STATE OF PRACTICE FOR MEDICAL IMAGING SOFTWARE

ASSESSING THE CURRENT STATE OF THE PRACTICE FOR MEDICAL IMAGING SOFTWARE

By

AO DONG.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the degree

Master of Engineering in Computing and Software

McMaster University

MASTER OF ENGINEERING (2021) McMaster University

(Computing and Software) Hamilton, Ontario

TITLE: Assessing the Current State of the Practice for Medical Imaging Software

AUTHOR: Ao Dong

SUPERVISOR: Dr. Spencer Smith

NUMBER OF PAGES: viii, **??**

Abstract

Abstract here

Acknowledgments

acknowledgements here

Contents

Chapter 1

Introduction

introduction

scientific computing (SC), also known as scientific computation or computational science

1.1 Motivation

1.2 Purpose

1.3 Scope

Chapter 2

Background

This chapter introduces several different categories of software, based on which we designed the processes to select the software domain and software candidates in Chapter 3. It also covers the software quality definitions used in the grading template in Appendix A and an overview of Analytic Hierarchy Process (AHP), a tool we used to compare and grading software products.

2.1 Software Categories

To narrow down the scopes when selecting software domains and software packages, we usually target specific software categories. In this section, we discuss three common software categories that are mentioned in Section 3.2, and also SC software. Section 3.1 and Section 3.2 present more details about why we prefer some of these categories.

2.1.1 Open Source Software

For Open Source software (OSS), its source code is openly accessible, and users have the right to study, change and distribute it under a license granted by the copyright holder. For many OSS projects, the development process is based on the collaboration of different contributors worldwide [4]. Accessible source code usually expose more “secrets” of a software project, such as the underlying logic of software functions, the styles that how the developers achieve their works, and the flaws and potential risks in the final product. Thus, it brings much more convenience to the researches analyzing the qualities of the project.

2.1.2 Freeware

Freeware is software that can be used free of charge. Unlike with OSS, the authors of freeware typically do not allow users to access or modify the source code of the software [13]. The term *freeware* should not be confused with *free software*, which is similar to OSS but with a few differences. To the end-users, the differences between freeware and OSS often do not bother them. The fact that these products are free of charge is likely to make them popular to a large base of users. However, software developers, end-users who wish to modify the source code, and researchers looking for inner characteristics of it may find the inaccessible source code to be a problem.

2.1.3 Commercial Software

“Commercial software is software developed by a business as part of its business” [7]. Typically speaking, the users are required to pay to access all of the features of commercial software, excluding access to the source code. However, some commercial software is also free of charge [7]. Based on our experience, most commercial software products are not OSS.

For some specific software, the backgrounds of commercial software developers often differ from the ones of non-commercial OSS. In such a case, the former is usually developed by software engineers, and the latter is likely to have developers who work in the domain of the software and are also end-users of the products. One example is software in Scientific Computing (SC), since the developers need to utilize their domain-specific during the development process [19].

2.1.4 Scientific Computing Software

Software development in SC depends on the knowledge of three areas - the knowledge of a specific engineering or science domain, the ability to mathematically build models and applying algorithms, and the capability to implement theoretical models and algorithms with computational tools. SC software is built with mathematical and computational tools to serve the purpose of solving scientific problems in a domain [11]. The majority of scientists developing their own software are self-taught programmers [19], so there may be a bigger Room for software quality improvement in SC domains.

2.2 Software Quality Definitions

The definitions of software qualities are from Smith et al. [16]. The order of the qualities follows the grading template in Appendix A.

- **Installability** The effort required for the installation, uninstallation, or reinstallation of a software or product in a specified environment.
- **Correctness & Verifiability** A program is correct if it behaves according to its stated. Verifiability is the extent to which a set of tests can be written and executed, to demonstrate that the delivered system meets the specification.
- **Reliability** The probability of failure-free operation of a computer program in a specified environment for a specified time, i.e. the average time interval between two failures also known as the mean time to failure (MTTF).
- **Robustness** Software possesses the characteristic of robustness if it behaves "reasonably" in two situations: i) when it encounters circumstances not anticipated in the requirements specification, and ii) when the assumptions in its requirements specification are violated.
- **Usability** The extent to which a product can be used by specified users to achieve

specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.

- **Maintainability** The effort with which a software system or component can be modified to i) correct faults; ii) improve performance or other attributes; iii) satisfy new requirements.
- **Reusability** The extent to which a software component can be used with or without adaptation in a problem solution other than the one for which it was originally developed.
- **Understandability** (To be completed)
- **Visibility & Transparency** The extent to which all of the steps of a software development process and the current status of it are conveyed clearly.

2.3 Analytic Hierarchy Process

To generate grading scores for a group of software packages, we use the AHP to pairwise compare them. This tool was developed by Thomas L. Saaty, and it has been widely used to make and analyze multiple criteria decisions [18]. The AHP organizes multiple criteria factors in a hierarchical structure and pairwise compares the alternatives to calculate relative ratios [14].

For a project with m criteria, we can use a $m \times m$ matrix A to record the relative importance between factors. By pairwise compare criterion i and criterion j, the value of A_{ij} is decided as follows, and the value of A_{ji} is $1/A_{ij}$ [14],

- $A_{ij} = 1$ if criterion i and criterion j are equally important;
- $A_{ij} = 9$ if criterion i is extremely more important than criterion j;
- A_{ij} equals to an integer value between 1 and 9 according the the relative importance of criterion i and criterion j.

The above process assumes that criterion i is not less important than criterion j, otherwise, we need to reverse i and j and determine A_{ji} first, then $A_{ij} = 1/A_{ji}$.

The priority vecotr w can be calculated by solving the following equation [14],

$$Aw = \lambda_{max} w, \tag{2.1}$$

where λ_{max} is the maximal eigenvalue of A.

In this project, w is approximated with the approach classic *mean of normalized values* [10],

$$w_i = \frac{1}{m} \sum_{j=1}^{m} \frac{A_{ij}}{\sum_{k=1}^{m} A_{kj}} \tag{2.2}$$

Suppose there are n alternatives, for criterion $i = 1, 2, ..., m$, we can create an $n \times n$ matrix B_i to record the relative preferences between these choices. The way of generating

B_i is similar to the one for A. However, unlike comparing the importance between criteria, we pairwise decide how much one alternative is more favored than the other. The same method is used to calculate the local priority vector for each B_i.

In this project, the 9 software qualities mentioned above are the criteria ($m = 9$), while 29 software packages ($n = 29$) are compared. The software are evaluated with the grading template in Appendix A and a subjective score is given for each quality. For a pair of qualities or software, i and j, such that i is not less significant than j, the pairwise comparison result of i versus j is converted from $min((score_i - score_j) + 1, 9)$.

Chapter 3

Methods

We designed a general process for evaluating the state of the practice of domain-specific software, that we instantiate to SC software for specific scientific domains.

Our method involves: 1) choosing a software domain (Section 3.1); 2) collecting and filtering software packages (Section 3.2); 3) grading the selected software (Section 3.3); 4) interviewing development teams for further information (Section 3.3).

Details of how we applied the method on the MI domain are expanded upon in Section 3.5.

3.1 Domain Selection

Our methods are generic, but our scope only included scientific domains due to the objective of our research, and thus we have only applied this method to scientific domains.

When choosing a candidate domain, we prefer a domain with a large number of active OSS. This is because we aim to finalize a list of 30 software packages [17] in the domain after screening out the ones not meeting our requirements, such as the ones without recent updates or certain functions. Thus the quantity of final-decision packages may not meet our initial expectation if we do not have enough software candidates to choose from. Another reason is that our grading method works much better for OSS. Moreover, we need the domain to have an active community developing and using SC software, making it easier to conduct interviews with developers.

With an adequate number of software packages in a selected domain, we may still find the software products with various purposes and fall into different sub-categories. So we should ask one question to ourselves - do we prefer a group of software all providing similar functions and features, or do we aim to cross-compare several sub-sections within the same domain? With that answered, we can better determine a favored domain.

Another aspect to consider is the team carrying out the research, more specifically, the domain experts - if there is any - in the team. In our team, researchers in the software engineering field usually lead the projects, and experts working in scientific domains support them. Having domain experts in the team provides significant benefits in selecting software packages and designing interview questions.

3.2 Software Product Selection

The process of selecting software packages contains two steps: i) identify software candidates in the chosen domain, ii) filter the list according to needs [17].

3.2.1 Identify Software Candidates

We can find candidate software in publications of the domain. Another source is to search various websites, such as GitHub, swMATH and the Google search results for software recommendation articles. Meanwhile, we should also include the suggested ones from domain experts [17].

3.2.2 Filter the Software List

The goal is to build a software list with a length of about 30 [17].

The only "mandatory" requirement is that the software must be OSS, as defined in Section 2.1.1. This is due to the grading process defined int Section 3.3.1. To evaluate all aspects of some software qualities, the source code should be accessible.

The other factors to filter the list can be optional and we should consider them according to the number of software candidates and the objectives of the research project.

One of the factors is the functions and purposes of the software. For example, we can choose a group of software with similar functions, so that the cross-comparison is between each individual of them. On the other hand, if our objective is to compare sub-categories in the domain, we should select candidates from each of the categories.

The empirical measurement tools listed in Section 3.3.2 have limitations that we can only apply them on projects using Git as the version control tool, so we prefer software with Git. Some manual steps in empirical measurement depend on a few metrics of GitHub, which makes projects held on GitHub more favored [17].

Some of the OSS projects may experience a lack of recent maintenance. So we can eliminate packages that have not been updated for a long time and unless they are still popular and highly recommended by the users in the domain [17].

No matter we set what standards to filter the packages, with domain experts in the team, we should value their opinions. For example, if a software package is not OSS and has not been updated for a long while, the domain experts may still identify it as a popular and widely used product. In this case, perhaps it is valuable to keep this software on the list.

3.3 Grading Software

We grade the selected software using a template (Section 3.3.1) and a specific empirical method (Section 3.3.2). Some technical details for the measurements are in Section 3.3.3.

3.3.1 Grading Template

The full grading template can be found in Appendix A. The template contains 101 questions that we use for grading software products.

We use the first section of the template to collect some general information about the software, such as the name, purpose, platform, programming language, publications about the software, the first release and the most recent change date, website, and source code repository of the product, etc. Information in this section helps us to understand the projects better and may be useful for further analysis, but it does not directly affect the grading scores for the packages.

We designed the next 9 sections in the template for the 9 software qualities mentioned in Section 2.2. For each quality, we ask several questions and the typical answers are among the choices of “yes”, “no”, “n/a”, “unclear”, a number, a string, a date, a set of strings, etc. The last question of each quality is asking for an overall score between 1 and 10 based on all the previous questions of this section, which is also the grading score for this quality. For some qualities, the empirical measurement sections on the template also affect this grading score.

All the last 3 sections on the template are about the empirical measurements. We use two command-line software tools git_stats and scc to extract information about the source code from the project repositories. For projects held on GitHub, we manually collect additional metrics, such as the stars of the GitHub repository, and the numbers of open and closed pull requests. Section 3.3.2 presents more details of how these empirical measurements affect software quality grading scores.

3.3.2 Empirical Measurements

We use two command-line tools for the empirical measurements. One is GitStats that generates statistics for git repositories and display outputs in the format of web pages [6]; the other one is Sloc Cloc and Code (as known as scc) [2], aiming to count the lines of code, comments, etc.

Both tools can measure the number of text-based files in a git repository, as well as lines of text in these files. Based on our experience, most text-based files in a software project repository contain programming source code and developers use them to compile and build software products, and a minority of these files are instructions and other documents. So we roughly regard the lines of text in text-based files as lines of programming code. The two tools usually generate similar but not identical results as for the above measurements. From our understanding, this minor difference is because of the different techniques that they use to detect if a file is a text-based or binary file.

Additionally, we also manually collect some information for projects held on GitHub, such as the numbers of stars, forks, people watching this repository, open pull request, and closed pull request.

These empirical measurements help us from two aspects. Firstly, with more statistical details, we can get an overview of a project faster and more accurately. For example, the number of commits over the last 12 months shows how active this project has been during this period, and the number of stars and forks may reveal its popularity. Secondly, the results may affect our decisions regarding the grading scores for some software qualities. For example, if the percentage of comment lines is low, we might want to double-check the understandability of the code, and if the ratio of open versus closed pull requests is high, perhaps we should pay more attention to the maintainability of the project.

3.3.3 Technical Details

To test the software on a “clean” system, we created a new virtual machine (VM) for each software and only installed the necessary dependencies before measuring. All 29 VM were created on the same computer. We only start the measuring of the next software after finishing a current one, and after grading each software, the VM is destroyed.

Generally speaking, we spend about two hours grading one software, unless there are technical issues and more time is needed to resolve them. In most of the situation, we finish all the measurements for one software on the same day.

3.4 Interview Methods

3.4.1 Interviewee Selection

For a software list with a length of roughly 30, we aim to interview about 10 development teams of these projects. Interviewing multiple individuals from each team should give us more comprehensive information about a project, but if there are difficulties in finding multiple willing participants, a single engineer well knowing the project can also be sufficient.

Ideally, we select projects after the grading measurements and prefer the projects with higher overall scores. However, if we do not find enough participants, we should also contact all teams on the list.

We try to find the contacts of the teams on the websites related to the software, such as the official web pages, repository websites, publication websites, and bio pages of the teams' institutions. For each candidate, we send at most two emails asking for their support and participation before receiving any replies.

3.4.2 Interview Question Selection

We have a list of 20 questions to guide our interviews with the development teams, which can be found in Appendix C.

Some of the questions are about the background of the software, the development teams, the interviewees, and the way they organize the projects. We also ask about their understandings of the users. Another part of the interview questions focuses on the major difficulties the team experience currently or in the past, and the solutions that they have found or will try in the future. We also discuss with interviewees the importance of documents to their projects and the current situations of these documents. One more proportion of the questions are about several specific software qualities, such as maintainability, understandability, usability, and reproducibility.

The interviews are supposed to be semi-structured based on the question list and we also ask follow-up questions when necessary. Based on our experience, the interviewees usually bring up some interesting ideas that we do not expect and we can expand on these topics for a few more details.

3.4.3 Interview Process

Since the members of the development teams are likely to be based around the world, so we organize these interviews as virtual meetings online with Zoom. After receiving consent from the interviewees, we also record our discussions to better transcribe them.

3.5 An Example of Applying the Method

This section shows how we applied our method in Section 3 on the Medical Imaging (MI) domain.

Based on the principles in Section 3.1, we selected the MI domain, because there are numerous software products and a great number of professionals use and/or develop such software in this domain. We decided to focus on MI software with the viewing function, so we needed a large number of software candidates in the domain. Being able to include MI domain experts in our team also made this domain more preferred.

By using the method in Section 3.2.1, we identified 48 MI software projects as the candidates from publications [1] [3] [8], online articles related to the domain [5] [9] [12], forum discussions related to the domain [15], etc.

Among the 48 software packages, there were several ones that we could not find their source code, such as MicroDicom, Aliza, and jivex, etc. These packages are likely to be freeware defined in Section 2.1.2 and not OSS. So following guidelines in Section 3.2.2 we removed them from the list. We focused on the MI software that could work as a MI viewer. Some of the software on the list were tool kits or libraries for other software to use as dependencies but not for end-users to view medical images, such as VTK, ITK, and dcm4che, etc. We also eliminated these from the list. After that, there were 29 software products left on the list. We still preferred projects using git and GitHub and being updated recently, but did not apply another filtering since the number of packages was already below 30. However, 27 out of the 29 software packages on the final list used git, and 24 of which were held on GitHub. Furthermore, 27 packages had the latest updates after the year 2018, and 23 after 2020.

Then we followed the steps in Section 3.3 to measure and grade the software. 27 out of the 29 packages can be installed on two or three different operating systems such as Windows, macOS, and Linux, and 5 of them are browser-based, making them platform-independent. However, in the interest of time, we only performed the measurements for each project by installing it on one of the platforms, most likely Windows.

Going through the interview process in Section 3.4, we started with the teams with higher scores on our list, and eventually contacted all of them. There are developers/architects from 8 teams having participated in our interviews so far. Before contacting any interviewee candidate, we received ethics clearance from the McMaster University Research Ethics Board (Appendix D).

Chapter 4

Measurement Results

For some qualities, it might be a good idea to cross-compare with the empirical scores.

4.1 Selected Software List

4.2 Installability

4.3 Correctness & Verifiability

4.4 Reliability

4.5 Robustness

4.6 Usability

4.7 Maintainability

4.8 Reusability

4.9 Understandability

4.10 Visibility & Transparency

Chapter 5

Interviews with Developers

5.1 Summary of Answers

- Start with one by one, with commonalities and interesting special cases.
- Shorten and summarize later.

5.2 Discussions

Any conclusions?

Chapter 6

Threat to Validity

Chapter 7

Recommendations

I think the recommendations can originate from both parts - measurements and interviews.

Chapter 8

Conclusions

No clues yet. Should be started at a later stage.

Bibliography

[1] Kari Björn. Evaluation of open source medical imaging software: A case study on health technology student learning experience. *Procedia Computer Science*, 121:724–731, 01 2017.

[2] Ben Boyter. Sloc cloc and code. `https://github.com/boyter/scc`, 2021. [Online; accessed 27-May-2021].

[3] Andreas Brühschwein, Julius Klever, Anne-Sophie Hoffmann, Denise Huber, Elisabeth Kaufmann, Sven Reese, and Andrea Meyer-Lindenberg. Free dicom-viewers for veterinary medicine: Survey and comparison of functionality and user-friendliness of medical imaging pacs-dicom-viewer freeware for specific use in veterinary medicine practices. *Journal of Digital Imaging*, 03 2019.

[4] James Edward Corbly. The free software alternative: Freeware, open source software, and libraries. *Information Technology and Libraries*, 33(3):65–75, Sep. 2014.

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

[6] Tomasz Gieniusz. Gitstats. `https://github.com/tomgi/git_stats`, 2019. [Online; accessed 27-May-2021].

[7] GNU. Categories of free and nonfree software. `https://www.gnu.org/philosophy/categories.html`, 2019. [Online; accessed 20-May-2021].

[8] Daniel Haak, Charles-E Page, and Thomas Deserno. A survey of dicom viewer software to integrate clinical research and medical imaging. *Journal of digital imaging*, 29, 10 2015.

[9] Mehedi Hasan. Top 25 best free medical imaging software for linux system. `https://www.ubuntupit.com/top-25-best-free-medical-imaging-software-for-linux-system/`, 2020. [Online; accessed 30-January-2020].

[10] Alessio Ishizaka and Markus Lusti. How to derive priorities in ahp: A comparative study. *Central European Journal of Operations Research*, 14:387–400, 12 2006.

[11] Hemant Kumar Mehta. *Mastering Python scientific computing: a complete guide for Python programmers to master scientific computing using Python APIs and tools*. Packt Publishing, 2015.

[12] Hamza Mu. 20 free & open source dicom viewers for windows. `https://medevel.com/free-dicom-viewers-for-windows/`, 2019. [Online; accessed 31-January-2020].

[13] The Linux Information Project. Freeware definition. `http://www.linfo.org/freeware.html`, 2006. [Online; accessed 20-May-2021].

[14] Thomas L. Saaty. How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, 48(1):9–26, 1990. Desicion making by the analytic hierarchy process: Theory and applications.

[15] Ravi Samala. Can anyone suggest free software for medical images segmentation and volume? `https://www.researchgate.net/post/Can_anyone_suggest_free_software_for_medical_images_segmentation_and_volume`, 03 2014. [Online; accessed 31-January-2020].

[16] Spencer Smith, Jacques Carette, Olu Owojaiye, Peter Michalski, and Ao Dong. Quality definitions of qualities. Manuscript in preparation, 2020.

[17] Spencer Smith, Jacques Carette, Olu Owojaiye, Peter Michalski, and Ao Dong. Methodology for assessing the state of the practice for domain x. Manuscript in preparation, 2021.

[18] Omkarprasad S. Vaidya and Sushil Kumar. Analytic hierarchy process: An overview of applications. *European Journal of Operational Research*, 169(1):1–29, 2006.

[19] Greg Wilson, Dhavide Aruliah, C. Titus Brown, Neil Chue Hong, Matt Davis, Richard Guy, Steven Haddock, Kathryn Huff, Ian Mitchell, Mark Plumbley, Ben Waugh, Ethan White, and Paul Wilson. Best practices for scientific computing. *PLoS Biology*, 12:e1001745, 01 2014.

Appendix A

Full Grading Template

appendix here

Appendix B

Summary of Measurements

appendix here

Appendix C

Interview Answers

appendix here

Appendix D

Ethics Approval

appendix here