Software development practices with AceFEM

Matevž Pintar[1], Tomaž Šuštar[1], and Tomaž Rodič[2]

[1] *Centre for Computational Continuum Mechanics (C3M), Tehnoloski park 21, 1000 Ljubljana, Slovenia*
[2] *University of Ljubljana, Faculty of Natural Sciences and Engineering, Askerceva cesta 12, 1000 Ljubljana, Slovenia*

Keywords: software development, functional programming, version control system, unit tests

1 Introduction

Scientists nowadays spend more and more time developing software to conduct their research, but majority of them have little formal education in good software development practices [2]. We believe that community of AceFEM [1] users is no exception to this trend. This comes from our experience as software developers with engineering background, creating finite element analysis software for industrial applications based on AceFEM. Our projects reach tens of thousands lines of code in size and continuous adoption of good software development practices has proved essential to manage their complexity. These practices are a set of tools and protocols that improve software reliability and user-friendliness, reduce the number of bugs and increase developer productivity. They have been nicely summarized in two recent papers by Wilson et. al. [2][3] and here we will present some of them aimed specifically at programming in Wolfram Language (WL). A general explanation will be supported by examples from one of the first open source projects using AceFEM framework [4].

2 Materials and Methods

The most basic method to successfully manage a complex (software) system is to make it as modular as possible. On the smaller scale of WL programming this can be achieved by embracing functional programming style that is native to the language. Functional programming means that every function accepts all inputs as parameters and returns its output without changing the global system state (i.e. values of symbols in the WL kernel). Following this approach consistently, enables us to easily chain functions together and create flexible hierarchical architecture of the program. Another important benefit is that such functions can be easily tested and debugged in isolation, independently from each other. Functions should be kept short and granular because then it is easier to follow the principle that each function does only one task and each task is done by only one function, therefore avoiding code duplication.

Designers of WL are known to put significant effort to choose meaningful names for built-in functions and pay special attention to backward compatibility between versions. We believe that it makes sense to imitate built-in function design and package layout for your own programming. For example, functions should accept somewhere between 0 and 5 arguments that are essential for its operation. Other non-essential values that control function behavior should be given through "named options" mechanism. In general re-implementing functionality (e.g. flow control structures, error reporting, etc.) that already exists in the language should be avoided.

To keep track of changes in the program code and facilitate collaboration it is essential to use version control system (VCS). VCS is a type of software that saves snapshots of files to a repository and that repository can also act as backup on remote location. Since VCS work best with plain-text files we recommend writing WL code directly in package files (.m or .wl) and avoiding metadata rich notebook (.nb) format.

3 Results

We have created a free and open source WL package "HeatTrans" [4] for non-stationary heat transfer simulation, based on AceFEM framework. AceFEM has a role of computational back-end and is more or less hidden from users view. The main purpose of the package is not its technical functionality but rather the demonstration of good practices of package development with AceFEM as described in the previous section.

The project file layout follows documented recommendations for WL applications and all source code, including AceGen code for finite element subroutines, is in plain-text format (.wl). Libraries for different finite element topologies and types are generated in a batch process, which is essential for painless updating of the whole project after some changes in AceGen code. User documentation is included in the package and gets seamlessly integrated in the WL documentation center at installation.

For "HeatTrans" development we have used Git (https://git-scm.com/) version control system and hosted it on GitHub page to facilitate collaboration and accessibility. GitHub page acts as project home page where the latest and previous released versions of the package can be downloaded. They are distributed in the "Paclet" format (.paclet) because this allows simple installation and updating with WL package manager functions. GitHub also provides convenient page to report bugs and project management tools.

Finally the project includes a set of unit tests implemented with WL testing framework (e.g. VerificationTest function) that can be run repeatedly many times during development process. As additional quality assurance measure VCS can be configured so that tests are automatically run at every commit and that commit is aborted if all tests don't pass.

4 Conclusions

We have presented a set of best practices for software development in Wolfram Language, based on a larger set of generic recommendations for scientific computing published by Willson et al. [2], [3]. They facilitate writing code that can be more easily understood and modified by other programmers including the author's future self. This is important since the code should be written for people, not computers. Good software development practices reduce mental overhead associated with keeping in mind all relevant aspects of code. This in turn increases the productivity of developers and reliability of resulting programs. The latter is especially important for reproducibility of scientific results obtained by such software.

References

[1] Korelc, J., Wriggers, P., 2016. Automation of Finite Element Methods. Springer International Publishing Switzerland, ISBN: 978-3-319-39003-1.

[2] Wilson G., Aruliah D.A., Brown C.T., Chue Hong N.P., Davis M., et al., 2014. Best Practices for Scientific Computing. PLoS Biol 12(1): e1001745. doi:10.1371/journal.pbio.1001745. https://doi.org/10.1371/journal.pbio.1001745

[3] Wilson G., Bryan J., Cranston K., Kitzes J., Nederbragt L., Teal T.K., 2017. Good enough practices in scientific computing. PLoS Comput Biol 13(6): e1005510. https://doi.org/10.1371/journal.pcbi.1005510

[4] C3M, 2018. "HeatTrans" - Package for non-stationary heat transfer simulation, https://github.com/c3m-labs/HeatTrans, accesed on: 30.10.2018.