State of the Practice for Medical Imaging Software Based on Open Source Repositories

W. Spencer Smith[1,*], Ao Dong[1], Jacques Carette[1], and Michael D. Noseworthy[2]

[1]McMaster University, Computing and Software Department, Canada
[2]McMaster University, Electrical & Computer Engineering Department, Canada
[*]Corresponding Author

June 10, 2024

Abstract

We present the state of the practice for Medical Imaging (MI) software based on data available in open source repositories. We selected 29 medical imaging projects from 48 candidates and assessed 9 software qualities (installability, correctness/ verifiability, reliability, robustness, usability, maintainability, reusability, understandability, and visibility/transparency) by answering 108 questions for each software project. Based on the quantitative data, we ranked the MI software with the Analytic Hierarchy Process (AHP). The top four software products are *3D Slicer*, *ImageJ*, *Fiji*, and *OHIF Viewer*. Our ranking is consistent with the community's ranking, with four of our top five projects also appearing in the top five of a list ordered by stars-per-year. We observed 88% of the documentation artifacts recommended by research software development guidelines. However, the current state of the practice deviates from the existing guidelines because of the rarity of some recommended artifacts (like test plans, requirements specification, code of conduct, code style guidelines, product roadmaps, and Application Program Interface (API) documentation).

Keywords: medical imaging, research software, software engineering, software quality, analytic hierarchy process

1 Introduction

We aim to study the state of software development practice for Medical Imaging (MI) software using data available in open source repositories. MI tools use images of the interior of the body (from sources such as Magnetic Resonance Imaging (MRI), Computed Tomography (CT), Positron Emission Tomography (PET) and Ultrasound) to provide information for diagnostic, analytic, and medical applications [2, 117, 121]. Figure 1, which shows an image of the brain, highlights the importance and value of MI. Through MI medical practitioners and researchers can noninvasively gain insights into the human body, including information on injuries and illnesses. Given the

importance of MI software and the high number of competing software projects, we wish to understand the merits and drawbacks of the current development processes, tools, and methodologies. We aim to assess through a software engineering lens the quality of the existing software with the hope of highlighting standout examples, and providing guidelines and recommendations for future development.

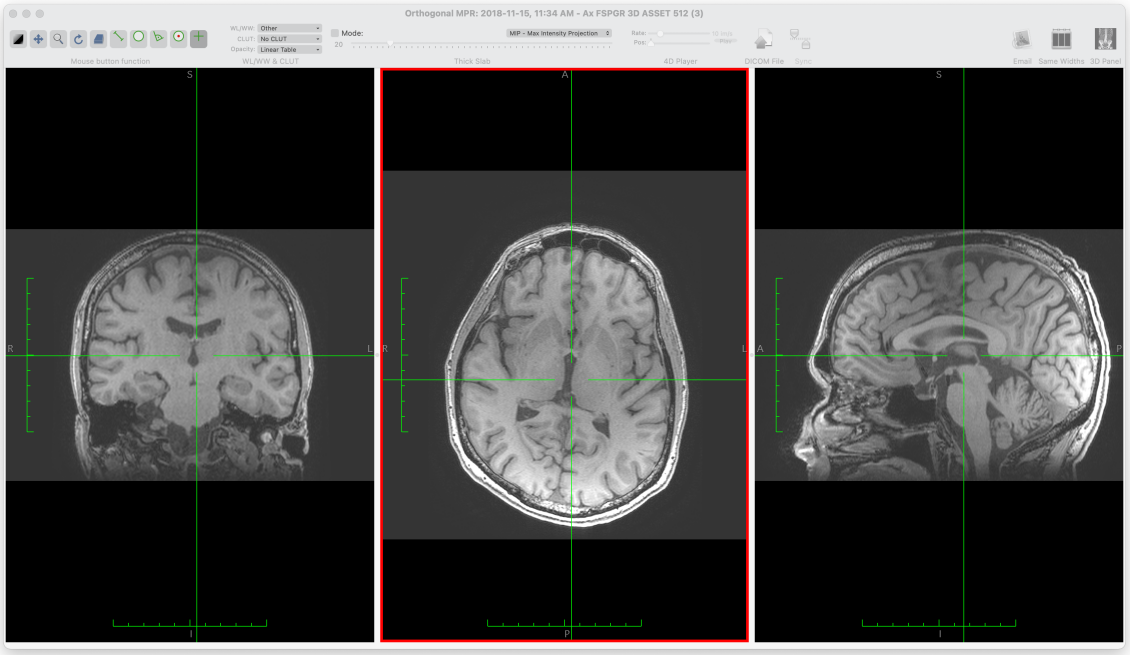

Figure 1: Example brain image showing a multi-planar reformat using Horos (free open-source medical imaging/DICOM viewer for OSX, based on OsiriX)

1.1 Research Questions

Not only do we wish to gain insight into the state of the practice for MI software, we also wish to understand the impact of the often cited gap, or chasm, between software engineering and research software [49, 105]. Although scientists spend a substantial proportion of their working hours on software development [35, 74], many developers learn software engineering skills by themselves or from their peers, instead of from proper training [35]. In the past many scientists have shown ignorance and indifference to standard software engineering concepts [35]. For instance, according to a survey by Prabhu in 2011 [74], more than half of their 114 subjects did not use a proper debugger when coding.

To gain insight, we devised 10 research questions, which can be applied to MI, as well as to other domains, of research software [97, 92]. We designed the questions to learn about the community's interest in, and experience with, software artifacts, tools, principles, processes, methodologies,

and qualities. When we mention artifacts we mean the documents, scripts and code that constitutes a software development project. Example artifacts include requirements, specifications, user manuals, unit tests, system tests, usability tests, build scripts, API (Application Programming Interface) documentation, READMEs, license documents, process documents, and code. Once we have learned what MI developers do, we then put this information in context by contrasting MI software against the trends shown by developers in other research software communities.

We based the structure of the paper on the research questions, so for each research question below we point to the section that contains our answer. We start with identifying the relevant examples of MI software for the assessment exercise:

RQ1: What MI software projects exist, with the constraint that the source code must be available for all identified projects? (Section 2)

RQ2: Which of the projects identified in RQ1 follow current best practices, based on evidence found by experimenting with the software and searching the artifacts available in each project's repository? (Section 2)

RQ3: How similar is the list of top projects identified in RQ2 to the most popular projects, as viewed by the scientific community? (Section 3.1)

RQ4: How do MI projects compare to research software in general with respect to the artifacts present in their repositories? (Section 3.2)

1.2 Scope

To make the project feasible, we only cover MI visualization software. As a consequence we are excluding many other categories of MI software, including Segmentation, Registration, Visualization, Enhancement, Quantification, Simulation, plus MI archiving and telemedicine systems (Compression, Storage, and Communication) [10, 7]. We also exclude Statistical Analysis and Image-based Physiological Modelling [116] and Feature Extraction, Classification, and Interpretation [51]. Software that provides MI support functions is also out of scope; therefore, we have not assessed the toolkit libraries VTK [88] and ITK [58]. Finally, Picture Archiving and Communication System (PACS), which helps users to economically store and conveniently access images [19], are considered out of scope.

1.3 Methodology

We designed a general methodology to assess the state of the practice for research software [97, 92]. Our methodology has been applied to MI software [21] and Lattice Boltzmann Solvers [60, 93]. This methodology builds off prior work to assess the state of the practice for such domains as Geographic Information Systems [102], Mesh Generators [103], Seismology software [95], and Statistical software for psychology [94]. In keeping with the previous methodology, we have maintained the constraint that the work load for measuring a given domain should be feasible for a team as small as one person, and for a short time, ideally around a person month of effort. We consider a person month

as 20 working days (4 weeks in a month, with 5 days of work per week) at 8 person hours per day, or $20 \times 8 = 160$ person hours.

With our methodology, we first choose a research software domain (in the current case MI) and identify a list of about 30 software packages. (For measuring MI we used 29 software packages.) The selected packages are vetted by a domain expert. We approximately measure the qualities of each package by filling in a grading template. The grader looks at the contents of the open source repository and repository based metrics, such as the number of files, number of lines of code, percentage of issues that are closed, etc. With the quantitative data in the grading template, we rank the software with the Analytic Hierarchy Process (AHP). Further details on interaction with the domain expert, software qualities, grading the software and AHP are found below and in [98].

1.3.1 Interaction With Domain Expert

The Domain Experts vet the proposed list of software packages and the AHP ranking. Doing these tasks with non-experts runs the risk of trusting inaccurate resources and overlooking the tacit knowledge of an expert. For the current assessment, our Domain Expert (and paper co-author) is Dr. Michael Noseworthy, Professor of Electrical and Computer Engineering at McMaster University, Co-Director of the McMaster School of Biomedical Engineering, and Director of Medical Imaging Physics and Engineering at St. Joseph's Healthcare, Hamilton, Ontario, Canada.

In advance of the first meeting with the Domain Expert, they are asked to create a list of top software packages in the domain. This is done to help the expert get in the right mind set in advance of the 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.

1.3.2 Software Qualities

Quality is defined as a measure of the excellence or worth of an entity. As is common practice, we do not think of quality as a single measure, but rather as a set of measures. That is, quality is a collection of different qualities, often called "ilities." For this study we selected 9 qualities to measure: installability, correctness/ verifiability, reliability, robustness, usability, maintainability, reusability, understandability, and visibility/transparency. With the exception of installability, all the qualities are defined in Ghezzi et al. (2003) [30]. Installability is defined as the effort required for the installation and/or uninstallation of software in a specified environment [46, 53].

1.3.3 Grading Software

We grade the selected software using an existing measurement template [97]. The template provides measures of the qualities listed in Section 1.3.2. For each software package, we fill in the template questions. To stay within the target of 160 person hours to measure the domain, we allocated between one and four hours for each package. Project developers can be contacted for help regarding installation, if necessary, but we impose a cap of about two hours on the installation process, to keep the overall measurement time feasible. Figure 2 shows an excerpt of the spreadsheet. The

spreadsheet includes a column for each measured software package. [22] provides the full set of measurement data.

Summary Information						
Software name?	3D Slicer	Ginkgo CADx	XMedCon	Weasis	ImageJ	DicomBrowser
Number of developers	100	3	2	8	18	3
Initial release date?	1998	2010	2000	2010	1997	2012
Last commit date?	02-08-2020	21-05-2019	03-08-2020	06-08-2020	16-08-2020	27-08-2020
Status?	alive	alive	alive	alive	alive	alive
License?	BSD	GNU LGPL	GNU LGPL	EPL 2.0	OSS	BSD
Software Category?	public	public	public	public	public	public
Development model?	open source	open source	open source	open source	open source	open source
Num pubs on the software?	22500	51	185	188	339000	unknown
Programming language(s)?	C++, Python, C	C++, C	C	Java	Java, Shell, Perl	Java, Shell
...	...	...	...	...	...	...
Installability						
Installation instructions?	yes	no	yes	no	yes	no
Instructions in one place?	no	n/a	no	n/a	yes	n/a
Linear instructions?	yes	n/a	yes	n/a	yes	n/a
Installation automated?	yes	yes	yes	yes	no	yes
messages?	n/a	n/a	n/a	n/a	n/a	n/a
Number of steps to install?	3	6	5	2	1	4
Numbe extra packages?	0	0	0	0	1	0
Package versions listed?	n/a	n/a	n/a	n/a	yes	n/a
Problems with uninstall?	no	no	no	no	no	no
...	...	...	...	...	...	...
Overall impression (1..10)?	10	8	8	7	6	7
...	...	...	...	...	...	...
Correctness/Verifiability						
...	...	...	...	...	...	...

Figure 2: Grading template example

The full template consists of 108 questions categorized under 9 qualities. We designed the questions to be unambiguous, quantifiable, and measurable with limited time and domain knowledge. We group the measures under headings for each quality, and one for summary information. The summary information (shown in Figure 2) is the first section of the template. This section summarizes general information, such as the software name, purpose, platform, programming language, publications about the software, the first release and the most recent change date, website, source code repository of the product, number of developers, etc. We only measure open source software that is ready for public use by other researchers (following the definition of the public category from [29]). Information in the summary section sets the context for the project, but it does not directly affect the grading scores.

For measuring each quality, we ask several questions and the typical answers are among the collection of "yes", "no", "n/a", "unclear", a number, a string, a date, a set of strings, etc. The grader assigns each quality an overall score, between 1 and 10, based on all the previous questions. 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. For instance, we are not currently doing any experiments to measure usability. Instead, we are looking for an indication that the developers

considered usability. We do this by looking for cues in the documentation, like a getting started manual, a user manual and a statement of expected user characteristics. We use two freeware tools to collect repository related data: GitStats [31] and Sloc Cloc and Code (scc) [12]. Details on how we measure each quality are provided in Smith et al. (2024) [98].

1.3.4 Analytic Hierarchy Process (AHP)

Saaty developed AHP in the 1970s, and people have widely used it since to make and analyze multiple criteria decisions [112]. AHP organizes multiple criteria in a hierarchical structure and uses pairwise comparisons between alternatives to calculate relative ratios [82]. AHP works with sets of n *options* and m *criteria*. In our project $n = 29$ and $m = 9$ since there are 29 options (software products) and 9 criteria (qualities). With AHP the sum of the grades (scores) for all products for a given quality will be 1.0. We rank the software for each of the qualities, and then we combine the quality rankings into an overall ranking based on the relative priorities between qualities.

2 Review

In this section we review the selected software packages. We used a two-step process for selecting the software packages: i) identify software candidates in the chosen domain; and, ii) filter the list to remove less relevant members [97].

We initially identified 48 MI candidate software projects [21] from the literature [11, 15, 32], online articles [24, 36, 61], and forum discussions [83]. To reduce the length of the list to a manageable number (29 in this case), we filtered the original list as follows:

1. We removed the packages with no source code available, such as *MicroDicom*, *Aliza*, and *jivex*.

2. We focused on the MI software that provides visualization functions, as described in Section 1.2. Furthermore, we removed seven packages that were toolkits or libraries, such as *VTK*, *ITK*, and *dcm4che*. We removed another three that were for PACS.

3. We removed *Open Dicom Viewer*, since it has not received any updates in a long time (since 2011).

The Domain Expert provided a list of his top 12 software packages. We compared his list to our list of 29. We found 6 packages were on both lists: *3D Slicer*, *Horos*, *ImageJ*, *Fiji*, *MRIcron* (we actually use the update version *MRIcroGL*) and *Mango* (we actually use the web version *Papaya*). Six software packages (*AFNI*, *FSL*, *Freesurfer*, *Tarquin*, *Diffusion Toolkit*, and *MRItrix*) were on the Domain Expert list, but not on our filtered list. When we examined those packages, we found they were out of scope, since their primary function was not visualization. The Domain Expert agreed with our final choice of 29 packages.

Table 1 shows the 29 software packages that we measured, along with summary data collected in the year 2020. We arrange the items in descending order of LOC. We found the initial release

dates (Rlsd) for most projects and marked the two unknown dates with "?". The date of the last update is the date of the latest update, at the time of measurement. We found funding information (Fnd) for only eight projects. For the Number Of Contributors (NOC) we considered anyone who made at least one accepted commit as a contributor. The NOC is not usually the same as the number of long-term project members, since many projects received change requests and code from the community. With respect to the OS, 25 packages work on all three OSs: Windows (W), macOS (M), and Linux (L). Although the usual approach to cross-platform compatibility was to work natively on multiple OSes, five projects achieved platform-independence via web applications. The full measurement data for all packages is available on Mendeley Data.

Figure 3 shows the primary languages versus the number of projects using them. The primary language is the language used for the majority of the project's code; in most cases projects also use other languages. The most popular language is C++, with almost 40% of projects (11 of 29). The two least popular choices are Pascal and Matlab, with around 3% of projects each (1 of 29).

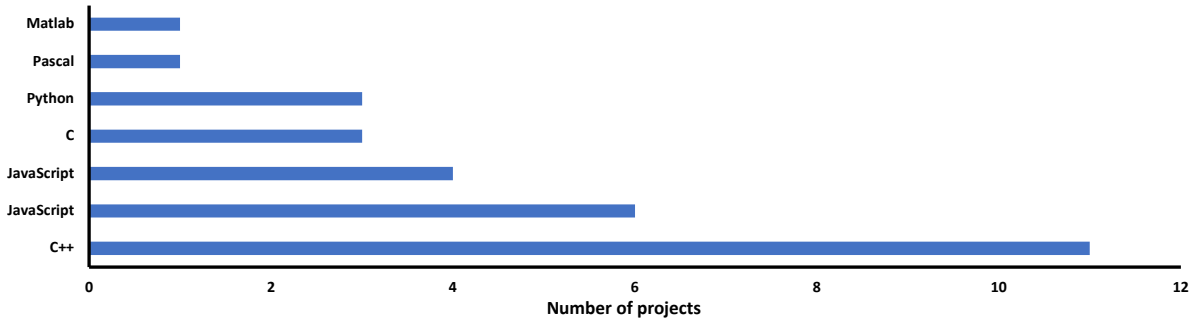

Figure 3: Primary languages versus number of projects

2.1 Installability

Figure 4 lists the installability scores. We found installation instructions for 16 projects. Among the ones without instructions, *BioImage Suite Web* and *Slice:Drop* do not need installation, since they are web applications. Installing 10 of the projects required extra dependencies. Five of them are web applications (as shown in Table 1) and depend on a browser; *dwv*, *OHIF Viewer*, and *GATE* needs extra dependencies to build; *ImageJ* and *Fiji* need an unzip tool; *MatrixUser* is based on Matlab; *DICOM Viewer* needs to work on a Nextcloud platform.

3D Slicer has the highest score because it had easy to follow installation instructions, and an automated, fast, frustration-free installation process. The installer added all dependencies automatically and no errors occurred during the installation and uninstallation steps. Many other software packages also had installation instructions and automated installers. We had no trouble installing the following packages: *INVESALIUS 3*, *Gwyddion*, *XMedCon*, and *MicroView*. We determined their scores based on the understandability of the instructions, installation steps, and user experience. Since *BioImage Suite Web* and *Slice:Drop* needed no installation, we gave them

Software	Rlsd	Updated	Fnd	NOC	LOC	OS			Web
						W	M	L	
ParaView [4]	2002	2020-10	✓	100	886326	✓	✓	✓	✓
Gwyddion [64]	2004	2020-11		38	643427	✓	✓	✓	
Horos [41]	?	2020-04		21	561617		✓		
OsiriX Lite [85]	2004	2019-11		9	544304		✓		
3D Slicer [50]	1998	2020-08	✓	100	501451	✓	✓	✓	
Drishti [54]	2012	2020-08		1	268168	✓	✓	✓	
Ginkgo CADx [119]	2010	2019-05		3	257144	✓	✓	✓	
GATE [47]	2011	2020-10		45	207122		✓	✓	
3DimViewer [107]	?	2020-03	✓	3	178065	✓	✓		
medInria [27]	2009	2020-11		21	148924	✓	✓	✓	
BioImage Suite Web [70]	2018	2020-10	✓	13	139699	✓	✓	✓	✓
Weasis [79]	2010	2020-08		8	123272	✓	✓	✓	
AMIDE [56]	2006	2017-01		4	102827	✓	✓	✓	
XMedCon [66]	2000	2020-08		2	96767	✓	✓	✓	
ITK-SNAP [120]	2006	2020-06	✓	13	88530	✓	✓	✓	
Papaya [77]	2012	2019-05		9	71831	✓	✓	✓	
OHIF Viewer [122]	2015	2020-10		76	63951	✓	✓	✓	✓
SMILI [18]	2014	2020-06		9	62626	✓	✓	✓	
INVESALIUS 3 [5]	2009	2020-09		10	48605	✓	✓	✓	
dwv [57]	2012	2020-09		22	47815	✓	✓	✓	✓
DICOM Viewer [3]	2018	2020-04	✓	5	30761	✓	✓	✓	
MicroView [44]	2015	2020-08		2	27470	✓	✓	✓	
MatrixUser [55]	2013	2018-07		1	23121	✓	✓	✓	
Slice:Drop [33]	2012	2020-04		3	19020	✓	✓	✓	✓
dicompyler [69]	2009	2020-01		2	15941	✓	✓		
Fiji [86]	2011	2020-08	✓	55	10833	✓	✓	✓	
ImageJ [80]	1997	2020-08	✓	18	9681	✓	✓	✓	
MRIcroGL [52]	2015	2020-08		2	8493	✓	✓	✓	
DicomBrowser [8]	2012	2020-08		3	5505	✓	✓	✓	

Table 1: Final software list (sorted in descending order of the number of Lines Of Code (LOC))

high scores. *BioImage Suite Web* also provided an option to download cache for offline usage, which was easy to apply.

GATE, *dwv*, and *DICOM Viewer* showed severe installation problems. We were not able to install them, even after a reasonable amount of time (2 hours). For *dwv* and *GATE* we failed to build from the source code, but we were able to proceed with measuring other qualities using a deployed on-line version for *dwv*, and a VM version for *GATE*. For *DICOM Viewer* we could not install the NextCloud dependency, and we did not have another option for running the software. Therefore, for *DICOM Viewer* we could not measure reliability or robustness. The other seven

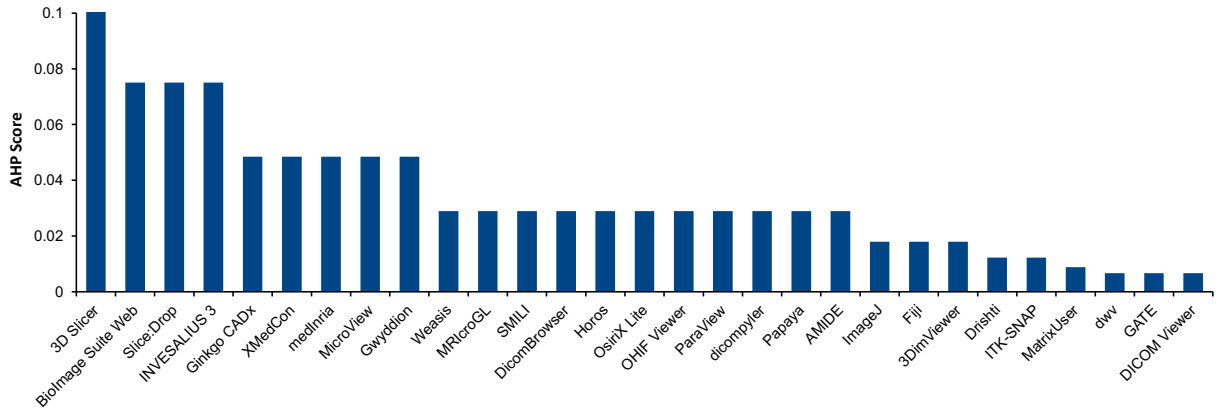

Figure 4: AHP installability scores

qualities could be measured, since they do not require installation.

MatrixUser has a lower score because it depends on Matlab. We assessed the score from the point of view of a user that would have to install Matlab and acquire a license. Of course, for users that already work within Matlab, the installability score should be higher.

2.2 Correctness & Verifiability

Figure 5 shows the scores of correctness and verifiability. Generally speaking, the packages with higher scores adopted more techniques to improve correctness, and had better documentation for us to verify against. For instance, we looked for evidence of unit testing, since it benefits most parts of the software's life cycle, such as designing, coding, debugging, and optimization [34]. We only found evidence of unit testing for about half of the projects. We identified five projects using CI/CD tools: *3D Slicer*, *ImageJ*, *Fiji*, *dwv*, and *OHIF Viewer*.

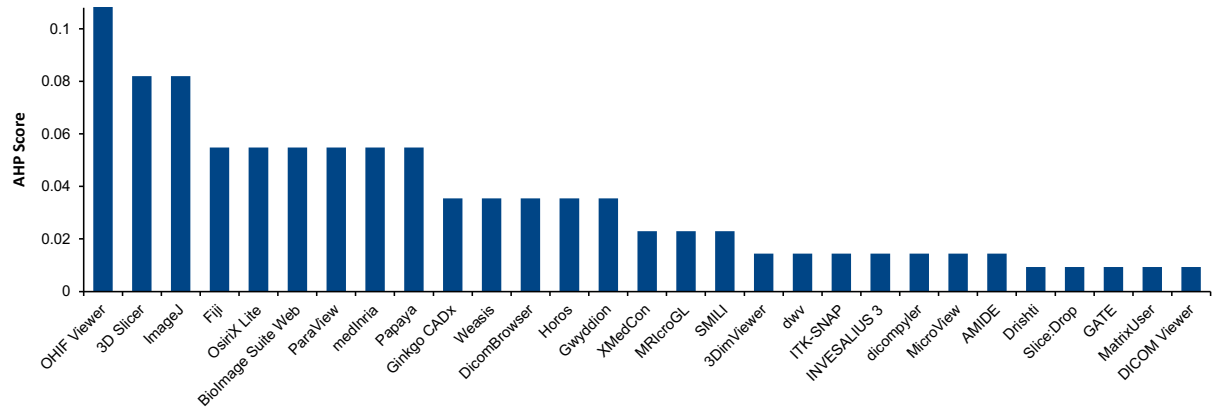

Figure 5: AHP correctness & verifiability scores

Even for some projects with well-organized documentation, requirements specifications and theory manuals were still missing. We could not identify theory manuals for all projects, and we did not find requirements specifications for most projects. The only requirements-related document we found was a road map of *3D Slicer*, which contained design requirements for upcoming changes.

2.3 Surface Reliability

Figure 6 shows the AHP results. As shown in Section 2.1, most of the software products did not "break" during installation, or did not need installation; *dwv* and *GATE* broke in the building stage, and the processes were not recoverable; we could not install the dependency for *DICOM Viewer*. Of the seven software packages with a getting started tutorial and operation steps in the tutorial, most showed no error when we followed the steps. However, *GATE* could not open macro files and became unresponsive several times, without any descriptive error message. We found that *Drishti* crashed when loading damaged image files, without showing any descriptive error message. We did not find any problems with the on-line version of *dwv*.

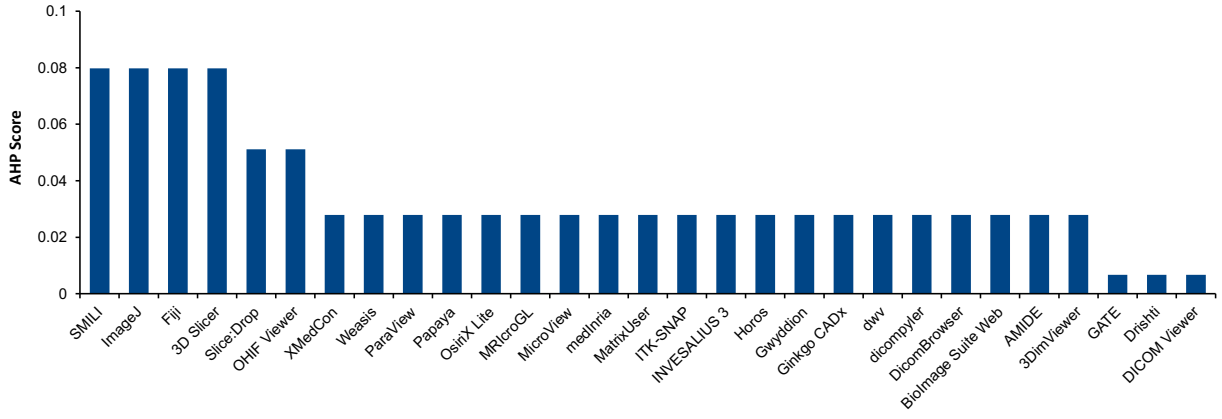

Figure 6: AHP surface reliability scores

2.4 Surface Robustness

Figure 7 presents the scores for surface robustness. The packages with higher scores elegantly handled unexpected/unanticipated inputs, typically showing a clear error message. We may have underestimated the score of *OHIF Viewer*, since we needed further customization to load data.

Digital Imaging and Communications in Medicine (DICOM) "defines the formats for medical images that can be exchanged with the data and quality necessary for clinical use" [9]. According to their documentation, all 29 software packages should support the DICOM standard. To test robustness, we prepared two types of image files: correct and incorrect formats (with the incorrect format created by relabelled a text file to have the ".dcm" extension). All software packages loaded the correct format image, except for *GATE*, which failed for unknown reasons. For the broken format, *MatrixUser*, *dwv*, and *Slice:Drop* ignored the incorrect format of the file and loaded it

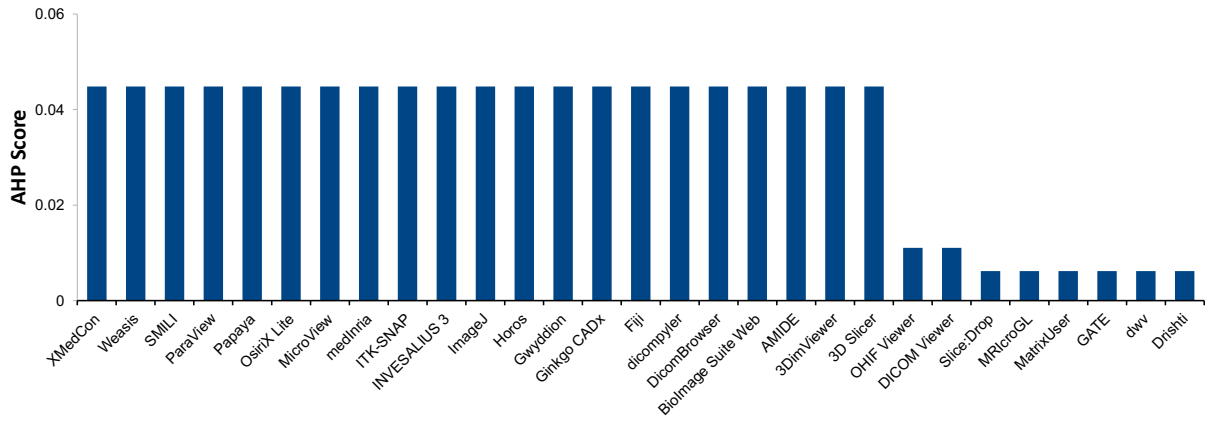

Figure 7: AHP surface robustness scores

regardless. They did not show any error message and displayed a blank image. *MRIcroGL* behaved similarly except that it showed a meaningless image. *Drishti* successfully detected the broken format of the file, but the software crashed as a result.

2.5 Surface Usability

Figure 8 shows the AHP scores for surface usability. The software with higher scores usually provided both comprehensive documented guidance and a good user experience. *INVESALIUS 3* provided an excellent example of a detailed and precise user manual. *GATE* also provided numerous documents, but unfortunately we had difficulty understanding and using them. We found getting started tutorials for only 11 projects, but a user manual for 22 projects. *MRIcroGL* was the only project that explicitly documented expected user characteristics.

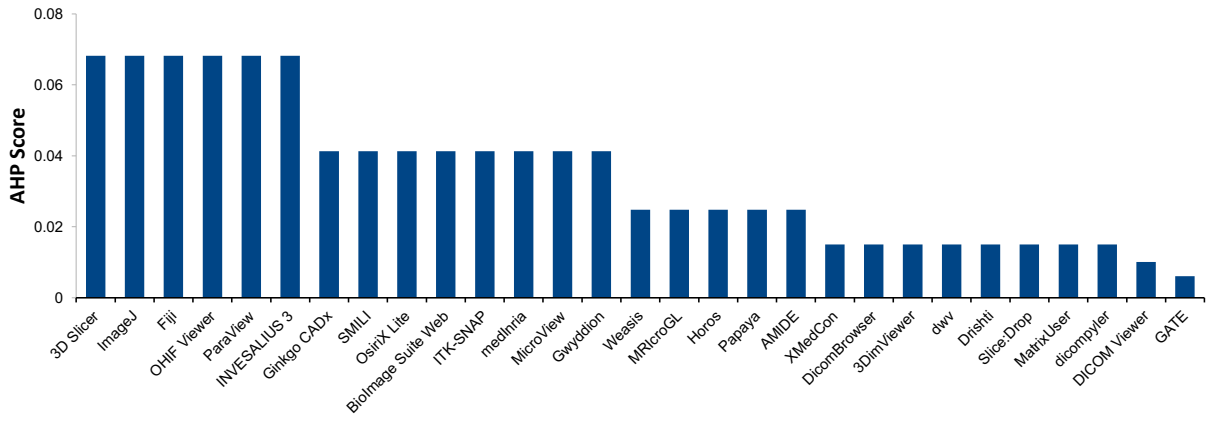

Figure 8: AHP surface usability scores

2.6 Maintainability

Figure 9 shows the ranking results for maintainability. We gave *3D Slicer* the highest score because we found it to have the most comprehensive artifacts. Only a few of the 29 projects had a product, developer's manual, or API documentation, and only *3D Slicer*, *ImageJ*, *Fiji* included all three documents. Moreover, *3D Slicer* has a much higher percentage of closed issues (92%) compared to *ImageJ* (52%) and *Fiji* (64%). Table 2 shows which projects had these documents, in descending order of their maintainability scores.

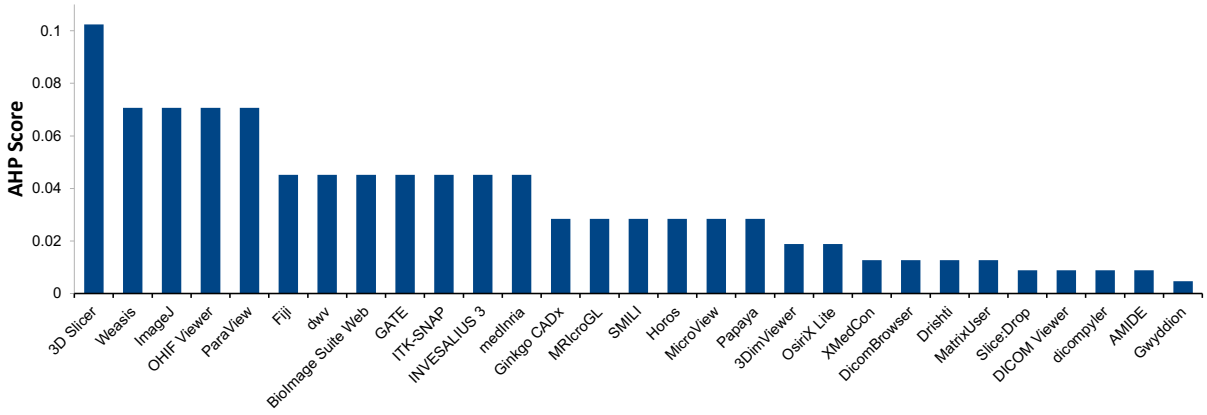

Figure 9: AHP maintainability scores

Software	Prod. Roadmap	Dev. Manual	API Doc.
3D Slicer	✓	✓	✓
ImageJ	✓	✓	✓
Weasis		✓	
OHIF Viewer		✓	✓
Fiji	✓	✓	✓
ParaView	✓		
SMILI			✓
medInria		✓	
INVESALIUS 3	✓		
dwv			✓
BioImage Suite Web		✓	
Gwyddion		✓	✓

Table 2: Software with the maintainability documents (listed in descending order of maintainability score)

Twenty-seven of the 29 projects used git as the version control tool, with 24 of these using GitHub. *AMIDE* used Mercurial and *Gwyddion* used Subversion. *XMedCon*, *AMIDE*, and *Gwyd-*

dion used SourceForge. *DicomBrowser* and *3DimViewer* used BitBucket.

2.7 Reusability

Figure 10 shows the AHP results for reusability. As described in Section 1.3.3, we gave higher scores to the projects with API documentation. As shown in Table 2, seven projects had API documents. We also assumed that projects with more code files and less LOC per code file are more reusable. Table 3 shows the number of text-based files by project, which we used to approximate the number of code files. The table also lists the total number of lines (including comments and blanks), LOC, and average LOC per file. We arranged the items in descending order of their reusability scores.

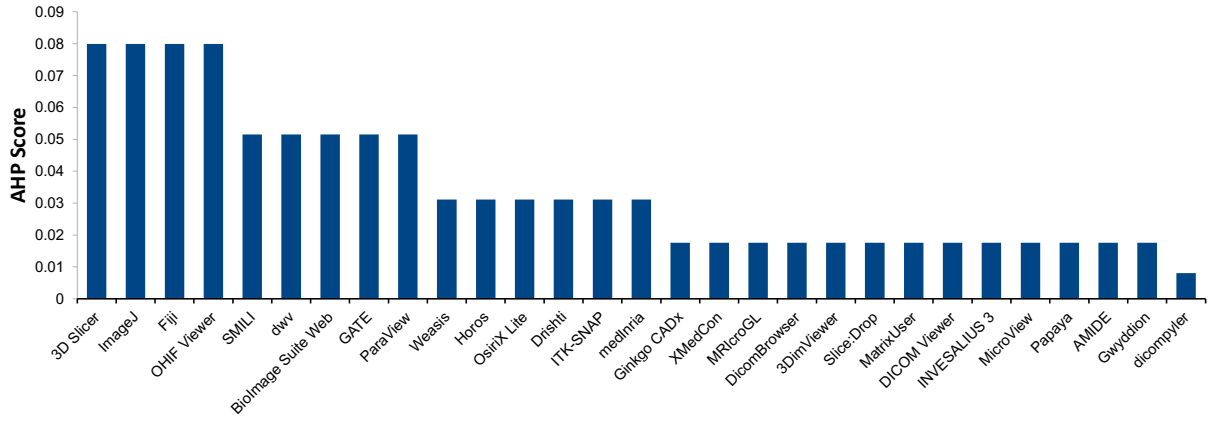

Figure 10: AHP reusability scores

2.8 Surface Understandability

Figure 11 shows the scores for surface understandability. All projects had a consistent coding style with parameters in the same order for all functions, modularized code, and, clear comments that indicate what is done, not how. However, we only found explicit identification of a coding standard for 3 out of the 29: *3D Slicer*, *Weasis*, and *ImageJ*. We also found hard-coded constants (rather than symbolic constants) in *medInria*, *dicompyler*, *MicroView*, and *Papaya*. We did not find any reference to the algorithms used in projects *XMedCon*, *DicomBrowser*, *3DimViewer*, *BioImage Suite Web*, *Slice:Drop*, *MatrixUser*, *DICOM Viewer*, *dicompyler*, and *Papaya*.

2.9 Visibility/Transparency

Figure 12 shows the AHP scores for visibility/transparency. Generally speaking, the teams that actively documented their development process and plans scored higher. Table 4 shows the projects that had documents for the development process, project status, development environment, and release notes, in descending order of their visibility/transparency scores.

Software	Text Files	Total Lines	LOC	LOC/file
OHIF Viewer	1162	86306	63951	55
3D Slicer	3386	709143	501451	148
Gwyddion	2060	787966	643427	312
ParaView	5556	1276863	886326	160
OsiriX Lite	2270	873025	544304	240
Horos	2346	912496	561617	239
medInria	1678	214607	148924	89
Weasis	1027	156551	123272	120
BioImage Suite Web	931	203810	139699	150
GATE	1720	311703	207122	120
Ginkgo CADx	974	361207	257144	264
SMILI	275	90146	62626	228
Fiji	136	13764	10833	80
Drishti	757	345225	268168	354
ITK-SNAP	677	139880	88530	131
3DimViewer	730	240627	178065	244
DICOM Viewer	302	34701	30761	102
ImageJ	40	10740	9681	242
dwv	188	71099	47815	254
MatrixUser	216	31336	23121	107
INVESALIUS 3	156	59328	48605	312
AMIDE	183	139658	102827	562
Papaya	110	95594	71831	653
MicroView	137	36173	27470	201
XMedCon	202	129991	96767	479
MRIcroGL	97	50445	8493	88
Slice:Drop	77	25720	19020	247
DicomBrowser	54	7375	5505	102
dicompyler	48	19201	15941	332

Table 3: Number of files and lines (sorted in descending order of reusability scores)

2.10 Overall Scores

As described in Section 1.3.4, for our AHP measurements, we have nine criteria (qualities) and 29 alternatives (software packages). In the absence of a specific real world context, we assumed all nine qualities are equally important. Figure 13 shows the overall scores in descending order. Since we produced the scores from the AHP process, the total sum of the 29 scores is precisely 1.0.

The top four software products *3D Slicer*, *ImageJ*, *Fiji*, and *OHIF Viewer* have higher scores in most criteria. *3D Slicer* has a score in the top two for all qualities; *ImageJ* ranks near the top

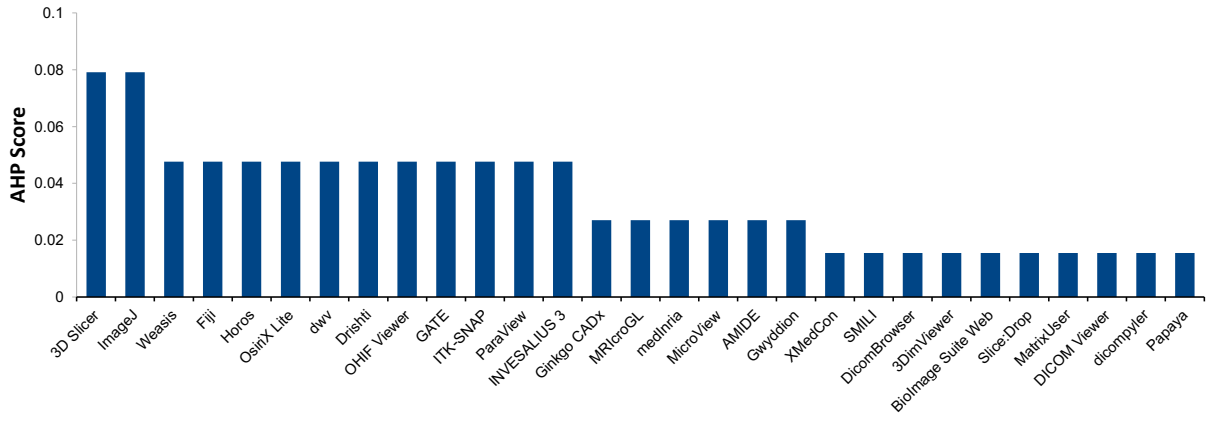

Figure 11: AHP surface understandability scores

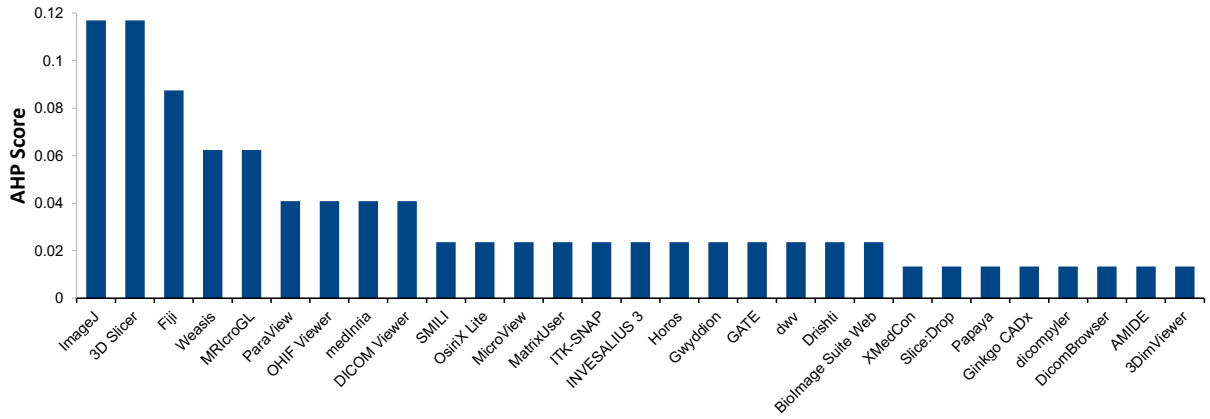

Figure 12: AHP visibility/transparency scores

for all qualities, except for correctness & verifiability. *OHIF Viewer* and *Fiji* have similar overall scores, with *Fiji* doing better in installability and *OHIF Viewer* doing better in correctness & verifiability. Given the installation problems, we may have underestimated the scores on reliability and robustness for *DICOM Viewer*, but we compared it equally for the other seven qualities.

3 Discussion

In Section 3.1, we compare our ranking to the MI software community's ranking, as a way to judge our results, and as a way to understand what the MI community currently values. To understand the state of the practice for MI software relative to other software, we compare MI software to other research software in Section 3.2. The comparison is based on the artifacts our measurement exercise found in the MI repositories, versus what is typically recommended for research software. We discuss threats to the validity of our data and conclusions in section (Section 3.3).

Software	Dev. Process	Proj. Status	Dev. Env.	Rls. Notes
3D Slicer	✓	✓	✓	✓
ImageJ	✓	✓	✓	✓
Fiji	✓	✓	✓	
MRIcroGL				✓
Weasis			✓	✓
ParaView		✓		
OHIF Viewer			✓	✓
DICOM Viewer			✓	✓
medInria			✓	✓
SMILI				✓
Drishti				✓
INVESALIUS 3				✓
OsiriX Lite				✓
GATE				✓
MicroView				✓
MatrixUser				✓
BioImage Suite Web			✓	
ITK-SNAP				✓
Horos				✓
dwv				✓
Gwyddion				✓

Table 4: Software with visibility/transparency related documents (listed in descending order of visibility/transparency score)

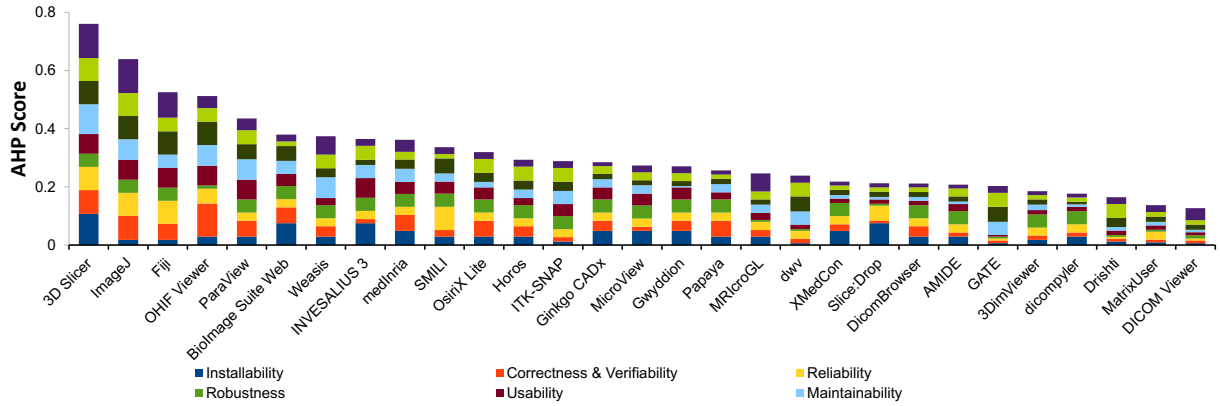

Figure 13: Overall AHP scores with an equal weighting for all 9 software qualities

3.1 Comparison to Community Ranking

To answer RQ3 about how our ranking compares to the popularity of projects as judged by the scientific community, we make two comparisons:

- A comparison of our ranking (from Section 2) with the community ratings on GitHub, as shown by GitHub stars, number of forks, and number of people watching the projects; and,
- A comparison of top-rated software from our methodology with the top recommendations from our domain expert (as mentioned in Section 1.3.1).

Table 5 shows our ranking of the 29 MI projects, and their GitHub metrics, if applicable. As mentioned in Section 2.6, 24 projects used GitHub. Since GitHub repositories have different creation dates, we collect the number of months each stayed on GitHub, and calculate the average number of new stars, people watching, and forks per 12 months. The items in Table 5 are listed in descending order of the average number of new stars per year. The non-GitHub items are listed in the order of our ranking. We collected all GitHub statistics in July 2021.

Generally speaking, most of the top-ranking MI software projects also received greater attention and popularity on GitHub. Between our ranking and the GitHub stars-per-year ranking, four of the top five software projects appear in both lists. Our top five packages are scattered among the first eight positions on the GitHub list. However, as discussed below, there are discrepancies between the two lists.

In some cases projects are popular in the community, but were assigned a low rank by our methodology. This is the case for *dwv*. The reason for the low ranking is that, as mentioned in Section 2.1, we failed to build it locally, and used the test version on its websites for the measurements. We followed the instructions and tried to run the command `yarn run test` locally, which did not work. In addition, the test version did not detect a broken DICOM file and displayed a blank image as described in Section 2.4. We might underestimate the scores for *dwv* due to uncommon technical issues.

We also ranked *DICOM Viewer* much lower than its popularity. As mentioned in Section 2.1, it depended on the NextCloud platform that we could not successfully install. Thus, we might underestimate the scores of its surface reliability and surface robustness.

Further reason for discrepancies between our ranking and the community's ranking is that we weighted all qualities equally. This is not likely how users implicitly rank the different qualities. As a result, some projects with high community popularity may have scored lower with our method because of a relatively higher (compared to the scientific community's implicit ranking) weighting of the poor scores for some qualities. A further explanation for discrepancies between our measures and the star measures may also be due to inaccuracy with using stars to approximate popularity. Stars are not an ideal measure because stars represent the community's feeling in the past more than they measure current preferences [106]. The issue with stars is that they tend only to be added, not removed. A final reason for inconsistencies between our ranking and the community's ranking is that, as for consumer products, more factors influence popularity than just quality.

As shown in Section 2, our domain experts recommended a list of top software with 12 software products. All the top 4 entries from the Domain Expert's list are among the top 12 ranked by our

Software	Comm. Rank	Our Rank	Stars/yr	Watches/yr	Forks/yr
3D Slicer	1	1	284	19	128
OHIF Viewer	2	4	277	19	224
dwv	3	19	124	12	51
ImageJ	4	2	84	9	30
ParaView	5	5	67	7	28
Horos	6	12	49	9	18
Papaya	7	17	45	5	20
Fiji	8	3	44	5	21
DICOM Viewer	9	29	43	6	9
INVESALIUS 3	10	8	40	4	17
Weasis	11	7	36	5	19
dicompyler	12	26	35	5	14
OsiriX Lite	13	11	34	9	24
MRIcroGL	14	18	24	3	3
GATE	15	24	19	6	26
Ginkgo CADx	16	14	19	4	6
BioImage Suite Web	17	6	18	5	7
Drishti	18	27	16	4	4
Slice:Drop	19	21	10	2	5
ITK-SNAP	20	13	9	1	4
medInria	21	9	7	3	6
SMILI	22	10	3	1	2
MatrixUser	23	28	2	0	0
MicroView	24	15	1	1	1
Gwyddion	25	16	n/a	n/a	n/a
XMedCon	26	20	n/a	n/a	n/a
DicomBrowser	27	22	n/a	n/a	n/a
AMIDE	28	23	n/a	n/a	n/a
3DimViewer	29	25	n/a	n/a	n/a

Table 5: Software ranking by our methodology versus the community (Comm.) ranking using GitHub metrics (Sorted in descending order of community popularity, as estimated by the number of new stars per year)

methodology. Three of the top four on both lists are the same: *3D Slicer*, *ImageJ*, and *Fiji*. *3D Slicer* is top project by both rankings (and by the GitHub stars measure as well). The Domain Expert ranked *Horos* as their second choice, while we ranked it twelfth. Our third ranked project, *OHIF Viewer* was not listed by the Domain Expert. Neither were the software packages that we ranked from fifth to eleventh (*ParaView*, *Weasis*, *medInria*, *BioImage Suite Web*, *OsiriX Lite*, *INVESALIUS*, and *Gwyddion*). The software mentioned by the Domain Expert that we did not rank were the six recommended packages that did not have visualization as the primary function (as discussed in Section 2). The differences between the list recommended by our methodology and the Domain Expert are not surprising. As mentioned above, our methodology weights all qualities equally, but that may not be the case for the Domain Expert's impressions. Moreover, although the Domain Expert has significant experience with MI software, they have not used all 29 packages

that were measured.

Although our ranking and the estimate of the community's ranking are not perfect measures, they do suggest a correlation between best practices and popularity. We do not know which comes first, the use of best practices or popularity, but we do know that the top ranked packages tend to incorporate best practices. The next sections will explore how the practices of the MI community compare to the broader research software community.

3.2 Comparison Between MI and Research Software for Artifacts

As part of filling in the measurement template (from Section 1.3.3), we summarized the artifacts observed in each MI package. Table 6 groups the artifacts by frequency into categories of common (20 to 29 (>67%) packages), uncommon (10 to 19 (33-67%) packages), and rare (1 to 9 (<33%) packages). Tables 2 and 4 show the details on which projects use which types of artifacts for documents related to maintainability and visibility, respectively.

Common	Uncommon	Rare
README (29)	Build scripts (18)	Getting Started (9)
Version control (29)	Tutorials (18)	Developer's manual (8)
License (28)	Installation guide (16)	Contributing (8)
Issue tracker (28)	Test cases (15)	API documentation (7)
User manual (22)	Authors (14)	Dependency list (7)
Release info. (22)	Frequently Asked Questions (FAQ) (14)	Troubleshooting guide (6)
	Acknowledgements (12)	Product roadmap (5)
	Changelog (12)	Design documentation (5)
	Citation (11)	Code style guide (3)
		Code of conduct (1)
		Requirements (1)

Table 6: Artifacts Present in MI Packages, Classified by Frequency (The number in brackets is the number of occurrences)

We answer RQ4 by comparing the artifacts that we observed in MI repositories to those observed and recommended for research software in general. Our comparison may point out areas where some MI software packages fall short of current best practices. This is not intended to be a criticism of any existing packages, especially since for practical reasons not every project has the resources to achieve the highest possible quality. However, rather than delve into the nuances of which software can justify compromising which practices, we will write our comparison under the ideal assumption that every project has sufficient resources to match best practices.

Table 7 (based on data from [92]) shows that MI artifacts generally match the recommendations found in nine current research software development guidelines:

- United States Geological Survey Software Planning Checklist [110],

- DLR (German Aerospace Centre) Software Engineering Guidelines [87],
- Scottish Covid-19 Response Consortium Software Checklist [13],
- Good Enough Practices in Scientific Computing [118],
- xSDK (Extreme-scale Scientific Software Development Kit) Community Package Policies [91],
- Trilinos Developers Guide [39],
- EURISE (European Research Infrastructure Software Engineers') Network Technical Reference [108],
- CLARIAH (Common Lab Research Infrastructure for the Arts and Humanities) Guidelines for Software Quality [113], and
- A Set of Common Software Quality Assurance Baseline Criteria for Research Projects [67].

In Table 7 each row corresponds to an artifact. For a given row, a checkmark in one of the columns means that the corresponding guideline recommends this artifact. The last column shows whether the artifact appears in the measured set of MI software, either not at all (blank), commonly (C), uncommonly (U) or rarely (R). We did our best to interpret the meaning of each artifact consistently between guidelines and specific MI software, but the terminology and the contents of artifacts are not standardized. The challenge even exists for the ubiquitous README file. The content of README files shows significant variation between projects [75]. Although some content is reasonably consistent, with 97% of README files contain at least one section describing the 'What' of the repository and 89% offering some 'How' content, other categories are more variable. For instance, information on 'Contribution', 'Why', and 'Who', appear in 28%, 26% and 53% of the analyzed files, respectively [75].

The frequency of checkmarks in Table 7 indicates the popularity of recommending a given artifact, but it does not imply that the most commonly recommended artifacts are the most important artifacts. Just because a guideline does not explicitly recommend an artifact, does not mean the guideline authors do not value it. They may have excluded it because it is out of the scope of their recommendations, or outside their experience. For instance, an artifact related to uninstall is only explicitly mentioned by one set of guidelines [113], but other guideline authors would likely see its value. They may simply feel that uninstall is implied by install, or they may have never asked themselves whether they need separate uninstall instructions.

	[110]	[87]	[13]	[118]	[91]	[39]	[108]	[113]	[67]	MI
LICENSE	✓	✓	✓	✓	✓		✓	✓	✓	C
README		✓	✓	✓	✓		✓	✓	✓	C
CONTRIBUTING		✓	✓	✓	✓		✓	✓	✓	R
CITATION				✓				✓	✓	U
CHANGELOG		✓		✓	✓		✓			U
INSTALL					✓		✓	✓	✓	U
Uninstall								✓		
Dependency List			✓		✓			✓		R
Authors							✓	✓	✓	U
Code of Conduct							✓			R
Acknowledgements							✓	✓	✓	U
Code Style Guide		✓					✓	✓	✓	R
Release Info.		✓				✓	✓			C
Prod. Roadmap						✓	✓	✓		R
Getting started					✓		✓	✓	✓	R
User manual			✓				✓			C
Tutorials							✓			U
FAQ							✓	✓	✓	U
Issue Track		✓	✓		✓	✓	✓		✓	C
Version Control		✓	✓	✓	✓	✓	✓	✓	✓	C
Build Scripts		✓		✓	✓	✓	✓		✓	U
Requirements		✓				✓			✓	R
Design Doc.		✓	✓		✓		✓	✓	✓	R
API Doc.					✓		✓	✓	✓	R
Test Plan		✓				✓				
Test Cases	✓	✓	✓		✓	✓	✓	✓	✓	U

Table 7: Comparison of Recommended Artifacts in Software Development Guidelines to Artifacts in MI Projects (C for Common, U for Uncommon and R for Rare)

Two of the items that appear in Table 6 do not appear in the software development guidelines shown in Table 7: Troubleshooting guide and Developer's manual. Although the guidelines do not name these two artifacts, the information contained within them overlaps with the recommended

artifacts. A Troubleshooting guideline contains information that would typically be found in a User manual. A Developer's guide overlaps with information from the README, INSTALL, Uninstall, Dependency List, Release Information, API documentation and Design documentation. In our current analysis, we have identified artifacts by the names given by the software guidelines and MI examples. In the future, a more in-depth analysis would look at the knowledge fragments that are captured in the artifacts, rather than focusing on the names of the files that collect these fragments together.

Although the MI community shows examples of 88% (23 of 26) of the practices we found in research software guidelines (Table 7), we did not observe three recommended artifacts: i) Uninstall, ii) Test plans, and iii) Requirements. Uninstall is likely an omission caused by the focus on installing software. Given the storage capacity of current hardware, developers and users are not generally concerned with uninstall. Moreover, as mentioned above, uninstall is not particularly emphasized in existing recommendations.

Test plans describe the scope, approach, resources, and the schedule of planned test activities [114]. The plan should cover details such as the makeup of the testing team, automated testing tools and technology to employ, the testing process, system tests, integration tests and unit tests. We did not observe test plans for MI software, but that doesn't mean plans weren't created; it means that the plans are not under version control. Test plans would have to at least be implicitly created, since we observed test cases with reasonable frequency for MI software (test cases are categorized as uncommon).

The other apparently neglected document is the requirements specification, which records the functionalities, expected performance, goals, context, design constraints, external interfaces and other quality attributes of the software [42]. If developers write requirements they typically are based on a template, which provide documentation structure, guidelines, and rules. Although there is no universally accepted template, examples exist [25, 42, 63, 78].

MI software is like other research software in its neglect of requirements documentation. Although requirements documentation is recommended by some [87, 39, 99], in practice research software developers often do not produce a proper requirements specification [37]. Sanders and Kelly [84] interviewed 16 scientists from 10 disciplines and found that none of the scientists created requirements specifications, unless regulations in their field mandated such a document. Requirements are the least commonly produced type of documentation for research software in general [65]. When looking at the pain points for research software developers, software requirements and management is the software engineering discipline that most hurts scientific developers, accounting for 23% of the technical problems [115]. The lack of support for requirements is likely due to the perception that up-front requirements are impossible for research software [17, 89], but if we drop the insistence on "up-front" requirements, allowing instead for the requirements to be written iteratively and incrementally, requirements are feasible [96]. Smith et al. provide a requirements template tailored to research software [100].

Table 7 shows several recommended artifacts that are rarely observed in practice. A theme among these rare artifacts is that, except for the user-focused getting started manual, they are developer-focused. The dependency list, which is a list of software library dependencies, was rarely observed, but this information is still likely present, just embedded in build scripts. The other developer-focused and rare artifacts are as follows:

- **A Contributing file** provides new contributors with the information that they need to start adding/modifying the repository's files. Abdalla [1] provides a simple template for creating an open-source contributor guideline.
- **A Developer Code of Conduct** explicitly states the expectations for developers on how they should treat one another [109]. The code outlines rules for communication and establishes enforcement mechanisms for violations. As Tourani et al. state, the developer code documents the spirit of a community so that anyone can comfortably contribute regardless of ethnicity, gender, or sexual orientation. Three popular codes of conduct are [109]: Contributor Covenant, Ubuntu Code of Conduct, and Django Code of Conduct. A code of conduct can improve inclusivity, which brings the benefit of a wider pool of contributors. For example, a code of conduct can improve the participation of women [90]. A standard of ethical behaviour can be captured in the code, for projects that are looking to abide by a code of ethics, such as the IEEE Code of Ethics [43], or the Professional Engineers of Ontario code of ethics [76, p. 23–24].
- **Code Style Guidelines** present standards for writing code. Style guides codify such elements as formatting, commenting, naming identifiers, best practices and dangers to avoid [16]. For instance, most coding style guides will specify using ALLCAPS when naming symbolic constants. Understandability improves under standardization, since developers spend more time on the content of the code, and less time distracted by its style. Three sample style guides are: Google Java Style Guide, PEP8 Style Guide for Python, and Google C++Style Guide. Linting tools, like flake8 for Python, can be used to enforced coding styles, like the PEP8 standard.
- **A Product Roadmap** explains the vision and direction for a product offering [62]. Although they have different forms, all roadmaps cover the following: i) Where are we now?, ii) Where are we going?, and iii) How can we get there? [72]. A product roadmap provides the following benefits: continuity of purpose, facilitation of collaboration, and assistance with prioritization [73]. Creating a roadmap involves the following steps: i) define and outline a strategic mission and product vision, ii) scan the environment, iii) revise and distill the product vision to write the product roadmap, and iv) estimate the product life cycle and evaluate the mix of planned development efforts [111].
- **Design Documentation** explicitly states the design goals and priorities, records the likely changes, decomposes the complex system into separate modules, and specifies the relationship between the modules. Design documentation shows the syntax of the interface for the modules, and in some cases also documents the semantics. Some potential elements of design documentation include the following:
  - Representation of the system design and class design using Unified Modelling Language class diagrams. This approach is suited to object-oriented design and designs that use patterns [28].
  - Rigorous documentation of the system design following the template for a Module Guide (MG) [71]. An MG organizes the modules in a hierarchy by their secrets.

  - An explanation of the design using data flow diagrams to show typical use cases for input transformation.
  - A table or graph showing the traceability between the requirements and the modules (or classes)
  - The syntax of the modules (or classes) by providing lists of the state variables, exported constants and all exported access programs for each module (or class). This shows the interface that can used to access each module's services.
  - A formal specification of the semantics of input/output relationships and state transitions for each module using a Module Interface Specification (MIS) [40]. An MIS is an abstract model that formally shows each module's access programs and the associated transitions and outputs based on their state, environment, and input variables. Previous work shows the example of an MIS for a mesh generator [23, 104].

- **API Documentation** shows developers the services or data provided by the software application (or library) through such resources as its methods or objects [59]. Understandability is improved by API documentation [59]. API documentation can be generated using tools like Doxygen, pydoc, and javadoc.

The rare artifacts for MI software are similar to the rare artifacts for Lattice Boltzmann solvers [60], except LBM software is more likely to have developer related artifacts, like Contributing, Dependency list, and Design documentation.

To improve MI software in the future, an increased use of checklists could help. Developers can use checklists to ensure they follow best practices. Some examples include checklists merging branches into master [14], checklists for saving and sharing changes to the project [118], checklists for new and departing team members [38], checklists for processes related to commits and releases [39] and checklists for overall software quality [108, 45].

The above discussion shows that, taken together, MI projects fall somewhat short of recommended best practices for research software. However, MI software is not alone in this. Many, if not most, research projects fall short of best practices. A gap exists in research software development practices and software engineering recommendations [105, 49, 68]. Johanson and Hasselbring observe that the state-of-the-practice for research software in industry and academia does not incorporate state-of-the-art SE tools and methods [48]. This causes sustainability and reliability problems [26]. Rather than benefit from capturing and reusing previous knowledge, projects waste time and energy "reinventing the wheel" [20].

3.3 Threats to Validity

Below we categorize and list the threats to validity that we have identified. Our categories come from an analysis of software engineering secondary studies by Ampatzoglou et al. [6], where a secondary study analyzes the data from a set of primary studies. Ampatzoglou et al. is appropriate because a common example of a secondary study is a systematic literature review. Our methodology is a systematic software review — the primary studies are the software packages, and our

work collects and analyzes these primary studies. We identified similar threats to validity in our assessment of the state of the practice of Lattice Boltzmann Solvers [93].

3.3.1 Reliability

A study is reliable if repetition of the study by different researchers using the original study's methodology would lead to the same results [81]. Reliability means that data and analysis are independent of the specific researcher(s) doing the study. For the current study the identified reliability related threats are as follows:

- One individual does the manual measures for all packages. A different evaluator might find different results, due to differences in abilities, experiences, and biases.
- The manual measurements for the full set of packages took several months. Over this time the software repositories may have changed and the reviewer's judgement may have drifted.

We reduced concern over the reliability risk associated with the reviewer's judgement by demonstrating that the measurement process is reasonably reproducible [101]. In previous work [101] we graded five software products by two reviewers. Their rankings were almost identical. As long as each grader uses consistent definitions, the relative comparisons in the AHP results will be consistent between graders.

3.3.2 Construct Validity

Runeson and Host [81] define construct validity as the adopted metrics representing what they are intended to measure. Our construct threats are often related to how we assume our measurements influences the various software qualities, as summarized in Section 1.3.3. Specifically, our construct validity related threats include the following:

- We make indirect measurement of software qualities since meaningful direct measures for qualities like maintainability, reusability and verifiability, are unavailable. We follow the usual assumption that developers achieve higher quality by following procedures and adhering to standards [114, p. 112].
- As mentioned in Section 2.1, we could not install or build *dwv*, *GATE*, and *DICOM Viewer*. We used a deployed on-line version for *dwv*, a VM version for *GATE*, but no alternative for *DICOM Viewer*. We might underestimate their rank due to these technical issues.
- Measuring software robustness only involved two pieces of data. This is likely part of the reason for limited variation in the robustness scores (Figure 7). We could add more robustness data by pushing the software to deal with more unexpected situations, like a broken Internet connection, but this would require a larger investment of measurement time.
- We may have inaccurately estimated maintainability by assuming a higher ratio of comments to source code improves maintainability. Moreover, we assumed that maintainability is improved if a high percentage of issues are closed, but a project may have a wealth of open issues, and still be maintainable.

- We assess reusability by the number of code files and LOC per file. This measure is indicative of modularity, but it does not necessarily mean a good modularization. The modules may not be general enough to be easily reused, or the formatting may be poor, or the understandability of the code may be low.
- The understandability measure relies on 10 random source code files, but the 10 files will not necessarily be representative.
- As discussed in Section 2.10, our overall AHP ranking makes the unrealistic assumption of equal weighting.
- We approximated popularity by stars and watches (Section 3.1), but this assumption may not be valid.
- In building Table 7 some judgement was necessary on our part, since not all guidelines use the same names for artifacts that contain essentially the same information.

3.3.3 Internal Validity

Internal validity means that discovered causal relations are trustworthy and cannot be explained by other factors [81]. In our methodology the internal validity threats include the following:

- In our search for software packages (Section 2), we may have missed a relevant package.
- Our methodology assumes that all relevant software development activities will leave a trace in the repositories, but this is not necessarily true. For instance, the possibility exists that CI usage was higher than what we observed through the artifacts (Section **??**). As another example, although we saw little evidence of requirements (Section **??**), maybe teams keep this kind of information outside their repos, possibly in journal papers or technical reports.

3.3.4 External Validity

If the results of a study can be generalized (applied) to other situations/cases, then the study is externally valid [81]. We are confident that our search was exhaustive. We do not believe that we missed any highly popular examples. Therefore, the bulk of our validity concerns are internal (Section 3.3.3). However, our hope is that the trends observed, and the lessons learned for MI software can be applied to other research software. With that in mind we identified the following threat to external validity:

- We cannot generalize our results if the development of MI software is fundamentally different from other research software.

Although there are differences, like the importance of data privacy for MI data, we found the approach to developing LBM software [93] and MI software to be similar. Except for the domain specific aspects, we believe that the trends observed in the current study are externally valid for other research software.

4 Conclusions

We analyzed the state of the practice for the MI domain with the goal of understanding current practice by answering our four research questions (Section 1.1). Our methods in Section 1.3 form a general process to evaluate domain-specific software, that we apply to the specific domain of MI software. We identified 48 MI software candidates, then, with the help of the Domain Expert selected 29 of them to our final list.

Section 2 lists our measurement results for ranking the 29 projects for nine software qualities. Our ranking results appear credible since they are mostly consistent with the ranking from the scientific community implied by the GitHub stars-per-year metric. As discussed in Section 3.1, four of the top five software projects appear in both our list and in the GitHub popularity list. Moreover, our top five packages appear among the first eight positions on the GitHub list. The noteworthy discrepancies between the two lists are for the packages that we were unable to install (*dwv* and *Dicom Viewer*).

Based on our grading scores *3D Slicer*, *ImageJ*, *Fiji* and *OHIF Viewer* are the top four software performers. However, the separation between the top performers and the others is not extreme. Almost all packages do well on at least a few qualities, as shown in Table 8, which summarizes the packages ranked first and second for each quality. Almost 70% (20 of 29) of the software packages appear in the top two for at least two qualities. The only packages that do not appear in Table 8, or only appear once, are *Papaya*, *MatrixUser*, *MRIcroGL*, *XMedCon*, *dicompyler*, *DicomBrowser*, *AMIDE*, *3DimViewer*, and *Drishti*. The shortness of this list suggests parity with respect to adoption of best practices for MI software overall.

When it comes to recommended software artifacts, the state of open source MI projects is healthy, with our surveyed examples showing 88% of the documentation artifacts recommended by research software development guidelines (Section 3.2). However, we did notice areas where practice seems to lag behind the research software development guidelines. For instance, the guidelines recommend three artifacts that were not observed: uninstall instructions, test plans, and requirements documentation. We observed the following recommended artifacts, but only rarely: contributing file, developer code of conduct, code style guidelines, product roadmap, design documentation, and API documentation (Section 3.2). Future MI projects may wish to consider incorporating more artifacts to potentially improve software quality.

Next steps. Mention threats to validity? Future work on pain points. Point to LBM methodology in that SOP paper?

Acknowledgements

We would like to thank Peter Michalski and Oluwaseun Owojaiye for fruitful discussions on topics relevant to this paper. We would also like to thank Jason Balaci for advice on web applications.

Conflict of Interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

Quality	Ranked 1st or 2nd
Installability	3D Slicer, BioImage Suite Web, Slice:Drop, INVESALIUS
Correctness and Verifiability	OHIF Viewer, 3D Slicer, ImageJ
Reliability	SMILI, ImageJ, Fiji, 3D Slicer, Slice:Drop, OHIF Viewer
Robustness	XMedCon, Weasis, SMILI, ParaView, OsiriX Lite, MicroView, medInria, ITK-SNAP, INVESALIUS, ImageJ, Horos, Gwyddion, Fiji, dicompyler, Dicom-Browser, BioImage Suite Web, AMIDE, 3DimViewer, 3D Slicer, OHIF Viewer, DICOM Viewer
Usability	3D Slicer, ImageJ, Fiji, OHIF Viewer, ParaView, INVESALIUS, Ginkgo CADx, SMILI, OsiriX Lite, BioImage Suite Web, ITK-SNAP, medInria, MicroView, Gwyddion
Maintainability	3D Slicer, Weasis, ImageJ, OHIF Viewer, ParaView
Reusability	3D Slicer, ImageJ, Fiji, OHIF Viewer, SMILI, dwv, BioImage Suite Web, GATE, ParaView
Understandability	3D Slicer, ImageJ, Weasis, Fiji, Horos, OsiriX Lite, dwv, Drishti, OHIF Viewer, GATE, ITK-SNAP, ParaView, INVESALIUS
Visibility and Transparency	ImageJ, 3D Slicer, Fiji
Overall Quality	3D Slicer, ImageJ

Table 8: Top performers for each quality (sorted by order of quality measurement)

References

[1] Safia Abdalla. A template for creating open source contributor guidelines. https://opensource.com/life/16/3/contributor-guidelines-template-and-tips, March 2016.

[2] U.S. Food & Drug Administration. Medical imaging. https://www.fda.gov/radiation-emitting-products/radiation-emitting-products-and-procedures/medical-imaging, 2021. [Online; accessed 25-July-2021].

[3] Aysel Afsar. Dicom viewer. https://github.com/ayselafsar/dicomviewer, 2021. [Online; accessed 27-May-2021].

[4] J. Ahrens, Berk Geveci, and Charles Law. Paraview: An end-user tool for large data visualization. *Visualization Handbook*, 01 2005.

[5] Paulo Amorim, Thiago Franco de Moraes, Helio Pedrini, and Jorge Silva. Invesalius: An interactive rendering framework for health care support. page 10, 12 2015.

[6] Apostolos Ampatzoglou, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, and Alexander Chatzigeorgiou. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. *Information and Software Technology*, 106, 02 2019.

[7] S. Angenent, Eric Pichon, and Allen Tannenbaum. Mathematical methods in medical image processing. *Bulletin (new series) of the American Mathematical Society*, 43:365–396, 07 2006.

[8] Kevin Archie and Daniel Marcus. Dicombrowser: Software for viewing and modifying dicom metadata. *Journal of digital imaging : the official journal of the Society for Computer Applications in Radiology*, 25:635–45, 02 2012.

[9] Medical Imaging Technology Association. About dicom: Overview. https://www.dicomstandard.org/about-home, 2021. [Online; accessed 11-August-2021].

[10] Isaac N. Bankman. Preface. In Isaac N. Bankman, editor, *Handbook of Medical Imaging*, Biomedical Engineering, pages xi – xii. Academic Press, San Diego, 2000.

[11] 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.

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

[13] Alys Brett, James Cook, Peter Fox, Ian Hinder, John Nonweiler, Richard Reeve, and Robert Turner. Scottish covid-19 response consortium. https://github.com/ScottishCovidResponse/modelling-software-checklist/blob/main/software-checklist.md, August 2021.

[14] Titus Brown. Notes from"how to grow a sustainable software development process (for scientific software)". http://ivory.idyll.org/blog/2015-growing-sustainable-software-development-process.html, June 2015.

[15] 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.

[16] David Carty. Follow google's lead with programming style guides. https://www.techtarget.com/searchsoftwarequality/feature/Follow-Googles-lead-with-programming-style-guides, July 2020.

[17] Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. Software development environments for scientific and engineering software: A series of case studies. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 550–559, Washington, DC, USA, 2007. IEEE Computer Society.

[18] Shekhar Chandra, Jason Dowling, Craig Engstrom, Ying Xia, Anthony Paproki, Ales Neubert, David Rivest-Hénault, Olivier Salvado, Stuart Crozier, and Jurgen Fripp. A lightweight rapid application development framework for biomedical image analysis. *Computer Methods and Programs in Biomedicine*, 164, 07 2018.

[19] Robert Choplin, J Boehme, and C Maynard. Picture archiving and communication systems: an overview. *Radiographics : a review publication of the Radiological Society of North America, Inc*, 12:127–9, 02 1992.

[20] Mario Rosado de Souza, Robert Haines, Markel Vigo, and Caroline Jay. What makes research software sustainable? an interview study with research software engineers. *CoRR*, abs/1903.06039, 2019.

[21] Ao Dong. Assessing the state of the practice for medical imaging software. Master's thesis, McMaster University, Hamilton, ON, Canada, September 2021.

[22] Ao Dong. Software quality grades for mi software. Mendeley Data, V1, doi: 10.17632/k3pcdvdzj2.1, August 2021.

[23] Ahmed H. ElSheikh, W. Spencer Smith, and Samir E. Chidiac. Semi-formal design of reliable mesh generation systems. *Advances in Engineering Software*, 35(12):827–841, 2004.

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

[25] ESA. ESA software engineering standards, PSS-05-0 issue 2. Technical report, European Space Agency, February 1991.

[26] S. Faulk, E. Loh, M. L. V. D. Vanter, S. Squires, and L. G. Votta. Scientific computing's productivity gridlock: How software engineering can help. *Computing in Science Engineering*, 11(6):30–39, Nov 2009.

[27] Pierre Fillard, Nicolas Toussaint, and Xavier Pennec. Medinria: Dt-mri processing and visualization software. 04 2012.

[28] E. Gamma, R. Helm, J. Vlissides, and I R Johnson. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[29] Marc-Oliver Gewaltig and Robert Cannon. Quality and sustainability of software tools in neuroscience. *Cornell University Library*, page 20 pp, May 2012.

[30] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

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

[32] 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.

[33] Daniel Haehn. Slice:drop: collaborative medical imaging in the browser. pages 1–1, 07 2013.

[34] Paul Hamill. *Unit test frameworks: Tools for high-quality software development.* O'Reilly Media, 2004.

[35] Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. How do scientists develop and use scientific software? In *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 1–8, 2009.

[36] 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].

[37] Dustin Heaton and Jeffrey C. Carver. Claims about the use of software engineering practices in science. *Inf. Softw. Technol.*, 67(C):207–219, November 2015.

[38] Michael A. Heroux and David E. Bernholdt. Better (small) scientific software teams, tutorial in Argonne training program on extreme-scale computing (ATPESC). https://press3.mcs.anl.gov//atpesc/files/2018/08/ATPESC_2018_Track-6_3_8-8_1030am_Bernholdt-Better_Scientific_Software_Teams.pdf, 2018.

[39] Michael A. Heroux, James M. Bieman, and Robert T. Heaphy. Trilinos developers guide part II: ASC softwar quality engineering practices version 2.0. https://faculty.csbsju.edu/mheroux/fall2012_csci330/TrilinosDevGuide2.pdf, April 2008.

[40] Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach.* International Thomson Computer Press, New York, NY, USA, 1995.

[41] horosproject.org. Horos. https://github.com/horosproject/horos, 2020. [Online; accessed 27-May-2021].

[42] IEEE. Recommended practice for software requirements specifications. *IEEE Std 830-1998*, pages 1–40, October 1998.

[43] Joint Task Force on Software Engineering Ethics IEEE-CS/ACM and Professional Practices. Code of ethics, IEEE computer society. https://www.computer.org/education/code-of-ethics, 1999.

[44] Parallax Innovations. Microview. https://github.com/parallaxinnovations/MicroView/, 2020. [Online; accessed 27-May-2021].

[45] Software Sustainability Institute. Online sustainability evaluation. https://www.software.ac.uk/resources/online-sustainability-evaluation, 2022.

[46] ISO/IEC. Systems and software engineering - systems and software quality requirements and evaluation (square) - system and software quality models. Standard, International Organization for Standardization, Mar 2011.

[47] Sama Jan, Giovanni Santin, Daniel Strul, S Staelens, K Assié, Damien Autret, Stéphane Avner, Remi Barbier, Manuel Bardiès, Peter Bloomfield, David Brasse, Vincent Breton, Peter Bruyndonckx, Irene Buvat, AF Chatziioannou, Yunsung Choi, YH Chung, Claude Comtat, Denise Donnarieix, and Christian Morel. Gate: a simulation toolkit for pet and spect. *Physics in medicine and biology*, 49:4543–61, 11 2004.

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

[49] Diane F. Kelly. A software chasm: Software engineering and scientific computing. *IEEE Software*, 24(6):120–119, 2007.

[50] Ron Kikinis, Steve Pieper, and Kirby Vosburgh. *3D Slicer: A Platform for Subject-Specific Image Analysis, Visualization, and Clinical Support*, volume 3, pages 277–289. 01 2014.

[51] Tae-Yun Kim, Jaebum Son, and Kwanggi Kim. The recent progress in quantitative medical image analysis for computer aided diagnosis systems. *Healthcare informatics research*, 17:143–9, 09 2011.

[52] Chris Rorden's Lab. Mricrogl. https://github.com/rordenlab/MRIcroGL, 2021. [Online; accessed 27-May-2021].

[53] Jörg Lenhard, Simon Harrer, and Guido Wirtz. Measuring the installability of service orchestrations using the square method. In *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*, pages 118–125. IEEE, 2013.

[54] Ajay Limaye. Drishti, a volume exploration and presentation tool. volume 8506, page 85060X, 10 2012.

[55] Fang Liu, Julia Velikina, Walter Block, Richard Kijowski, and Alexey Samsonov. Fast realistic mri simulations based on generalized multi-pool exchange tissue model. *IEEE Transactions on Medical Imaging*, PP:1–1, 10 2016.

[56] Andy Loening. Amide. https://sourceforge.net/p/amide/code/ci/default/tree/amide-current/, 2017. [Online; accessed 27-May-2021].

[57] Yves Martelli. dwv. https://github.com/ivmartel/dwv, 2021. [Online; accessed 27-May-2021].

[58] Matthew McCormick, Xiaoxiao Liu, Julien Jomier, Charles Marion, and Luis Ibanez. Itk: Enabling reproducible research and open science. *Frontiers in neuroinformatics*, 8:13, 02 2014.

[59] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. Application programming interface documentation: What do software developers want? *Journal of Technical Writing and Communication*, 48(3):295–330, 2018.

[60] Peter Michalski. State of the practice for lattice boltzmann method software. Master's thesis, McMaster University, Hamilton, Ontario, Canada, September 2021.

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

[62] Jürgen Münch, Stefan Trieflinger, and Dominic Lang. Product roadmap – from vision to reality: A systematic literature review. In *2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 1–8, 2019.

[63] NASA. Software requirements DID, SMAP-DID-P200-SW, release 4.3. Technical report, National Aeronautics and Space Agency, 1989.

[64] D Nevcas and P Klapetek. Gwyddion: an open-source software for spm data analysis. *Cent Eur J Phys*, 10, 01 2012.

[65] Luke Nguyen-Hoan, Shayne Flint, and Ramesh Sankaranarayana. A survey of scientific software development. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 12:1–12:10, New York, NY, USA, 2010. ACM.

[66] E Nolf, Tony Voet, Filip Jacobs, R Dierckx, and Ignace Lemahieu. (x)medcon * an opensource medical image conversion toolkit. *European Journal of Nuclear Medicine and Molecular Imaging*, 30:S246, 08 2003.

[67] Pablo Orviz, Álvaro López García, Doina Cristina Duma, Giacinto Donvito, Mario David, and Jorge Gomes. A set of common software quality assurance baseline criteria for research projects, 2017.

[68] Oluwaseun Owojaiye, W. Spencer Smith, Jacques Carette, Peter Michalski, and Ao Dong. State of sustainability for research software (poster). In *SIAM-CSE 2021 Conference on Computational Science and Engineering, Minisymposterium: Software Productivity and Sustainability for CSE*, March 2021.

[69] A. Panchal and R. Keyes. Su-gg-t-260: Dicompyler: An open source radiation therapy research platform with a plugin architecture. *Medical Physics - MED PHYS*, 37, 06 2010.

[70] Xenophon Papademetris, Marcel Jackowski, Nallakkandi Rajeevan, Robert Constable, and Lawrence Staib. Bioimage suite: An integrated medical image analysis suite. 1, 01 2005.

[71] D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.

[72] R. Phaal, C.J.P. Farrukh, and D.R. Probert. Developing a technology roadmapping system. In *A Unifying Discipline for Melting the Boundaries Technology Management:*, pages 99–111, 2005.

[73] Roman Pichler. Working with an agile product roadmap. https://www.romanpichler.com/blog/agile-product-roadmap/, May 2012.

[74] Prakash Prabhu, Thomas B. Jablin, Arun Raman, Yun Zhang, Jialu Huang, Hanjun Kim, Nick P. Johnson, Feng Liu, Soumyadeep Ghosh, Stephen Beard, Taewook Oh, Matthew Zoufaly, David Walker, and David I. August. A survey of the practice of computational science. SC '11, New York, NY, USA, 2011. Association for Computing Machinery.

[75] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. Categorizing the content of github readme files, 2018.

[76] Professional Engineers Act. Professional engineers act, rso 1990, c p. 28. https://canlii.ca/t/5568z, Dec 2021.

[77] UTHSCSA Research Imaging Institute. Papaya. https://github.com/rii-mango/Papaya, 2019. [Online; accessed 27-May-2021].

[78] Suzanne Robertson and James Robertson. *Mastering the Requirements Process*, chapter Volere Requirements Specification Template, pages 353–391. ACM Press/Addison-Wesley Publishing Co, New York, NY, USA, 1999.

[79] Nicolas Roduit. Weasis. https://github.com/nroduit/nroduit.github.io, 2021. [Online; accessed 27-May-2021].

[80] Curtis Rueden, Johannes Schindelin, Mark Hiner, Barry Dezonia, Alison Walter, and Kevin Eliceiri. Imagej2: Imagej for the next generation of scientific image data. *BMC Bioinformatics*, 18, 11 2017.

[81] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, Dec 2009.

[82] 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.

[83] 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].

[84] Rebecca Sanders and Diane Kelly. Dealing with risk in scientific software development. *IEEE Software*, 4:21–28, July/August 2008.

[85] Pixmeo SARL. Osirix lite. https://github.com/pixmeo/osirix, 2019. [Online; accessed 27-May-2021].

[86] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, Jean-Yves Tinevez, Daniel White, Volker Hartenstein, Kevin Eliceiri, Pavel Tomancak, and Albert Cardona. Fiji: An open-source platform for biological-image analysis. *Nature methods*, 9:676–82, 06 2012.

[87] Tobias Schlauch, Michael Meinel, and Carina Haupt. Dlr software engineering guidelines, August 2018.

[88] Will Schroeder, Bill Lorensen, and Ken Martin. *The visualization toolkit*. Kitware, 2006.

[89] Judith Segal and Chris Morris. Developing scientific software. *IEEE Software*, 25(4):18–20, July/August 2008.

[90] Vandana Singh, Brice Bongiovanni, and William Brandon. Codes of conduct in open source software—for warm and fuzzy feelings or equality in community? *Software Quality Journal*, 2021.

[91] Barry Smith, Roscoe Bartlett, and xSDK Developers. xsdk community package policies, Dec 2018.

[92] Spencer Smith and Peter Michalski. Digging deeper into the state of the practice for domain specific research software. In *Proceedings of the International Conference on Computational Science, ICCS*, pages 1–15, June 2022.

[93] Spencer Smith, Peter Michalski, Jacques Carette, and Zahra Keshavarz-Motamed. State of the practice for Lattice Boltzmann Method software. *Archives of Computational Methods in Engineering*, 31(1):313–350, Jan 2024.

[94] Spencer Smith, Yue Sun, and Jacques Carette. Statistical software for psychology: Comparing development practices between cran and other communities, 2018.

[95] Spencer Smith, Zheng Zeng, and Jacques Carette. Seismology software: state of the practice. *Journal of Seismology*, 22, 05 2018.

[96] W. Spencer Smith. A rational document driven design process for scientific computing software. In Jeffrey C. Carver, Neil Chue Hong, and George Thiruvathukal, editors, *Software Engineering for Science*, chapter Section I – Examples of the Application of Traditional Software Engineering Practices to Science, pages 33–63. Taylor & Francis, 2016.

[97] W. Spencer Smith, Jacques Carette, Peter Michalski, Ao Dong, and Oluwaseun Owojaiye. Methodology for assessing the state of the practice for domain X. https://arxiv.org/abs/2110.11575, October 2021.

[98] W. Spencer Smith, Ao Dong, Jacques Carette, and Michael D. Noseworthy. State of the practice for medical imaging software. https://arxiv.org/abs/2405.12171, May 2024.

[99] W. Spencer Smith and Nirmitha Koothoor. A document-driven method for certifying scientific computing software for use in nuclear safety analysis. *Nuclear Engineering and Technology*, 48(2):404–418, April 2016.

[100] W. Spencer Smith, Lei Lai, and Ridha Khedri. Requirements analysis for engineering computation: A systematic approach for improving software reliability. *Reliable Computing, Special Issue on Reliable Engineering Computation*, 13(1):83–107, February 2007.

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

[102] W. Spencer Smith, Adam Lazzarato, and Jacques Carette. State of the practice for gis software, 2018.

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

[104] W. Spencer Smith and Wen Yu. A document driven methodology for improving the quality of a parallel mesh generation toolbox. *Advances in Engineering Software*, 40(11):1155–1167, November 2009.

[105] Tim Storer. Bridging the chasm: A survey of software engineering practice in scientific programming. *ACM Comput. Surv.*, 50(4):47:1–47:32, August 2017.

[106] Keenan Szulik. Don't judge a project by its github stars alone. https://blog.tidelift.com/dont-judge-a-project-by-its-github-stars-alone, December 2017.

[107] TESCAN. 3dimviewer. https://bitbucket.org/3dimlab/3dimviewer/src/master/, 2020. [Online; accessed 27-May-2021].

[108] Carsten Thiel. EURISE network technical reference. https://technical-reference.readthedocs.io/en/latest/, 2020.

[109] Parastou Tourani, Bram Adams, and Alexander Serebrenik. Code of conduct in open source projects. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 24–33, 2017.

[110] USGS. USGS (united states geological survey) software plannning checklist. https://www.usgs.gov/media/files/usgs-software-planning-checklist, December 2019.

[111] Jarno Vähäniitty, Casper Lassenius, and Kristian Rautiainen. An approach to product roadmapping in small software product businesses. Helsinki, Finland, 2002. University of Technologie, Software Business and Engineering Institute.

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

[113] Maarten van Gompel, Jauco Noordzij, Reinier de Valk, and Andrea Scharnhorst. Guidelines for software quality, CLARIAH task force 54.100. https://github.com/CLARIAH/software-quality-guidelines/blob/master/softwareguidelines.pdf, September 2016.

[114] Hans van Vliet. *Software Engineering (2nd ed.): Principles and Practice*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[115] I. S. Wiese, I. Polato, and G. Pinto. Naming the pain in developing scientific software. *IEEE Software*, pages 1–1, 2019.

[116] Wikipedia contributors. Medical image computing — Wikipedia, the free encyclopedia, 2021. [Online; accessed 25-July-2021].

[117] Wikipedia contributors. Medical imaging — Wikipedia, the free encyclopedia, 2021. [Online; accessed 25-July-2021].

[118] Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K. Teal. Good enough practices in scientific computing. *CoRR*, abs/1609.00037, 2016.

[119] Gert Wollny. Ginkgo cadx. https://github.com/gerddie/ginkgocadx, 2020. [Online; accessed 27-May-2021].

[120] Paul A. Yushkevich, Joseph Piven, Heather Cody Hazlett, Rachel Gimpel Smith, Sean Ho, James C. Gee, and Guido Gerig. User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability. *Neuroimage*, 31(3):1116–1128, 2006.

[121] Xiaofeng Zhang, Nadine Smith, and Andrew Webb. 1 - medical imaging. In David Dagan Feng, editor, *Biomedical Information Technology*, Biomedical Engineering, pages 3–27. Academic Press, Burlington, 2008.

[122] Erik Ziegler, Trinity Urban, Danny Brown, James Petts, Steve D. Pieper, Rob Lewis, Chris Hafey, and Gordon J. Harris. Open health imaging foundation viewer: An extensible open-source framework for building web-based imaging applications to support cancer research. *JCO Clinical Cancer Informatics*, (4):336–345, 2020. PMID: 32324447.