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 Author: Please fill in 1 or more \ccsdesc macro

Keywords and phrases Author: Please fill in \keywords macro

Contents

1 Introduction

The scope of this document is to conduct a current state of practice analysis of software in a scientific computing domain for the purpose of understanding quality and assisting in future development. We observe product, artifact, and process quality in our analysis. We leave the assessment of scientific computing software using performance benchmarks to other projects, such as the work of Kågström et al. [1998]. The methodology presented here is part of an overall assessment of the impact of model driven engineering and code generation on the sustainability of scientific computing software. The research proposal for this assessment, which contains a more detailed description of this state of practice exercise, can be found here. The research questions that this exercise will answer are found in Section 2. Section 3 outlines the steps to be taken.

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 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. Use AHP process to rank the software packages. (Section 10)
13. Identify a short list of top software packages, typically four to six, for deeper exploration according to the AHP rankings of the measurements.
14. *Domain Experts*: Vet AHP ranking and short list. (Meeting Agenda with Domain Experts)
15. With short list:
    a. Survey developers (Questions to Developers)
    b. Usability experiments (Experiments)
    c. Modifiability experiments (Experiments)
16. Rank short list. (Section 11) [To be completed - add pairwise comparison tool —PM]
17. 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]. The chosen domain must have the following properties:

1. The domain must have a well-defined, stable theoretical underpinning. Standard textbooks, preferably understandable by an upper undergraduate student, is a good sign of this.
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 about 15, candidate software packages in the domain that qualify as 'research software'.

Some examples of domains that fit these criteria are finite element analysis, quantum chemistry, seismology, as well as mesh generators and processors. These are described in [Szabo and Actis, 1996], [Veryazov et al., 2004], [Smith et al., 2018] and [Smith et al., 2016b], respectively.

5 How to Identify Candidate Software

The candidate software can be found through search engine queries targeting authoritative lists of software from the domain, such as lists on GitHub and swMATH, and by searching domain related publications. Domain experts are also asked for their suggestions and are asked to review the list. The candidate software should have the following properties:

1. Major function(s) must fall within the identified domain.
2. Must have viewable source code.
3. Ideally have a git repository or ability to gather empirical measures found in Section 8.
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 in the priority order listed, stopping once a manageable number of projects is reached:

1. Scope: The functionality of the domain software should be increasingly specific.
2. Usage: The installation procedure should appear to be clear or easy to figure out.
3. Age: Ideally the latest release or source code commit should be relatively recent for software in the domain unless the candidate software appears to be well recommended and currently in use.

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

7 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. Likewise, variabilities are goals, theories, models, definitions and assumptions that are different between family members. The final result of the domain analysis will be tables of commonalities, variabilities, and parameters of variation of a program family, which Parnas [1976] defines 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". 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 that we want to answer. 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 will orient the reader as to what raw data is collected. Some of this data can be observed from Git 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. The 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. A guide for installing and running them can be found here. 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

	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})		

Sheet1

Figure 1 Top of 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, 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.

Some guidance on applying sections of the template:

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.
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, please follow these steps:

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 survey results is conducted 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 features of specific software packages.

12 Using Data to Rank Family Members

Describe AHP process (or similar).

A Appendix

A.1 Survey for the Selected Projects

[Several questions are borrowed from Jegatheesan2016, and needed to be cited later. —AD]

A.1.1 Information about the developers and users

1. Interviewees' current position/title? degrees?
2. Interviewees' contribution to/relationship with the software?
3. Length of time the interviewee has been involved with this software?
4. How large is the development group?
5. What is the typical background of a developer?
6. How large is the user group?
7. What is the typical background of a user?

A.1.2 Information about the software

1. [General —AD] What is the most important software quality(ies) to your work? (set of selected qualities plus "else")
2. [General —AD] Are there any examples where the documentation helped? If yes, how it helped. (yes*, no)
3. [General —AD] Is there any documentation you feel you should produce and do not? If yes, what is it and why? (yes*, no)
4. [Completeness —AD] Do you address any of your quality concerns using documentation? If yes, what are the qualities and the documents. (yes*, no)
5. [Visibility/Transparency —AD] Is there a certain type of development methodologies used during the development? ({Waterfall, Scrum, Kanban, else})
6. [Visibility/Transparency —AD] Is there a clearly defined development process? If yes, what is it. ({yes*, no})
7. [Visibility/Transparency —AD] Are there any project management tools used during the development? If yes, what are they. ({yes*, no})
8. [Visibility/Transparency —AD] Going forward, will your approach to documentation of requirements and design change? If not, why not. ({yes, no*})

9. [Correctness and Verifiability —AD] During the process of development, what tools or techniques are used to build confidence of correctness? (string)
10. [Correctness and Verifiability —AD] Do you use any tools to support testing? If yes, what are they. (e.g. unit testing tools, regression testing suites) ({yes*, no})
11. [Correctness and Verifiability —AD] Is there any document about the requirements specifications of the program? If yes, what is it. ({yes*, no})
12. [Portability —AD] Do you think that portability has been achieved? If yes, how? ({yes*, no})
13. [Maintainability —AD] How was maintainability considered in the design? (string)
14. [Maintainability —AD] What is the maintenance type? (set of {corrective, adaptive, perfective, unclear})
15. [Reusability —AD] How was reusability considered in the design? (string)
16. [Reusability —AD] Are any portions of the software used by another package? If yes, how they are used. (yes*, no)
17. [Reproducibility —AD] Is reproducibility important to you? (yes*, no)
18. [Reproducibility —AD] Do you use tools to help reproduce previous software results? If yes, what are they. (e.g. version control, configuration management) (yes*, no)
19. [Completeness —AD] Is any of the following documents used during the development? (yes*, no)
20. [General —AD] Will this experience influence how you develop software? Do you see yourself maintaining the same level of documentation, tool support as you go forward? (string)
    - Module Guide
    - Module Interface Specification
    - Verification and Validation Plan
    - Verification and Validation Report

A.2 Empirical Measures Considerations - Raw Data

Measures that can be extracted from on-line repos.

[Still at brainstorm stage. —AD]

- number of contributors
- number of watches
- number of stars
- number of forks
- number of clones
- number of commits
- number of total/code/document files
- lines of total/logical/comment code
- lines/pages of documents (can pdf be extracted?)
- number of total/open/closed/merged pull requests
- number of total/open/closed issues
- number of total/open/closed issues with assignees

Instead of only focus on the current status of the above numbers, we may find the time history of them to be more valuable. For example, the number of contributors over time, the number of lines of code over time, the number of open issues over time, etc.

A.3 Empirical Measures Considerations - Processed Data

Metrics that can be calculated from the raw data.

[Still at brainstorm stage. —AD]

- percentage of total/open/closed issues with assignees - Visibility/Transparency
- lines of new code produced per person-month - Productivity
- lines/pages of new documents produced per person-month - Productivity
- number of issues closed per person-month - Productivity
- percentage of comment lines in the code - maintainability [Not Ao's qualities —AD]

In the above calculations, a month can be determined to be 30 days.

A.4 Empirical Measures Considerations - Tool Tests

[This section is currently a note of unorganized contents. Most parts will beremoved or relocated. —AD]

[This citation needs to be deleted later. It's here because my compiler doesn't work with 0 citations —AD] Emms [2019]

Most tests were done targeting to the repo of 3D Slicer GitHub repo

A.4.1 git-stats

GitHub repo

Test results: http://git-stats-slicer.ao9.io/ the results are output as webpages, so I hosted for you to check. Data can be downloaded as spreadsheets.

A.4.2 scc

GitHub repo

A.4.3 git-of-theseus

GitHub repo

Test results: It took about 100 minutes for one repo on a 8 core 16G ram Linux machine. It only outputs graphs.

A.4.4 hercules

GitHub repo

Test results: this one seems to be promising, but the installation is complicated with various errors.

A.4.5 git-repo-analysis

GitHub repo

A.4.6 HubListener

GitHub repo

The data that HubListener can extract.

Raw:

- Number of Files

- Number of Lines
- Number of Logical Lines
- Number of Comments

Cyclomatic: Intro
- Cyclomatic Complexity

Halstead: Intro
- Halstead Effort
- Halstead Bugs
- Halstead Length
- Halstead Difficulty
- Halstead Time
- Halstead Vocabulary
- Halstead Volume

Test results: HubListener works well on the repo of itself, but it did not work well on some other repos.

A.4.7 gitinspector

GitHub repo

Test results: it doesn't work well. Instead of creating output results, it prints the results directly in the console.

References

Steve Emms. 16 best free linux medical imaging software. https://www.linuxlinks.com/medicalimaging/, 2019. [Online; accessed 02-February-2020].

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.

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.

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

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

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.