Boosting Group-level Synergies by Using a Shared Modeling Framework

Yunus Sevinchan[1 [0000−0003−3858−0904]],
Benjamin Herdeanu[1,2 [0000−0001−6343−3004]], Harald Mack[1 [0000−0001−7787−9496]],
Lukas Riedel[1,3,4 [0000−0002−4667−3652]], and Kurt Roth[1,3 [0000−0003−2634−8825]]

[1] Institute of Environmental Physics, Heidelberg University, Germany
{first.last}@iup.uni-heidelberg.de — ts.iup.uni-heidelberg.de
[2] Heidelberg Graduate School for Physics, Heidelberg University, Germany
[3] Interdisciplinary Center for Scientific Computing, Heidelberg University, Germany
[4] Heidelberg Graduate School of Mathematical and Computational Methods for the Sciences, Heidelberg University, Germany

Abstract. Modern software engineering has established sophisticated tools and workflows that enable distributed development of high-quality software. Here, we present our experiences in adopting these workflows to collectively develop, maintain, and use research software, specifically: a modeling framework for complex and evolving systems. We exemplify how sharing this modeling framework within our research group helped conveying software engineering best practices, fostered cooperation, and boosted synergies. Together, these experiences illustrate that the adoption of modern software engineering workflows is feasible in the dynamically changing academic context, and how these practices facilitate reliability, reproducibility, reusability, and sustainability of research software, ultimately improving the quality of the resulting scientific output.

Keywords: Research software · Software quality · Reproducibility · Complex systems · Modeling · Scientific computing

1 Introduction

Software has become an integral part of modern research, be it for the control of experiments, analysis of data, or computer simulations. In recent years, however, the research community has become aware of issues relating to the reliability and reusability of software developed and used for scientific purposes [2,6,10]. These shortcomings are mainly attributed to the unique characteristics of scientific software development and a resulting disconnect between the research and software engineering communities [5]. Ultimately, these issues impede the progress of scientific research. At the same time, the Open-source Software (OSS) community has grown vigorously and is successfully developing high-quality software. This is made possible in large parts by platforms like GitHub and GitLab, which greatly simplify collaboration on software projects, and by the adoption of software engineering workflows which have proven valuable for efficiency of development, long-term maintainability, and reliability.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

Although OSS development always had a major role in scientific computing, these approaches are now gaining more traction in the wider scientific community [3]. Additionally, several institutions have been formed which aim to counteract the negative effects arising from current practices in the development and use of research software. For example, to increase the recognition and visibility of reliable and sustainable research software projects, a number of relatively young journals like the Journal of Open Research Software and the Journal of Open Source Software focus solely on publishing high-quality open-source research software. Furthermore, the extension of FAIR principles to research software is currently being discussed [8] with the goal to improve the findability, accessibility, interoperability, and reusability of software used in the scientific context.

When studying complex and evolving systems – the research field in which we operate –, computer models[5] are considered the main research method. Given the nature of the research questions in this field, models for a huge diversity of phenomena and ever-changing situations need to be conceived. This is in contrast to other research fields like weather prediction, oil recovery, and combustion optimization where the challenge is to investigate a specific scenario with the highest attainable efficiency using numerical simulations of well established systems, like flow-, transport-, and reaction-equations.

Subsequently, the implementation and analysis of these models often requires the development of custom software, thus entailing the same challenges as experienced for other research software. For instance, we observed that code was frequently written from scratch because the work involved in understanding and adapting an existing model or simulation tool was higher than writing it anew. This not only led to redundant implementations but also to the repetition of mistakes, and overall unreliable and unsustainable code. In effect, software was written by a single researcher for the purpose of their project and discarded soon afterwards, missing out on the collaborative aspect of modern research.

In this paper, we present the approaches we took on the level of our research group to address these issues. The aim was not only to improve the quality of the software we developed and used, but to also boost synergies within the group. We did so by collaboratively developing a modeling framework, using it throughout the group, and creating associated communication structures and workflows. We perceived such a framework to be the ideal focus point for promoting collaboration. At the same time, we saw this as a way to gain experience in developing reliable and sustainable software that not only adheres to best practices in software engineering, but also advocates them to young researchers.

The modeling framework resulting from this effort is called Utopia [12] and aims to be the central tool in all stages of a modeling-based research workflow, providing solutions for common problems based on state-of-the-art programming practices. Utopia is publicly available[6] under an open-source license. Since the

[5] With the term *computer model* we denote a conceptualization of a real-world system that is investigated via computer simulations. When solely the implementation in code is concerned, we use the term *model implementation*.

[6] https://ts-gitlab.iup.uni-heidelberg.de/utopia/utopia

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

start of the project in early 2018, it formed the basis for more than thirty new modeling projects in our group and has been used as a teaching tool in graduate physics courses. In the context of this paper, the specific details of the framework are less relevant than the conceptual role it fulfilled in improving research software quality and enhancing collaboration within the group: It provided a platform to work on, a common language to communicate in, and a shared goal to motivate everyone involved.

As such, other modeling frameworks may fulfill the same role as Utopia did for us. One example is the NetLogo programming language and development environment [13], which can be used to study a wide variety of agent-based models.

In the following, we first aggregate the synergies we see arising from sharing a modeling framework (section 2) and outline which software engineering workflows we regard as feasible to adopt in such a context (section 3). We then describe the experiences we had in developing Utopia as a group, working with it as researchers, and using it for academic teaching (section 4). By sharing these experiences, we hope to contribute to the effort of improving the responsible development and use of research software.

2 Synergies

Synergies in research environments surely are of a wide variety and depend very much on the necessities and methods of the respective fields. Here, we focus on the investigation of complex and evolving systems using computer models, where common workflows allow the use of a shared modeling framework.

In our experience, a typical workflow in this field can be represented as a four-stage process:

1. **Conceptualize** a research question into a model system
2. **Implement** the model system as a computer model
3. **Generate** simulation data using the model implementation
4. **Analyze** the simulation data, extract results, and reflect on the model

The above aims to categorize the workflow into stages that have different qualitative demands on the methods, the software tools, and the interaction with colleagues, and subsequently varying potential for synergies. In practice, such a workflow is hardly ever sequential, but an iterative and self-referential process. Furthermore, the wide diversity of phenomena under investigation requires a high flexibility in all stages of the workflow, allowing for constant redefining, redesigning, reimplementation, and analysis. While this formulation originates from observations within our group and the study of complex and evolving systems, we believe that scientific research in other areas might be represented in a similar fashion, especially when working with computer models (e.g., [4]).

We call a framework that covers all the stages of the typical workflow a *comprehensive modeling framework*. By modularizing the tools on the level of the whole research workflow, it abstracts away technicalities and allows researchers to think on the level of the model system they want to investigate.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

Simulation Infrastructure We identified the following software functionalities as common infrastructure during the aforementioned modeling workflow: *(i)* storing and managing simulation data, *(ii)* configuring simulations, i.e., passing parameters to the simulation, *(iii)* performing parameter sweeps, *(iv)* data processing and analysis routines, and *(v)* data visualization. In cases where the typical workflow of the involved researchers is comparable, as sketched above, sharing these infrastructure tools via a framework becomes beneficial and greatly reduces the time and effort needed to get operational in all its stages. Importantly, the framework can provide a more reliable and more sophisticated feature set, which would not be feasible to be implemented by an individual developer.

One example are parameter sweeps, as are often required for sensitivity analysis of a model's behavior. A framework-level implementation can generate the set of configurations programmatically, based on user input, and can make use of naive parallelization to speed up the generation of simulation data, while being completely independent of the model implementation itself. Having such a feature available as part of a framework is a direct benefit to all framework users.

In situations where more flexibility is required than can be offered by the framework, the use of open data standards helps avoiding a lock-in effect which would be detrimental to the use of the software and the researchers' freedom.

Modeling Techniques The investigation of complex and evolving systems has generated a number of modeling techniques, the most notable being *(i)* cellular automata, which can be used as a representation of spatially extended systems, *(ii)* individual- or agent-based models, which represent individual entities with a varying range of autonomy, agency, and perception, and *(iii)* network-based models, which put the focus on these agents' interactions with each other.

By integrating frequently used modeling techniques into a shared framework, the knowledge about common pitfalls can be used to avoid the repetition of mistakes. Furthermore, taking into account the central importance of correctness and efficiency of these implementations, they can be thoroughly tested, reviewed by multiple developers, and optimized for performance. We see these aspects as the key arguments for an implementation on the framework level, thus achieving a higher reliability, efficiency, and usability compared to software developed in an uncoordinated manner.

A modeling framework should also allow researchers to leverage the great variety of research software already available in the scientific field, which often have been tried and tested for decades and hence attained a quality virtually unreachable by most individual researchers. These might for example provide efficient implementations of graph data structures or numerical algorithms. A tight integration of existing solutions is often desirable, but needs to be such that particularities are abstracted away to not demand too much additional knowledge from the user of the framework.

Feature Sharing A shared framework allows to implement new features as part of the framework rather than in an isolated manner, making them easily

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

accessible to other users. Such a framework-level implementation often requires a higher degree of generality for the feature in order to make it useful for other users. At the expense of additional work in abstracting the functionality, this facilitates collaboration with other developers, ultimately improving reliability, usability, integration, and performance.

A Common Conceptual Language When using the same modeling framework in a research group, not only the code base is shared, but also the conceptual language in which discussions and the exchange of ideas take place. This can be beneficial in multiple stages of the research workflow. For instance, when designing an implementation of the model system, the modeling techniques provided by the framework supply a set of abstractions that can be used when talking about the model, e.g., the concept of what constitutes an entity or the vocabulary describing a specific algorithm.

Benefits also arise from an easier understanding of code written by other researchers. The overall amount of code is reduced by the use of the simulation infrastructure and the framework-level implementations of the modeling techniques. This not only facilitates a closer and more efficient interaction between researchers on the level of the implementation, but also simplifies reusing model implementations.

Transparency In a research context, model implementations are typically privately developed until they are made public alongside a publication. Within a research group, we see several positive effects of making the source code of model implementations openly available right from the beginning of development: By reading the code written by other members of the research group, possible implementation approaches can be exchanged and spark discussions. Furthermore, members of the research group are encouraged to learn from other developers' implementations, which pertains not only to modeling ideas but also to software engineering practices in general. This openness is an important aspect in a dynamic environment where people and problems change frequently.

While the transparency brings significant benefits, it also demands a careful consideration of collateral effects, including premature diffusion of novel ideas – even within the research group – and the potential for surveillance. We see a common understanding of intellectual property as a key requirement for this approach. Consequently, having model implementations visible probably works well within a research group, but not necessarily in a large community.

3 Adopting Software Engineering Workflows

Developing research software collaboratively allows the use of software engineering workflows that rely on divided responsibilities and interactions with other group members. In the following, we highlight the aspects we perceive as particularly beneficial for the quality of collaboratively developed research software, while not posing a prohibitively large learning or management overhead.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

Effective Use of Version Control Systems Version control systems (VCS) are not only valuable to an individual researcher but enable workflows that form the basis of efficient collaboration. Platforms like GitHub and GitLab go beyond the mere hosting of VCS repositories: they provide task management tools, communication channels, contribution workflows, automation services, and interfaces to other services that allow for custom extensions.

Research software can and should make use of these platform-level tools that have proven themselves highly valuable to a successful software engineering workflow. While such a procedure may pose additional work for a single researcher, there are considerable gains to be expected for consistent work as a group.

Code Review Code review serves the purpose of improving the quality and maintainability of software [9]. Reviewers inspect source code changes not only with respect to potential defects that may not be found through static code checks, but also suggest improvements to the implementation, its readability, or its integration into the larger project. Furthermore, such a process can serve to transfer knowledge about the code base between the authors and the reviewers, while at the same time strengthening their team identity [1].

When working with VCS and collaborative version control platforms, code review can be conveniently carried out on so-called *Pull Requests* or *Merge Requests*, i.e., on the proposed changes to a code base, prior to merging them into the main branch of a repository. The platforms usually present the suggested changes alongside the code they will replace, identify code ownership, and allow reviewers to comment on the changes and propose improvements. This platform-assisted code review is already well-established in the OSS community. In the context of software engineering for research software, the same benefits can be expected, and would directly address issues like reliability, consistency, and general code quality. Given the often heterogeneous software engineering skills in an academic context, knowledge transfer mediated by code review may also gain an important role.

Testing & Automation While software testing is a cornerstone of professional software development, testing of scientific software is intrinsically more difficult. Kanewala and Bieman [7] identify the challenges of testing scientific software to be either due to characteristics of the research software itself – like the oracle problem –, or to be caused by "cultural differences" between the software engineering community and scientists. They come to the conclusion that while some challenges are unique to scientific software, others can be overcome by incorporating existing testing techniques from software engineering into their development workflows.

In modeling-based research, tests are required both for the framework code and the model implementations. When collaboratively developing software, the importance of both of these can be emphasized, and the implementation of tests can be simplified to become more accessible to novice developers. At the same time, techniques that make the testing of model implementations possible can be

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

conveyed, while also identifying in which areas further testing techniques would be required. Putting a larger emphasis on these parts of software development can also inspire the implementation workflow itself, e.g., by promoting test-driven development of models.

With growing size of a software project, automation becomes a crucial part of the development process. The term refers to a set of actions that are automatically carried out, e.g., when pushing code to the remote server or when merging a feature into the main branch. With these tools being readily available via the same platform that the software is developed on, automation becomes easily accessible. Benefits for the development of research software and the individual model developers are that associated tests are carried out automatically, taking that burden off the researcher, and making it easier to detect breaking changes in both the framework and the model implementations.

4 Utopia – Three Case Studies

The context of these case studies is our research group that focuses on the investigation of complex, chaotic, and evolving environmental systems. Development of Utopia started early 2018 with a team of four PhD candidates[7], two MSc students and two BSc students. Utopia has been used in more than thirty projects, with usual durations of one year and up to fifteen simultaneous projects. Students joining our group typically have a physics background and entry-level programming experience, mostly in Python, but are unfamiliar with version control, testing, or other software engineering workflows.

4.1 Developing the Utopia Framework

In this case study, we focus on the experiences from the collective development of the Utopia framework and the adoption of software engineering workflows throughout this process.

VCS Workflows & Code Review Utopia was developed in a GitLab project on a self-hosted GitLab instance. We chose to use the *GitHub Flow* branching model that has little management overhead and focuses on the main branch always being in a working state: Feature branches start off the main branch and merge back into it. Each feature branch corresponds to the implementation of a single, well-isolated task, which often is planned in a GitLab Issue.

We adopted a change-based code review process for the Utopia framework. All code changes require a review by at least one other developer before being allowed to be merged into the main branch. Code review takes place in the GitLab Merge Request interface, where the author of the request gives a brief description of the changes, why they were necessary, and how they were achieved, and points out changes which require further discussion. The author then assigns a

[7] YS, BH, HM, and LR

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

person for review, typically someone who is already involved in the task or who is familiar with the affected framework structures. The reviewer then goes through the changes and may comment directly on the code to discuss the changes or suggest improvements. Moreover, they may include other developers into the review process. If everybody involved approves of the changes, the Merge Request is merged and the corresponding Issue is closed automatically.

We perceived the review process as highly beneficial as it considerably improved the resulting code in terms of consistency and reliability. All Merge Requests that included substantial code changes or additions profited from code review, be it through the detection of potential defects or improvements relating to the robustness of the implementation. Alongside, the documentation (both of the code itself and the usage of the overall framework) profited from a thorough review process because reviewers could use it as a starting point and evaluate how well the changes could be understood by someone who did not implement the code. Furthermore, the review process helped conveying knowledge about Utopia's structure, inner workings, and agreed-upon guidelines, thus gaining an important role in increasing maintainability of the project.

These benefits greatly offset the additional time and effort that developers needed to invest into code review.

Testing & Automation Thorough testing procedures are commonplace in software engineering and a prerequisite for projects to grow in a sustainable fashion. Accordingly, we require of all Utopia features to have a corresponding test implemented that covers the relevant use cases.

We make use of GitLab CI/CD for automated test execution and building of the framework in different build modes. Furthermore, the pipeline generates a code coverage report, and deploys a preview of the documentation and a ready-to-use Docker image of Utopia. All these automations proved highly valuable in the code review process where another requirement for merge approval is that the pipeline passes successfully and the code coverage report was inspected. That way, the pipeline provides a "ground truth" irrespective of the individual developers' systems and allows to easily detect regressions in seemingly unrelated parts of the framework.

While not feasible in all situations, this workflow also allowed test-driven development approaches which we found helpful when addressing previously undetected bugs: the bug can then first be reproduced by a new test case, which fails initially; subsequently, the code is adjusted such that the test passes.

Partaking in Framework Development Through the focus of the framework to make the modeling workflow as convenient as possible, exposure to low-level code was inevitably reduced. Potential contributors thus developed the perception of not having enough insight into the inner workings of the framework to suggest changes or improvements to it.

As a counteraction, we organized multiple *Coding Weeks* which were preceded by a planning phase with regular meetings. During the planning phase, new

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

contributors got to know those parts of the framework that were relevant for a particular improvement or new feature. Depending on the task, they then worked alone or in a small group to implement it, add test cases and documentation, and jointly review all of it. Overall, these events proved to be not only a group-forming experience, but also very productive, especially when it came to the coordination and implementation of larger features. One important long-term effect was that people who took part in a Coding Week were more likely to also contribute thereafter.

While it needs to be acknowledged that not everyone who is using the framework is also interested in improving it, we think that experience with these collaborative workflows has value beyond the currently undertaken project. We currently try to lower initial thresholds for new contributors by keeping everyone involved in the development of Utopia as much as possible and suggesting starting points for contributions.

4.2 Using Utopia throughout the Research Group

In this case study, the focus is on usage of Utopia for the implementation and investigation of models of complex and evolving systems, the synergies that developed from using the shared modeling framework, and the integration of software engineering workflows into that process.

Individual model implementations were part of a single, group-internal GitLab project, separate from the framework project.

Learning Curve To make the entry as easy as possible and thereby facilitate usage of the shared framework, we put a focus on providing detailed introductory guides. These were meant especially for students entering the research group for a BSc or MSc project, which have a typical duration of 3–4 months or about one year, respectively. The aim was to accustom these young researchers with the Utopia framework such that they can install it within a day, are capable of using essential features within a week, and can start with a model implementation soon after.

While learning to use a new framework may constitute a considerable overhead, we had positive experiences with the adoption of Utopia in the group and the speed with which models were implemented or adapted. Despite the fact that Utopia models are implemented in C++, most newcomers were quick to setup a new model using the corresponding guides, and subsequently start to implement the desired model dynamics using the abstractions and modeling techniques provided by the framework.

While we encouraged newcomers to search the documentation and the GitLab project in case of questions, some personal assistance was almost always necessary to help resolving particularities in the installation or the setup of models. Intermittently, this created a substantial additional work load for the more experienced developers. For reducing this work load, it proved crucial to reflect even seemingly minor problems in the project documentation and develop a shared corpus of knowledge among framework users.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

Synergies A central motivation for using a shared modeling framework was to boost the development of synergies within our research group. For us, this proved to be a successful and overall beneficial approach, albeit not without challenges and a certain maintenance cost. For instance, owing to the different schedules of research projects, we experienced some diffusion of knowledge and loss of experience. Thus, this approach hinges on a critical mass of users and developers carrying on the required knowledge to maintain the shared code base and offer assistance to new group members.

Simulation Infrastructure, Modeling Techniques & Feature Sharing The aim of eliminating repeated implementation of simulation infrastructure, modeling techniques, and associated tools was largely achieved and the benefits extended over the whole modeling workflow, from conceptualization to data analysis and visualization. Sharing the simulation infrastructure was a very clear benefit, because the overlap of required software tools was very high within the research group. Also, some of the more sophisticated features like parameter sweeps with multi-node cluster support, uniform manager structures for the provided modeling techniques, or generalized data post-processing and plotting could not have been feasibly developed outside such a framework. Having these features implemented into the framework effectively addressed most of the issues we observed in solitary software development. Especially for the PhD candidates, Utopia improved work efficiency in all stages of the modeling workflow by alleviating points of friction and providing a flexible, scalable, optimized, and reliable development environment.

However, in some cases, the generalization of features required considerably more effort than if these features would have been implemented directly where they were needed; in that sense, we sometimes fell prey to "premature generalization", thinking that a framework-level feature would always provide a benefit. Also, discoverability of features proved to be more challenging than expected: Despite documentation, users sometimes did not associate the functionality they desired with the description of an already existing feature, which in some cases led to a redundant implementation by the users.

Common Language & Transparency Having all code openly accessible for everyone in the group proved to be one important factor for improving collaboration: Group members frequently inspected existing implementations to learn about representation details and model dynamics. This not only facilitated discussions, but also provided a means of knowledge transfer, often extending to programming language features and software engineering best practices. Notably, a number of projects are under way that build on existing model implementations. The common conceptual language and the use of the framework made this process efficient by abstracting away most of the code that relates to simulation infrastructure and modeling techniques.

Regarding transparency, we observed some reluctance in newcomers to share model ideas and code with the rest of the group. We ascribe this partly to the need to learn the corresponding workflows, but also to a certain insecurity

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

regarding code quality. This reluctance typically subsided after feeling more integrated into the group and having experienced the benefits of sharing code with other group members.

Software Engineering Workflows Aside the synergies developing in the group, we also wanted to promote software engineering workflows during model development. Our observations varied for experienced and novice developers.

Experienced model developers profited significantly from the easy availability of the testing framework and automation, and adopted VCS workflows for their model implementations, e.g., by working on their models in a task-based fashion with frequent and granular Merge Requests.

For larger models, the implementation of tests proved indispensable. Here, the framework assisted mainly by simplifying the implementation of C++ unit tests and providing an easy-to-use interface to generate and validate simulation data; these approaches were often viable to assert the expected microscopic and macroscopic model behavior, respectively. Furthermore, by including all tests into the GitLab CI/CD alongside the framework, not only the model behavior was tested automatically but also the effect of framework changes on the models, which was crucial when working on more intricate parts of the framework.

Unlike for framework code, we did not require in-depth code review for individual model implementations, mainly due to the high work load such a policy would generate for reviewers. Nevertheless, experienced developers often requested code review from others; given the abstractions the framework provided, this was feasible for smaller Merge Requests or when the author asked for feedback on specific parts of their implementation. Despite the restrictions, this proved to be a workable compromise between a researcher's responsibility for their model and the work load for others, thus still providing mutual benefits.

For novice developers, our experiences were mixed: the adoption of SE workflows by newcomers was not as natural as we had hoped, but the benefits were typically acknowledged and, in the long run, some of the procedures found their way into their work with the framework. We observed the adoption to be impeded mostly by the learning required to understand the new procedures and their benefits. With a physics background, many of the workflows are completely new to MSc- and BSc-level students. Especially in the first weeks and months of a project, the focus typically is on the scientific literature and the model formulation and implementation itself, not on software engineering aspects. We found that adoption increases only once developers become more proficient and see these methods as solutions to new problems arising during development.

For example, learning to use VCS and GitLab and adapting personal workflows to accommodate these was often perceived as a substantial investment without clear benefits for the students' own projects. This was despite their acknowledgment that these tools are useful in the development of the framework or in software development in general. In effect, for new users, a decoupling from the rest of the repository and group was quite common in the first weeks

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

or months of a project: Once a working setup was achieved, development often continued on a local feature branch for the new model and with a fixed (i.e., outdated) version of the framework. We made similar observations regarding the implementation of tests: It was generally acknowledged that having tests would be beneficial, but their implementation was typically not a priority.

For individual model development, we deliberately decided against imposing strict workflows. Instead, we promoted their benefits and tried to reduce entry barriers by providing guides, personal assistance, and framework tools that made adoption easier. Thereby, we aimed at eliciting the motivation to learn and integrate these tools, rather than enforcing their use. This proved successful on a longer time scale and with developers becoming more experienced with Utopia: After the first few months of many MSc projects, the projects were kept up-to-date, GitLab became more regularly used, and test implementations became more frequent. We believe that without Utopia, many novice developers would not have considered adopting these software engineering techniques and would not have reached the same level in their actual research. Utopia not only promoted the utility of these tools, but also reduced the barrier of using them.

Currently, we aim to improve group-level interactions through having regular meetings for questions and discussions on the framework and model implementations. Similar as with the Coding Weeks, we find these meetings to have a group-building effect, and the higher interaction having a positive effect on cooperation. As part of this process, we are also promoting pair-wise code review alongside the propositions made in [11].

4.3 Using Utopia in Teaching

Possible use cases for Utopia in academic teaching range from the generation of simulation data using existing models to the implementation and investigation of new models, as it is done in the regular research context. So far, Utopia was used for teaching in an MSc physics lecture (summer term 2019) and is currently (winter term 2019/20) used in an MSc physics seminar. With the seminar still under way at the time of writing of this paper, we present the experiences from using Utopia as part of the mandatory exercises accompanying the *Complex, Chaotic, and Evolving Environmental Systems* lecture.

There, students used Utopia to run and analyze well-established models of spatially distributed complex systems: the Forest Fire Model, the Contagious Disease Model, and the Predator-Prey Model. The focus was on the investigation of the behavior of the models, not on their implementation, which is why we provided these implementations alongside with a detailed description of the implemented dynamics and available configuration parameters. The exercises instructed students to generate simulation data for changing configuration parameters and subsequently use the framework to analyze the data and understand the model behavior, e.g., by extracting macroscopic fixpoints or by investigating power spectra to characterize spatial structures.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

To make the setup as easy and machine-agnostic as possible, we used the automatically deployed and ready-to-use Docker image of Utopia[8], which contained all relevant model binaries. Furthermore, a Jupyter Notebook server was set up to run inside the Docker container, allowing students to work with Utopia from an interface that is more easily accessible than the command line. Setting up the Docker environment and working through a tutorial on the usage of Utopia constituted the first exercise of the lecture. We subsequently dedicated one session of the tutorials to questions arising from this exercise; this was sufficient to get students operational on their own computers.

We as teachers profited from not having to implement standalone models and bare-bones simulation infrastructure, but having a fully-fledged modeling framework available that we already had experience with. Furthermore, any effort put into model implementation or documentation enhancements was a direct benefit to other framework users.

For students, benefits included having a single and simple-to-use interface with which to investigate all provided models. While learning to use Utopia created additional challenges compared to using isolated model scripts, we see working with a modeling framework as a valuable skill and perceived this to be an appealing aspect for students interested in going into modeling-based research.

5 Conclusion

Developing flexible yet reliable software is a challenge. This is particularly true in the academic context where people remain for just a few years and typically have little experience with modern software engineering workflows. At the same time, the nature of highly dynamic research fields puts high demands on the quality of research software, e.g., in terms of reproducibility and reliability.

In the above case studies, we described our experiences with collectively developing and using Utopia, a modeling framework for complex and evolving systems. Our primary aim was to demonstrate that this approach is feasible and how it can boost synergies within the research group. Adopting these practices not only resulted in beneficial effects on the group's dynamics, but improved the quality of the developed research software and the resulting research output. We also showed that in order to sustain these workflows, the group with its ever new members has to be motivated and instructed in events like Coding Weeks and regular question-discussion meetings. We aim to further promote both this approach and the Utopia framework itself, being aware that software frameworks need to be continuously maintained and improved in order to survive.

Taken together, we exemplified how modern software engineering workflows allow to successfully tackle a large and rapidly changing research field in a comprehensive way which is otherwise only feasible for larger and more permanent institutions. We trust that the collaborative philosophy that is facilitated by such approaches will propagate via its users and thereby contribute to sustainable development and responsible use of research software.

[8] https://hub.docker.com/r/ccees/utopia

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32

Acknowledgments

We thank the reviewers of this submission for their thoughtful remarks and suggestions and Maria Blöchl for valuable feedback on an earlier version of this manuscript. We are grateful to all contributors and users of Utopia to have participated in this collective effort.

References

1. Bacchelli, A., Bird, C.: Expectations, outcomes, and challenges of modern code review. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 712–721. ICSE '13, IEEE Press (2013)
2. Hannay, J.E., MacLeod, C., Singer, J., Langtangen, H.P., Pfahl, D., Wilson, G.: How do scientists develop and use scientific software? In: 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering. IEEE (2009). https://doi.org/10.1109/secse.2009.5069155
3. Heaton, D., Carver, J.C.: Claims about the use of software engineering practices in science: A systematic literature review. Information and Software Technology **67**, 207–219 (2015). https://doi.org/10.1016/j.infsof.2015.07.011
4. Helbing, D., Bialetti, S.: How to do agent-based simulations in the future: From modeling social mechanisms to emergent phenomena and interactive systems design (Oct. 2013). In: Understanding Complex Systems, chap. Agent-Based Modeling, pp. 25–70. Springer, Berlin/Heidelberg (2012)
5. Johanson, A., Hasselbring, W.: Software engineering for computational science: Past, present, future. Computing in Science & Engineering (2018). https://doi.org/10.1109/mcse.2018.108162940
6. Joppa, L.N., McInerny, G., Harper, R., Salido, L., Takeda, K., O'Hara, K., Gavaghan, D., Emmott, S.: Troubling trends in scientific software use. Science **340**(6134), 814–815 (2013). https://doi.org/10.1126/science.1231535
7. Kanewala, U., Bieman, J.M.: Testing scientific software: A systematic literature review. Information and Software Technology **56**(10), 1219–1232 (2014). https://doi.org/10.1016/j.infsof.2014.05.006
8. Lamprecht, A.L., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Martin Del Pico, E., Dominguez Del Angel, V., van de Sandt, S., Ison, J., Martinez, P.A., McQuilton, P., Valencia, A., Harrow, J., Psomopoulos, F., Gelpi, J.L., Chue Hong, N., Goble, C., Capella-Gutierrez, S.: Towards FAIR principles for research software. Data Science pp. 1–23 (2019). https://doi.org/10.3233/DS-190026
9. McIntosh, S., Kamei, Y., Adams, B., Hassan, A.E.: An empirical study of the impact of modern code review practices on software quality. Empirical Software Engineering **21**(5), 2146–2189 (2015). https://doi.org/10.1007/s10664-015-9381-9
10. Nguyen-Hoan, L., Flint, S., Sankaranarayana, R.: A survey of scientific software development. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '10. ACM Press (2010)
11. Petre, M., Wilson, G.: Code review for and by scientists. `arXiv:1407.5648v2 [cs.SE]` (2014), https://arxiv.org/abs/1407.5648v2
12. Riedel, L., Herdeanu, B., Mack, H., Sevinchan, Y., Weninger, J.: Utopia: A comprehensive and collaborative modeling framework for complex and evolving systems. Journal of Open Source Software (2020), https://joss.theoj.org/papers/8ce6d2bc26c0c6553c5ce5aff38d83c3, under review
13. Wilensky, U.: NetLogo (1999), http://ccl.northwestern.edu/netlogo/

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_32