Track 1 Paper: Good Usability Practices in Scientific Software Development

Francisco Queiroz
Tecgraf Institute, PUC-Rio
Department of Arts & Design, PUC-Rio
Rio de Janeiro, Brazil

Raniere Silva
Software Sustainability Institute UK
School of Computer Science
University of Manchester
Manchester, United Kingdom

Jonah Miller
Perimeter Institute of Theoretical Physics
Waterloo, Canada
University of Guelph
Guelph, Canada

Sandor Brockhauser
European XFEL GmbH
Schenefeld, Germany

Hans Fangohr
University of Southampton
United Kingdom
European XFEL GmbH
Schenefeld, Germany

Abstract—**Scientific software often presents very particular requirements regarding usability, which is often completely overlooked in this setting. As computational science has emerged as its own discipline, distinct from theoretical and experimental science, it has put new requirements on future scientific software developments. In this paper, we discuss the background of these problems and introduce nine aspects of good usability. We also highlight best practices for each aspect with an emphasis on applications in computational science.**

Keywords—*Best Practices, Usability, Scientific Software, Computational Science, Software for Science.*

I. INTRODUCTION

Scientific software development is a field of growing importance but lacks a widespread methodology. Scientists generally have little or no training in software engineering but tend to be main developers of computational science codes. They face a number of challenges including: quickly changing requirements due to the research nature of the work, competition between maintainable and performance code, and lack of metrics that would reward investment into sustainable software [1], [2]. Of particular detriment is the pressure to rapidly produce scientific publications [3], [4]. It may be possible to overcome this publication pressure when funding agencies are convinced that it is worth investing directly in software software for computationally intensive fields. The Science and Technology Facilities Council (STFC) in the UK Collaborative Computational Projects (http://www.ccp.ac.uk/about.html) sets a good example.

In this work we focus on *usability*, a particular aspect of software development and design. Usability is one of the attributes of sustainable software and can be defined as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of us" [5, p.3]. Without proper usability, a software cannot be distributed and applied even within its targeted domain. More importantly, its unusable software can easily result in non-reproducible science and the violation of the FAIR principles [6]. Unfortunately, usability is often neglected in scientific software development [7], and is of mixed perceived importance to users and developers [8], [9]. Scientific software usage and development present many challenges for usability design that can be related to development models, user-base needs and specialization, professional practices, technical constraints, and scientific demands [10]. Computational science is therefore an idiosyncratic field with unique and, occasionally, counterintuitive usability requirements. There are, nevertheless, a significant number of informative case studies and guidelines on the subject [7], [11]–[24]. Supported by those references and informed by first-hand experience, we discuss usability challenges and how to address them.

II. GOOD PRACTICES

A. Think Beyond Graphical User Interfaces

Graphical user interfaces (GUIs) have made software user-friendly and arguably fosteed the popularization of software in general. However, scientific software might require alternatives that, if not more intuitive, are more appropriate and efficient depending on the user's needs — especially if they involve entering a large amounts of data, and reading the data from many files, or running on a shared or distributed architectures. Command-Line Interfaces (CLIs) are popular in computational science because they often allow for quick repetition of tasks [25] and scriptability. The analysis of large datasets can be significantly easier and more productive when done through command-line input than through visual-based interfaces [12].[1] Moreover, GUIs can be extremely cumbersome on distributed infrastructures such as supercomputers. To implement a GUI for distributed code, a graphical frontend must connect via network to a distributed backend. Although many scientific visualization tools such as VisIt [26] and Paraview [27], [28] have implemented this scheme, full rendering via GUI can

---

[1]Of course using a CLI does not guarantee ease of use. One must still follow usability best practices when designing the CLI.

be impractical and computationally intensive renders are often performed "headless" without the GUI [26]. Because of these difficulties, most distributed scientific codes completely lack a graphical frontend or separate computation and visualization into separate and subsequent stages in the workflow. Similarly, complex experimental protocols combining data analysis and scientific instrumentation control can be designed with separate User Interaction points in the complete workflow [29].

Even for software where daily use relies on a GUI, such as text processors and web browsers, there are times when having a CLI for some tasks is a time saver. For example, users of LaTeX, or the LibreOffice or Chrome CLIs can convert a text document into PDF format from the command line.

B. Keep UI Code Separate From Scientific Calculation

Simulation (or any scientific calculation) should not be embedded in User Interface (UI) code [30]. This rule is particularly true for scientific software primarily because, as previously stated, scientific software should be usable via a number of alternative interfaces, such as GUI and CLI. Moreover, it should be possible to access these interfaces both locally and over a network (e.g., via ssh). Keeping the scientific calculation code wrapped into functions that are called by the UI should make reconfiguration and customization more convenient [31], make porting the functionalities to another UI easier and make integration with other software simpler.

C. Keep the Configuration in a File

Some tasks requires researchers to provide a long list of parameters to define their computational problem, and the software they are using may not provide default values for the parameters, or the default parameters need to be overriden. In these situations, it is very handy to have the ability to store some or all of the parameters in a configuration file that the software can read at the begining of every execution. Alternatively, if a command line tool asks for input parameters from the standard input, which requires continuous user interaction, it can be modified to be scriptable. In a script, the configuration parameters are stored next to the execution command itself [32].

Configuration files have the advantage of being *declarative* and *automatically verifiable*. The file defines a *state* which the program will start from or try to achieve, rather than a procedure which leads to that state. Moreover, a parser can automatically check to see if state is valid. The former is good for reproducibility because it is (ideally) unambiguous even years later [33]. The latter is good for accuracy because the code can check if the parameters in the file are sensible [23]. This state-based approach can also make parallelism easier to automatically reason about [34] and some parallel runtime environments have made use of this property [35], [36].

Domain Specific Languages (DSLs), on the other hand, provide additional flexibility not present in a plain configuration file. They allow the user to *programmatically* define new behaviour for the code. This can be a major advantage since it often enables the code to be extended to unforeseen use-cases without major rewrites. DSLs can also allow the user to interact with the code at runtime, which can be helpful for debugging, prototyping, and visualization [23]. The syntax and rules of the DSL can also provide the same error checking as a parameter file.

While defining a new domain specific language (DSL) requires the development of a parser for the language, this extra effort can be avoided by *embedding* the domain specific language in an existing general purposes language. This has been demonstrated by a number of projects recently, and Python is a common choice as the general purpose language. In this context, the domain specific language is given through a Python module that the user imports into their generic Python program, and which provides commands, objects and operations that are specific to the domain in question. The Python program then becomes the (very flexible) configuration file for the computational problem.

However, some care is required when designing such DSLs: the elements of the DSL must be constructed so that users cannot combine them in ways that would take the tool outside its range of applicability. This could be achieved through explicit assert statements in the DSL's implementation or appropriate (often Object Oriented) design. If the code author lacks the experience or time available to achieve this, it is important to document the assumptions made for use of the DSL so that it is not used incorrectly inadvertently by others in the future. For example, the yt project [37] is a DSL for scientific visualization and data analysis built in Python. If the data fed into yt doesn't satisfy the correct assumptions, yt could produce spurious visualization artifacts or incorrectly integrate a quantity over the domain. Therefore the authors of yt take extreme care to document their API, sanitize their inputs and throw informative error messages when incorrect data is fed into the tool. A major part of this process is unit testing the DSL's functionality

Some codes combine both plain configuration files *and* DSLs. For example, the Einstein Toolkit [38],[2] a code for relativistic astrophysics, uses configuration files for day-to-day simulations. However, it also provides a low-level DSL, Kranc [39], for defining systems of equations to solve.

D. Design for Small, Incremental Changes

Making incremental changes is considered a best practice for scientific software development [25], and the same principle applies to user interfaces. Ideally, *UIs should be planned for extensibility and frequent changes* as new requisites emerge. Through incremental changes, software is more likely to *stay attuned to users' needs*, not forcing them to radically change the way they work [19].

Regarding constant updates and addition of new functionalities, UI components that can be easily extended might offer interesting solutions. This is the case for map3D, a scientific visualization software for displaying and editing three-dimensional models and associated data [40]. During development, pop-up menus were implemented for providing the necessary flexibility, allowing developers to add new commands and submenus as the software development and requisites evolved [11]. It is worth mentioning that web-based applications might take advantage of the modularity allowed by frontend design methodologies such as Atomic Design [41],

---

[2]For which one of us is a developer.

making it easier to configure user interfaces as the project advances.

The parameter files and DSLs described in section II-C are particularly good for satisfying this design constraint. For example, the Einstein Toolkit [38] packages low-level code in modules. Each module must declare which functionality it adds, which relevant parameters can be set in the parameter file, and how these parameters depend on other modules. The parameter file parser then automatically adds these options to the parameter file at compile time. The yt project [37] provides a DSL for scientific visualization. This DSL interacts with the low-level code only through function calls and so functionality can easily be incrementally added by the introduction of new DSL language features or functions.

E. Facilitate and Register User Activity and Environment

There are a number of ways through which usability can be enhanced based on *past* user activity. First, providing access to a list of recent commands and allowing users to re-execute them can help users save time. This is a major reason for the popularity of command-line interfaces [25]. A very popular implementation of this concept is the ability to access previously typed commands by pressing the up arrow key or do a reverse search on the history of executed commands. Users can also press the right and left arrow keys to navigate through a previous command and edit it to suit their needs. Second, it might be a good idea to give users quick access to frequently used commands [42]. In some environments, the tab key is used to roll among frequent used commands or to auto complete a command. Third, logging user activity might help users identify and support research reproducibility [24] by exporting the history to a file.

After registering user activity, developers can go further and log the user environment, i.e. compiled binary, configuration files, input files and output files, used when running the program. This is useful in scientific software since the output of any experiment can be different because of different implementations (or compiler optimizations) of the Basic Linear Algebra Subprograms (BLAS), LAPACK (Linear Algebra Package) or any other library used when performing the experiment. This automatic logging not only provides users quick access to their exact configuration for debugging purposes but also allows the computation to be reproduced years after it was run for the first time. One example framework is Formaline [43]. For software developers working with Python, we mention the related packages ReciPy [44] and Sumatra [45].

F. Learn About How Users Work

Guidelines and case studies often recommend the adoption of a user-centered design process that seeks to develop a firm understanding of how scientists do their work before developing a piece of software. This understanding can be acquired by learning the meanderings of scientific work [12], or through a participatory design approach in which users are actively involved in the design process [16], [46]–[48]. It is also important to *analyze the scientific work within the environment where it actually takes place* [13] and evaluate existing tools which are already in use [14]. In this last case it might be advantageous to adopt preexisting industry standards (e.g.: keyboard shortcuts for common functionalities, iconography, etc.).

When designing user interfaces for scientific software, it is a good idea to *address specific users or user-bases* rather than aim for a general solution [14], [19]. Ideally, GUIs should be open to user customization and adjustable to personal preferences and professional specialization [14], [49]. However, users should not be overwhelmed by an excessive number of customizable parameters — some of which can be unimportant or meaningless to their specific case. Instead, there should be an additional section for setting advanced parameters [24].

As a user base grows, users may have suggestions for improving the UI or the underlying scientific code. If the code is open source, it can be extremely advantageous to transform these *users* into *developers* so that they can bring their user experience and domain expertise to bear [50]. Additionally, it is advisable that scientific domain experts are brought into the design process for informing *domain best practices* [15], [21] and evaluating the tool [20].

G. Be Minimalistic, but Look Out for Exceptional Needs

Designers should be attentive to information that is particularly relevant in scientific software, but that could be eluded otherwise. *Metadata*, for instance, is often required to be readable and easy to access [17], [18], [20], [25], [47], [51].

Also, despite recent trends favoring flat design over skeuomorphism (i.e.: visual design that imitates the appearance of real-world objects), software versions of physical instruments might benefit from adopting the looks of their real-world counterparts [52], making it easier for users to recognize and learn about their functioning. An example for that approach is LabViEW's set of GUI components mimicking dials, knobs and meters [53].

Finally, minimalism should *emphasize, rather than conceal, critical information* such as system malfunctioning [54], emergency information [55], and situations where awareness and response under time pressure are essential. For instance, the Sky software for astronomical visualization reduces users' cognitive load by simplifying three-dimensional visualization data as a two-dimensional projection [46].

H. Design for Precision

In order to achieve satisfying results, scientific work often demands precision regarding user's input. A possible means for that would be continuously constraining users' input and providing feedback on it. The Dynamic Dragging Interface, for instance, makes user of force-feedback input devices to help users selecting sections of 3D brain visualization [20]. Another solution would be not accepting the input when the input device, such as a stylus or mouse, moves too fast.

It can be advantageous to have two input modes for the same action — one designed for accuracy, and another for speed. In map3D, for instance, geometric models could be moved, rotated and scaled through dial boxes (accurate, but slower) or, alternatively, via mouse (less accurate, but faster) [11]. Another approach would be to give users a way to switch between fast and precise working modes. For instance,

by activating a 'snap' mode where the mouse cursor snaps to a gridline, objects or other elements on screen.

I. Contextualize User Actions

Work in scientific software can involve a number of different and/or sequential tasks to be performed by users. In those cases, it is desirable to contextualize users' actions, facilitating their access to functions that are relevant to their current tasks and preventing their access to functions that are not.

The Petri Net Toolbox for MATLAB, for instance, features a button that toggles between *Draw Mode*, for creating and editing models, and *Explore Mode*, for simulation and analysis [42]. When switching between modes, GUI elements are displayed or hidden depending on their relevance to the selected mode. This approach is known as a design pattern named *Disabled Irrelevant Things* [56].

Another possible approach is the *Window Per Task* [56] design pattern, in which tasks are distributed across individual screens containing the appropriate commands for that task only. Pharmaceutical biology software Lipid-Pro makes use of this design pattern by organizing its tasks into separate panels [7].

III. Summary

Throughout the previous section, we have presented a nonexhaustive set of good practices in usability for scientific software, taking in consideration challenging aspects of scientific software development and use such as the lack of attention to software engineering; the need for reproducibility; the handling of large amounts of data; the complexity of actions and parameters involved in scientific work; frequent changes in requirements; particularities of scientific work and its environment; the need for accessing and responding to critical information; and the importance of precision.

By adopting the presented practices, developers should be able to deliver applications that are more usable, robust and more appropriate for scientific work.

Acknowledgment

References

[1] J. Segal, "Some problems of professional end user developers," in *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, ser. VLHCC '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 111–118. [Online]. Available: http://dx.doi.org/10.1109/VLHCC.2007.17

[2] D. F. Kelly, "A software chasm: Software engineering and scientific computing," *IEEE Softw.*, vol. 24, no. 6, pp. 120–119, Nov. 2007. [Online]. Available: http://dx.doi.org/10.1109/MS.2007.155

[3] G. Wilson, "Where's the Real Bottleneck in Scientific Computing?" *American Scientist*, vol. 94, no. 1, pp. 5+, 2006. [Online]. Available: http://dx.doi.org/10.1511/2006.1.5

[4] S. Killcoyne and J. Boyle, "Managing chaos: Lessons learned developing software in the life sciences," *Computing in Science Engineering*, vol. 11, no. 6, pp. 20–29, Nov 2009. [Online]. Available: http://dx.doi.org/10.1109/MCSE.2009.198

[5] C. Venters, L. Lau, M. K. Griffiths, V. Holmes, R. R. Ward, and J. Xu, "The blind men and the elephant: Towards a software sustainability architectural evaluation framework," figshare, Tech. Rep. 790758, 2013, http://dx.doi.org/10.6084/m9.figshare.790758.

[6] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. G. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. C. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific Data*, vol. 3, pp. 160 018+, Mar. 2016. [Online]. Available: http://dx.doi.org/10.1038/sdata.2016.18

[7] Z. Ahmed, S. Zeeshan, and T. Dandekar, "Developing sustainable software solutions for bioinformatics by the butterfly paradigm [version 2; referees: 2 approved]," *F1000Research*, vol. 3, no. 71, 2014. [Online]. Available: http://dx.doi.org/10.12688/f1000research.3681.2

[8] L. Nguyen-Hoan, S. Flint, and R. Sankaranarayana, "A survey of scientific software development," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10. New York, NY, USA: ACM, 2010, pp. 12:1–12:10. [Online]. Available: http://doi.acm.org/10.1145/1852786.1852802

[9] M. Hucka and M. J. Graham, "Software search is not a science, even among scientists," *CoRR*, vol. abs/1605.02265, 2016. [Online]. Available: http://arxiv.org/abs/1605.02265

[10] F. Queiroz and R. Spitz, "The lens of the lab: Design challenges in scientific software," *International Journal of Design Management and Professional Practice*, vol. 10, no. 3, pp. 17–45, 2016. [Online]. Available: https://doi.org/10.18848/2325-162X/CGP/v10i03/17-45

[11] R. S. MacLeod, C. R. Johnson, and M. A. Matheson, "Visualization of cardiac bioelectricity-a case study," in *Visualization, 1992. Visualization '92, Proceedings., IEEE Conference on*, Oct 1992, pp. 411–418. [Online]. Available: http://dx.doi.org/10.1109/VISUAL.1992.235178

[12] R. R. Springmeyer, "Applying observations of work activity in designing prototype data analysis tools," in *Visualization, 1993. Visualization '93, Proceedings., IEEE Conference on*, Oct 1993, pp. 228–235. [Online]. Available: https://doi.org/10.1109/VISUAL.1993.398873

[13] C. Pancake, "Improving the usability of numerical software through user-centered design," Corvallis, OR, USA, Tech. Rep., 1996. [Online]. Available: https://web.engr.oregonstate.edu/~pancake/papers/ImpUsab.pdf

[14] H. Javahery, A. Seffah, and T. Radhakrishnan, "Beyond power: Making bioinformatics tools user-centered," *Commun. ACM*, vol. 47, no. 11, pp. 58–63, Nov. 2004. [Online]. Available: http://doi.acm.org/10.1145/1029496.1029527

[15] m. c. schraefel, G. V. Hughes, H. R. Mills, G. Smith, T. R. Payne, and J. Frey, "Breaking the book: Translating the chemistry lab book into a pervasive computing lab environment," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '04.

New York, NY, USA: ACM, 2004, pp. 25–32. [Online]. Available: http://doi.acm.org/10.1145/985692.985696

[16] C. Letondal and W. E. Mackay, "Participatory programming and the scope of mutual responsibility: Balancing scientific, design and software commitment," in *Proceedings of the Eighth Conference on Participatory Design: Artful Integration: Interweaving Media, Materials and Practices - Volume 1*, ser. PDC 04. New York, NY, USA: ACM, 2004, pp. 31–41. [Online]. Available: http://dx.doi.org/10.1145/1011870.1011875

[17] T. Talbott, M. Peterson, J. Schwidder, and J. D. Myers, "Adapting the electronic laboratory notebook for the semantic era," in *Proceedings of the 2005 International Symposium on Collaborative Technologies and Systems, 2005.*, May 2005, pp. 136–143. [Online]. Available: http://dx.doi.org/10.1109/ISCST.2005.1553305

[18] C. Macaulay, D. Sloan, X. Jiang, P. Forbes, S. Loynton, J. R. Swedlow, and P. Gregor, "Usability and user-centered design in scientific software development," *IEEE Softw.*, vol. 26, no. 1, pp. 96–102, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1109/MS.2009.27

[19] D. D. Roure and C. Goble, "Software design for empowering scientists," *IEEE Software*, vol. 26, no. 1, pp. 88–95, Jan 2009. [Online]. Available: http://dx.doi.org/10.1109/MS.2009.22

[20] D. F. Keefe, "Integrating visualization and interaction research to improve scientific workflows," *IEEE Computer Graphics and Applications*, vol. 30, no. 2, pp. 8–13, March 2010. [Online]. Available: http://doi.org/10.1109/MCG.2010.30

[21] P. De Matos, J. A. Cham, H. Cao, R. Alcantara, F. Rowland, R. Lopez, and C. Steinbeck, "The Enzyme Portal: a case study in applying user-centred design methods in bioinformatics," *BMC Bioinformatics*, vol. 14, p. 103, 2013. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-14-103

[22] H. Fangohr, M. Albert, and M. Franchin, "Nmag micromagnetic simulation tool: Software engineering lessons learned," in *Proceedings of the International Workshop on Software Engineering for Science*, ser. SE4Science '16. New York, NY, USA: ACM, 2016, pp. 1–7. [Online]. Available: http://doi.acm.org/10.1145/2897676.2897677

[23] M. Beg, R. A. Pepper, and H. Fangohr, "User interfaces for computational science: a domain specific language for oommf embedded in python," *AIP Advances*, vol. 7, p. 056025, 2017. [Online]. Available: http://doi.org/10.1063/1.4977225

[24] M. List, P. Ebert, and F. Albrecht, "Ten simple rules for developing usable software in computational biology," *PLOS Computational Biology*, vol. 13, no. 1, pp. 1–5, 01 2017. [Online]. Available: http://dx.doi.org/10.1371%2Fjournal.pcbi.1005265

[25] G. Wilson, D. A. Aruliah, C. Titus Brown, N. P. Chue Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson, "Best practices for scientific computing," *PLoS Biol*, vol. 12, no. 1, Jan. 2014. [Online]. Available: https://doi.org/10.1371/journal.pbio.1001745

[26] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, and P. Navrátil, "VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data," in *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, Oct 2012, pp. 357–372.

[27] J. Ahrens, B. Geveci, and C. Law, "Paraview: An end-user tool for large data visualization," *The Visualization Handbook*, vol. 717, 2005. [Online]. Available: https://doi.org/10.1016/B978-012387582-2/50038-1

[28] U. Ayachit, *The Paraview Guide (Full Color Version): A Parallel Visualization Application*. Kitware, Incorporated, 2015. [Online]. Available: https://books.google.com/books?id=Z8wHrgEACAAJ

[29] S. Brockhauser, O. Svensson, M. W. Bowler, M. Nanao, E. Gordon, R. M. F. Leal, A. Popov, M. Gerring, A. A. McCarthy, and A. Gotz, "The use of workflows in the design and implementation of complex experiments in macromolecular crystallography," *Acta Crystallographica Section D*, vol. 68, no. 8, pp. 975–984, Aug 2012. [Online]. Available: https://doi.org/10.1107/S090744491201863X

[30] D. Kelly, D. Hook, and R. Sanders, "Five recommended practices for computational scientists who write software," *Computing in Science Engineering*, vol. 11, no. 5, pp. 48–53, Sept 2009. [Online]. Available: https://doi.org/10.1109/MCSE.2009.139

[31] B. F. Bastos, V. M. Moreira, and A. T. A. Gomes, "Rapid prototyping of science gateways in the brazilian national hpc network." in *IWSG*, ser. CEUR Workshop Proceedings, T. Kiss, Ed., vol. 993. CEUR-WS.org, 2013. [Online]. Available: http://dblp.uni-trier.de/db/conf/iwsg/iwsg2013.html#BastosMG13

[32] E. Potterton, P. Briggs, M. Turkenburg, and E. Dodson, "A graphical user interface to the *CCP*4 program suite," *Acta Crystallographica Section D*, vol. 59, no. 7, pp. 1131–1137, Jul 2003. [Online]. Available: https://doi.org/10.1107/S0907444903008126

[33] J. W. Lloyd, "Practical advtanages of declarative programming," in *1994 Joint Conference on Declarative Programming, GULP-PRODE'94 Peñiscola, Spain, September 19-22, 1994, Volume 1*, 1994, pp. 18–30.

[34] M. M. Chakravarty, "On the massively parallel execution of declarative programs," Ph.D. dissertation, Berlin Institute of Technology, 1997. [Online]. Available: http://www.cse.unsw.edu.au/~chak/papers/diss.ps.gz

[35] A. Buss, Harshvardhan, I. Papadopoulos, O. Pearce, T. Smith, G. Tanase, N. Thomas, X. Xu, M. Bianco, N. M. Amato, and L. Rauchwerger, "Stapl: Standard template adaptive parallel library," in *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, ser. SYSTOR '10. New York, NY, USA: ACM, 2010, pp. 14:1–14:10. [Online]. Available: http://doi.acm.org/10.1145/1815695.1815713

[36] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken, "Legion: Expressing locality and independence with logical regions," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 66:1–66:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389086

[37] M. J. Turk, B. D. Smith, J. S. Oishi, S. Skory, S. W. Skillman, T. Abel, and M. L. Norman, "yt: A Multi-code Analysis Toolkit for Astrophysical Simulation Data," *ApJS*, vol. 192, p. 9, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1088/0067-0049/192/1/9

[38] F. Löffler, J. Faber, E. Bentivegna, T. Bode, P. Diener, R. Haas, I. Hinder, B. C. Mundim, C. D. Ott, E. Schnetter, G. Allen, M. Campanelli, and P. Laguna, "The Einstein Toolkit: A Community Computational Infrastructure for Relativistic Astrophysics," *Class. Quantum Grav.*, vol. 29, no. 11, p. 115001, 2012. [Online]. Available: http://dx.doi.org/10.1088/0264-9381/29/11/115001

[39] S. Husa, I. Hinder, and C. Lechner, "Kranc: a Mathematica application to generate numerical codes for tensorial evolution equations," *Comput. Phys. Commun.*, vol. 174, pp. 983–1004, 2006. [Online]. Available: http://dx.doi.org/10.1016/j.cpc.2006.02.002

[40] CIBC, 2016, map3d: Interactive scientific visualization tool for bio-engineering data. Scientific Computing and Imaging Institute (SCI), Download from: http://www.sci.utah.edu/cibc/software.html.

[41] B. Frost, "Designing systems — atomic design by brad frost," 2016, (Accessed on 08/17/2017). [Online]. Available: http://atomicdesign.bradfrost.com/chapter-1/

[42] J. Júlvez, M. H. Matcovschi, and O. Pastravanu, "Matlab tools for the analysis of petri net models," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, Sept 2014, pp. 1–12. [Online]. Available: https://doi.org/10.1109/ETFA.2014.7005053

[43] C. Schnetter, E. Ott, "Formaline: The provenance of computational astrophysics simulations and their results (part i)," http://blowingupstars.blogspot.ca/2016/06/formaline-provenance-of-computational.html, 2016, online Access. http://blowingupstars.blogspot.ca/2016/06/formaline-provenance-of-computational.html. Accessed Julu 5, 2017.

[44] e. a. Robin Wilson, "Recipy," https://github.com/recipy/recipy, 2015, accessed: 2017-07-04.

[45] A. P. Davison, M. Mattioni, D. Samarkanov, and B. Teleńczuk, "Sumatra: A Toolkit for Reproducible Research," in *Implementing Reproducible Research*, V. Stodden, F. Leisch, and R. D. Peng, Eds. Chapman & Hall, 2014, ch. 3, pp. 57–78, https://pypi.python.org/pypi/Sumatra.

[46] C. R. Aragon, S. S. Poon, G. S. Aldering, R. C. Thomas, and R. Quimby, "Using visual analytics to maintain situation awareness in astrophysics," in *2008 IEEE Symposium on Visual Analytics Science and Technology*, Oct 2008, pp. 27–34. [Online]. Available: https://doi.org/10.1109/VAST.2008.4677353

[47] A. K. Thomer, M. B. Twidale, J. Guo, and M. J. Yoder, "Co-designing scientific software: Hackathons for participatory interface design," in *Proceedings of the 2016 CHI Conference Extended Abstracts on*

Human Factors in Computing Systems, ser. CHI EA '16. New York, NY, USA: ACM, 2016, pp. 3219–3226. [Online]. Available: http://doi.acm.org/10.1145/2851581.2892549

[48] D. R. Luna, D. A. R. Lede, C. M. Otero, M. R. Risk, and F. G. B. de Quirós, "User-centered design improves the usability of drug-drug interaction alerts: Experimental comparison of interfaces," *Journal of Biomedical Informatics*, vol. 66, pp. 204 – 213, 2017. [Online]. Available: https://doi.org/10.1016/j.jbi.2017.01.009

[49] M. W. Gertz, D. B. Stewart, and P. K. Khosla, "A human machine interface for distributed virtual laboratories," *IEEE Robotics Automation Magazine*, vol. 1, no. 4, pp. 5–13, Dec 1994. [Online]. Available: https://doi.org/10.1109/100.388265

[50] M. J. Turk, "Scaling a code in the human dimension," in *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, ser. XSEDE '13. New York, NY, USA: ACM, 2013, pp. 69:1–69:7. [Online]. Available: http://doi.acm.org/10.1145/2484762.2484782

[51] S. M. Baxter, S. W. Day, J. S. Fetrow, and S. J. Reisinger, "Scientific software development is not an oxymoron," *PLoS Comput Biol*, vol. 2, no. 9, pp. 1–4, 09 2006. [Online]. Available: https://dx.doi.org/10.1371%2Fjournal.pcbi.0020087

[52] K. R. Foster, "Software tools [technology 1998 analysis and forecast]," *IEEE Spectrum*, vol. 35, no. 1, pp. 52–56, Jan 1998. [Online]. Available: https://doi.org/10.1109/6.645980

[53] "Ni pxi and labview deliver unrivaled performance, flexibility, and value for automated test," 2013. [Online]. Available: http://www.ni.com/white-paper/14300/en/

[54] H. Morais, P. Vancraeyveld, A. H. B. Pedersen, M. Lind, H. Jóhannsson, and J. Østergaard, "Sospo-sp: Secure operation of sustainable power systems simulation platform for real-time system state evaluation and control," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2318–2329, Nov 2014. [Online]. Available: https://doi.org/10.1109/TII.2014.2321521

[55] H. T. Ferguson, S. Gesing, and J. Nabrzyski, "Measuring usability in decision tools supporting collaborations for environmental disaster response," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Jan 2016, pp. 2872–2881. [Online]. Available: https://doi.org/10.1109/HICSS.2016.360

[56] Z. Ahmed, "Designing flexible GUI to increase the acceptance rate of product data management systems in industry," *CoRR*, vol. abs/1103.1134, 2011. [Online]. Available: http://arxiv.org/abs/1103.1134

[57] D. S. Katz, K. E. Niemeyer, S. Gesing, L. Hwang, W. Bangerth, S. Hettrick, R. Idaszak, J. Salac, N. Chue Hong, S. Núñez Corrales, A. Allen, R. S. Geiger, J. Miller, E. Chen, A. Dubey, and P. Lago, "Report on the Fourth Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE4)," *ArXiv e-prints*, May 2017. [Online]. Available: https://arxiv.org/abs/1705.02607v2