

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	3
2	Research Questions	3
3	Overview of Steps in Assessing Quality of the Domain Software	3
4	How to Identify the Domain	4
5	How to Identify Candidate Software	4
6	How to Initially Filter the Software List	5
7	Domain Analysis	5
8	Empirical Measures	6
8.1	Raw Data	6
8.2	Processed Data	6
8.3	Tool Tests	6
8.3.1	git-stats	6
8.3.2	scc	6
9	Measure Using Measurement Template	7
10	Analytic Hierarchy Process	7

11 Rank Short List	7
12 Quality Specific Measures	8
12.1 Installability [owner —OO]	8
12.2 Correctness [owner —OO]	8
12.3 Verifiability/Testability [owner —OO]	8
12.4 Validatability [owner —OO]	8
12.5 Reliability [owner —OO]	8
12.6 Robustness [owner —PM]	8
12.7 Performance [owner —PM]	8
12.8 Usability [owner —JC]	8
12.9 Maintainability [owner —PM]	8
12.10 Reusability [owner —PM]	8
12.11 Portability [owner —PM]	8
12.12 Understandability [owner —JC]	8
12.13 Interoperability [owner —AD]	8
12.14 Visibility/Transparency [owner —AD]	8
12.15 Reproducibility [owner —SS]	8
12.16 Productivity [owner —AD]	8
12.17 Sustainability [owner —SS]	8
12.18 Completeness [owner —AD]	8
12.19 Consistency [owner —AD]	8
12.20 Modifiability [owner —JC]	8
12.21 Traceability [owner —JC]	8
12.22 Unambiguity [owner —SS]	8
12.23 Verifiability [owner —SS]	8
12.24 Abstract [owner —SS]	8
13 Using Data to Rank Family Members	8
A Appendix	8
A.1 Survey for the Selected Projects	8
A.1.1 Information about the developers and users	9
A.1.2 Information about the software	9
A.2 Empirical Measures Considerations - Raw Data	10
A.3 Empirical Measures Considerations - Processed Data	10
A.4 Empirical Measures Considerations - Tool Tests	10
A.4.1 git-stats	11
A.4.2 scc	11
A.4.3 git-of-theseus	11
A.4.4 hercules	11
A.4.5 git-repo-analysis	11
A.4.6 HubListener	11
A.4.7 gitinspector	12

1 Introduction

Purpose and scope of the document. [Needs to be filled in. Should reference the overall research proposal, and the “state of the practice” exercise in particular. Reference questions we are trying to answer. —SS]

Note the following formatting conventions of this document. Red text denotes a link to information within the document. Purple text denotes a link to supplementary information that is external to this document.[needs to be added to intro to explain convention —PM]

2 Research Questions

1. What artifacts are recommended for research software development? [remove —PM]
2. What artifacts do current software projects produce? [keep —PM]
3. What artifacts are produced in existing software development processes. [remove —PM]
4. What information is captured by current projects artifacts? [keep —PM]
5. What information is necessary to capture for sustainability and productivity? [need to review - subjective, can we capture this from gathering state of practice data? —PM]
6. What commonalities exist among software artifacts? [keep —PM]
7. What are sustainable and productive methods to produce softifacts? [remove —PM]
8. What are the consequences of technology tools towards sustainability and productivity [keep, reword: what technology tools are used? what do they capture? benefits? —PM]
9. What are the common factors that lead to sustainability and productivity? [need to review, reword. How do we gather sustainability from SOP data? —PM]
10. What relationships exist, if any, between 'high ranking' software projects and sustainability/productivity? [need to review —PM]
11. What skills/knowledge are needed by future developers? [copied from original - review —PM]
12. What are the common pain points? [copied from original - review —PM]
13. How can the development process be improved? [copied from original - review —PM]

3 Overview of Steps in Assessing Quality of the Domain Software

1. Start with state of practice research questions. (Section 2) [To be completed —PM]
2. Identify the domain. (Section 4) [To be completed based on suggestions —PM]
3. *Domain Experts*: Create a top ten list of software packages in the domain. (Meeting Agenda with Domain Experts)
4. Brief the Domain Experts on the overall objective, research proposal, research questions, measurement template, survey for short list projects, usability tests, performance benchmarks, maintainability experiments. (Meeting Agenda with Domain Experts)
5. Identify broad list of candidate software packages in the domain. (Section 5) [To be reviewed —PM]
6. Preliminary filter of software packages list. (Section 6) [To be reviewed —PM]
7. *Domain Experts*: Review domain software list. (Meeting Agenda with Domain Experts)
8. Domain Analysis. (Section 7)[To be reviewed —PM]
9. *Domain Experts*: Vet domain analysis. (Meeting Agenda with Domain Experts) [This was part of the original Steps in Assessing Quality list. We need to add it to the Meeting Agenda —PM]
10. Gather source code and documentation for each prospective software package.

11. Collect empirical measures. (Section 8) [To be completed - go over how to clean this up in meeting, or through an issue —PM]
12. Measure using measurement template. (Section 9) [To be reviewed —PM]
13. Use AHP process to rank the software packages. (Section 10) [To be reviewed —PM]
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. Survey developers (Questions to Developers)
 - b. Usability experiments (User Experiments)
 - c. Performance benchmarks[note: this is still in consideration —PM]
 - d. Maintainability experiments[note: these experiments need to be completed - to be discussed at next meeting —PM]
17. Rank short list. (Section 11) [To be completed —PM]
18. Document answers for research questions.

[The domain expert is involved in multiple steps in the process. How best to get their feedback? The domain experts are busy and are unlikely to devote significant time to the project. We need to quickly get to the point. Maybe something around task based inspection? Directed interview? —SS]

4 How to Identify the Domain

[add introduction —PM] [add a definition in the introduction for what we mean by research software —PM]

1. The domain must fall within the research software scope.
2. The domain must be well understood.
3. There must be a community of people studying the domain.
4. A preliminary search, or discussion with experts, suggests that there will be numerous, at least about 15, candidate software packages to study.
5. The software packages must have open source options.

[SS:"Add examples of well understood domains. We can include pointers to previous "state of the practice" projects as examples." —PM]

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 through 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. ["Ideally have the latest release or source code commit within the last 5 years." How about this one? Do we care about the "vintage" software? —AD][I tested some software that has not been updated in about 8 years, as this software was mentioned in multiple places and looked professional. That being said, age should be noted as something that

is considered along with other qualities. "Ideally the latest release or source code commit should be relatively recent, such as within the last 5 years, unless the candidate software appears to be well recommended and currently in use." —PM]

6 How to Initially Filter the Software List

The initial list of candidate software packages should first be filtered to a manageable size. Our target is about 30 packages. We want the list to include packages that are not marked as incomplete and are at least somewhat organized and understandable. We also want the list to be a good representation of the software domain, with packages having enough in common with respect to functionality that it makes sense to compare the software. The initial list of candidate software should be filtered using the following properties:

1. Organization - The software and any related documentation should appear to be easy to gather and understand.
2. Available documentation - The purpose of the software and the installation and usage procedures should appear to be moderately clear or easy to find. [What if installation or usage procedures are unavailable? I think we're measuring that if the software have installation instructions and user manuals, so perhaps they should be kept even if these docs are missing. —AD] [How are we installing the software if such documents and procedures are missing? Should we expect users to be able to install? I did not encounter a situation where even basic installation steps were missing. —PM]
3. Status - The software cannot be marked as incomplete or in an initial development phase.
4. [What if there are still more candidates than we need? Should we also filter out the "older" or "less popular" ones if needed? —AD][Good question. Let's discuss during meeting. —PM]

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 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. The final result of the analysis will be tables of commonalities, variabilities and parameters of variation.

Steps to produce a commonality analysis:

1. Write an Introduction
2. Write the Overview of Domain
3. List Commonalities - Split into sections for the simulation step, input information, output information, and nonfunctional requirements
4. List Variabilities - Split into sections for the simulation step, input information, output information, and nonfunctional requirements
5. List Parameters of Variation - Split into sections for the simulation step, input information, output information, system constraints, and nonfunctional requirements
6. Add Terminology, Definitions, Acronyms

A sample commonality analysis for Lattice Boltzmann Solvers can be found [here](#).

8 Empirical Measures

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).
- Percentage of identified issues that are closed.
- Percentage of code that is comments.

8.3 Tool Tests

The following tools are used to gather some of the raw data from the repositories:

8.3.1 git-stats

[GitHub repo](#)

8.3.2 scc

[GitHub repo](#)

9 Measure Using Measurement Template

For each software package fill out one column in the [Measurement Template](#) following these steps: [\[Maybe a one or two sentences introduction to the template? —AD\]](#)

1. Gather the summary information for rows 3 to 20
2. Using the GitStats tool found in Section [8.3.1](#) gather the measurements for rows 105 to 112 [\[Will it be easier to use the section names instead of row numbers? Otherwise it can be hard to maintain. —AD\]](#)
3. Using the SCC tool found in Section [8.3.2](#) gather the measurements for rows 114 to 118
4. If the software package is found on git, gather the measurements for rows 120 to 124
5. Review installation documentation and attempt to install the software package
6. Gather the measurements for rows 22 to 37
7. Gather the measurements for correctness and verifiability in rows 39 to 48 [\[for 7 to 14, maybe we can put them together, to something like "measurements for the followings sections..." —AD\]](#)
8. Gather the measurements for surface reliability in rows 50 to 56
9. Gather the measurements for surface robustness in rows 58 to 61
10. Gather the measurements for surface usability in rows 63 to 68
11. Gather the measurements for maintainability in rows 70 to 78
12. Gather the measurements for reusability in rows 80 to 83
13. Gather the measurements for surface understandability in rows 85 to 94
14. Gather the measurements for visibility and transparency in rows 96 to 101

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. \[2016\]](#) 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

Rank using pairwise comparison of short list software packages with respect to usability survey results.

[\[SS: let's ask the people that do the usability/performance/maintainability experiments to do an AHP pair-wise comparison between the short-list software packages —PM\]](#)

12 Quality Specific Measures

- 12.1 **Installability** [owner —OO]
- 12.2 **Correctness** [owner —OO]
- 12.3 **Verifiability/Testability** [owner —OO]
- 12.4 **Validatability** [owner —OO]
- 12.5 **Reliability** [owner —OO]
- 12.6 **Robustness** [owner —PM]
- 12.7 **Performance** [owner —PM]
- 12.8 **Usability** [owner —JC]
- 12.9 **Maintainability** [owner —PM]
- 12.10 **Reusability** [owner —PM]
- 12.11 **Portability** [owner —PM]
- 12.12 **Understandability** [owner —JC]
- 12.13 **Interoperability** [owner —AD]
- 12.14 **Visibility/Transparency** [owner —AD]
- 12.15 **Reproducibility** [owner —SS]
- 12.16 **Productivity** [owner —AD]
- 12.17 **Sustainability** [owner —SS]
- 12.18 **Completeness** [owner —AD]
- 12.19 **Consistency** [owner —AD]
- 12.20 **Modifiability** [owner —JC]
- 12.21 **Traceability** [owner —JC]
- 12.22 **Unambiguity** [owner —SS]
- 12.23 **Verifiability** [owner —SS]
- 12.24 **Abstract** [owner —SS]

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