater best structures
Water refinement (water)
Nwater models generated
Water scoring
Water analysis
Post-processing
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

Figure 3 Internal workflow of HADDOCK.

The actual computations are performed using the CNS (Crystallography & NMR System) engine (http://cns-online.org), which comes with its own test suite. All the docking and refinement protocols in HADDOCK are coded at the level of CNS scripts. This means that development does not require coding in the CNS source code (Fortran77) unless new functions need to be implemented.

4.1 Architecture

The architecture is quite static for HADDOCK, the server drives local work, which drives the grid submission/polling engines. It is unlikely to change much. HADDOCK can exploit both grid HTC or local cluster resources for distributing the workload. It has also run on HPC systems under several HPC-Europe and HPC-Europe2 programmes.

4.2 Design

Current: 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 level to the portal are offered that only expose a limited or more extended sets of input options depending on the complexity of the scenario.
Future: Usability and scenario-specific web forms is something that will remain central in any future development.

4.3 Release Process

Current: New features are implemented in local development versions of the software and then added to the web front end. The latter part is currently the limiting factor in releasing new versions because of the old framework on which the portal has been built. We are currently rewriting the web portal into a Flask framework which should allow much faster implementation of new features in the web portal.
Incremental bug fixing alongside minor feature development are constantly implemented. These are first tested in a development version of the web portal,

which can also be accessed by users for testing purposes. Once these have been properly tested and validated, the changes are pushed to the production version of the web portal and the stand alone software distribution is updated accordingly. This process does not result in a new version release since it remains rather transparent to the end users (i.e. no new input forms are typically added to the server).

Future: All changes will be properly documented (using Git with good messages and self-contained commits), clearly separating for fixes and features, to ease making release notes.

4.4 Review Process

Current: A number of 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).

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 the test suite should be demonstrated (see testing process below).

4.5 Feature request handling

Current: New features are often added as result of new challenges and scientific questions. Currently users will 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 will make us of the “issues” mechanism offered by GitHub to add and track feature requests. GitHub allows to classify issues in various categories, including “enhancement,” which will be used for feature requests. This will allow referring to them when committing software in the HADDOCK GitHub repository that addresses a feature request.

4.6 Bug tracking

Current: Bug tracking has been done via internal documents shared between core developers. The input is typically collected via direct emailing (see Feature request handling) and via the BioExcel support forum.

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

4.7 Testing requirements

Current: 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.
Future: For new features, a single, self-contained haddockparameter file should be provided to allow simple testing via the "file upload" interface of the server.

4.8 Testing process

Current: HADDOCK's 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. The current examples are:

- protein-DNA
- protein-ligand
- protein-peptide-ensemble
- protein-peptide
- protein-protein docking using NMR chemical shift perturbation restraints
- protein-protein docking using NMR diffusion anisotropy restraints
- protein-protein docking using NMR pseudo-contact shift restraints
- single structure refinement with NMR restraints
- protein-trimer docking
- solvated protein-protein docking
- multi-body docking of a C4 tetramer with a coarse grained representation
- protein-protein docking using cryo-EM restraints

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.
Future: 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.

4.9 Development process details

Future: Any new development should be linked to an identified feature documented as an issue in GitHub. This will allow 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
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 will be 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.

4.10 Integration approach

The HADDOCK developers will work on their own separate branch of the code using standard GitHub mechanisms for this. Integration will happen by merging into the main branch following the development process described above.

4.11 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 stand alone version is kept as the same level as the web server version (currently HADDOCK2.2), since this allow for a much smoother user support by pointing users to the web portal in case of local installation problems or setup issues.

Currently we support two versions of the web portal for HADDOCK 2.1 and 2.2, because of third party software connecting directly to the 2.1 version of the portal. Developers have been contacted to upgrade to the latest version.

Future: For future releases of the web portal and associated stand alone 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 years after the release of a new version of those. While two years might seem rather long, 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.

4.12 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.

4.13 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 https://google.github.io/styleguide/pyguide.html.

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

5 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.[12] It was reported to be able to scale up to several million threads on a 16.32 PFLOPS supercomputer with almost 100% efficiency.[13]

However, even such highly-parallel ab initio code can only treat relatively small systems (few thousands of atoms), while many applications, in particular biological ones, require to process 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[14] 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 due to the fact that 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.

In order 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 4.

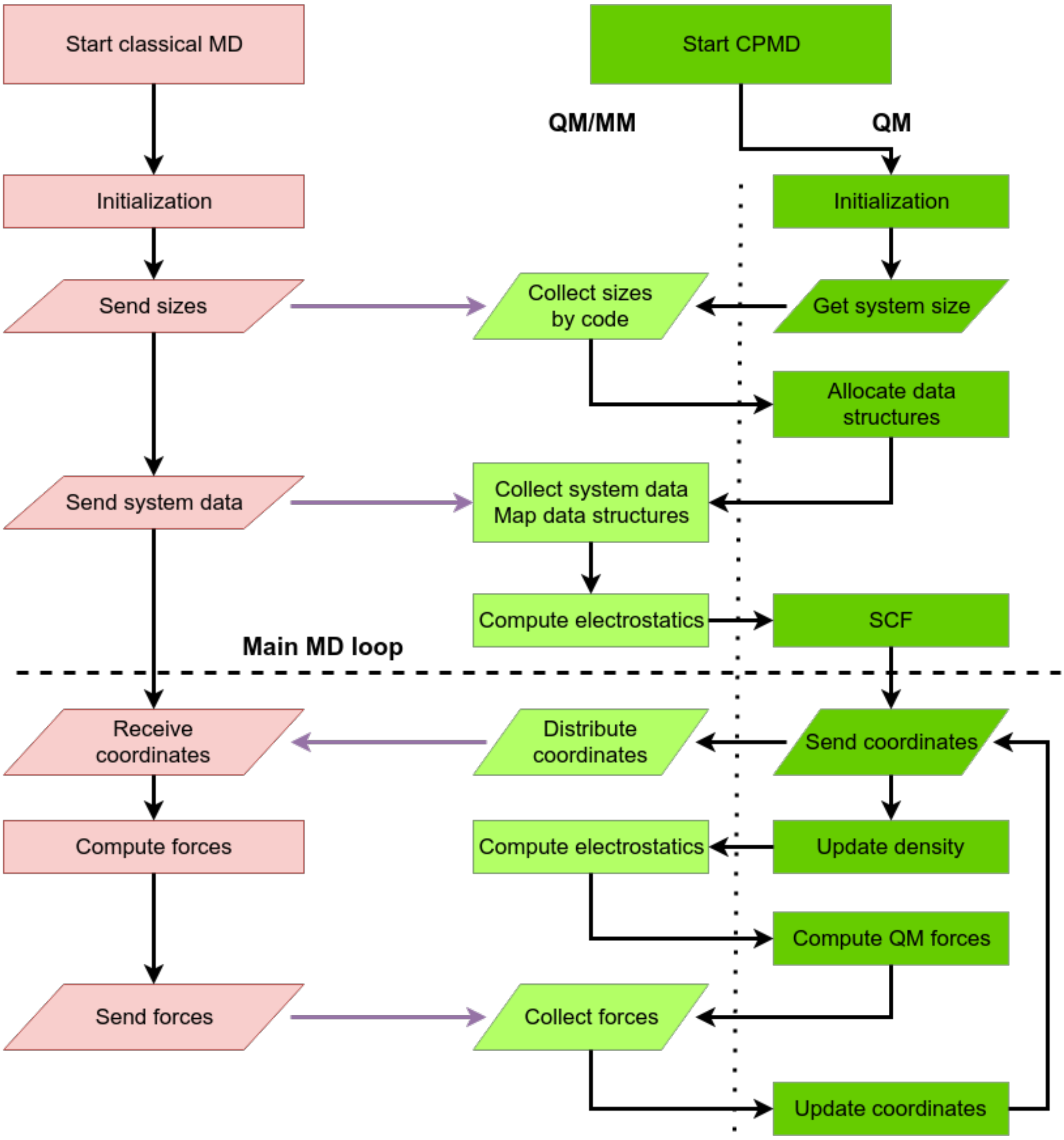

Figure 4 CPMD QM/MM-enabled simulation workflow.

5.1 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 run independently occasionally exchanging data. In order 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) overcome possible licensing conflicts between the CPMD's proprietary license and the one of the classical molecular dynamics code.

5.2 Copyright and License

Future: Discussion about under which kind of license to release the new QM/MM interface is ongoing among the developer team. The Lesser GNU General Public License (LGPL) v2.1 appears currently the preferred choice even if this would prevent 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.

5.3 Version control

Git is used for version control. For the initial stage of the development we are using the merge-based Github workflow. A stable master branch will be used to create builds. All changes should be 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 two-level testing procedure is planned to be used as soon as the initial development stage is over. 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 will be triggered. Since, currently we do not have enough coding base to build integration tests we are currently using only unit tests. 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.

5.4 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 not to be sufficient.

5.5 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.) have to be covered by unit tests. Failing to comply with this requirement 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 in order to assure the portability of the code.

Once a viable product exists, its performance overhead compared to e.g. a pure QM calculation will be measured to see if performance optimization is warranted.

5.6 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. As soon as the development has reached the release state, the development process will become more open.

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 5.

5.7 Code commenting

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

5.8 Code standards and styles

The code is being developed to comply with Fortran2008 standard. Coding style is loosely based on the Best Practices proposed here: http://www.fortran90.org/src/best-practices.html. 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) in order to make the maintenance of the code easier.

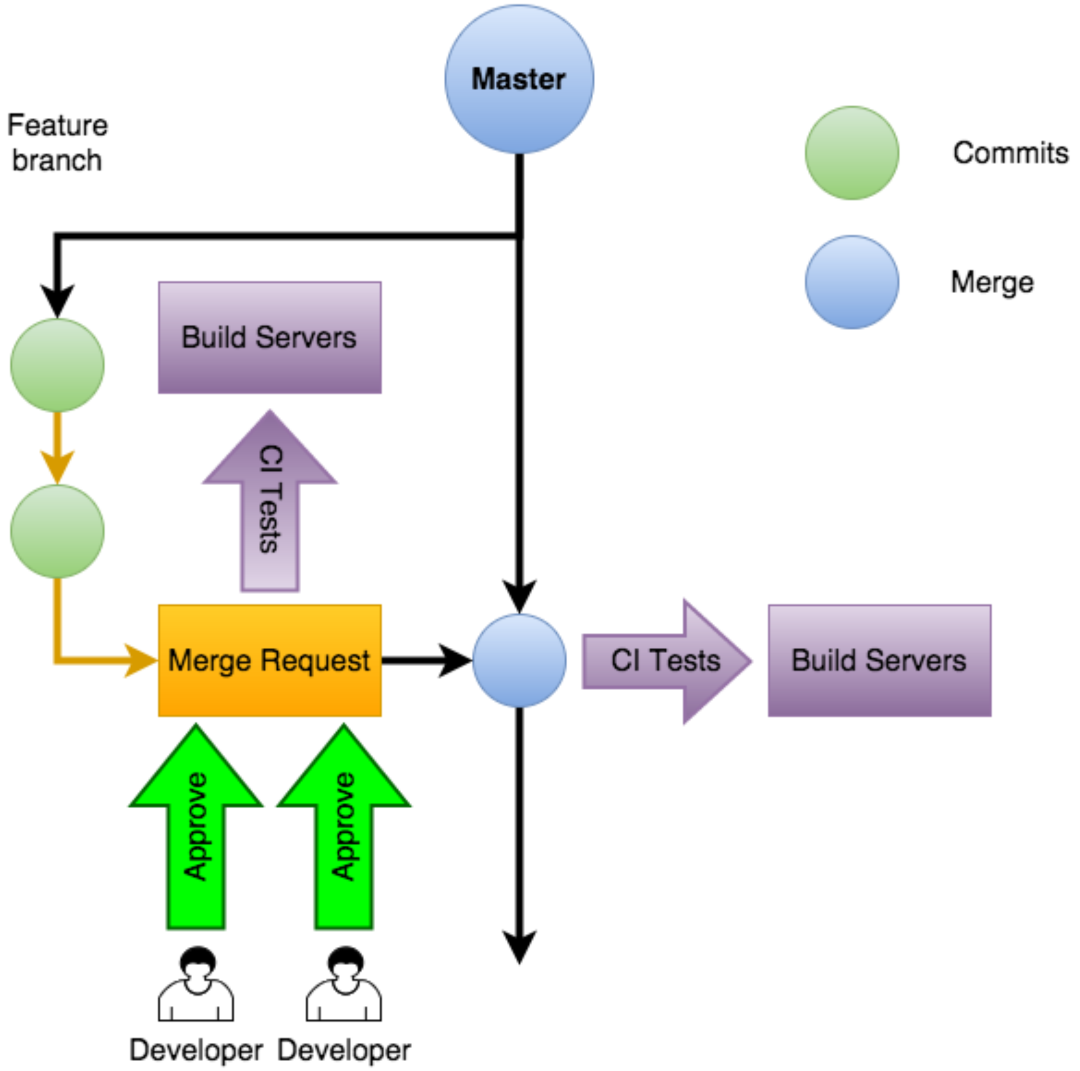

Figure 5 QM/MM development workflow during initial MiMiC development phase.

6 Development process details for pmx

pmx is a Python-based software package [15, 16] 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.[17] The basic scheme underlying the workflow of the automated hybrid protein structure/topology generation is depicted in Figure 6. 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 is currently under development.

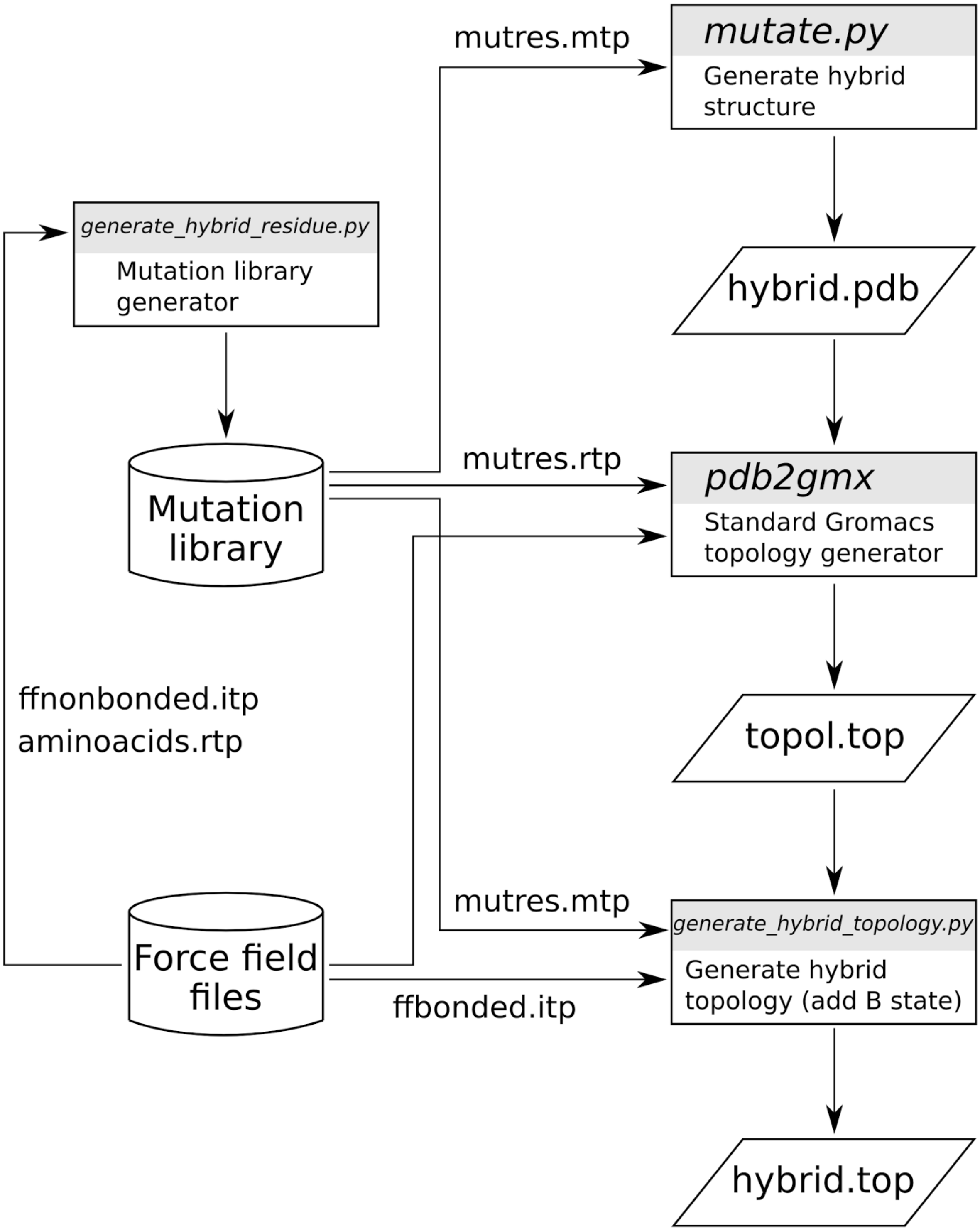
mutres.mtp
mutate.py
Generate hybrid structure
generate_hybrid_residue.py
Mutation library generator
hybrid.pdb
Mutation library
mutres.rtp
pdb2gmx
Standard Gromacs topology generator
ffnonbonded.itp
aminoacids.rtp
topol.top
mutres.mtp
generate_hybrid_topology.py
Generate hybrid topology (add B state)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

Figure 6 Workflow of the pmx-based hybrid structure/topology generation for an amino acid mutation.

6.1 Architecture

pmx implements a number of classes for the structure and topology handling (Figure 7). The setup for the alchemical single topology simulations of the amino acid, dna or ligand modifications comprises a number of scripts (Figure 6) that utilize the data structures in Figure 6. 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.

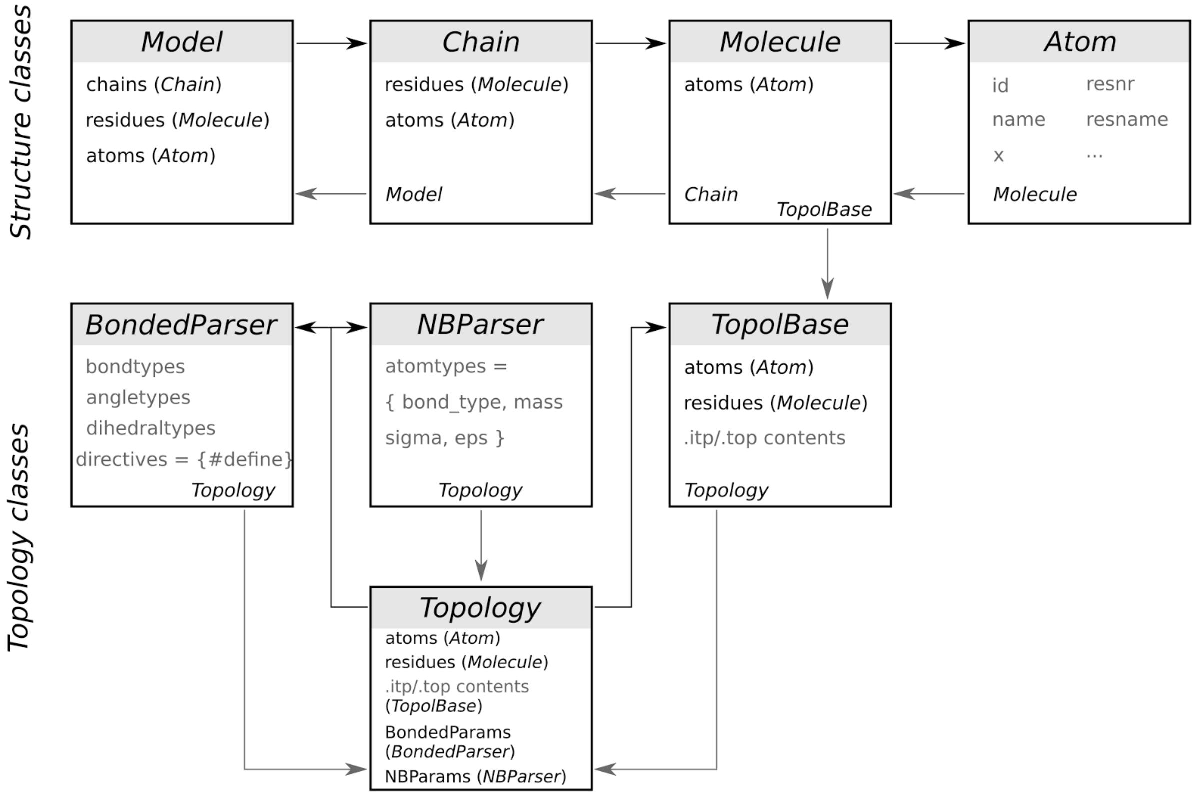

Figure 7 The main classes underlying the pmx architecture.

6.2 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.

6.3 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.

6.4 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

6.5 Review Process

pmx is actively developed by one person, thus no independent review process is currently implemented.

Future: when resources permit, and the benefit to enough users is clear, add a code review process

6.6 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.

6.7 Testing process

Currently the testing is performed internally after generating new mutation libraries and software releases.

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.

6.8 Release Process

The releases are guided by the implementation of new features and bug-fixes. The updated versions are immediately pushed onto GitHub. The version control and documentation of the changes is provided by the Git version control system.

6.9 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.

6.10 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.

6.11 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.

6.12 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. The requests can be sent by email directly to the developers or posted on GitHub. Major novel features to be added in the near future are support for nucleic acids and ligands.

6.13 Code commenting/standards/styles

pmx is written in Python 2. The code does not adhere to any specific style recommendations. The pmx source code contains a bare minimum of comments, but tries to maintain an understandable class, method and variable naming convention. This has proven a robust and extendable framework since the first release in 2010.

7 Concluding remarks

Each of the software sub-teams within Work Package 1 of BioExcel has described their current software-development process, including plans for quality assurance and testing, along with improvements they plan to make. These plans and this document are intended to ensure the long-term usefulness of the pilot codes, with a particular view to smoothing over expected hardware-related turbulence in the exascale era. Progress in biomolecular science requires that hardware and human resources are used efficiently, and must be based on results that are capable of reproduction in the long term, and the BioExcel software development processes are likely to achieve this, given the resources available.

In project month 17, two further deliverables are due. Deliverable 1.2 will make a project release of all the codes, implementing appropriate stages of the work in the Description of Action. Deliverable 1.3 will describe a roadmap of future hardware relevant to exascale development, with a focus on the interests of the biomolecular simulation community, and long-term development plans for each pilot code. If appropriate based on subsequent developments, improvements to this specification may be noted then.

8 References

1. Kelly, D.F., *A Software Chasm: Software Engineering and Scientific Computing.* IEEE Software, 2007. **24**(6): p. 120-119.
2. Killcoyne, S. and J. Boyle, *Managing Chaos: Lessons Learned Developing Software in the Life Sciences.* Computing in science & engineering, 2009. **11**(6): p. 20-29.
3. Segal, J. and C. Morris, *Developing Scientific Software.* IEEE Software, 2008. **25**(4): p. 18-20.

4. Wilson, G., et al., *Best Practices for Scientific Computing.* PLoS Biol, 2014. **12**(1): p. e1001745.
5. Boehm, B.W., *Seven basic principles of software engineering.* Journal of Systems and Software, 1983. **3**(1): p. 3-24.
6. Kroll, P. and P. Kruchten, *The rational unified process made easy: a practitioner's guide to the RUP*. 2003: Addison-Wesley Longman Publishing Co., Inc. 397.
7. 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.
8. Dominguez, C., R. Boelens, and A.M.J.J. Bonvin, *HADDOCK: A Protein−Protein Docking Approach Based on Biochemical or Biophysical Information.* Journal of the American Chemical Society, 2003. **125**(7): p. 1731-1737.
9. de Vries, S.J., et al., *HADDOCK versus HADDOCK: New features and performance of HADDOCK2.0 on the CAPRI targets.* Proteins: Structure, Function, and Bioinformatics, 2007. **69**(4): p. 726-733.
10. de Vries, S.J., M. van Dijk, and A.M.J.J. Bonvin, *The HADDOCK web server for data-driven biomolecular docking.* Nat. Protocols, 2010. **5**(5): p. 883-897.
11. van Zundert, G.C.P., et al., *The HADDOCK2.2 Web Server: User-Friendly Integrative Modeling of Biomolecular Complexes.* Journal of Molecular Biology.
12. Car, R. and M. Parrinello, *Unified Approach for Molecular Dynamics and Density-Functional Theory.* Physical Review Letters, 1985. **55**(22): p. 2471-2474.
13. 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.
14. 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.
15. Seeliger, D. and B.L. de Groot, *Protein Thermostability Calculations Using Alchemical Free Energy Simulations.* Biophysical Journal, 2010. **98**(10): p. 2309-2316.
16. Gapsys, V., et al., *pmx: Automated protein structure and topology generation for alchemical perturbations.* Journal of Computational Chemistry, 2015. **36**(5): p. 348-354.
17. 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.