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

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

[1*]Computing and Software Department, McMaster University, 1280 Main Street West, Hamilton, L8S 4K1, Ontario, Canada.
[2]Electrical and Computer Engineering Department, McMaster University, 1280 Main Street West, Hamilton, L8S 4K1, Ontario, Canada.

*Corresponding author(s). E-mail(s): smiths@mcmaster.ca;
Contributing authors: carette@mcmaster.ca; nosewor@mcmaster.ca;

Abstract

We present the state of the practice for Medical Imaging (MI) software based on data available in open source repositories. We selected 29 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. Using the Analytic Hierarchy Process (AHP) on the quantitative data, we ranked the MI software. The top five are *3D Slicer*, *ImageJ*, *Fiji*, *OHIF Viewer*, and *ParaView*. This is consistent with the community's view, with four of these also appearing in the top five using GitHub metrics (stars-per-year). In general, the quality and quantity of documentation present in a project correlates quite well with its popularity.

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

1 Introduction

We 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 critical information for diagnostic, analytic, and medical applications. Given its importance, we want to understand the merits and drawbacks of the current development processes, tools, and methodologies. We use a software engineering lens to assess the quality of existing MI software.

1.1 Research Questions

As well as state of the practice (SOP) for MI software, we would like to understand the impact of the often cited gap (chasm!) between recommended software engineering practices and that used for most research software [1]. Although scientists spend a substantial proportion of their working times on software development [2, 3], few are formally trained [2].

Our investigation is based on the following four research questions:

RQ1: What MI **open source** software projects exist? (Section 2)

RQ2: Based on quantitative measurements of each project's development practices, which projects follow best practices? (Section 2)

RQ3: How similar are the top projects identified in RQ2 to the most popular projects as viewed by the community? (Section 3.1)

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

1.2 Scope

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

1.3 Methodology

We have a standard set of questions designed to assess the qualities of any research software project [4, 5]. This has been applied to MI software and Lattice Boltzmann Solvers [6]. This builds off prior work to assess the state of the practice for such domains as Geographic Information Systems [7], Mesh Generators [8], Seismology software [9], and Statistical software for psychology [10]. We maintain the previous constraint that the work load for measuring a given domain should take around one person-month's worth of effort (20 working days at 8 person-hours per day).

We identify a list of potential packages (through online searches) which is then filtered and vetted by a domain expert. We aim for roughly 30 packages. For each remaining package, we measure its qualities by filling in a grading template [4]. This

data is used to rank the projects with the Analytic Hierarchy Process (AHP). We summarize further details on the interaction with the domain expert, software qualities, grading the software and AHP below and in longer form in Smith et al. (2024) [11].

1.3.1 Domain Expert

The Domain Expert vets the proposed list because online resources can be inaccurate. The expert also vets the AHP ranking. For the current assessment, our Domain Expert is [Details of our domain expert removed for double-blind].

In advance of the first meeting with the Domain Expert, they are asked to independently create a list of top software packages in the domain. This helps get the expert's knowledge refreshed in advance of the meeting.

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) [12]. Installability is defined as the effort required for the installation and/or uninstallation of software in a specified environment [13].

1.3.3 Grading

We use an existing template [4] that is designed to measure the aforementioned qualities. To stay within our given measurement time frame, each package gets up to five hours of time. Project developers can be contacted for help regarding installation, if necessary, but we impose a cap of about two hours on the installation process. Figure 1 shows an excerpt of the measurement spreadsheet. The rows are the measures and the columns correspond to the software packages. [The full data is available on Mendeley; link will be provided after refereeing.]

The full template consists of 108 questions over 9 qualities. These questions are designed to be unambiguous, quantifiable, and measurable with constrained time and domain knowledge.

The grader, after answering questions for each quality assigns an overall score (between 1 and 10) based on the answers. Several of the qualities use the word "surface" to highlight that these particular qualities are a shallow measure. For example, usability is not measured using user studies. Instead, we look for signs that the developers considered usability. We use two freeware tools to collect repository related data: GitStats and Sloc Cloc and Code (scc). Further details on quality measurement are provided in Smith et al. (2024) [11].

1.3.4 Analytic Hierarchy Process (AHP)

Developed by Saaty in the 1970s, AHP is widely used to analyze multiple criteria decisions [14]. AHP organizes multiple criteria in a hierarchical structure and uses

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						
...	...	...	...	...	...	...

Fig. 1 Grading template example

pairwise comparisons between alternatives to calculate relative ratios [15]. 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

We initially identified 48 candidate software projects from the literature [16–18], online articles [19–21], and forum discussions [22]. Then we filtered as follows:

1. Removed the packages with no source code available, such as *MicroDicom*, *Aliza*, and *jivex*.
2. Focused on MI software that provides visualization functions. We removed seven packages that were toolkits or libraries, such as *VTK*, *ITK*, and *dcm4che*, and another three that were for PACS.
3. Removed *Open Dicom Viewer* as it has not received any updates since 2011.

The Domain Expert provided a list 12 software packages. We found 6 packages were on both lists: *3D Slicer*, *Horos*, *ImageJ*, *Fiji*, *MRIcron* (we use its descendant *MRIcroGL*) and *Mango* (we use the web version *Papaya*). The remaining six packages were on our out-of-scope list. The Domain Expert agreed with our final choice of 29 packages – see Table 1. This contains summary data collected in the year 2020.

The projects are sorted in descending order of lines of code. 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 [removed for blind review]

Software	Rlsd	Updated	Fnd	NOC	LOC	OS			Web
						W	M	L	
ParaView [23]	2002	2020-10	✓	100	886326	✓	✓	✓	✓
Gwyddion [24]	2004	2020-11		38	643427	✓	✓	✓	
Horos [25]	?	2020-04		21	561617		✓		
OsiriX Lite [26]	2004	2019-11		9	544304		✓		
3D Slicer [27]	1998	2020-08	✓	100	501451	✓	✓	✓	
Drishti [28]	2012	2020-08		1	268168	✓	✓	✓	
Ginkgo CADx [29]	2010	2019-05		3	257144	✓	✓	✓	
GATE [30]	2011	2020-10		45	207122		✓	✓	
3DimViewer [31]	?	2020-03	✓	3	178065	✓	✓		
medInria [32]	2009	2020-11		21	148924	✓	✓	✓	
BioImage Suite Web [33]	2018	2020-10	✓	13	139699	✓	✓	✓	✓
Weasis [34]	2010	2020-08		8	123272	✓	✓	✓	
AMIDE [35]	2006	2017-01		4	102827	✓	✓	✓	
XMedCon [36]	2000	2020-08		2	96767	✓	✓	✓	
ITK-SNAP [37]	2006	2020-06	✓	13	88530	✓	✓	✓	
Papaya [38]	2012	2019-05		9	71831	✓	✓	✓	
OHIF Viewer [39]	2015	2020-10		76	63951	✓	✓	✓	✓
SMILI [40]	2014	2020-06		9	62626	✓	✓	✓	
INVESALIUS 3 [41]	2009	2020-09		10	48605	✓	✓	✓	
dwv [42]	2012	2020-09		22	47815	✓	✓	✓	✓
DICOM Viewer [43]	2018	2020-04	✓	5	30761	✓	✓	✓	
MicroView [44]	2015	2020-08		2	27470	✓	✓	✓	
MatrixUser [45]	2013	2018-07		1	23121	✓	✓	✓	
Slice:Drop [46]	2012	2020-04		3	19020	✓	✓	✓	✓
dicompyler [47]	2009	2020-01		2	15941	✓	✓		
Fiji [48]	2011	2020-08	✓	55	10833	✓	✓	✓	
ImageJ [49]	1997	2020-08	✓	18	9681	✓	✓	✓	
MRIcroGL [50]	2015	2020-08		2	8493	✓	✓	✓	
DicomBrowser [51]	2012	2020-08		3	5505	✓	✓	✓	

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

The programming languages used in order of decreasing popularity are C++, JavaScript, Java, C, Python, Pascal, Matlab. The most popular language is C++, for 11 of 29 projects; Pascal and Matlab were each used for a single project.

2.1 Installability

Figure 2 lists the installability scores. We found installation instructions for 16 projects, but two did not need them (*BioImage Suite Web* and *Slice:Drop*) as they are web applications. 10 of the projects required extra dependencies: Five depend on a specific browser; *dwv*, *OHIF Viewer*, and *GATE* needs extra libraries to build; *ImageJ* and *Fiji* need an unzip tool; *MatrixUser* needs Matlab; *DICOM Viewer* needs a Nextcloud platform.

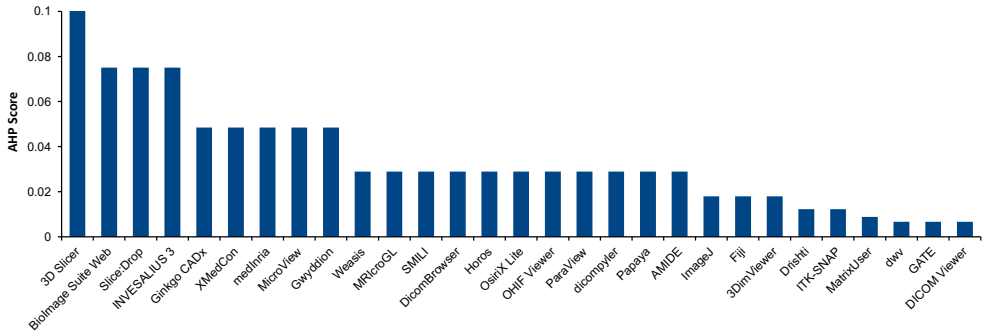

Fig. 2 AHP installability scores

The scores are based on the ease of following the installation instructions, and automated installation and uninstallation process. There were no issues for all but the bottom four, just various degrees of ease and automation. *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 virtual machine version for *GATE*. For *DICOM Viewer* we could not install the NextCloud dependency, and thus could not measure reliability nor robustness for it.

MatrixUser depends on Matlab, whose installation is not easy. For users who already have Matlab, this score should be higher.

2.2 Correctness & Verifiability

The packages with higher scores for correctness and verifiability (see Figure 3) used a wider array of techniques to improve correctness, and had better documentation to witness this. For instance, we looked for evidence of unit testing, and found evidence for only about half of the projects. We identified five projects using continuous integration tools: *3D Slicer*, *ImageJ*, *Fiji*, *dwv*, and *OHIF Viewer*.

Even for projects with well-organized documentation, requirements specifications and theory manuals were still missing. The only requirements-related document we found was a road map of *3D Slicer*, which contained design requirements for upcoming changes.

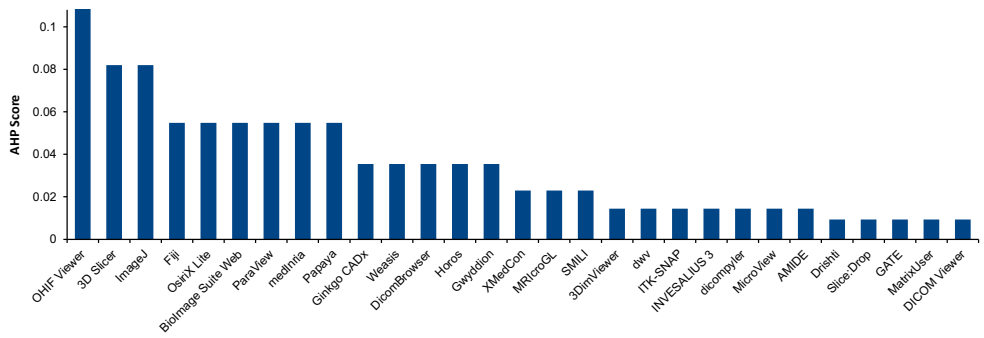

Fig. 3 AHP correctness & verifiability scores

2.3 Surface Reliability

Figure 4 shows our results. We were able to follow the steps in the tutorials that existed (seven packages had them.) 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.

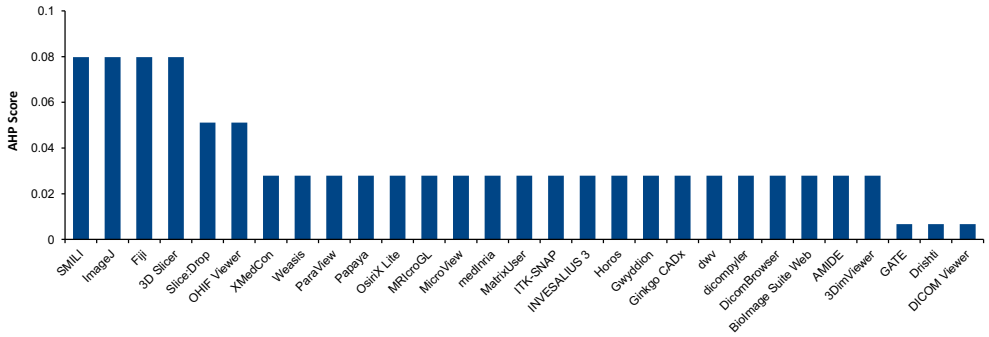

Fig. 4 AHP surface reliability scores

2.4 Surface Robustness

Figure 5 presents the scores for surface robustness. The packages with higher scores gracefully handled unexpected or unanticipated inputs, typically showing a clear error message. We may have underscored *OHIF Viewer*, since we needed further customization to load data.

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 relabelling 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*,

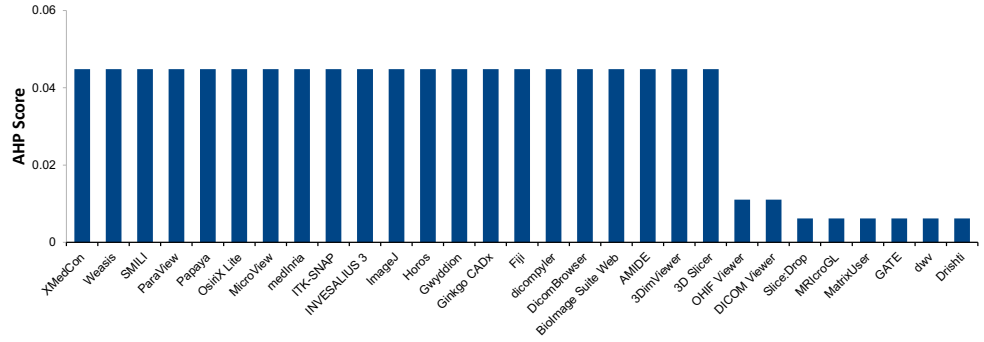

Fig. 5 AHP surface robustness scores

dwv, and *Slice:Drop* ignored the incorrect format, 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 6 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.

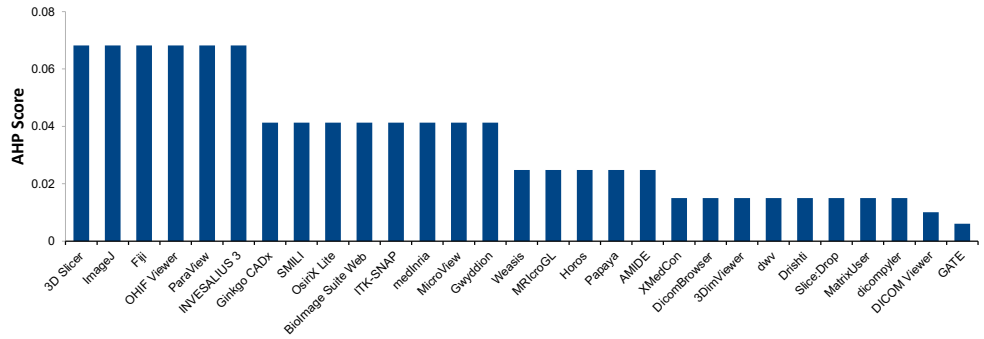

Fig. 6 AHP surface usability scores

2.6 Maintainability

Figure 7 shows the ranking results for maintainability. We gave *3D Slicer* the highest score because we found it had the most comprehensive artifacts. Only a few of the 29

projects had a product, developer's manual, or API (Application Programming Interface) documentation, and only *3D Slicer*, *ImageJ*, *Fiji* included all three documents (see Table 2 for the full data). Moreover, *3D Slicer* has a much higher percentage of closed issues (92%) compared to *ImageJ* (52%) and *Fiji* (64%).

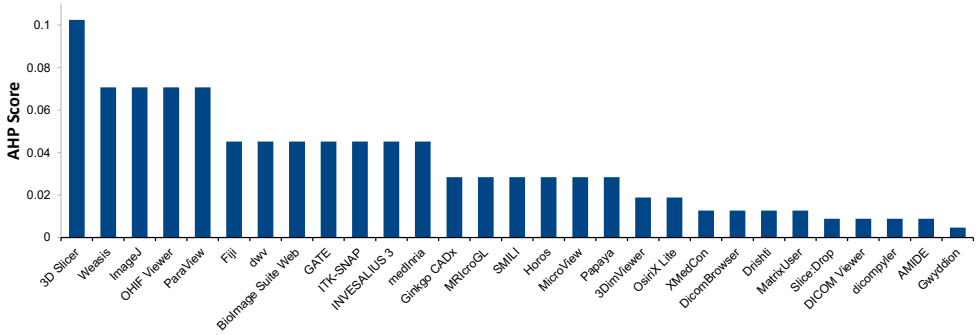

Fig. 7 AHP maintainability scores

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 for version control, with 24 of these using GitHub. *AMIDE* used Mercurial and *Gwyddion* used Subversion. *XMedCon*, *AMIDE*, and *Gwyddion* used SourceForge. *DicomBrowser* and *3DimViewer* used BitBucket.

2.7 Reusability

API documents (see Table 2) also help maintainability, which explains some of the scores shown in Figure 8. We have assumed that smaller code files are likely more reusable – see Table 3 for the details.

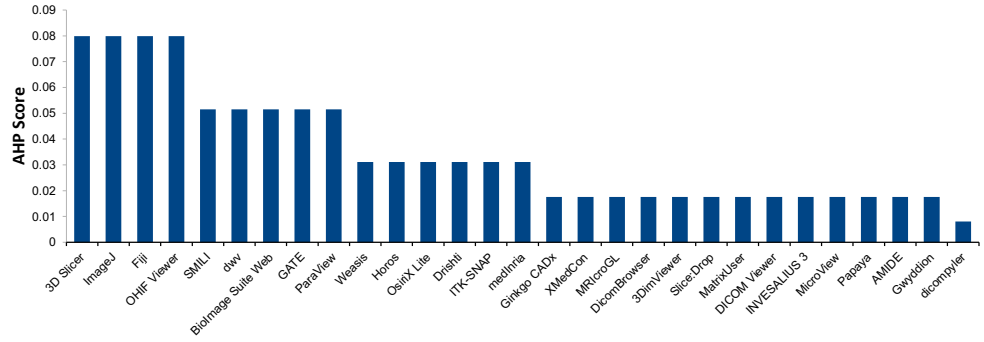

Fig. 8 AHP reusability scores

2.8 Surface Understandability

Figure 9 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*.

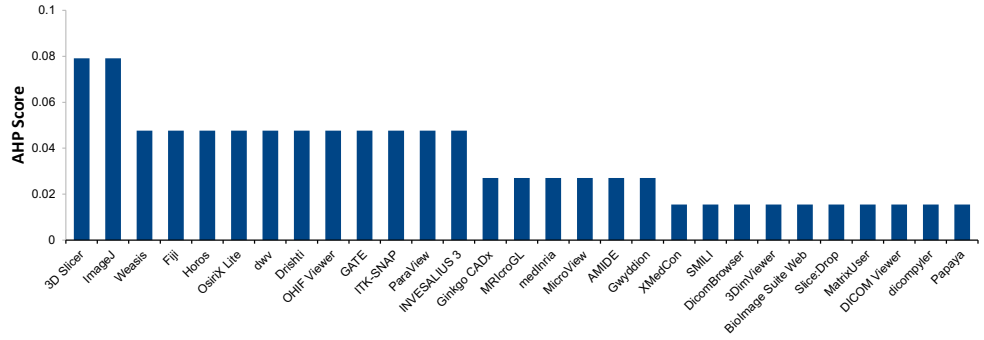

Fig. 9 AHP surface understandability scores

2.9 Visibility/Transparency

Figure 10 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.

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 (by reusability scores)

2.10 Overall Scores

In the absence of a specific real world context, we assumed all nine qualities are equally important. Figure 11 shows the overall scores in descending order.

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 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

We first compare our ranking to a (proxy for) the community's ranking. We then compare the state of the practice for MI with that of other research software. In particular we provide details on recommended artifacts that are rarely observed for MI software. Section 3.3 presents threats to the validity of our data and conclusions.

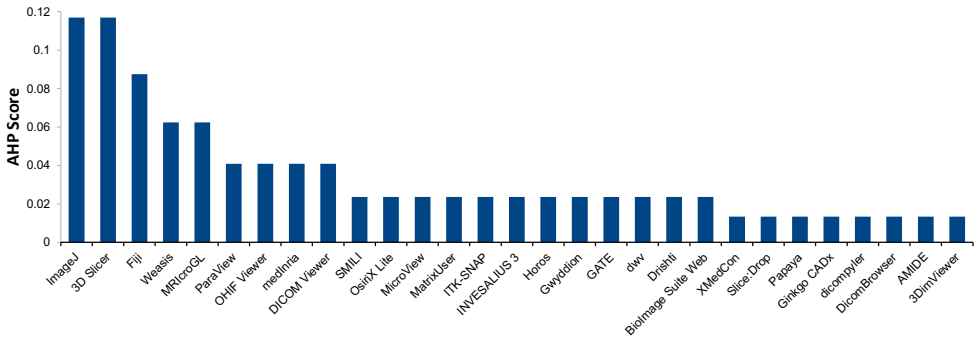

Fig. 10 AHP visibility/transparency scores

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)

3.1 Comparison to Community Ranking

We use GitHub stars, number of forks and number of people watching the projects are proxies for community ranking – see Table 5 for statistics collected in July 2021. Recall that 24 projects use GitHub. Our ranking and GitHub popularity, at least for the top five projects, seems to line up fairly well.

We ranked some popular packages fairly low, such as *dwv*. This is because we were unable to build it locally, even though we followed its installation instructions. However, we were able to use its web version for the rest of the measurements. Additionally, this version did not detect a broken DICOM file and instead displayed a blank image

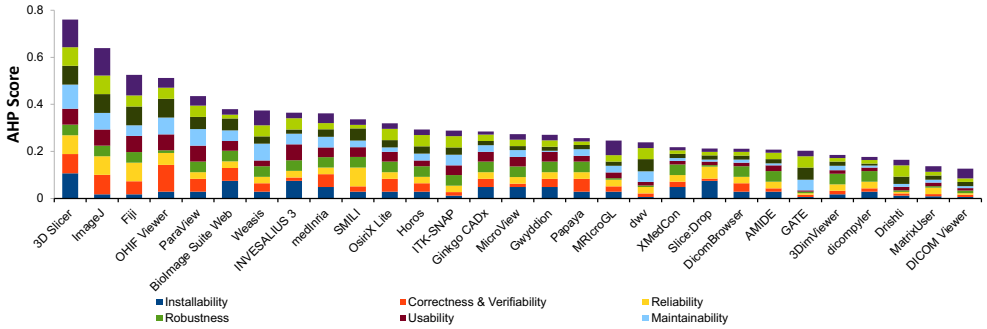

Fig. 11 Overall AHP scores with an equal weighting for all 9 software qualities

(Section 2.4). *DICOM Viewer* ranked low as we were unable to install the NextCloud platform.

Another likely reason for discrepancies is that we weighted all qualities equally. This is not likely how users implicitly rank the different qualities. This would require a broader user study to properly assess. Furthermore our measures of popularity are only *proxies* which are biased towards past rather than current preferences [52], as these are monotonically increasing quantities. Finally there are often more factors than just quality that influence the popularity of "consumer" products.

Although both rankings are imperfect measures, they nevertheless suggest a correlation between best practices and popularity. We don't know if this is causal, in either direction (i.e. if best practices enable popularity or if popularity increases the need for using more software development best practices).

3.2 Software Artifacts

We use nine research software development guidelines to compare recommended software artifacts versus those present in MI software. These guidelines are:

- United States Geological Survey Software Planning Checklist [53],
- DLR (German Aerospace Centre) Software Engineering Guidelines [54],
- Scottish Covid-19 Response Consortium Software Checklist [55],
- Good Enough Practices in Scientific Computing [56],
- xSDK (Extreme-scale Scientific Software Development Kit) Community Package Policies [57],
- Trilinos Developers Guide [58],
- EURISE (European Research Infrastructure Software Engineers') Network Technical Reference [59],
- CLARIAH (Common Lab Research Infrastructure for the Arts and Humanities) Guidelines for Software Quality [60], and
- A Set of Common Software Quality Assurance Baseline Criteria for Research Projects [61].

In Table 6 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,

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)

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 [62]. 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 [62].

Table 7 presents our measurements for MI software. The table 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.

Note that "popularity" in Table 6 does no imply that these oft recommended artifacts are the most important. Guidelines are often brief, to encourage adoption, and thus even guidelines that mention the need for installation instructions rarely mention uninstallation instructions. Two items in Table 7 do not appear in any guidelines: *Troubleshooting guide* and *Developer's manual*. However the information within

	[53]	[54]	[55]	[56]	[57]	[58]	[59]	[60]	[61]	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 6 Comparison of Recommended Artifacts in Software Development Guidelines to Artifacts in MI Projects (C for Common, U for Uncommon and R for Rare)

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

these documents overlaps with the recommended artifacts. Troubleshooting information often can be found in a User Manual, while the information in a "Developer's Manual" is often scattered amongst many other documents.

Three of the 26 recommended artifacts were never observed in the MI software: i) Uninstall, ii) Test plans, and iii) Requirements. It is possible that some of these were created but never put under version control.

Neglecting requirements documentation is unfortunately common for research software, and MI software is no exception to this trend. Although such documentation is recommended by some [54, 58, 63], in practice this is rare [64]. Sanders and Kelly [65] 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 [66].

This is unfortunate as when scientific developers are surveyed on their pain points, Wiese et al. [67] found that software requirements and management is the software engineering discipline that most hurts them, accounting for 23% of the technical problems reported by study participants. Further adding to the misfortune, there is a widespread perception that up-front requirements are impossible for research software [68, 69]. Fortunately a more agile approach to requirements is feasible [70], and research-software specific templates exist [71].

A theme emerges amongst the artifacts rarely observed in practice: they are developer-focused (a list of library dependencies, a contributor's guide, a developer Code of Conduct, coding style guidelines, product roadmap, design documentation and API documentation).

Other communities use checklists to help with best practices. Examples include checklists for merging branches [72], for saving and sharing changes to the project [56], for new and departing team members [73], for processes related to commits and releases [58] and for overall software quality [59, 74].

MI projects fall somewhat short of recommended best practices, but are not alone amongst research software projects. This gap has been documented before [1, 75], and is known to cause sustainability and reliability problems [76], and to waste development effort [77].

3.3 Threats to Validity

We follow Ampatzoglou et al. [78]'s analysis of threats to validity in software engineering secondary studies.

3.3.1 Reliability

A study is reliable if repeating it by another researcher, using the same methodology, would lead to the same results [79]. We identify the following threats:

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

The measurement process used has previously been shown to be reasonably reproducible. [80] reports grading 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

Construct validity is when the adopted metrics represent what they are intended to measure [79]. We have identified the following potential issues:

- 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 [81, p. 112].
- We could not install or build *dwv*, *GATE*, and *DICOM Viewer*. We used a deployed on-line version for *dwv*, a virtual machine version for *GATE*, but no alternative for *DICOM Viewer*.
- Robustness measurements involve only two pieces of data, leading to limited variation in the robustness scores (Figure 5). Our measurement-time budget limited what we could achieve here.
- Our maintainability proxies (higher ratio of comments to source, high percentage of closed issues) have not been validated.
- While smaller modules tend to be easier to reuse, small modules are not necessarily good modules, nor understandable modules.
- The understandability measure relies on 10 random source code files, but the 10 files will not necessarily be representative.
- Our overall AHP ranking makes the unrealistic assumption of equal weighting.
- We approximated popularity by stars and watches.
- Table 6 required judgement as 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 [79]. We identify the following:

- Our search (Section 2) could have missed a relevant package.
- Our methodology assumes that development activities will leave a trace in the repositories, but this is not necessarily true. For instance, we saw little evidence of requirements (Section 3.2), but teams might keep this kind of information outside their repos.

3.3.4 External Validity

If the results of a study can be generalized to other situations, then the study is externally valid [79]. In other words, 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 [6] and MI software to be similar. Except for the domain specific aspects, the trends observed in the current study are similar to that for other research software.

4 Conclusions

Our analysis of the state of the practice for MI domain along nine software qualities strongly indicates that "higher quality" is consistent with community ranking proxies. Although our quality measures are rather shallow, we see this as an advantage. The shallow measures are a proxy for the importance of *first impressions* for software adoption.

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, DicomBrowser, 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)

Our grading scores indicate that *3D Slicer*, *ImageJ*, *Fiji* and *OHIF Viewer* are the overall top four. 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*.

While we did find a reasonable amount of documentation, especially when consider all MI projects, there were definitely some holes. Some important documentation (test plans and requirements documentation) was missing, and other (contributors' guide,

code of conduct, code style guidelines, product roadmap, design documentation, and API documentation) (Section 3.2) were rare.

A deeper understanding of the needs of the MI community will require data beyond what is available in repositories. Future work should involve interviewing MI developers to better understand their "pain points".

Acknowledgements

We would like to thank Peter Michalski and Oluwaseun Owojaiye for fruitful discussions on topics relevant to this paper.

Conflict of Interest

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

References

[1] Storer, T.: Bridging the chasm: A survey of software engineering practice in scientific programming. ACM Comput. Surv. **50**(4), 47–14732 (2017) https://doi.org/10.1145/3084225

[2] Hannay, J.E., MacLeod, C., Singer, J., Langtangen, H.P., Pfahl, D., Wilson, G.: How do scientists develop and use scientific software? In: 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, pp. 1–8 (2009). https://doi.org/10.1109/SECSE.2009.5069155

[3] Prabhu, P., Jablin, T.B., Raman, A., Zhang, Y., Huang, J., Kim, H., Johnson, N.P., Liu, F., Ghosh, S., Beard, S., Oh, T., Zoufaly, M., Walker, D., August, D.I.: A survey of the practice of computational science. SC '11. Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/2063348.2063374 . https://doi.org/10.1145/2063348.2063374

[4] Smith, W.S., Carette, J., Michalski, P., Dong, A., Owojaiye, O.: Methodology for Assessing the State of the Practice for Domain X. https://arxiv.org/abs/2110.11575 (2021)

[5] Smith, S., Michalski, P.: Digging deeper into the state of the practice for domain specific research software. In: Proceedings of the International Conference on Computational Science, ICCS, pp. 1–15 (2022)

[6] Smith, S., Michalski, P., Carette, J., Keshavarz-Motamed, Z.: State of the practice for Lattice Boltzmann Method software. Archives of Computational Methods in Engineering **31**(1), 313–350 (2024) https://doi.org/10.1007/s11831-023-09981-2

[7] Smith, W.S., Lazzarato, A., Carette, J.: State of the Practice for GIS Software (2018)

[8] Smith, W.S., Lazzarato, D.A., Carette, J.: State of the practice for mesh generation and mesh processing software. Advances in Engineering Software **100**, 53–71 (2016)

[9] Smith, S., Zeng, Z., Carette, J.: Seismology software: state of the practice. Journal of Seismology **22** (2018) https://doi.org/10.1007/s10950-018-9731-3

[10] Smith, S., Sun, Y., Carette, J.: Statistical Software for Psychology: Comparing Development Practices Between CRAN and Other Communities (2018)

[11] Smith, W.S., Dong, A., Carette, J., Noseworthy, M.D.: State of the Practice for Medical Imaging Software. https://arxiv.org/abs/2405.12171 (2024)

[12] Ghezzi, C., Jazayeri, M., Mandrioli, D.: Fundamentals of Software Engineering, 2nd edn. Prentice Hall, Upper Saddle River, NJ, USA (2003)

[13] 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)

[14] Vaidya, O.S., Kumar, S.: Analytic hierarchy process: An overview of applications. European Journal of Operational Research **169**(1), 1–29 (2006) https://doi.org/10.1016/j.ejor.2004.04.028

[15] Saaty, T.L.: How to make a decision: The analytic hierarchy process. European Journal of Operational Research **48**(1), 9–26 (1990) https://doi.org/10.1016/0377-2217(90)90057-I . Desicion making by the analytic hierarchy process: Theory and applications

[16] Björn, K.: Evaluation of open source medical imaging software: A case study on health technology student learning experience. Procedia Computer Science **121**, 724–731 (2017) https://doi.org/10.1016/j.procs.2017.11.094

[17] Brühschwein, A., Klever, J., Hoffmann, A.-S., Huber, D., Kaufmann, E., Reese, S., Meyer-Lindenberg, A.: 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 (2019) https://doi.org/10.1007/s10278-019-00194-3

[18] Haak, D., Page, C.-E., Deserno, T.: A survey of dicom viewer software to integrate clinical research and medical imaging. Journal of digital imaging **29** (2015) https://doi.org/10.1007/s10278-015-9833-1

[19] Emms, S.: 16 Best Free Linux Medical Imaging Software. https://www.linuxlinks.com/medicalimaging/. [Online; accessed 02-February-2020] (2019)

[20] Hasan, M.: Top 25 Best Free Medical Imaging Software for Linux System. https://www.ubuntupit.com/top-25-best-free-medical-imaging-software-for-linux-system/. [Online; accessed 30-January-2020] (2020)

[21] Mu, H.: 20 Free & open source DICOM viewers for Windows. https://medevel.com/free-dicom-viewers-for-windows/. [Online; accessed 31-January-2020] (2019)

[22] Samala, R.: 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. [Online; accessed 31-January-2020] (2014)

[23] Ahrens, J., Geveci, B., Law, C.: Paraview: An end-user tool for large data visualization. Visualization Handbook (2005)

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

[25] horosproject.org: Horos. GitHub. [Online; accessed 27-May-2021] (2020)

[26] SARL, P.: OsiriX Lite. GitHub. [Online; accessed 27-May-2021] (2019)

[27] Kikinis, R., Pieper, S., Vosburgh, K.: 3D Slicer: A Platform for Subject-Specific Image Analysis, Visualization, and Clinical Support, vol. 3, pp. 277–289 (2014). https://doi.org/10.1007/978-1-4614-7657-3_19

[28] Limaye, A.: Drishti, a volume exploration and presentation tool, vol. 8506, p. 85060 (2012). https://doi.org/10.1117/12.935640

[29] Wollny, G.: Ginkgo CADx. GitHub. [Online; accessed 27-May-2021] (2020)

[30] Jan, S., Santin, G., Strul, D., Staelens, S., Assié, K., Autret, D., Avner, S., Barbier, R., Bardiès, M., Bloomfield, P., Brasse, D., Breton, V., Bruyndonckx, P., Buvat, I., Chatziioannou, A., Choi, Y., Chung, Y., Comtat, C., Donnarieix, D., Morel, C.: Gate: a simulation toolkit for pet and spect. Physics in medicine and biology **49**, 4543–61 (2004) https://doi.org/10.1088/0031-9155/49/19/007

[31] TESCAN: 3DimViewer. BitBucket. [Online; accessed 27-May-2021] (2020)

[32] Fillard, P., Toussaint, N., Pennec, X.: Medinria: Dt-mri processing and visualization software (2012)

[33] Papademetris, X., Jackowski, M., Rajeevan, N., Constable, R., Staib, L.: Bioimage suite: An integrated medical image analysis suite **1** (2005)

[34] Roduit, N.: Weasis. GitHub. [Online; accessed 27-May-2021] (2021)

[35] Loening, A.: AMIDE. Sourceforge. [Online; accessed 27-May-2021] (2017)

[36] Nolf, E., Voet, T., Jacobs, F., Dierckx, R., Lemahieu, I.: (x)medcon * an opensource medical image conversion toolkit. European Journal of Nuclear Medicine and Molecular Imaging **30**, 246 (2003) https://doi.org/10.1007/s00259-003-1284-0

[37] Yushkevich, P.A., Piven, J., Cody Hazlett, H., Gimpel Smith, R., Ho, S., Gee, J.C., Gerig, G.: User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability. Neuroimage **31**(3), 1116–1128 (2006)

[38] Research Imaging Institute, U.: Papaya. GitHub. [Online; accessed 27-May-2021] (2019)

[39] Ziegler, E., Urban, T., Brown, D., Petts, J., Pieper, S.D., Lewis, R., Hafey, C., Harris, G.J.: 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) https://doi.org/10.1200/CCI.19.00131 https://doi.org/10.1200/CCI.19.00131. PMID: 32324447

[40] Chandra, S., Dowling, J., Engstrom, C., Xia, Y., Paproki, A., Neubert, A., Rivest-Hénault, D., Salvado, O., Crozier, S., Fripp, J.: A lightweight rapid application development framework for biomedical image analysis. Computer Methods and Programs in Biomedicine **164** (2018) https://doi.org/10.1016/j.cmpb.2018.07.011

[41] Amorim, P., Moraes, T., Pedrini, H., Silva, J.: Invesalius: An interactive rendering framework for health care support, p. 10 (2015). https://doi.org/10.1007/978-3-319-27857-5_5

[42] Martelli, Y.: dwv. GitHub. [Online; accessed 27-May-2021] (2021)

[43] Afsar, A.: DICOM Viewer. GitHub. [Online; accessed 27-May-2021] (2021)

[44] Innovations, P.: Microview. GitHub. [Online; accessed 27-May-2021] (2020)

[45] Liu, F., Velikina, J., Block, W., Kijowski, R., Samsonov, A.: Fast realistic mri simulations based on generalized multi-pool exchange tissue model. IEEE Transactions on Medical Imaging **PP**, 1–1 (2016) https://doi.org/10.1109/TMI.2016.2620961

[46] Haehn, D.: Slice:drop: collaborative medical imaging in the browser, pp. 1–1 (2013). https://doi.org/10.1145/2503541.2503645

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

37 (2010) https://doi.org/10.1118/1.3468652

[48] Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B., Tinevez, J.-Y., White, D., Hartenstein, V., Eliceiri, K., Tomancak, P., Cardona, A.: Fiji: An open-source platform for biological-image analysis. Nature methods **9**, 676–82 (2012) https://doi.org/10.1038/nmeth.2019

[49] Rueden, C., Schindelin, J., Hiner, M., Dezonia, B., Walter, A., Eliceiri, K.: Imagej2: Imagej for the next generation of scientific image data. BMC Bioinformatics **18** (2017) https://doi.org/10.1186/s12859-017-1934-z

[50] Lab, C.R.: MRIcroGL. GitHub. [Online; accessed 27-May-2021] (2021)

[51] Archie, K., Marcus, D.: 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 (2012) https://doi.org/10.1007/s10278-012-9462-x

[52] Szulik, K.: Don't judge a project by its GitHub stars alone. https://blog.tidelift.com/dont-judge-a-project-by-its-github-stars-alone (2017)

[53] USGS: USGS (United States Geological Survey) Software Plannning Checklist. https://www.usgs.gov/media/files/usgs-software-planning-checklist (2019)

[54] Schlauch, T., Meinel, M., Haupt, C.: DLR Software Engineering Guidelines. Zenodo (2018). https://doi.org/10.5281/zenodo.1344612 . https://doi.org/10.5281/zenodo.1344612

[55] Brett, A., Cook, J., Fox, P., Hinder, I., Nonweiler, J., Reeve, R., Turner, R.: Scottish Covid-19 Response Consortium. https://github.com/ScottishCovidResponse/modelling-software-checklist/blob/main/software-checklist.md (2021)

[56] Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., Teal, T.K.: Good enough practices in scientific computing. CoRR **abs/1609.00037** (2016)

[57] Smith, B., Bartlett, R., Developers, x.: xSDK Community Package Policies. figshare (2018). https://doi.org/10.6084/m9.figshare.4495136.v6 . https://figshare.com/articles/journal_contribution/xSDK_Community_Package_Policies/4495136/6

[58] Heroux, M.A., Bieman, J.M., Heaphy, R.T.: Trilinos Developers Guide Part II: ASC Softwar Quality Engineering Practices Version 2.0. https://faculty.csbsju.edu/mheroux/fall2012_csci330/TrilinosDevGuide2.pdf (2008)

[59] Thiel, C.: EURISE Network Technical Reference. https://technical-reference.

readthedocs.io/en/latest/ (2020)

[60] Gompel, M., Noordzij, J., Valk, R., Scharnhorst, A.: Guidelines for Software Quality, CLARIAH Task Force 54.100. https://github.com/CLARIAH/software-quality-guidelines/blob/master/softwareguidelines.pdf (2016)

[61] Orviz, P., García, Á.L., Duma, D.C., Donvito, G., David, M., Gomes, J.: A set of common software quality assurance baseline criteria for research projects. https://digital.csic.es/handle/10261/160086 (2017). https://doi.org/10.20350/digitalCSIC/12543

[62] Prana, G.A.A., Treude, C., Thung, F., Atapattu, T., Lo, D.: Categorizing the Content of GitHub README Files (2018)

[63] Smith, W.S., Koothoor, N.: A document-driven method for certifying scientific computing software for use in nuclear safety analysis. Nuclear Engineering and Technology **48**(2), 404–418 (2016) https://doi.org/10.1016/j.net.2015.11.008

[64] Heaton, D., Carver, J.C.: Claims about the use of software engineering practices in science. Inf. Softw. Technol. **67**(C), 207–219 (2015) https://doi.org/10.1016/j.infsof.2015.07.011

[65] Sanders, R., Kelly, D.: Dealing with risk in scientific software development. IEEE Software **4**, 21–28 (2008)

[66] Nguyen-Hoan, L., Flint, S., Sankaranarayana, R.: A survey of scientific software development. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM '10, pp. 12–11210. ACM, New York, NY, USA (2010). https://doi.org/10.1145/1852786.1852802 . http://doi.acm.org/10.1145/1852786.1852802

[67] Wiese, I.S., Polato, I., Pinto, G.: Naming the pain in developing scientific software. IEEE Software, 1–1 (2019) https://doi.org/10.1109/MS.2019.2899838

[68] Carver, J.C., Kendall, R.P., Squires, S.E., Post, D.E.: 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, pp. 550–559. IEEE Computer Society, Washington, DC, USA (2007). https://doi.org/10.1109/ICSE.2007.77

[69] Segal, J., Morris, C.: Developing scientific software. IEEE Software **25**(4), 18–20 (2008)

[70] Smith, W.S.: A rational document driven design process for scientific computing software. In: Carver, J.C., Hong, N.C., Thiruvathukal, G. (eds.) Software Engineering for Science, pp. 33–63. Taylor & Francis, Oxfordshire (2016). Chap. Section I – Examples of the Application of Traditional Software Engineering

[71] Smith, W.S., Lai, L., Khedri, R.: Requirements analysis for engineering computation: A systematic approach for improving software reliability. Reliable Computing, Special Issue on Reliable Engineering Computation **13**(1), 83–107 (2007) https://doi.org/10.1007/s11155-006-9020-7

[72] Brown, T.: 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 (2015)

[73] Heroux, M.A., Bernholdt, D.E.: 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). https://doi.org/https://figshare.com/articles/journal_contribution/ATPESC_Software_Productivity_03_Better_Small_Scientific_Software_Teams/6941438

[74] Institute, S.S.: Online sustainability evaluation. https://www.software.ac.uk/resources/online-sustainability-evaluation (2022)

[75] Johanson, A.N., Hasselbring, W.: Software engineering for computational science: Past, present, future. Computing in Science & Engineering **Accepted**, 1–31 (2018)

[76] Faulk, S., Loh, E., Vanter, M.L.V.D., Squires, S., Votta, L.G.: Scientific computing's productivity gridlock: How software engineering can help. Computing in Science Engineering **11**(6), 30–39 (2009) https://doi.org/10.1109/MCSE.2009.205

[77] Souza, M.R., Haines, R., Vigo, M., Jay, C.: What makes research software sustainable? an interview study with research software engineers. CoRR **abs/1903.06039** (2019) arXiv:1903.06039

[78] Ampatzoglou, A., Bibi, S., Avgeriou, P., Verbeek, M., Chatzigeorgiou, A.: Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. Information and Software Technology **106** (2019) https://doi.org/10.1016/j.infsof.2018.10.006

[79] Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14**(2), 131–164 (2009) https://doi.org/10.1007/s10664-008-9102-8

[80] Smith, W.S., Lazzarato, A., Carette, J.: State of practice for mesh generation software. Advances in Engineering Software **100**, 53–71 (2016)

[81] Vliet, H.: Software Engineering (2nd Ed.): Principles and Practice. John Wiley

& Sons, Inc., New York, NY, USA (2000)