Methodology for Assessing the State of the Practice for Domain X

Spencer Smith
McMaster University, Canada
smiths@mcmaster.ca

Jacques Carette
McMaster University, Canada
carette@mcmaster.ca

Olu Owojaiye
McMaster University, Canada
owojaiyo@mcmaster.ca

Peter Michalski
McMaster University, Canada
michap@mcmaster.ca

Ao Dong
McMaster University, Canada
donga9@mcmaster.ca

Abstract

...

2012 ACM Subject Classification Software and its engineering; Software and its engineering → Software product lines; General and reference → Empirical studies

Keywords and phrases software quality, domain analysis, scientific computing software, research software, empirical measures, analytic hierarchy process

Contents

1 Introduction

Research software uses computing to simulate mathematical models of real world systems so that we can better understand and predict those systems's behaviour. A small set of examples of important research software includes the following: designing new automotive parts, analyzing the flow of blood in the body, and determining the concentration of a pollutant released into the groundwater. As these examples illustrate, research software can be used for tackling important problems that impact such areas as manufacturing, financial planning, environmental policy, and the health, welfare and safety of communities.

Given the importance of research software, scientists and engineers are pushing for methods and tools to sustainably develop hiqh quality software. This is evident from the existence of such groups as the Software Sustainability Institute (SEI) and Better Scientific Software (BSS). Sustainability promoting groups such as these are necessary because unfortunately the current state-of-the-practice for research software often does not incorporate state-of-the-art Software Engineering (SE) tools and methods (Johanson and Hasselbring, 2018). The lack of SE tools and methods contributes to sustainability and reliability problems (Faulk et al., 2009). Problems with the current state of the practice are evident from embarrassing failures, like a retraction of derived molecular protein structures (Miller, 2006), false reproduction of sonoluminescent fusion (Post and Votta, 2005), and fixing and then reintroducing the same error in a large code base three times in 20 year (Milewicz and Raybourn, 2018). To improve this situation, we need to first fully understand the current state of the practice for research software.

The purpose of our proposed methodology is to understand how software quality is impacted by the software development principles, processes and tools currently used within research software communities. Since research software is so broad a category, we will reduce the scope of our methodology to focus on one specific research software domain at a time. This "state of the practice for domain X" exercise will build off of prior work on measuring/assessing the state of software development practice in several research domains. We will update the work that was done previously for domains such as Geographic Information Systems (Smith et al., 2018a), Mesh Generators (Smith et al., 2016a), Seismology software (Smith et al., 2018c), and Statistical software for psychology (Smith et al., 2018b).

In the previous "state of the practice" project, we measured 30 software projects for each domain, but the measures were relatively shallow. With this re-boot we will still target about 30 software examples from each domain, but we will be collecting more data by adding more measures and experiments. In keeping with the previous project, we still have the constraint that the work load for applying the methodology to a given domain needs to be feasible for a team as small as one individual, and for a time that is short, ideally 3 to 4 months of part-time work per domain.

To start with, the previous set of questions were critically assessed and modified. In addition, the following data is part of the new methodology:

- Characterization of the functionality provided by the software in the domain via a commonality analysis.
- Usability testing for a sample of software in the domain.
- Empirical software engineering related data, such as the number of files, number of lines of code, percentage of issues that are closed, etc.

Unlike the previous measurement process, the new methodology will involves and engages a domain expert partner throughout. We did not previously engage the domain expert with

the rationale that we wished to exclude potential bias. However, this advantage is not worth not being able to evaluate the functionality/usability of the software. Moreover, not having an expert makes publication more difficult, since there is no one to provide advice on how best to approach journals and publishers.

In the proposed methodology, the collected data will be combined to rank the software within each domain using the Analytic Hierarchy Process (AHP) (Saaty, 1980). As in the past process, we will use AHP to develop a list of software ranked by quality. However, in the new process we will not stop with this list. The domain expert will be consulted to verify the ordering, and to discuss the decisions that led to the ranking. The AHP process will be used to facilitate a conversation with the domain experts as a means to deepen our understanding of the software in the domain, and the needs of typical developers.

GitHub will be used to coordinate the work of the team of people that will be involved in this project. In addition to the project record left on GitHub, the final data will be exported to Mendeley (as was done for the previous quality measurement exercise, for instance the grades for GIS software are summarized at https://data.mendeley.com/datasets/6kprpvv7r7/1.)

The scope of this methodology includes observations on product, artifact and process quality for research softwares. We leave the assessment of research software using performance benchmarks to other projects, such as the work of Kågström et al. (1998).

Note the following formatting conventions of this document. Red text denotes a link to sections of this document. Cyan text denotes a URL link. Blue text denotes a citation.

2 Research Questions

The following are the research questions that we wish to answer for each of our selected domains. In addition to answering the questions for each domain, we also wish to combine the data from multiple domains to answer these questions for research software in general.

1. What artifacts are present in current software packages?
2. What tools (development, dependencies, project management) are used by current software packages?
3. What principles, processes, and methodologies are used in the development of current software packages?
4. What are the pain points for developers working on research software projects? What aspects of the existing processes, methodologies and tools do they consider as potentially needing improvement? How should processes, methodologies and tools be changed to improve software development and software quality?
5. For research software developers, what specific actions are taken to address the following:
   a. usability
   b. traceability
   c. modifiability
   d. maintainability
   e. correctness
   f. understandability
   g. unambiguity
   h. reproducibility
   i. visibility/transparency
6. How does software designated as high quality by this methodology compare with top rated software by the community?

3 Overview of Steps in Assessing Quality of the Domain Software

To answer the above research questions (Section 2), we systematically measure the quality of the software through data collection and a series of experiments. An overview of the measurement process is given in the following steps, starting from determining a domain that is suitable for measurement:

1. Identify the domain. (Section 4)
2. *Domain Experts*: Create a top ten list of software packages in the domain. (Meeting Agenda with Domain Experts)
3. Brief the Domain Experts on the overall objective, research proposal, research questions, measurement template, survey for short list projects, usability tests, maintainability experiments. (Meeting Agenda with Domain Experts)
4. Identify broad list of candidate software packages in the domain. (Section 5)
5. Preliminary filter of software packages list. (Section 6)
6. *Domain Experts*: Review domain software list. (Meeting Agenda with Domain Experts)
7. Domain Analysis. (Section 7)
8. *Domain Experts*: Vet domain analysis. (Meeting Agenda with Domain Experts)
9. Gather source code and documentation for each prospective software package.
10. Collect empirical measures. (Section 8)
11. Measure using measurement template. (Section 9)
12. Survey developers (Questions to Developers)
13. Use AHP process to rank the software packages. (Section 10)
14. Identify a short list of top software packages, typically four to six, for deeper exploration according to the AHP rankings of the measurements.
15. *Domain Experts*: Vet AHP ranking and short list. (Meeting Agenda with Domain Experts)
16. With short list:
    a. Usability experiments (Experiments)
    b. Modifiability experiments (Experiments)
17. Rank short list. (Section 11) [To be completed - add pairwise comparison tool —PM]
18. Document answers for research questions.

4 How to Identify the Domain

A domain of research software must be identified. Research software is defined in this exercise as "software that is used to generate, process or analyze results that [are intended] to appear in a publication" (Hettrick et al., 2014) in a scientific or engineering context. Research software is a more general term that covers what is often called scientific computing. To be applicable for the methodology described in this document, the chosen domain must have the following properties:

1. The domain must have well-defined and stable theoretical underpinning. A good sign of this is the existence of standard textbooks, preferably understandable by an upper year undergraduate student.
2. There must be a community of people studying the domain.
3. The software packages must have open source options.
4. A preliminary search, or discussion with experts, suggests that there will be numerous, at least 15, candidate software packages in the domain that qualify as 'research software'.

Some examples of domains that fit these criteria are finite element analysis (Szabo and Actis, 1996), quantum chemistry (Veryazov et al., 2004), seismology (Smith et al., 2018c), as well as mesh generators (Smith et al., 2016b).

5 How to Identify Candidate Software

Once the domain of interest is identified, the candidate software for measuring can be found through search engine queries targeting authoritative lists of software. Potential places to search include GitHub, swMATH and domain related publications, such as review articles. Domain experts are also asked for their suggestions and are asked to review the initial draft of the software list.

When forming the list and reviewing the candidate software the following properties should be considered:

1. The software functionality must fall within the identified domain.
2. The source code must be viewable.
3. The empirical measures listed in Section 8 should ideally be available, which implies a preference for GitHub-style repositories.
4. The software cannot be marked as incomplete or in an initial development phase.

6 How to Initially Filter the Software List

If the list of software is too long (over around 30 packages), then steps need to be taken to create a more manageable list. To reduce the length of the list, the following filters are applied. The filters are applied in the priority order listed, with the filtering process stopped once the list size is manageable.

1. Scope: Software is removed by narrowing what functionality is considered to be within the scope of the domain.
2. Usage: Software packages are eliminated if their installation procedure is not clear and easy to follow.
3. Age: The older software packages (age being measured by the last date when a change was made) are eliminated, except in the cases where an older software package appears to be highly recommended and currently in use.

Copies of both the initial and filtered lists should be kept for traceability purposes.

7 Domain Analysis

Since each domain we will study will have a reasonably small scope, we will be able to view the software as constituting a program family. The concept of a program family is defined by Parnas (1976) as "a set of programs whose common properties are so extensive that it is advantageous to study the common properties of the programs before analyzing individual members". Studying the common properties within a family of related programs is termed a domain analysis.

The domain analysis consists of a commonality analysis of the family of software packages. Its purpose is to show the relationships between these packages, and to facilitate an understanding of the informal specification and development of them. Weiss (1997) defines commonality analysis as an approach to defining a family by identifying commonalities,

variabilities, and common terminology for the family. Commonalities are goals, theories, models, definitions and assumptions that are common between family members. Variabilities are goals, theories, models, definitions and assumptions that differ between family members. Associated with each variability are its parameters of variation, which summarize the possible values for that variability, along with their potential binding time. The binding time is when the value of the variability is set. It could be set as specification time, build time (when the program is being compiled) or run time (when the code is executing).

The final result of the domain analysis will be tables of commonalities, variabilities, and parameters of variation of a program family. Smith et al. (2008) present a template for conducting a commonality analysis, which was referred to when conducting this work. Weiss (1998) describes another commonality analysis technique for deciding the members of a program family which readers are encouraged to be familiar with. Smith and Chen (2004) and Smith et al. (2017) are examples of a commonality analysis for a family of mesh generating software and a family of material models, respectively. The steps to produce a commonality analysis are:

1. Write an Introduction
2. Write the Overview of Domain
3. List Commonalities
4. List Variabilities
5. List Parameters of Variation
6. Add Terminology, Definitions, Acronyms

A sample commonality analysis for Lattice Boltzmann Solvers can be found here.

8 Empirical Measures

Some quality measurements rely on the gathering of raw and processed empirical data. We focus on data that is reasonably easy to collect, which we combine and analyze for this assessment. The measures that are collected relate to the research questions (Section 2). For instance, we collect some of the data to see how large a project is. Other measures intend to ascertain a project's popularity, or how active the project is.

Section 8.1 orients the reader on what raw data is collected. Some of this data can be observed from GitHub repository metrics. The rest can be collected using freeware tools. GitStats is used to measure the number of binary files as well as the number of added and deleted lines in a repository. This tool is also used to measure the number of commits over different intervals of time. Sloc Cloc and Code (scc) is used to measure the number of text based files as well as the number of total, code, comment, and blank lines in a repository. These tools were selected due to their installability, usability, and ability to gather the empirical measures listed below. Details on installing and running the tools can be found in our Guide to Empirical Measures. Section 8.2 introduces the required processed data, which is calculated using the raw data.

8.1 Raw Data

The following raw data measures are extracted from repositories:

- Number of stars.
- Number of forks.
- Number of people watching the repository.

- Number of open pull requests.
- Number of closed pull requests.
- Number of developers.
- Number of open issues.
- Number of closed issues.
- Initial release date.
- Last commit date.
- Programming languages used.
- Number of text-based files.
- Number of total lines in text-based files.
- Number of code lines in text-based files.
- Number of comment lines in text-based files.
- Number of blank lines in text-based files.
- Number of binary files.
- Number of total lines added to text-based files.
- Number of total lines deleted from text-based files.
- Number of total commits.
- Numbers of commits by year in the last 5 years. (Count from as early as possible if the project is younger than 5 years.)
- Numbers of commits by month in the last 12 months.

8.2 Processed Data

The following measures are calculated from the raw data:

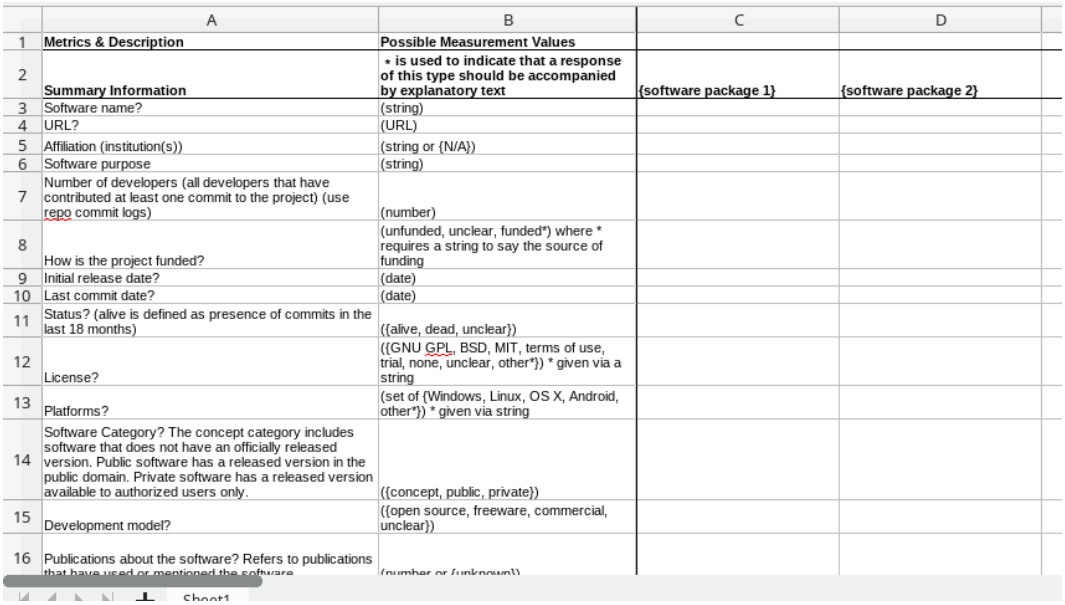
- Status of software package (dead or alive). Alive is defined as the presence of repository commits or software package version releases in the last 18 months.
- Percentage of identified issues that are closed.
- Percentage of code that is comments.

9 Measure Using Measurement Template

The Measurement Template is used to track measurements and quality scores for all of the software packages in the domain. For each software package, we fill out one column of the template. This process can take between 2 to 5 hours for each package. Project developers can be contacted for help regarding installation, if necessary, but a cap of about 2 hours should be imposed on the entire installation process, to keep the overall measurement time feasible.

Most of the data to be collected should be straightforward from reviewing the measurement template. However, in some cases some extra guidance is necessary to eliminate ambiguity, as follows:

1. Initial release date: Mark the release year if an exact date is not available.
2. Publications about the software: A list of publications can be found directly on the website of some software packages. For others use Google Scholar or a similar index.
3. Is there evidence that performance was considered?: Search the software artifacts for any mention of speed, storage, throughput, performance optimization, parallelism, multi-core processing, or similar considerations. The search function on GitHub can help.



	A	B	C	D
1	**Metrics & Description**	**Possible Measurement Values**		
2	**Summary Information**	*** is used to indicate that a response of this type should be accompanied by explanatory text**	**{software package 1}**	**{software package 2}**
3	Software name?	(string)		
4	URL?	(URL)		
5	Affiliation (institution(s))	(string or {N/A})		
6	Software purpose	(string)		
7	Number of developers (all developers that have contributed at least one commit to the project) (use repo commit logs)	(number)		
8	How is the project funded?	(unfunded, unclear, funded*) where * requires a string to say the source of funding		
9	Initial release date?	(date)		
10	Last commit date?	(date)		
11	Status? (alive is defined as presence of commits in the last 18 months)	({alive, dead, unclear})		
12	License?	({GNU GPL, BSD, MIT, terms of use, trial, none, unclear, other*}) * given via a string		
13	Platforms?	(set of {Windows, Linux, OS X, Android, other*}) * given via string		
14	Software Category? The concept category includes software that does not have an officially released version. Public software has a released version in the public domain. Private software has a released version available to authorized users only.	({concept, public, private})		
15	Development model?	({open source, freeware, commercial, unclear})		
16	Publications about the software? Refers to publications that have used or mentioned the software	(number or {unknown})		

Figure 1 Top of Measurement Template

4. Getting started tutorial: Sometimes this is found within another artifact, like the user manual.
5. Continuous integration: Search the software artifacts for any mention of continuous integration. The search function on GitHub can help.

For completing the template, the following steps should be followed:

1. Gather the summary information into the top section of the document (Figure 1)
2. Using the GitStats tool that is described in Section 8 gather the measurements for the Repo Metrics - GitStats section found near the bottom of the document
3. Using the SCC tool that is also described in Section 8 gather the measurements for the Repo Metrics - SCC section found near the bottom of the document
4. If the software package is found on git, gather the measurements for the Repo Metrics - GitHub section found near the bottom of the document
5. Review installation documentation and attempt to install the software package on a virtual machine
6. Gather the measurements for installability
7. Gather the measurements for correctness and verifiability
8. Gather the measurements for surface reliability
9. Gather the measurements for surface robustness
10. Gather the measurements for surface usability
11. Gather the measurements for maintainability
12. Gather the measurements for reusability
13. Gather the measurements for surface understandability
14. Gather the measurements for visibility and transparency
15. Assign a score out of ten for each quality. The score can be measured using the Measurement Template Impression Calculator. For each quality measurement, the file indicates the appropriate score to assign the measurement based on possible measurement values.

10 Analytic Hierarchy Process

The Analytical Hierarchy Process (AHP) is a decision-making technique that can be used when comparing multiple options by multiple criteria. In our work AHP is used for comparing and ranking the software packages of a domain using the quality scores that are gathered in the Measurement Template. AHP performs a pairwise analysis between each of the quality options using a matrix which is then used to generate an overall score for each software package for the given criteria. Smith et al. (2016a) shows how AHP is applied to ranking software based on quality measures. We have developed a tool for conducting this process. The tool includes an AHP JAR script and a sensitivity analysis JAR script that is used to ensure that the software package rankings are appropriate with respect to the uncertainty of the quality scores. The README file of the tool is found here. This file outlines the requirements for, and configuration and usage of, the JAR scripts. The JAR scripts, source code, and required libraries are located in the same folder as the README file.

11 Rank Short List

An AHP pairwise comparison between the short list packages with respect to usability and modifiability experiment results is conducted, and then reviewed by the domain expert. The tool can be found here. The usability and modifiability survey can be found here. This step is intended to start a conversation about the relative quality of short-listed software packages.

References

S. Faulk, E. Loh, M. L. V. D. Vanter, S. Squires, and L. G. Votta. Scientific computing's productivity gridlock: How software engineering can help. *Computing in Science Engineering*, 11(6):30–39, Nov 2009. ISSN 1521-9615. doi: 10.1109/MCSE.2009.205.

Simon Hettrick, Mario Antonioletti, Leslie Carr, Neil Chue Hong, Stephen Crouch, David De Roure, Iain Emsley, Carole Goble, Alexander Hay, Devasena Inupakutika, et al. Uk research software survey 2014. 2014.

Arne N. Johanson and Wilhelm Hasselbring. Software engineering for computational science: Past, present, future. *Computing in Science & Engineering*, Accepted:1–31, 2018.

Bo Kågström, Per Ling, and Charles Van Loan. Gemm-based level 3 blas: High-performance model implementations and performance evaluation benchmark. *ACM Transactions on Mathematical Software (TOMS)*, 24(3):268–302, 1998.

Reed Milewicz and Elaine M. Raybourn. Talk to me: A case study on coordinating expertise in large-scale scientific software projects. *ArXiv e-prints*, September 2018.

Greg Miller. SCIENTIFIC PUBLISHING: A Scientist's Nightmare: Software Problem Leads to Five Retractions. *Science*, 314(5807):1856–1857, 2006. doi: 10.1126/science.314.5807.1856. URL http://www.sciencemag.org.

David Lorge Parnas. On the design and development of program families. *IEEE Transactions on Software Engineering*, (1):1–9, 1976.

D. E. Post and L. G. Votta. Computational Science Demands a New Paradigm. *Physics Today*, 58(1):35–41, January 2005. doi: 10.1063/1.1881898.

T. L. Saaty. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. McGraw-Hill Publishing Company, New York, New York, 1980.

Dan Sholler, Igor Steinmacher, Denae Ford, Mara Averick, Mike Hoye, and Greg Wilson. Ten simple rules for helping newcomers become contributors to open projects. *PLOS Computational Biology*, 15(9):1–10, 09 2019. doi: 10.1371/journal.pcbi.1007296. URL https://doi.org/10.1371/journal.pcbi.1007296.

W. Spencer Smith and Chien-Hsien Chen. Commonality and requirements analysis for mesh generating software. In F. Maurer and G. Ruhe, editors, *Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE 2004)*, pages 384–387, Banff, Alberta, 2004.

W Spencer Smith, Jacques Carette, and John McCutchan. Commonality analysis of families of physical models for use in scientific computing. In *Proceedings of the First International Workshop on Software Engineering for Computational Science and Engineering (SECSE08)*, 2008.

W. Spencer Smith, Adam Lazzarato, and Jacques Carette. State of practice for mesh generation software. *Advances in Engineering Software*, 100:53–71, October 2016a.

W Spencer Smith, D Adam Lazzarato, and Jacques Carette. State of the practice for mesh generation and mesh processing software. *Advances in Engineering Software*, 100:53–71, 2016b.

W. Spencer Smith, John McCutchan, and Jacques Carette. Commonality analysis for a family of material models. Technical Report CAS-17-01-SS, McMaster University, Department of Computing and Software, 2017.

W. Spencer Smith, Adam Lazzarato, and Jacques Carette. State of the practice for GIS software. https://arxiv.org/abs/1802.03422, February 2018a.

W. Spencer Smith, Yue Sun, and Jacques Carette. Statistical software for psychology: Comparing development practices between CRAN and other communities. https://arxiv.org/abs/1802.07362, 2018b. 33 pp.

W. Spencer Smith, Zheng Zeng, and Jacques Carette. Seismology software: State of the practice. *Journal of Seismology*, 22(3):755–788, May 2018c.

BA Szabo and RL Actis. Finite element analysis in professional practice. *Computer methods in applied mechanics and engineering*, 133(3-4):209–228, 1996.

Valera Veryazov, Per-Olof Widmark, Luis Serrano-Andrés, Roland Lindh, and Björn O Roos. 2molcas as a development platform for quantum chemistry software. *International journal of quantum chemistry*, 100(4):626–635, 2004.

David M Weiss. Defining families: The commonality analysis. *submitted to IEEE Transactions on Software Engineering*, 1997. URL http://www.research.avayalabs.com/user/weiss/Publications.html.

David M Weiss. Commonality analysis: A systematic process for defining families. In *International Workshop on Architectural Reasoning for Embedded Systems*, pages 214–222. Springer, 1998. URL citeseer.ist.psu.edu/13585.html.