Digging Deep to Assess the State of the Practice for Different Research Software Domains

Spencer Smith[1] [0000−0002−0760−0987]

McMaster University, 1280 Main Street West, Hamilton ON L8S 4K1, Canada
smiths@mcmaster.ca
http://www.cas.mcmaster.ca/~smiths/

Abstract. To improve software development methods and tools for research software, we first need to understand the current state of the practice. Therefore, we have developed a methodology for assessing the state of the software development practices for a given research software domain. The methodology is applied to one domain at a time in recognition that software development in different domains is likely to have adopted different best practices. Moreover, providing a means to measure different domains facilitates comparison of development practices between domains. For each domain we wish to answer questions such as: i) What artifacts (documents, code, test cases, etc.) are present? ii) What tools are used? iii) What principles, process and methodologies are used? iv) What are the pain points for developers? v) What actions are used to improve qualities like maintainability and reproducibility? To answer these questions, our methodology prescribes the following steps: i) Identify the domain; ii) Identify a list of candidate software packages; iii) Filter the list to a length of about 30 packages; iv) Gather source code and documentation for each package; v) Collect repository related data on each software package, like number of stars, number of open issues, number of lines of code; vi) Fill in the measurement template (the template consists of 108 questions to assess 9 qualities (including the qualities of installability, usability and visibility)); vii) Interview developers (the interview consists of 20 questions and takes about an hour); viii) Rank the software using the Analytic Hierarchy Process (AHP); and, ix) Analyze the data to answer the questions posed above. A domain expert should be engaged throughout the process, to ensure that implicit information about the domain is properly represented and to assist with conducting an analysis of the commonalities and variabilities between the 30 selected packages. Using our methodology, spreadsheet templates and AHP tool, we estimate (based on our experience with using the process) the time to complete an assessment for a given domain at 173 person hours.

The abstract should briefly summarize the contents of the paper in 150–250 words.

Keywords: First keyword · Second keyword · Another keyword.

1 Introduction

- research software is vital for making progress in science and engineering, but it is difficult to develop - this has promoted many studies to understand the challenges of research software development - the previous studies have usually used interviews, surveys and case studies - the interviews and surveys recruit participants from all domains of research software - the case studies, by their nature, focus on specific examples - what is missing is a methodology that: - goes beyond surveys to look at the actual artifacts in the software repos - focuses on one domain at a time - by focusing on one domain at a time we can: - provide useful knowledge/advice to the developers in that domain - build up a set of measures of different domains to compare and contrast them

Research software uses computing to simulate mathematical models of real world systems so that we can better understand and predict those systems' 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 high 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 [**?**]. The lack of SE tools and methods contributes to sustainability and reliability problems [**?**]. Problems with the current state of the practice are evident from embarrassing failures, like a retraction of derived molecular protein structures [**?**], false reproduction of sonoluminescent fusion [**?**], and fixing and then reintroducing the same error in a large code base three times in 20 year [**?**]. To improve this situation, we need to first fully understand the current state of the practice for research software development.

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. To emphasize that the method is generic, we will label the specific domain as X within this document.

This "state of the practice for domain X" exercise builds off of prior work on measuring/assessing the state of software development practice in several research domains. We have updated the work that was done previously for domains such as Geographic Information Systems [**?**], Mesh Generators [**?**], Seismology software [**?**], and Statistical software for psychology [**?**]. Initial tests of the new methodology have been done for medical image analysis software [**?**] and for Lattice Boltzmann Method (LBM) software [**?**].

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 still target about 30 software examples from each domain, but we are now collecting more data. 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 around a person month per domain.[1]

To begin our re-boot of the previous methodology we critically assessed, and subsequently modified, our previous set of measures. In addition, the following data has been added to the new methodology:

- Characterization of the functionality provided by the software in the domain via a commonality analysis.
- Project repository related data, such as the number of files, number of lines of code, percentage of issues that are closed, etc.
- Interviews with software developers in domain X.

Unlike the previous measurement process, the new methodology involves and engages a domain expert partner throughout. We did not previously engage the domain expert with the rationale that we wished to eliminate potential bias. However, this advantage is not worth the inability to evaluate the functionality of the software. Moreover, not having an expert makes navigating the on-line resources difficult, since on-line resources are often silent on information that is implicit to domain experts. Furthermore, not every statement found on-line will necessarily be accurate. The importance of the domain expert is particularly noteworthy when it comes time for publication and dissemination of the state of the practice assessment. Throughout this document the person (or persons) that provide domain expertise will be designated as the *Domain Expert*.

In the proposed methodology, the collected data is combined to rank the software within each domain using the Analytic Hierarchy Process (AHP) [**?**]. As in the previous measurement exercise, we use AHP to develop a list of software ranked by quality. However, in the new process we do not stop with this list. The Domain Expert is consulted to verify the ordering, and to discuss the decisions that led to the ranking. The AHP process is used to facilitate a conversation with the Domain Expert as a means to deepen our understanding of the software in the domain, and the needs of typical developers.

So that the collected data for a given domain can benefit the scientific community, our recommendation is that all collected data be made public. For instance, the data collection for each domain can be put on a GitHub repository. In addition to the project record left on GitHub, the final data can be exported to Mendeley Data. As an example, the measurements for the state of the practice for GIS software are available on Mendeley. Ideally the full analysis of the state of the practice for domain X will also be published in a suitable journal, allowing for dissemination/feedback and communication.

[1] A person month is considered to be 20 working days (4 weeks in a month, with 5 days of work per week) at 8 person hours per day, or $20 \cdot 8 = 160$ person hours.

The scope of this methodology includes observations on product, artifact (documentation, test scripts, etc.) and process quality for research software. We leave the assessment of the performance of research software, for instance using benchmarks, to other projects, such as the work of [**?**]. Currently we are also leaving experiments to measure usability and modifiability as future work, as discussed in Section 14.

The full methodology is presented in the sections that follow. Section 2 highlights the research questions that are to be answered for each measured domain. These questions are answered by the data collected using the process outlined in Section 3. The major steps in the process are outlined in Sections 4 – 12. Following this, the time required for assessing a single domain is estimated in Section 13.

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. 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 (X). (Section 4)
2. Ask the Domain Expert to create a top ten list of software packages in the domain. (Section 5)
3. Meet with the Domain Expert to brief them on the overall objective, research proposal, research questions, measurement template, and developer interviews. (Section 5)
4. Identify the broad list of candidate software packages in the domain. (Section 6)
5. Preliminary filter of software packages list. (Section 7)
6. Review software list with Domain Expert. (Section 5)
7. Domain Analysis (with the help of the Domain Expert). (Section 8)
8. Ask Domain Expert to vet domain analysis. (Section 5)
9. Gather source code and documentation for each prospective software package.
10. Collect repository based information. (Section 9)
11. Measure using measurement template. (Section 10)
12. Survey developers. (Section 11)
13. Use AHP process to rank the software packages. (Section 12)
14. Ask Domain Expert to vet AHP ranking. (Section 5)
15. Answer research questions (from Section 2) and document answers.

4 How to Identify the Domain?

A domain of research software must be identified to begin the assessment. 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" [**?**] in a scientific or engineering context. Research software is a more general term for 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, close to 30, 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 [**?**].

5 Interaction With Domain Expert

As mentioned in the introduction (Section 1), the Domain Expert is an important member of the state of the practice assessment team. Pitfalls exist if non-domain experts attempt to acquire an authoritative list of software, or perform a commonality analysis, or try to definitively rank the software. The main source of problems for non-domain experts is that they can only rely on information that is available on-line, but on-line data has two potential problems: i) the on-line resources could have false or inaccurate information; and, ii) the on-line resources could leave out relevant information that is so in-grained with experts that nobody thinks to explicitly record the information.

Domain experts may be recruited from academia or industry. The only requirements are knowledge of the domain and a willingness to be engaged in the assessment process. The Domain Expert does not have to be a software developer, but they should be a user of domain software. Given that the domain experts are likely to be busy people, the measurement process cannot put to much of a burden on their time.

In advance of the first meeting with the Domain Expert (Step 3 in Section 3) the expert is asked to create a top ten list of software packages in the domain. This is done to help the expert get in the right mind set in advance of the first meeting. Moreover, by doing the exercise in advance, we avoid the potential pitfall of the expert approving the discovered list of software without giving it adequate thought. The emphasis during the first meeting is for the Domain Expert to learn what is expected of them. The discussion should also cover avenues for publication and dissemination.

The Domain Experts are asked to vet the collected data and analysis. In particular, they are asked to vet the proposed list of software packages, the domain analysis and the AHP ranking. These interactions can be done either electronically or with in-person (or virtual) meetings.

6 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 9 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.

7 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. (The Domain Expert can be consulted on this question, if necessary.)

Copies of both the initial and filtered lists, along with the rationale for shortening the list, should be kept for traceability purposes.

8 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 [**?**] 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. [**?**] 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. [**?**] present a template for conducting a commonality analysis, which was referred to when conducting this work. [**?**] describes another commonality analysis technique for deciding the members of a program family. [**?**] and [**?**] 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 an Overview of the Domain Knowledge
3. List Commonalities
4. List Variabilities
5. List Parameters of Variation
6. Add Terminology, Definitions, Acronyms

9 Repository Based Measures

Some quality measurements rely on gathering raw and processed data from software repositories. We focus on data that is reasonably easy to collect, which we combine and analyze. The measures that are collected relate to the research questions (Section 2). For instance, we collect data to see how large a project is, to ascertain a project's popularity, and to determine whether to project is being actively developed.

Section 9.1 lists the raw data that 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 Appendix **??**. Section 9.2 introduces the required processed data, which is calculated using the raw data.

9.1 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.

9.2 Processed Data

The following measures are calculated from the raw data:

- Status of software package as either dead or alive, where 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.

The time frame of 18 months was selected as the separating point between alive and dead projects because this is the usual timeframe for operating system updates.

10 Measure Using Measurement Template

The Measurement Template is found in Appendix **??**. This template is used to track measurements and quality scores for all of the software packages in the domain. For each software package, we fill-in the template questions. This process can take between 1 to 4 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. To save time, a blank measurement template spreadsheet has been prepared, with the measures as rows. An excerpt of the spreadsheet is shown in Figure 1. A column should be added to this template for each software package to be measured.

Fig. 1. Excerpt of the Top Section of the Measurement Template (Summary Information)

The full template consists of 108 questions categorized under 9 qualities. The questions were designed to be unambiguous, quantifiable and measurable with limited time and domain knowledge. The measures are grouped under headings for each quality, and one for summary information. The summary information (shown in Figure 1) is the first section of the template. This section summarizes general information, such as the software name, number of developers, etc. We follow the definitions given by [**?**] for the software categories. Public means software intended for public use. Private means software aimed only at a specific group, while the concept category is used for software written simply to demonstrate algorithms or concepts. The three categories of development models are: open source, where source code is freely available under an open source license; free-ware, where a binary or executable is provided for free; and, commercial, where the user must pay for the software product.

Following the summary section are sections to measure 9 qualities: 1. installability; 2. correctness and verifiability; 3. surface reliability; 4. surface robustness; 5. surface usability; 6. maintainability; 7. reusability; 8. surface understandability; and, 9. visibility/transparency. Definitions of these qualities are available in a working document on software quality. Several of the qualities use the word "surface". This is to highlight that, for these qualities in particular, the best that we can do is a shallow measure of the quality. For instance, we are not currently doing any experiments to measure usability. Instead, we are looking for an indication that usability was considered by the developers. We do this by looking for cues in the documentation, like a getting started manual, a user manual and documentation of expected user characteristics.

Most of the data to be collected should be straightforward from reviewing the measurement template. However, in a few cases 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.
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. In some cases, `yaml` files will provide a hint that continuous integration is employed.

To fill-in the spreadsheet 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 9 gather the measurements for the Repo Metrics - GitStats section found near the bottom of the spreadsheet.
3. Using the SCC tool that is also described in Section 9 gather the measurements for the Repo Metrics – SCC section found near the bottom of the spreadsheet.
4. If the software package is found on git, gather the measurements for the Repo Metrics - the GitHub section found near the bottom of the spreadsheet.
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, found in Appendix **??**. For each quality measurement, the file indicates the appropriate score to assign the measurement based on possible measurement values.

As in [**?**], Virtual machines (VMs) are used to provide an optimal testing environments for each package. VMs were used because it is easier to start with a fresh environment without having to worry about existing libraries and conflicts. Moreover, when the tests are complete the VM can be deleted, without any impact on the host operating system. The most significant advantage of using VMs is to level the playing field. Every software install starts from a clean slate, which removes "works-on-my-computer" errors. When filling in the measurement template spreadsheet, the the details for each VM should be noted, including hypervisor and operating system version.

11 Survey Developers

In the previous state of the practice measurement process [**?**,**?**,**?**], we only based our assessment on information available in on-line software repos. However, this

approach meant we weren't able to learn about the development process, the attitudes of the developers, the pain points for developers and how the developers handle modifiability, reproducibility and usability. Therefore, in the reboot of the measurement process, we have explicitly added a stage for interviewing research software developers.

We designed a list of 20 questions to guide our interviews, which can be found in Appendix **??**. Some questions are about the background of the software, the development teams, the interviewees, and how they organize the projects. We also ask about the developer's understandings of the users. Some questions focus on the current and past difficulties, and the solutions the team has found, or will try. We also discuss the importance and current situations of documentation. A few questions are about specific software qualities, such as maintainability, understandability, usability, and reproducibility. The interviews are semi-structured based on the question list; we ask follow-up questions when necessary. Based on our experience, the interviewees usually bring up some exciting ideas that we did not expect, and it is worth expanding on these topics.

Our methodology suggests requesting interviews with a developer from each of the 30 software package. Requests for interviews are sent to all packages so as to not cause a potential bias by singling out any subset of the list. Moreover, since not every developer will agree to the interview request, asking 30 times will typically yield a reasonable number of responses. In our experience, the response rate is between 15% and 30%. In some cases multiple developers from the same project will agree to be interviewed. When sending out interview requests, we recommend finding the contacts on the projects' website, or code repository, or publications, or the biographic pages of the teams' institutions. We send at most two interview request emails to a contact for each software package. Meeting will typically be held using on-line meeting software, like Zoom or Teams. This facilitates recording and automatic transcription of the meetings.

The interviewees should follow a process where they can make informed consent. The interviews should follow standard ethics guideline of asking for consent before interviewing, recording, and including participant details in the report. The interview process presented here was approved by the McMaster University Research Ethics Board under the application number MREB#: 5219.

12 Analytic Hierarchy Process

The Analytical Hierarchy Process (AHP) is a decision-making technique that can be used when comparing multiple options by multiple criteria. AHP focuses on pair-wise comparisons between all options for all criteria. The advantage of pair-wise comparisons is that they facilitates a separation of concerns. Rather than worry about the entire problem, the decision maker can focus on one comparison at a time. 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 (Appendix **??**). AHP performs a pairwise analysis between each of the 9 quality options for each of the 30 software packages. This

results in a matrix, which is used to generate an overall score for each software package for the given criteria. [?] 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 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.

13 Estimate of Time Required

Table 1 estimates the time required (in person hours) to complete a state of the practice assessment for domain X. The table assumes that the domain has already been decided and the Domain Expert has been recruited. The time spent by the Domain Expert is not included in the numbers shown in the table, since the amount of time that the domain expert will work independently of the rest of the assessment team will be small. Moreover, this amount of time will vary greatly depending on the preferred work habits of the Domain Expert. The table follows the steps outlined in Section 3. Time is not included for reviewing the methodology. Moreover, it is assumed that the template spreadsheets linked in this document, and the developed AHP tool, will be employed, rather than developing new tools. The person hours given are a rough estimate, based on our experience completing assessments for medical image analysis software [?] and for Lattice Boltzmann Method (LBM) software [?]. These two domains were assessed at the same time as designing the methodology presented in this document. We did our best to estimate the time spent on measurement and separate it from the time spend on design and development. The estimate assumes 30 software packages will be measured; the numbers will need to be adjusted if the total packages changes.

The total number of person hours is 173 hours. This is close to our goal of 1 month of person hours (160 hours). The amount of time spent by the Domain Expert can be estimated by summing the Domain Expert items in Table 1 and adding an estimate of the time that they will independently spend on their assigned tasks. If we assume that the Domain Expert will spend 2 hours on the domain analysis and another 2 hours with answering questions, the Domain Expert time will be about 12 person hours.

14 Future Work

As explained in the introduction (Section 1), our eventual goal is to improve the state of practice of software development, which requires a baseline means for measuring the current state of the practice. Now that we have a methodology for assessing the state of practice for a given domain, the next task is to complete the measurements for multiple domains. Using the previous and updated

Table 1. Estimated Person Hours for Assessing the State of Practice for Domain X

Task	**Hours**
Initial 1 hour meeting with the Domain Expert plus meeting prep	5
Identify broad list of candidate software (Section 6)	12
Filter software list (Section 7) (10 minutes per package for 30 packages)	5
Review software list with Domain Expert (Section 5)	2
Domain analysis (with help of Domain Expert) (Section 8)	20
Vet domain analysis with Domain Expert (Section 5)	3
Gather source code and documentation for each package (10 minutes per package for 30 packages))	5
Collect repository based data (Section 9) (10 minutes per package for 30 packages)	5
Measure using measurement template (Section 10) (2.5 hours per repo for 30 repos)	75
Solicit developers for interviews	2
Conduct interviews (1.5 hour interviews with 10 developers (assuming 1 in 3 developers agree to an interview))	15
AHP ranking	2
Work with Domain Expert to vet AHP ranking	2
Analyze data and answer research questions	20
Total	**173**

methodologies, we have measured the state of practice for the following domains: Geographic Information Systems [**?**], Mesh Generators [**?**], Seismology software [**?**], Statistical software for psychology [**?**], medical image analysis software [**?**] and for Lattice Boltzmann Method software [**?**]. Future domains for measurement could include finite element software, computational medicine, machine learning, ordinary differential equation solvers, computer graphics, stoichiometry, etc.

With the wealth of data from assessing the state of practice for multiple domains, the next step is a meta-analysis. We would look at how the different domains compare. What lessons from one domain could be applied in other domains? What (if any) differences exist in the pain points between domains? Are there differences in the tools, processes, and documentation between domains?

The current methodology is constrained by limited resources. A 4 hour cap on the measurement time for each software package limits what can be assessed. Within this limit, we can't measure some important qualities, like usability and modifiability. In the future, we propose a more time-consuming process that would capture these other quality measures. To improve the feasibility, the more time consuming measurements would not have to be completed for all 30 packages. Instead, a short list could be identified using the output of the AHP ranking to select the top projects, or to select a sample of interesting projects across the quality spectrum.

If we can add measures for modifiability and usability, we can start to measure the quality impact of software development processes, tools and techniques.

For instance, we have a project (called Drasil [**?**]), which facilitates a development process that focuses on knowledge capture, followed by generation of code and other software artifacts from the captured knowledge. To understand the advantages and disadvantages of Drasil, we could measure its quality. In particular, we would like to see the impact of a generative process on the quality of usability and modifiability.

14.1 Measuring Usability in the Future

In the future, we propose an experiment for assessing the usability of a given software package. Some initial thoughts on how this might be done are recorded in this section. To do the experiment we need an experimental subject, who will be required to complete tasks with the software being studied. The interaction with the software will allow the study subject to experience the software's usability. The tasks for the subject to complete would vary by domain; therefore, the tasks would be selected with the help of the Domain Expert. Criteria for selecting candidate tasks are as follows:

1. Tasks should be executable for subjects with novice to intermediate experience.
2. All tasks should take no more than one hour.
3. Tasks should include the basic/common use cases of the software package.
4. Include tasks that require sequential or hierarchical steps for completion.

Once the study subject has experience with the software, they will be in a position to judge its usability. We will measure the usability using a standardized usability questionnaire, like the System Usability Scale questionnaire or the Post-Study System Usability Questionnaire.

As a starting point for the experiment design, the procedure could be something like the following:

1. Survey participants to collect pre-experiment data (background, experience of subjects (especially with domain software)).
2. Participants perform tasks based on task defined by Domain Experts.
3. Observe the study subjects (take notes, record sessions (screen recorder), watch for body languages and verbal cues).
4. Survey the study subjects to collect feedback (post-experiment interview), complete usability questionnaire.
5. Prepare a summary report of the experimental results.

14.2 Measuring Modifiability in the Future

The next experiment is designed to gather qualitative data regarding the modifiability of each software package. This proposed experiment also involves experimental subjects/participants, who in this case are asked to modify a set of software packages. The specific modifications requested will again depend on the

software domain. In advance of the experiment the Domain Expert will be asked the likely changes for software in this domain. We emphasize likely changes, instead of any changes because software cannot be designed so that everything is equally easy to change [**?**]. The procedure could be along the following lines:

1. Domain Expert lists all likely changes that a developer might be asked to make in a software package in the domain.
2. Survey study participants to collect pre-experiment data (background, experience of subjects (especially with domain software)).
3. Participants perform modification tasks for likely changes on each software package being studied.
4. Observe the study subjects (take notes, record sessions (screen recorder), watch for body languages and verbal cues).
5. Record time needed to make the changes.
6. Confirm through testing that the modified software has the correct behaviour.
7. Survey the study subjects to collect feedback (post-experiment interview).
8. Prepare a summary report of the experimental results.

The study subjects should make the same changes in multiple pieces of software. The reporting of the results will focus more on the relative time differences between the set of software packages, rather than the absolute time to make any given change. To remove biases caused by the participants experience, the different study subjects should use a different order as they go through the list of software packages. Details for this experiment still need to be resolved, such as how to take the participants prior knowledge into account, especially with respect to their programming experience.

15 Concluding Remarks

We have outlined a methodology for assessing the state of the practice for any given research software domain. (Although the scope of the current work has been on research software, there is little in the methodology that is specific to research software, except for the interview question related to the quality of reproducibility.) When applying the methodology to a given domain, we provide a means to answer the following questions: i) What artifacts (documents, code, test cases, etc.) are present? ii) What tools are used? iii) What principles, process and methodologies are used? iv) What are the pain points for developers? v) What actions are used to improve qualities like maintainability and reproducibility? vi) What specific actions are taken to achieve the qualities of usability, traceability, modifiability, maintainability, correctness, understandability, unambiguity, reproducibility and visibility/transparency? vii) How does software designated as high quality by this methodology compare with top rated software by the community?

The methodology depends on the engagement of a Domain Expert. The Domain Expert's role is to ensure that the assessment is consistent with the culture

of the community of practitioners in the domain. The Domain Expert also has an important role to play with the domain analysis. For each domain we conduct a domain analysis to look at the commonalities, variabilities and parameters of variation, for the family of software in the domain. The domain analysis means that software can be compared not just based on its quality, but also based on its functionality.

The methodology follows a systematic procedure that begins with identifying the domain and ends with answering the research questions posed above. In between we collect an authoritative list of about 30 software packages. For each package in the list we fill in our measurement template. The template consists of repository related data (like number of open issues, number of lines of code, etc.) and 108 measures/questions related to 9 qualities: installability, correctness/verifiability, reliability, robustness, usability, maintainability, reusability, understandability and visibility/transparency. Filling in the template requires installing the software, running simple tests (like completing the getting started instructions (if present)), and searching the code, documentation and test files.

The data for each domain is used to rank the software package according to each quality dimension using AHP. The ranking is not intended to identify a single best software package. Instead the ranking is intended to provide insights on the top set of software for each quality. The top performers can be contrasted with the lesser performers to gain insight into what practices in the domain are working. Deeper insight can be obtained by combining this data with the interview data from asking each recruited developer 20 questions.

Combining the quantitative data from the measurement template with the interview results, along with the domain experts knowledge, we can determine the current state of the practice for domain X. Using our methodology, spreadsheet templates and AHP tool, we estimate (based on our experience with using the process) the time to complete an assessment for a given domain at 173 person hours.

Acknowledgements Please place your acknowledgments at the end of the paper, preceded by an unnumbered run-in heading (i.e. 3rd-level heading).

References

1. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
2. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016). `https://doi.org/10.10007/1234567890`
3. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999)
4. Author, A.-B.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010)
5. LNCS Homepage, `http://www.springer.com/lncs`. Last accessed 4 Oct 2017