Department of Computing and Software

Faculty of Engineering — McMaster University

State of the Practice for Developing Oceanographic Software

by

S. Smith, Y. Sun and J. Carette

C.A.S. Report Series **CAS-15-02-SS**

Department of Computing and Software **January 2015**
Information Technology Building
McMaster University
1280 Main Street West Hamilton, Ontario, Canada L8S 4K1

State of the Practice for Developing Oceanographic Software

Spencer Smith*, Yue Sun and Jacques Carette

Technical Report CAS-15-02-SS
Department of Computing and Software
McMaster University

January 24, 2015

Abstract

Developing high quality scientific computing software for oceanography is challenging. Our study looks to quantify the success that has been achieved in this domain with the goal of identifying areas for improvement. We compared 30 scientific computing software tools. The tools are subdivided into the categories of R packages, Matlab toolboxes and oceanographic model frameworks. Our analysis showed that R packages and oceanographic model frameworks have advantages in qualities related to development; Matlab toolboxes have the best installability; Matlab toolboxes and R packages did well in showing correctness and verifiability. The quality of the oceanographic frameworks suffered because of poor installability. For quality improvement, recommendations include reducing the manual steps for installation of oceanographic model frameworks, adding a specific section with examples and their expected output to the documentation template for R packages, consolidating existing projects into fewer software products, adopting the process and tools used by other successful scientific communities and reaching out to the broader open source community for assistance. For documentation, we suggested a software requirements specification, a design document, a programmer's manual, submission guidelines, a coding standard and literate programming. For tools, we suggested the use of version control software, GitHub and **doxygen**.

Keywords: Software quality, R, Matlab, Oceanographic model framework

*Computing and Software Department, McMaster University, smiths@mcmaster.ca

1 Introduction

Scientists often develop their own software because of the need for domain specific knowledge during development (Wilson et al., 2013). However, many of these scientists are self-taught and lack a complete knowledge of software development best practices (Kelly, 2013). To help these scientists, Wilson et al. (2013) gives a list of general advice. Although the advice is certainly useful, it is based on the authors's personal experiences within their own domains. The advice might not apply equally to software development in other specific scientific communities.

A first look at the state of practice in a specific scientific community is provided by Gewaltig and Cannon (2014). The work of Gewaltig and Cannon focuses on the computational neuroscience domain. Gewaltig and Cannon (2014) provide a high level comparison of existing neuroscience software packages, but little data is given on the specific metrics for the comparison. By building on the idea of studying the software in specific scientific communities, while incorporating detailed measurements, we hope to identify the current state of practice in the oceanography communities.

We selected to study oceanography software because of the large amount of sophisticated numerical software developed for this field. Moreover, we hope to contribute in a small way to this critically important research community. We recognize the central importance of oceanography software for a scientific understanding of global warming, extreme weather events, utilization of marine food resources and the history and future of our planet. We did a similar study of psychometrics software, as summarized in Smith et al. (2015).

In our project, we are targeting 30 scientific computing software packages, including software developed within different local communities and education institutions. The software options were selected objectively from the combination of four external authoritative lists: Tech.plym.ac.uk (2014) and Woodshole.er.usgs.gov (2014), which summarize software toolboxes written in Matlab; CRAN (2014b), which presents 4 R packages, and Paul G. Myers, who provided 16 oceanographic model frameworks (personal communication, November 2013).

Combining our own ideas and suggestions from Wilson et al. (2013) and Gewaltig and Cannon (2014), we made a software quality grading sheet, which allowed us to measure each software package with respect to certain software qualities. In this paper, we use the software engineering term *software qualities*, to refer to properties of software such as installability, reliability, maintainability, portability and so on (see Section 2.2 for more details). When we speak of "software qualities," we mean the union of these properties. After grading each software package, we used AHP (Saaty, 1994) to quantify the ranking between packages via pair-wise comparisons.

The software packages are only analyzed with respect to software engineering qualities. We will not make comparisons based on functionality. We have made this choice since we do not have the domain expertise to judge the functionality of the software. Fortunately, our lack of domain knowledge is an advantage, since it means we can be objective. We do not have any initial biases; we can and reach our conclusions through a systematic and objective grading process.

The details of our method and the selected software packages are presented in Sections 2 and 3, respectively. Information on how the R packages, Matlab toolboxes and Oceanographic model frameworks compare to each other is given in Section 4. Recommendations and conclusions can be found in Sections 5 and 6.

2 Method

This section begins with an overview of the overall process we used to select, measure and compare oceanographic software. This is followed by definitions of the terminology used to construct our software quality grading sheet, including the definitions of the relevant software qualities. This section ends with background information on the Analytic Hierarchy Process (AHP), since it forms an important part of our ranking process.

2.1 Overall Process

The process used for the current study is that same as that used in Smith et al. (2015). The steps are as follows:

1. Choose the scientific computing related domain. Here we chose oceanography as our domain, since software development is active in this community, as evidenced by the software lists referenced in the previous section. Moreover, we can benefit by having access to the code, since the majority of oceanography software is open source and non-commercial.

2. Select 30 software packages from the authoritative lists. For the oceanographic domain this is the combination of four lists mentioned in Section 1. We selected 4 R packages hosted by CRAN, 10 Matlab toolboxes and another 16 oceanographic model frameworks. Within each category the software packages were selected randomly.

3. Create a software quality grading template based on software engineering qualities (see Section 2.2) . We employed the same template as used by Smith et al. (2015).

4. Grade each of the selected software package using the software quality grading sheet. We did an experiment with different people grading the same software, to see if the rankings are reproducible. The result showed that people's standards of grading may vary, but the overall ranking of software remained stable, due to the use of relative comparisons, rather than absolute measures.

5. Apply AHP (see Section 2.4) on the grading sheet. As was explained under the previous bullet, people may have different standards about grades; therefore, AHP was employed to smooth out the differences through its emphasis on relative comparison.

6. Analyze the AHP results for different weights between the qualities, to account for different quality priority profiles.

2.2 Software qualities

The definitions of software qualities come from Ghezzi et al. (2002). The qualities are presented in the order of a typical measurement exercise. Information on how the quality will be measured in the current study is given, where relevant.

- *Installability* is a measure of the ease of installing the software. This quality is measured by the amount and quality of information provided by developers. High installability implies detailed and well organized installation instructions, simplicity for the user and an automated installation process.

- *Correctness and Verifiability* measure the amount of confidence a user can place in the behaviour of the software. Software that reuses trustworthy libraries builds more confidence than when the software uses self-developed libraries. Judging correctness requires a complete, consistent and unambiguous specification of the requirements. A well explained example (with input, expected output and instructions) is helpful too, so that users can verify that the software produces the expected results on their own machine.

- *Reliability* measures how dependable the software is. Reliable software has a high probability of meeting its stated requirement under a given usage profile over a given span of time. Given that we are not domain experts, in our study we can only do a surface measure of reliability.

- *Robustness* considers whether the software can gracefully recover from unexpected input.

- *Performance* is a measure of the time and storage resources required by the software. Since we are not domain experts, we did not measure performance directly. In stead, we looked for whether there was evidence that performance was considered. Potential evidence includes a makefile (or other source file) that shows evidence of employing a performance profiler.

- *Usability* is a measure of the ease of using the software. This quality is related to the user interface and to the quality of the training material provided to users. With respect to the user interface, we look to see whether the interface has a consistent look and feel for the platform it is on and whether the principle of visibility (Herr, 2002) has been respected. Visibility means that the user can easily find the interface elements that provide the functionality that they need. With respect to training material, we look for documentation to include a user manual, a getting started tutorial, and standard examples (with input and expected output). Usability is also improved if the software has a good user support model (e.g. forum).

- *Maintainability* is a measure of the ease with which the software can be corrected or updated. Maintainability benefits future developers, as opposed to current users. To

measure this quality, we look to for the following: explicit tracking of version history, change logs, a developer's guide, issue tracking tools and a concurrent versioning system.

- *Reusability* is a measure of the ease of reusing code in another software packages. We consider a software package to have good reusability when part of the software is used by another package and when the API (Application Program Interface) is documented.

- *Portability* is the ability of software to run on different platforms. For each package, we determine the intended portability from the developer's statements and measure the portability by running the software on the supported platforms.

- *Understandability (of the code)* measures how quickly and completely a developer can grasp the implementation of the code. With the time available to measure this quality, we were only able to do a surface measurement. This was accomplished by looking at whether the code has consistent indentation and formatting style, whether constants are hard coded, whether the code is modularized, etc. Understandability is assumed to be improved by providing a coding standard, or design document.

- *Interoperability* is defined as whether a software package is designed to work with other software or external systems. This quality was measured by checking whether an external system exists and whether an external API document is provided.

- *Visibility/Transparency* is a measure of the ease of determining the status of the development of a software package. This quality is measured by looking for documentation on the development process. Another metric is the examiner's subjective feeling about the ease of accessing information about the software package.

- *Reproducibility* measures whether sufficient information is available that another party could independently verify the software's results (Davison, 2012). To follow the scientific method, and bring confidence in the science, oceanography software should be reproducible. For reproducibility, documentation of the process of verification and validation is required, including details of the development and testing environment, operating system and version number. If possible, test data and automated tools for capturing the experimental context should also be provided.

Using the above qualities, we built a grading sheet, as shown in Appendix A and at the following url: `https://github.com/adamlazz/DomainX`. Each selected software package was marked using this sheet. As an example, Table 1 shows how the quality of installability was measured.

2.3 Software classification

We captured the status of each software project using the following definitions:

Question (Allowed responses)
Are there installation instructions? (Yes/No)
Are the installation instructions linear? (Yes/No)
Is there something in place to automate the installation? (Yes*/No)
Is there a specified way to validate the installation, such as a test suite? (Yes*/No)
How many steps were involved in the installation? (Number)
How many software packages need to be installed before or during installation? (Number)
Run uninstall, if available. Were any obvious problems caused? (Unavail/Yes*/No)

Table 1: Installability metrics, where Unavail means uninstall is not available and the superscript * means that this response should be accompanied by explanatory text.

- Alive: Software package, related documentation or website has been updated within the last 18 month.
- Dead: The last update of the above information was 18 month ago or longer.
- Unclear: No clear information related to the date is given.

The categories of our selected software are given by Gewaltig and Cannon (2012). This paper is an older version of Gewaltig and Cannon (2014). We have chosen to use this earlier classification system because it is simpler and the extra category from Gewaltig and Cannon (2014) does not occur in our selected software.

- Public: Software that is ready for use by the public.
- Private: Software that is designed for a specific group of people.
- Concept: Software that is simply aimed at demonstrating algorithms or concepts.

2.4 Analytic Hierarchy Process

AHP gives us a technique for prioritizing qualities and comparing between software packages, even though the methods of measuring each quality are very different. There are no common units that allow for the measurement of one quality, say maintainability and comparing it to another, such as portability. However, the central idea of AHP is that for a pairwise comparison of criteria or alternatives, it is possible to determine their relative weighting. Pairwise comparisons allow decision makers to focus on only two qualities at a time. Rather than use their judgement for a complete ranking of the qualities, they only need to compare two at a time and let the mathematics behind AHP produce the final priority ranking. Similarly when ranking the software packages for a given quality,

it is only necessary to compare the relative success between two packages at a time. An overview of AHP can be found in Triantaphyllou (1995).

By using AHP, people can evaluate n options (the 30 software packages in our case) with respect to m criteria (the 13 software qualities in our case). The criteria are prioritized based on the relative comparisons between their importance to form an $m \times m$ decision matrix. There is a row for each criterion. The columns follow the same ordering with respect to their correspondance to the criteria. The next step is to build, for each of the criterion, a pairwise analysis between the options. This forms, for each criterion, an $n \times n$ matrix a. Following the same pattern as the decision matrix, the rows and columns of a correspond to each option. Each entry in a shows the relative success, in achieving the given criterion, of the option corresponding to the row versus the option corresponding to the column. Therefore, the diagonal entries for a, and the decision matrix, are all equal to 1. The entries in the lower triangle of the matrix are usually the reciprocals of the corresponding entries in the upper triangle. A scale from 1 to 9 is used for assigning weights for the entries of the decision matrix and the matrix a, as summarized in (Triantaphyllou, 1995).

We do not present a definitive ranking of qualities in this paper, since this ranking will vary between projects. One project may prioritize maintainability, while another might prioritize portability. We will however investigate, in Section 4.14, the consequences of using different quality priority profiles on the overall quality ranking.

In our project, we focus on ranking the 30 software options for each of the 13 software qualities given in Section 2.2. We do this by making objective measures, as discussed in Section 2, of each quality for each option. Each measurement corresponds to a score from one to ten. We turn these scores to pair-wise scores using the following steps:

1. We have 2 scores A and B.
2. If $A = 10$ and $B = 1$ or $A = 1$ and $B = 10$, the value that is 10 is reset to 9.
3. When $A \geq B$, Pair-wise score $= A - B + 1$.
4. When $A < B$, Pair-wise score $= 1/(B - A + 1)$.

For example, if installablity of two software packages A and B are marked as 8 and 2, respectively, then the pair-wise score for A versus B is 7, which means that the software A is strongly favoured for this quality. The comparison of B versus A will result in $1/7$.

3 Selected Software

We selected the software packages so that we can have a clear view of the difference between oceanographic model frameworks, Matlab toolboxes and R packages hosted by CRAN. The 30 selected oceanographic software packages are summarized in Tables 2–4. Some summary points about the data are as follows:

- All of them are public software.
- 4 of them are developed using R and hosted by the CRAN community (Table 2); 10 of them are stand alone Matlab toolboxes (Table 3); and 16 of them are oceanographic model frameworks, maintained by different communities (Table 4).
- 14 of the oceanographic models use FORTRAN; 1 of them use C and 1 of them uses both C++ and Java.
- All the software packages from CRAN are considered as alive, using the definition given in Section 2.3. 4 of the Matlab toolboxes are alive and 6 are dead. For the oceanographic model frameworks, 15/16 are alive and 1/16 is dead. The selected R packages and oceanographic model frameworks seem to be better maintained by their communities than the Matlab toolboxes.
- Source code for all R packages and Matlab toolboxes are available. 2 out of 16 oceanographic model frameworks provide partial source code, while all the others provided full source code.

Name	Released	Updated	Status	Language
oce Kelley (2014)	2009	2014	Alive	R
seacarb Gattuso et al. (2014)	2003	2014	Alive	R
ocean Jones (2014)	2013	2014	Alive	R
OceanView Soetaert (2014)	2014	2014	Alive	R

Table 2: CRAN (R) software packages

Name	Released	Updated	Status	Language
SeaGrid USGS (2013)	Unclear	2013	Alive	Matlab
GSW McDougall and Barker (2013)	2009	2013	Alive	Matlab
STAPLOT Byrne (1999)	Unclear	1999	Dead	Matlab
sbPOM Jordi (2012)	2009	2012	Dead	FORTRAN, Matlab
WAFO WAFO (2011)	1993	2011	Dead	Matlab
ADCP Côté et al. (2011)	Unclear	2011	Dead	Matlab
JLAB Lilly (2012)	2012	2012	Dead	Matlab
RSLICE Evans (2007)	2005	2007	Dead	Matlab
TSG-QC IRD (2013)	2010	2013	Alive	Matlab
M_Map Pawlowicz (2013)	2005	2013	Alive	Matlab

Table 3: Matlab Toolboxes

Name	Released	Updated	Status	Source code	Language
NEMO NEMO (2014)	1988	2014	Alive	Available	FORTRAN
GOLD NOAA-GFDL (2012)	2012	2012	Alive	Available	FORTRAN
MOM NOAA-GFDL (2014)	2004	2014	Alive	Available	FORTRAN
MPAS Team (2013)	2013	2013	Alive	Available	FORTRAN
GOTM Umlauf et al. (2013)	Unclear	2013	Alive	Available	FORTRAN
MITgcm MITgcm (2013)	Unclear	2013	Alive	Available	FORTRAN
CICE COSIM (2013)	1998	2013	Alive	Available	FORTRAN
ADCIRC Luettich and Westerink (2013)	2008	2013	Alive	Available	FORTRAN
ROMS ROMS (2014)	2003	2014	Alive	Available	FORTRAN
BOM Berntsen and Avlesen (2011)	2002	2011	Dead	Available	FORTRAN
MOHID MOHID (2014)	2004	2014	Alive	Partial	FORTRAN
ODV ODV (2014)	2006	2014	Alive	Partial	C++/Java
SUNTANS Fringer et al. (2013)	2011	2013	Alive	Available	C
ESCM Eby (2013)	2003	2013	Alive	Available	FORTRAN
HYCOM HYCOM (2013)	2000	2013	Alive	Available	FORTRAN
FVCOM FVCOM (2013)	2011	2013	Alive	Available	FORTRAN

Table 4: Oceanographic model frameworks

4 AHP results

In this section, the AHP results will be presented for each quality. In the charts that follow, red bars (using vertical lines) stand for R packages, blue bars (using slashes) stand for Matlab toolboxes, and green bars (using horizontal lines) stand for oceanographic model frameworks. The detailed grading results are given in Appendix B and are available on-line at `https://github.com/adamlazz/DomainX`.

4.1 Installability

So as to not have problems caused by a pre-existing build system, each software package was installed on its own virtual machine. During the installation process we checked whether there are install instructions and whether these instructions are organized linearly. We also did the following: checked for the use of automated tools for installation, looked for test cases for validation, counted the number of steps required for installation, and counted the number of external libraries needed. At the end of the installation and testing, if an

uninstaller was available, we ran it and assessed how successfully the software was removed.

Figure 1 shows the AHP result for installability. Some key findings between the different types of software are as follows:

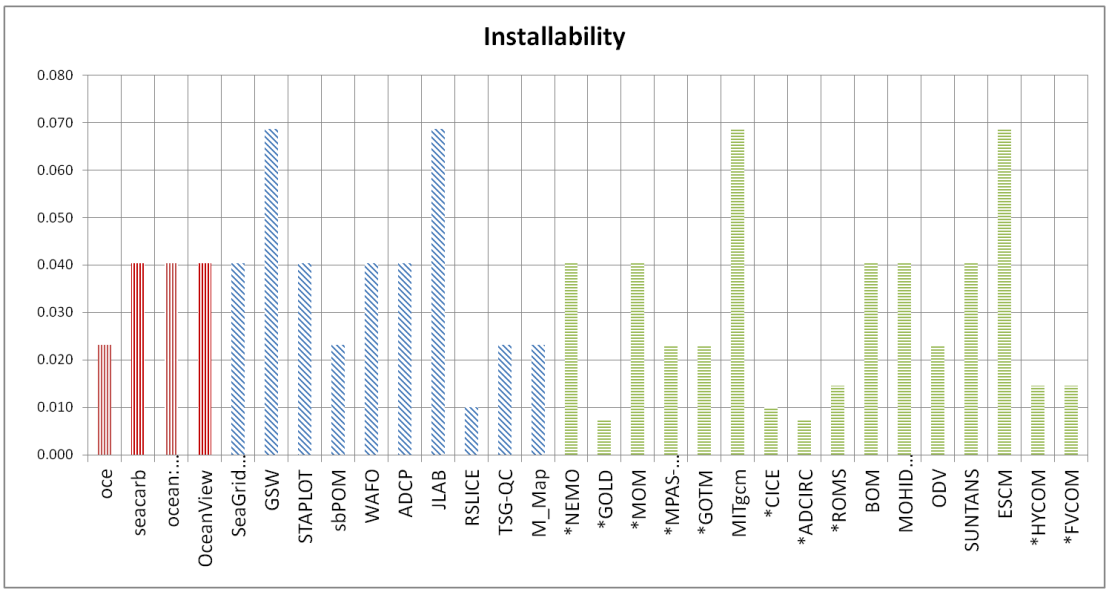

Figure 1: AHP result for installability (Packages that were not installed successfully are marked with an *)

- R packages hosted by CRAN: The selected R packages did well on the quality of installability. The explanation for this, in part, is the presence of the CRAN community's general installation instructions (CRAN, 2014a) for all the packages it maintains. Unfortunately, in some case links to the general instructions were not given in the software package page. This may cause confusion to beginners. In other cases, such as for **oce**, the software installation information is given on its own website. Most of the packages installed easily and automatically, except for **oce**, which needed a few steps to be done manually. The uninstall process went well in all cases. A draw back of R packages for installability is that none of the packages provide a standard suite of test cases specifically for the verification of the installation.

- Matlab toolboxes: The overall results of the installability of Matlab toolboxes is as good as for the R packages. Most of them have well organized installation instructions and automated installers. 5 out of 10 toolboxes did not provide an automated installer, but given the nature of Matlab this is not a major issue. Users can generally include a new toolbox simply by placing the relevant files directly into their search path. The installability of Matlab is certainly aided by the fact that it is a commercial product. 2 out of 10 Matlab toolboxes (**GSW** and **JLab**) provided a standard test suite specifically for the verification of the installation. The other toolboxes apparently did not to consider this issue.

- Oceanographic model frameworks: The results of installability in the oceanographic model frameworks were uneven. For the good examples shown in the Figure 1, they provided everything that we are looking for. The situation for test suites is better than for the other types of software; 7 out 16 frameworks (e.g. **NEMO**, **MOM**, etc.) provided relevant data and methods to validate the installation process. The software that ranked lower was because of unorganized installation instructions (3 out of 16), too many steps involved in installation (3 out of 16 had over 10 steps) and too many libraries (4 out of 16 used over 5 libraries) to be manually installed. Little information about the libraries is given during the installation of most software, which caused significant trouble during our experiment. Due to our limited time and the fact that we are not oceanographic domain experts, we were not able to install 10 out of 16 oceanographic model frameworks. These packages are marked with a * in Figure 1. As shown on many user forums, for example **ROMS**'s user forum, problems occurring during installation is a significant issue for a large number of users. Paul G. Myers (personal communication, July 2014) mentioned that the oceanographic community is relatively small; therefore, the installation procedure for new users is usually conducted by a supervisor that is already well established in the community. People outside the community will have problems in installing the software. As a point of comparison, we have done a similar measurement for the state of the practice in the domain of psychometric software (Smith et al., 2015), where we are also not domain experts. All 30 selected software packages from that domain were installed successfully, and 24 out of 30 software tools were installed without any trouble. The installation process of oceanographic model frameworks is an area for improvement within the oceanography community.

4.2 Correctness and Verifiability

We used several metrics to assess correctness and verifiability. First we checked for the use of trustworthy libraries. Second we looked for the presence of a requirements specification. The specification did not have to be written in a rigorous or formal style, as long as it documented the physical and mathematical models used in the software. (If we had required a formal requirements specification, the number of projects providing one would have been zero.) The third step was to look for the use of any confidence building tools or techniques, such as literate programming, automated testing, symbolic execution, or assertions in the code. The final step was to run (if possible) any examples provided in the documentation on a virtual machine, to verify that the output matches the documented expectations.

The results for correctness and verifiability of the oceanographic domain are shown in Figure 2, with the comparison as follows:

- R packages hosted by CRAN: A wide range of results were observed for the 4 R packages. On the positive side, CRAN facilitates correctness and verifiability, as evidenced by oce and OceanView. This success is partly attributable to the CRAN

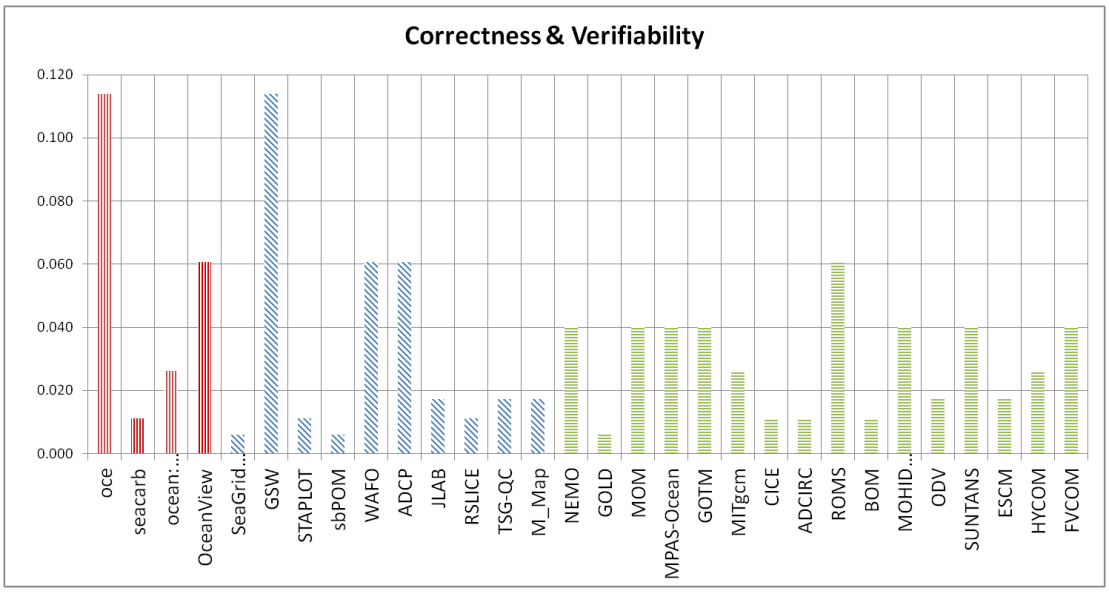

Figure 2: AHP result for correctness and verifiability

Repository policy. This policy, along with the requirement of documenting dependencies and reverse dependencies between extensions, aids in building a collection of trustworthy R extensions. Another positive for CRAN is the quality and completeness of the documentation, which is a consequence of the requirement of Rd (R documentation) files. Rd is a markup language that can be processed to create documentation in LaTeX, HTML and text formats (R Core Team, 2014). On the negative side, the CRAN policies and tools are not enough to guarantee correctness and verifiability, as evidenced by seacarb and ocean. The developers of R extensions have to include certain sections of documentation, but the content of these sections is up to them, which means considerable variability in quality. Further variability in quality is a result of the CRAN policy that vignettes are optional. The two highest scoring R packages were the only two where vignettes were included. Vignettes provide additional documentation in a more free format than the Rd documentation allows. Vignettes can act as user manuals, tutorials, and extended examples. Many vignettes are written with Sweave (Leisch, 2002), which is a Literate Programming (LP) (Knuth, 1984) tool for R. All the R packages have examples on how to use the software, but half (**seacarb** and **ocean**) did not provide expected output data.

- Matlab toolboxes: Many of the Matlab toolboxes provided inadequate information for building confidence in the correctness and verifiability of the software packages. We found that only **GSW**, **ADCP** and **TSGQC** mentioned that they used existing libraries during development. In their defence, they all reused the computational framework provided by Matlab. Only 3 out of 10 provided a requirement specification and 4 out of 10 provided a reference manual. 8 packages presented standard examples and half of them also showed the expected output. **GSW**, **WAFO** and **ADCP** were

the best Matlab examples.

- Oceanographic model frameworks: The results for oceanographic frameworks were mostly fine, but there were no stand out examples. 4 out of 16 mentioned standard libraries were used; 6 of the them provided requirement specification documents and 14 had reference manuals to build up confidence. 3 out of 16 provided standard examples, but the rest of them often lacked proper instructions, explanations and data.

Little evidence of verification via testing was found among the three classes of software. From the oceanographic model frameworks the notable exceptions were **FVCOM** and **GOTM**, which include large sets of model verification simulations. Unfortunately, the model verifications do not seem to be part of automated regression tests. From CRAN, the best example of testing is the diagnostic checks done on each of the R extensions. To verify that the extensions will work, the R package checker, `R CMD check`, is used. The tests done by `R CMD check` include the following: R syntax checks, Rd syntax and completeness checks and, if available, example checks, which verify that the examples produce the expected output (R Core Team, 2014). Although these checks are valuable, they focus on syntax not on semantics. As observed by Wickham (2011), the development practices of R programmers does not usually appear to include automatically re-running test cases.

For the 30 packages considered, Literate Programming (LP) tools, which can facilitate code verification, were only used in two of the R packages. Unfortunately, in both cases LP was not used to assist with code verification. As was also observed in Smith et al. (2015) for psychometrics software, literate R documentation focuses on examples to illustrate the use of the extension, not on showing the R implementation along with its explanation. In the spirit of Knuth's original work, LP can be used to document the code to convince a human reader of the correctness of the implementation (Knuth, 1984). In scientific computing, LP can be used to improve verifiability by maintaining the documentation and code together in one source (Nedialkov, 2010), but the CRAN community does not appear to follow this approach.

4.3 Reliability (Surface)

The reliability of each software package was checked by installing and running it on a virtual machine using the instructions given with the software. We checked if the software breaks during installation or during initial tutorial testing. Installability has already been presented in Section 4.1. The difference here is that we are not measuring the quality of the installation process or instructions, but rather the results of the installation and initial testing. For practical reasons, our measure of reliability has to be considered as a surface measure, since we only have time for a small set of test cases. The results for reliability of the oceanographic domain are shown in Figure 3, and the comparison between different types of software are as follows:

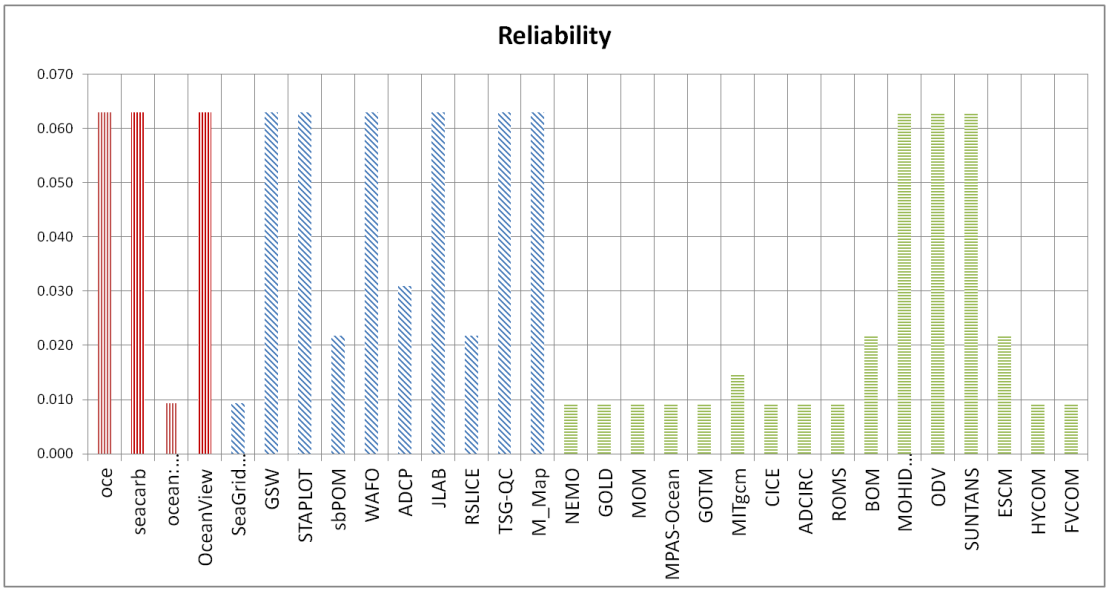

Figure 3: AHP result for reliability

- R packages hosted by CRAN: For all of the R packages there was no problem during installation, although there was one problem during the initial tutorial testing. For **ocean** one of its functions could not be found. The overall result is positive for this type of software.

- Matlab toolboxes: Most Matlab toolboxes did extremely well in this quality. For 6 out of 10 toolboxes, no problem happened either during installation or during initial tutorial testing. There was only 1 problem during installation – **SeaGrid** did not support running on R2010b (Win64). For those packages that ranked lower, the reason is that they failed to provide enough instructions for tutorial testing, or to present the input data.

- Oceanographic model frameworks: This type of software has problems in reliability because of the difficulty in installation. As mentioned previously, we had problems installing 10 out of 16 software tools. (We invested between 4 and 8 hours of time in attempting to install each of the challenging software package). The problems are often caused by too short of a description about installing the NETCDF library, compilation aborts and different kinds of makefile error. Similar problems are listed on the community's user forum (e.g. **ROMS**'s user forum), which shows reliability in installation is indeed a problem for this type of software.

4.4 Robustness (Surface)

The robustness of each package was checked by running it for situations the developers may not have anticipated. We checked whether the software can gracefully handle garbage

input, with a graceful response including the issuing of an appropriate error message. For the packages that accept text files, one robustness test was to modify the expected end of line character. Like reliability, this is also an example of surface checking of a quality. With reference to Figure 4, the comparison between the different types of software for the quality of robustness is as follows:

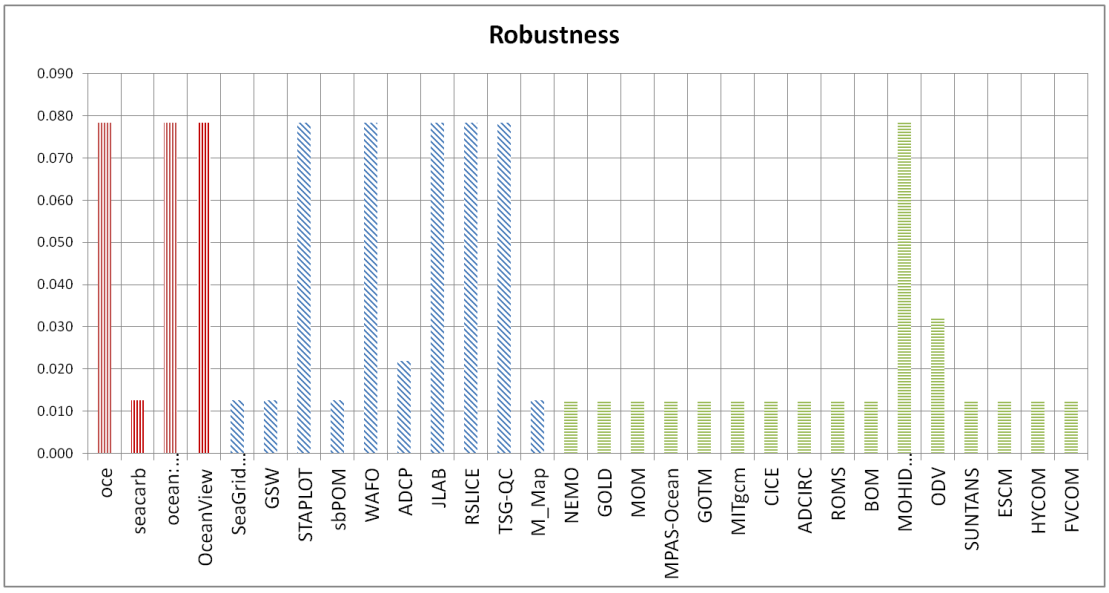

Figure 4: AHP result for robustness

- R packages hosted by CRAN: 3 out of 4 R packages hosted by CRAN handle unexpected input reasonably; they provide information or warnings when the user inputs invalid data. One package, **Seacarb**, did not check the validity of its inputs; we noticed that it allows users to input negative salinity. R packages do not use text files as input file; therefore, a change of format in a text file is not a issue for them.

- Matlab toolboxes and oceanographic model frameworks: half of the selected Matlab toolboxes and 1 oceanographic model framework did well on robustness. **ODV** had a problem when we tried to open an invalid input file. For the rest of software packages that ranked lower, they either did not provide the correct format of input data or we were unable to run the software, since we were unable to properly install it. If we had been able to install more of the oceanographic frameworks, their robustness measures would likely have increased.

4.5 Performance (Surface)

We checked performance by looking for evidence of a profiler, or whether the package is optimized for different environments. The results are shown in Figure 5, with the result comparison as follows:

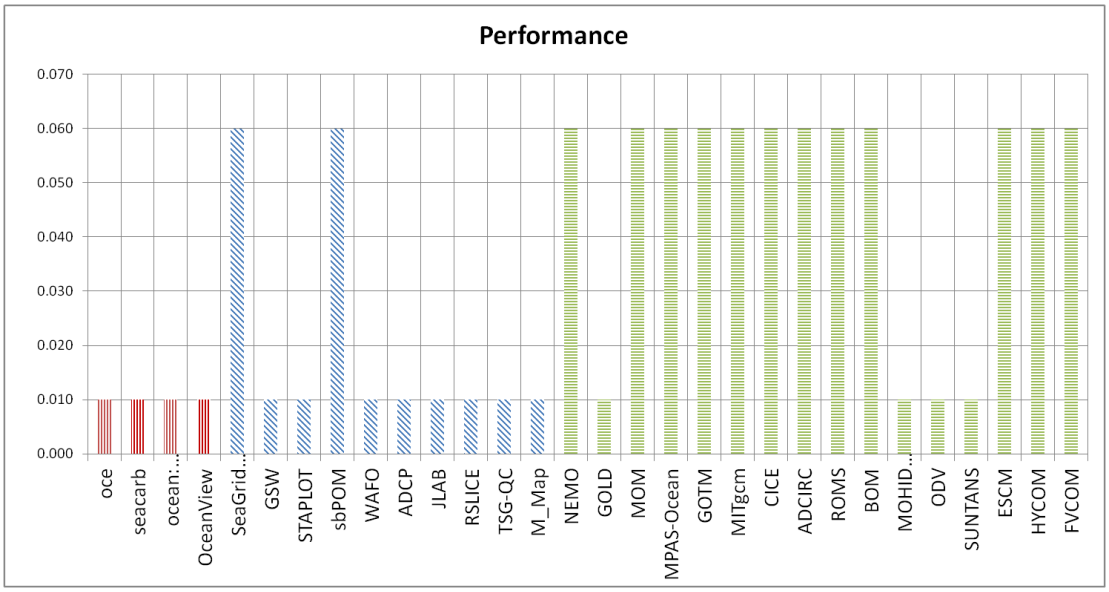

Figure 5: AHP result for performance

- R packages hosted by CRAN: No evidence related to performance was found in the selected R packages.
- Matlab toolboxes and oceanographic model frameworks: 2 Matlab toolboxes and 12 oceanographic model frameworks have different compilation option for different environment, which showed their consideration for optimizing their programs for different platforms.

4.6 Usability (Surface)

We asked seven questions to assess usability: *i*) Is there a getting started tutorial? *ii*) Is there a well-explained standard example? *iii*) Is there a user manual? *iv*) Does the application have the usual "look and feel" for the platform it is on? *v*) Are there any functions that are not easily visible through the interface? *vi*) Are expected user characteristics documented? *vii*) What is the user support model? The results for usability of the oceanographic domain are shown in Figure 6. With respect to a comparison between types of software, we saw the following:

- R packages hosted by CRAN: The overall results for R are reasonably good. All of the R packages gave detailed explanations for their examples and provided well-organized user manuals. This minimum level of usability is ensured by the CRAN repository policy and the required use of Rd files. Usability is further improved by the use of **Sweave** vignettes. Of the R examples, **oce** is the best. However, only half of them provided getting started tutorials. The common user support model is an email address. **oce** and **ocean** also add support through GitHub.

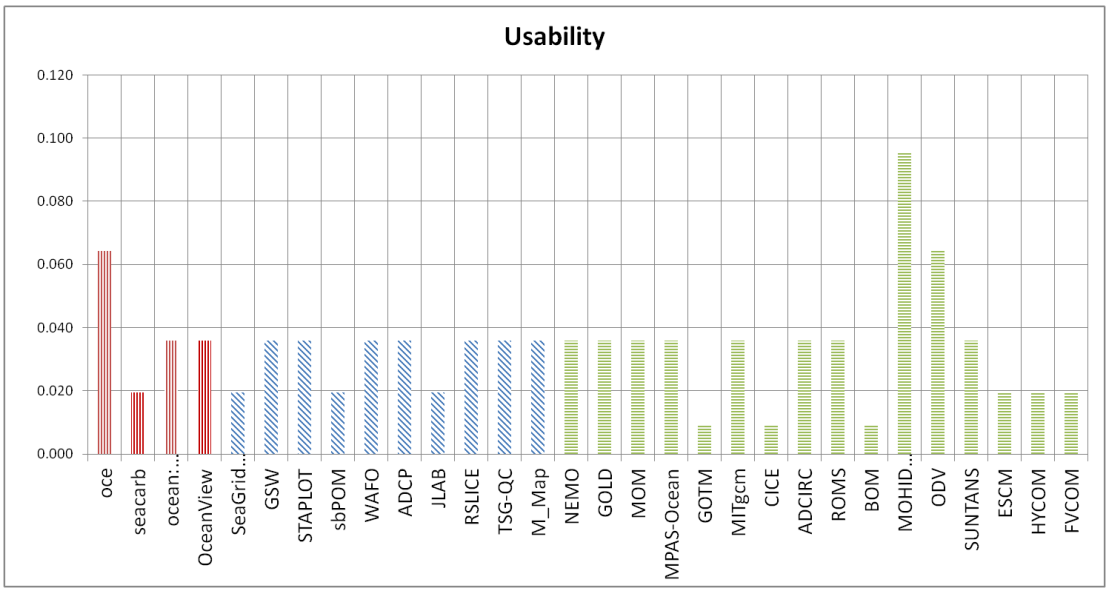

Figure 6: AHP result for usability

- Matlab toolboxes: Matlab toolboxes all did about the same with respect to this quality. 9 out of 10 had getting started tutorials; all of them provided explanations for their examples and 7 of them had good user manuals. For the user support model, only 1 of them (**sbPOM**) provided a user forum, and only **WAFO** uses Google code and an FAQ in addition to email support.

- Oceanographic model frameworks: Oceanographic model frameworks performed similarly to the Matlab toolboxes. **MOHID** is the best example here. Overall the model framework software did better than the other categories with respect to the user support model. 3 out of 16 had an FAQ. 6 out of 16 provided user forums and 3 of them employed Google code or GitHub. 2 of the packages (**NEMO** and **ADCIRC**) have organized in-person user meetings.

4.7 Maintainability

Maintainability was measured by looking for evidence that there is a history of multiple versions of the software. If the software has been maintained through several iterations, this is a good sign that it is maintainable. Other indications of maintainability include instructions on how to contribute code, and the presence of a change log. If there was a change log, we identified the most common types of maintenance (corrective, adaptive or perfective). Maintainability was also measured by the tools used in the software development. For instance, what issue tracking tool was employed? Does it show that major bugs have been fixed? We also looked at the concurrent versioning system in use, as this is important to both industry and open source project (Wilson et al., 2013). The software design was also inspected to determine if the modular decomposition is based, at least

on the surface, on likely changes, as advocated by Parnas (1972). In a similar spirit, we looked for evidence of code clones, since maintaining similar code in more than one place is challenging.

The results for maintainability are shown in Figure 7. The conclusion we draw from the data are as follows:

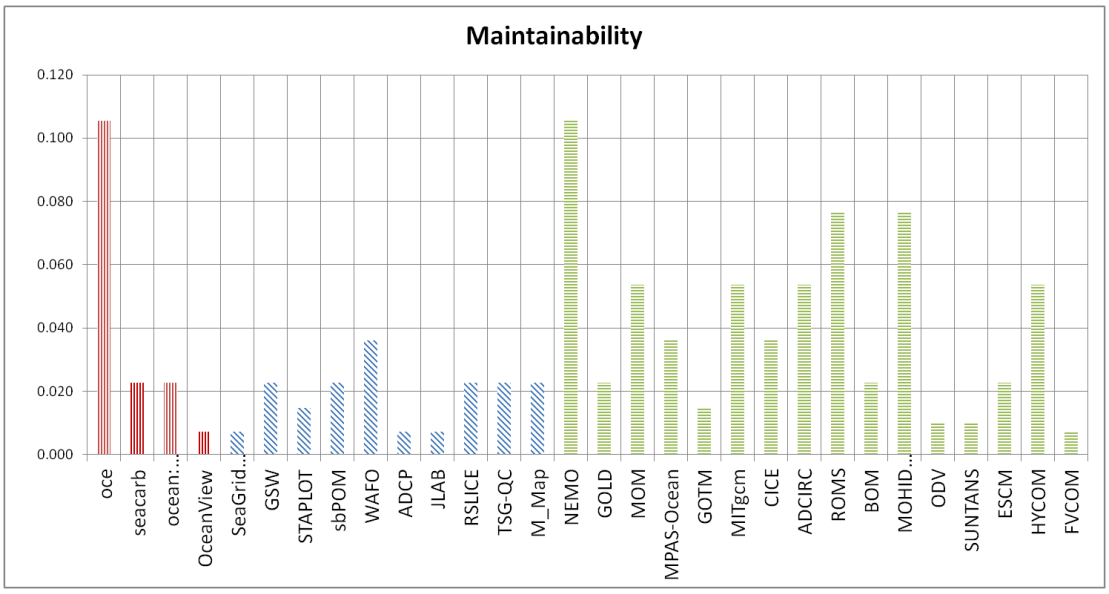

Figure 7: AHP result for maintainability

- R packages hosted by CRAN: **oce** is a very good example for this type of software; other R packages ranked slightly lower. 3 out of 4 packages provided a history of multiple versions of the software and 1 of them gave information about how to make contributions. 2 out of 4 provided change logs; 2 out of 4 indicate use of an issue tracking tool (GitHub) and concurrent versioning systems (GitHub). All of them consider maintainability in code with no obvious evidence of code clones.

- Matlab toolboxes: Matlab toolboxes did fine with respect to this quality and showed a very consistent result. 4 out of 10 provided a history of multiple versions; only **RSLICE** provided details on how to make contributions to the software package; 6 of the packages provided change logs. The common type of maintenance is corrective and perfective. Only **WAFO** used an issue tracking tool (Google code) and 3 toolboxes employed SVN as their concurrent versioning system.

- Oceanographic model frameworks: Oceanographic model frameworks tended to do better than the other two types of software in maintainability. 12 out of 16 provided the version history of their software, 9 out of 16 gave information about how to contribute and provided change logs. 9 of them clearly showed that they made use of issue tracking tools, and 10 of them employed a concurrent versioning system.

Developers outside a given software community will likely find it easier to contribute to oceanographic model frameworks than to the Matlab toolboxes.

4.8 Reusability

Reusability was measured by looking for any evidence that a portion of the software is used by another package. Other evidence of reusability is also looked for, such as a carefully documented Application Program Interface (API). The results for reusability of the oceanographic domain are as follows (Figure 8 shows the AHP results):

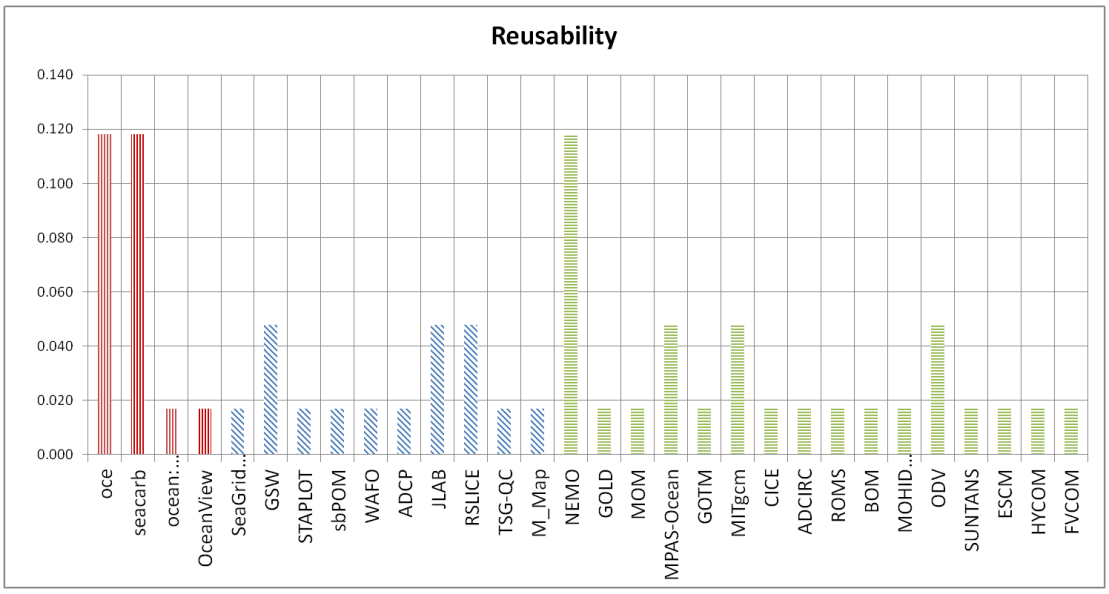

Figure 8: AHP result for reusability

- R packages hosted by CRAN: R packages have great reusability. There are clear evidences that 2 out of 4 selected packages have been reused by other software packages hosted by CRAN. Also, the reference manual required by CRAN can serve as an API document, which helps others to reuse the package.
- Matlab toolboxes and oceanographic model frameworks: Only **NEMO** showed evidence that it has been reused by other projects. For all the other software packages, only 6 of them provided API documents.

4.9 Portability

Portability was measured by checking which platforms the software is advertised to work on, assessing how people are expected to handle portability, and whether evidence in the documentation shows portability has been achieved. The results for portability, based on the AHP result shown in Figure 9, are as follows:

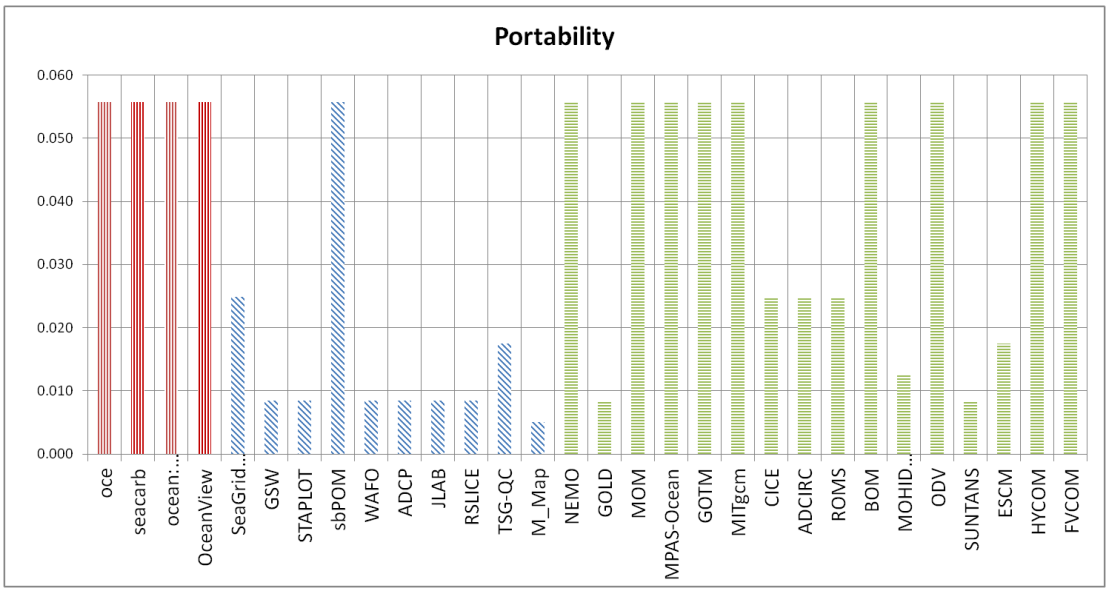

Figure 9: AHP result for portability

- R packages hosted by CRAN: R packages handle portability consistently well. They all kept different versions of the software for different platforms. Also, software checking results from CRAN showed that the software worked well on different platforms.
- Matlab toolboxes: Little evidence was available to show that portability has been achieved. Only 2 of the selected Matlab toolboxes handled portability by branches in the repository. However, this cannot really be considered as a negative if the developers are not interested in targeting other environments.
- Oceanographic model frameworks: The portability measure for oceanographic model frameworks is positive. 11 out of 16 packages used different makefile variables to achieve portability. Portability was clearly considered in designing most of these packages.

4.10 Understandability (Surface)

To measure understandability we focused on code formatting style, existence of the use of coding standards and comments, identifier and parameters style, use of constants, meaningful file names, whether code is modularized and whether there is a design document. Figure 10 summarizes the AHP results. The differences between the communities are as follows:

- R packages hosted by CRAN: R packages did fine on this quality. The common problem in R packages is that no coding standard is provided and there are limited comments in the code. The understandability of the R packages could be improved by

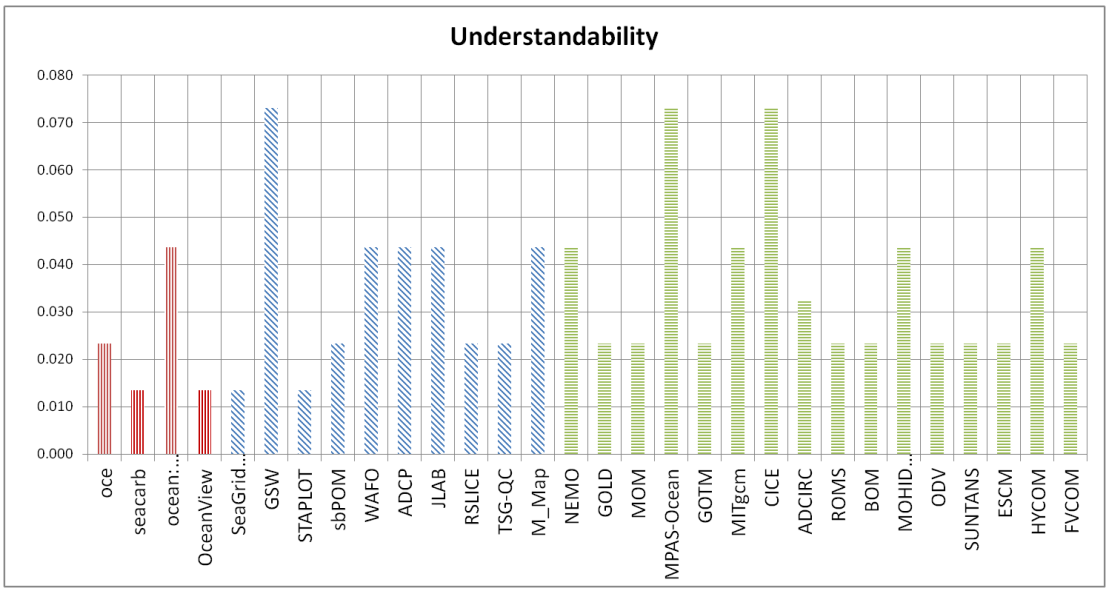

Figure 10: AHP result for understandability

using **Sweave** for literate programming. As discussed in Section 4.2 for the quality of verifiability, R has the tools for maintaining the code and its documentation together, but this does not appear to be a common practice within CRAN.

- Matlab toolboxes: Matlab toolboxes did consistently well on this quality. Their main problem is a lack of coding standards and the absence of a design document.
- Oceanographic model frameworks: 7 out of 16 projects here presented coding standard, and 4 out of 16 provided design document (not strictly), which ensured better average understandability over the other two types of software.

4.11 Interoperability

Interoperability was measured by looking for the following: an external systems the software communicates with, a workflow that uses other software, and a documented API. The main findings for interoperability are as follows (Figure 11 shows the AHP results):

- R packages hosted by CRAN: R packages had positive performance on this quality because of the nature of R and the requirement of documentation for CRAN. There is no obvious evidence that these software packages communicate with external system, but it is very clear that 3 out of 4 have existing workflows that use other R packages.
- No sign of interoperability was found in the Matlab toolboxes.
- Oceanographic model frameworks: 2 out 16 packages can communicate with external systems; half of the packages have workflows that use other software. The problem with this type of software is that no API is documented.

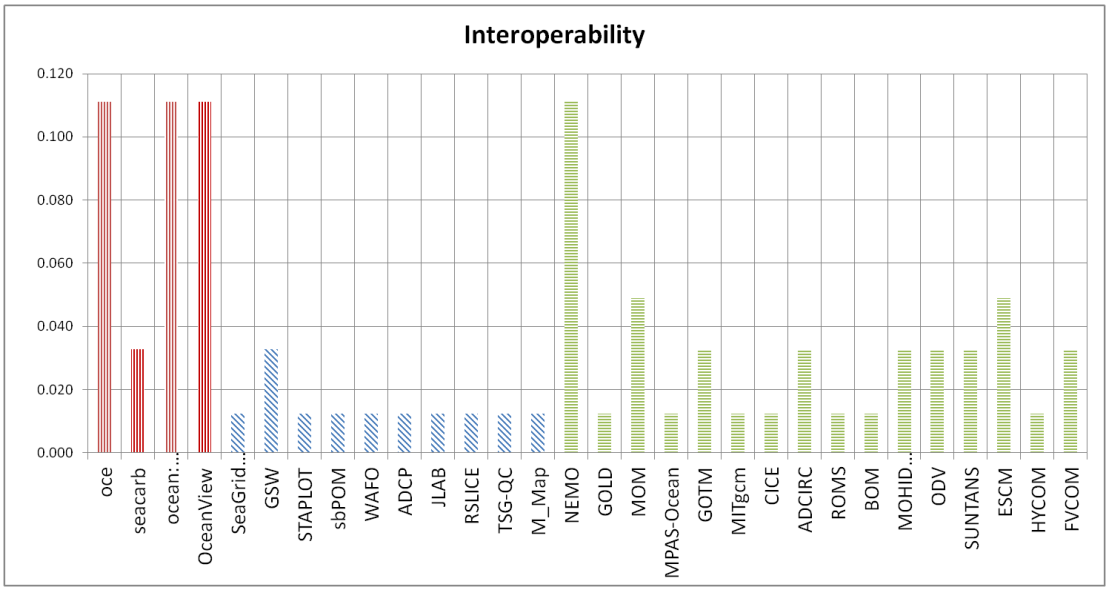

Figure 11: AHP result for interoperability

4.12 Visibility/Transparency

Visibility was measured by looking for documentation of the development process. A subjective score was also assigned as a visibility metric based on our experience searching for the information we needed for grading the software. This score reflects the relative ease for users to find useful informations from the software website and documentation. The results for visibility (transparency) follow (Figure 12 shows the AHP scores):

- R packages hosted by CRAN and Matlab toolboxes: No software packages provided information about the development process. However, packages like **oce** provided a decent way to present information about the software, which made it easy for us to check it in comparison to other software packages. The good R and Matlab packages have well organized websites with examples and instructions.

- Oceanographic model frameworks: Oceanographic model frameworks did better than the other two categories with respect to presenting useful information to the user. 4 out of 16 oceanographic model frameworks provided information about the development process, although this information was not very detailed.

4.13 Reproducibility

Reproducibility was assessed by looking for the following: a record of the environment used for development and testing, test data for verification and automated tools to capture the experimental context. The AHP results for reproducibility are given in Figure 13.

All the R packages provided a record of the environment for testing. R packages also benefit from the use of Sweave for the vignettes (if present), since LP can be used to

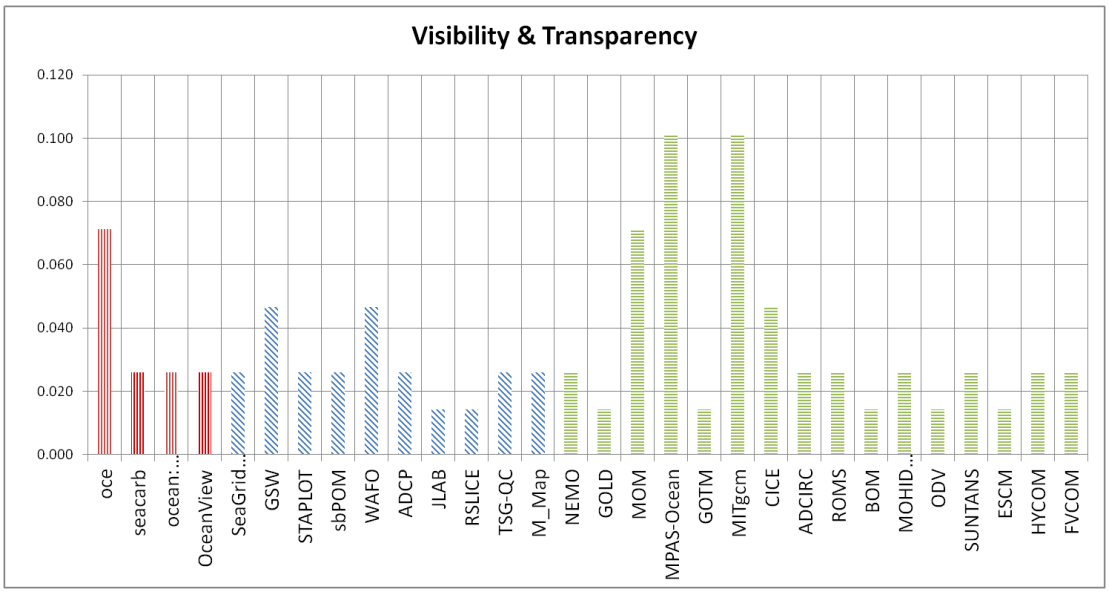

Figure 12: AHP result for visibility (transparency)

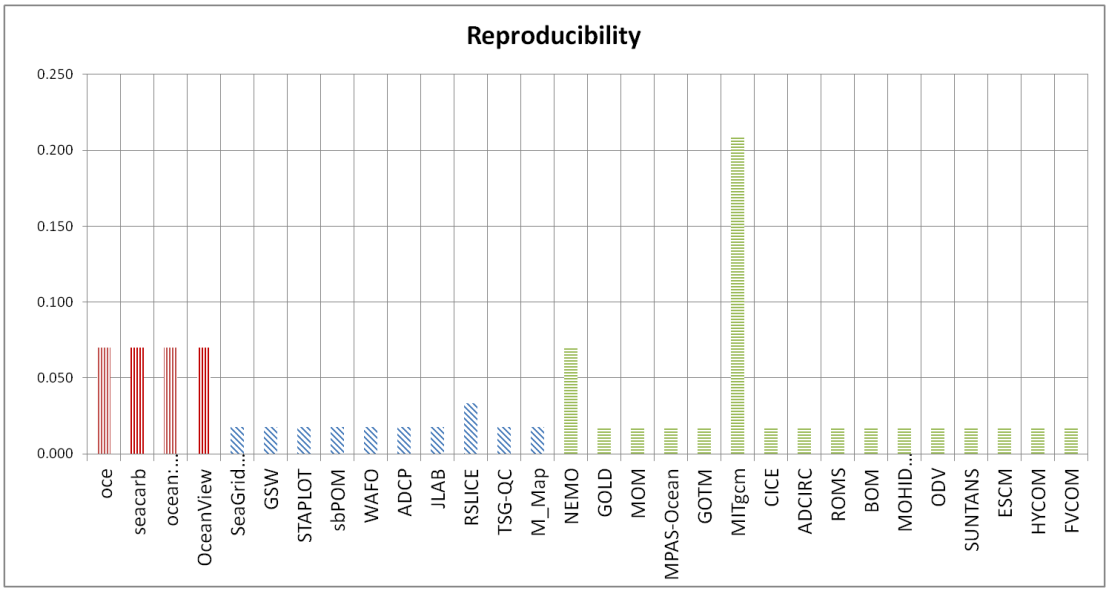

Figure 13: AHP result for reproducibility

document reproducible results by maintaining the code and its documentation together. Little relevant information was given by the Matlab toolboxes; **NEMO** provided information about the testing environment and **MITgcm** provided not only testing records but also test data and some scripts that help to capture the data. However, most software packages do not seem to have considered reproducibility.

4.14 Overall Quality

We first use the same weight for each quality in our AHP table, as shown in Figure 14. In this case, oceanographic model frameworks ranked slightly lower for the reason that we were not able to run 10 of them successfully, which meant their reliability and robustness was not accurately measured.

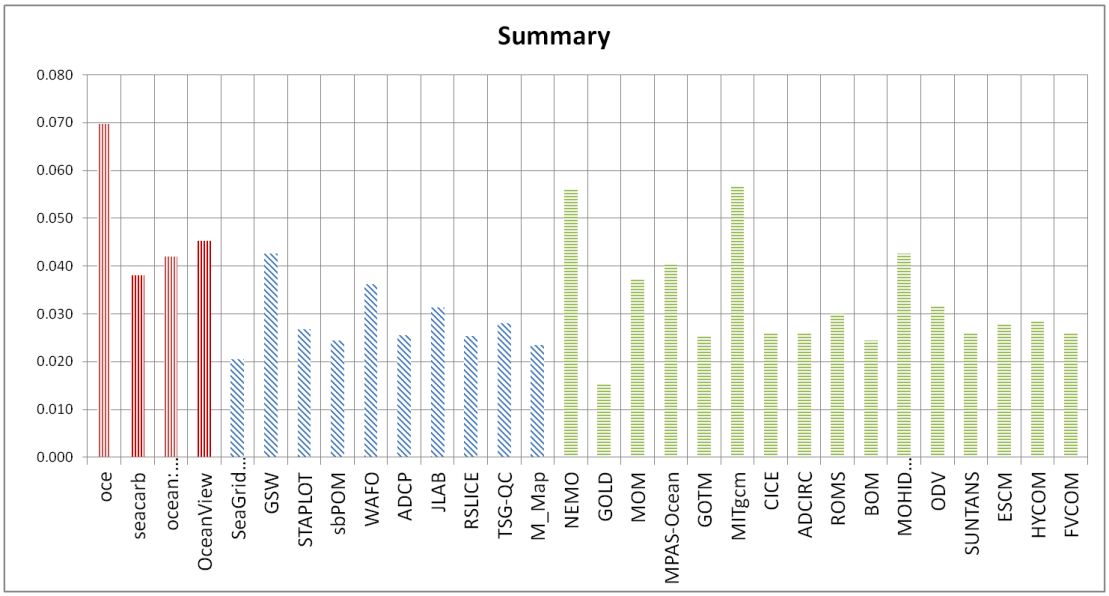

Figure 14: Summary (All qualities have the same weight)

To minimize the influence of the installability problem mentioned above, we gave reliability and robustness a weight of 1, while all other qualities were set to 9. Figure 15 shows the AHP results. As for the previous analysis, the best examples are **oce**, **MITgcm** and **NEMO**, but the overall results do not favor R as strongly as they did in the previous analysis.

5 Recommendation

Although for practical reasons it may not be feasible, or necessary in all cases, our recommendations are written with the intention that all software packages are aiming for the highest possible success on each quality. Individual projects will need to determine for themselves whether there is any justification for quality improvement efforts. The recommendations begin with specific thoughts on each of the qualities. This is followed by a discussion of potential strategies that could potentially improve multiple qualities. The strategies proposed are intended to emulated the success achieved by other scientific software projects.

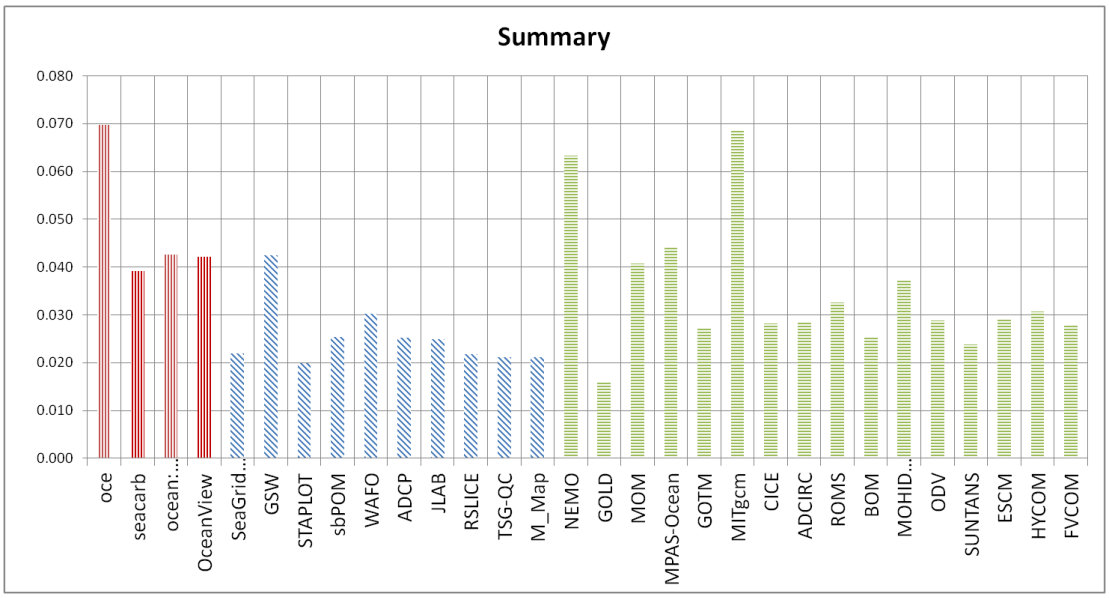

Figure 15: Summary (Reduced weight for reliability and robustness)

5.1 Recommendations to Improve Each Quality

We found that all the selected software did well on the quality of robustness. However, for the other qualities, there are examples that show room for improvement. Recommendations for each quality are listed below.

- Installability: Developers should give detailed install instructions and a standard suite of test cases for verification of installation. Specifically for the oceanographic model frameworks, the amount of work to be done manually should be reduced and more detailed information about installing prerequisite library should be given. **MITgcm** is a good example for future developers to start from.

- Correctness and Verifiability: The way CRAN organized reference manuals, requirement specifications, and information about the libraries used is worth learning. This positive aspect of CRAN was even more strongly supported in our survey of psychometrics software (Smith et al., 2015). More test data and instructions for test examples should be given for each type of software. Package **Oce** gives detailed examples with output figure on its own website; other packages can learn from this. A specific recommendation for CRAN would be to add a specific section in their documentation template for packages to present their example/output.

- Usability: CRAN should require a detailed getting started tutorial for each software tool, in addition to the user manual. Developers who wish to see a good getting started tutorial should look to **GSW**. Matlab toolboxes and R packages should provide a better user support model than simply email support. For user support, **NEMO** and **ROMS** are the good examples.

- Maintainability: For open source project, concurrent versioning and issue tracking tool are strongly suggested (Wilson, 2006). Github was successfully used in developing 4 out of 30 packages; it also has an issue tracking tool within it. People within oceanography domain should make greater use of Github in the development process. Information like change log or how to contribute should also be presented.

- Reusability: A well documented API should be given. Packages like **MAPS-Ocean** employed **doxygen** to generate API document. Developers can make use of similar tools if they want to publish API documents.

- Understandability: Coding standards and design documentation should be provided for all packages. **NEMO** is an example of good practice that others can follow. **NEMP** not only gives a coding standard, but also provides a development policy. This kind of documentation can make developers within a community cooperate consistently. Developers should write comments in their R packages. Ideally, **Sweave** could be used to write literate documentation, as discussed in the next section.

- Visibility and Transparency: Projects that look forward to the involvement of future developers should provide the development process in their developer guide. Matlab toolboxes should organize a better way to present information (introduction of software, relevant document, code, user support, etc.). **Oce**, **MITgcm** and **ROMS** are good examples to follow. The way they organized their website makes it easy for users to find information.

- Reproducibility: Software developers should start tracking the environment used for development and testing, to make the software result more reproducible. There was no evidence that any packages made use of automated tools to capture experimental context. One option to consider in the future is Madagascar (2014). **MITgcm** provided a script for running the data automatically. Developers could follow this example to improve their software's reproducibility.

5.2 Strategies to Emulate the Success of Others

For achieving truly impressive software quality for oceanographic software, the hard part has already been completed. A rich collection of software is already available, representing a significant investment of time and energy. The oceanography community has developed sophisticated models and has implemented complex algorithms. The only part that seems to be missing, for many of the examples considered, is a complete software development process and infrastructure. In particular, a process and an infrastructure that supports software installation, automated testing, and contributions from new developers. Installation in particular, is particularly noteworthy for the oceanographic model frameworks studied. Since we could not complete this critical first step for many packages, we were unable to fully evaluate the software. In the case of potential software users, if they cannot install it, then they cannot reap the rewards of all the hard work that has preceded them.

Excellent examples of fully worked out software development processes and infrastructures can be found in scientific computing. For instance, **GRASS** (GRASS Development Team, 2014), an open source Geographic Information System (GIS), scored very highly for all software qualities when it was assessed following the same methods used in this study (Lazzarato et al., 2015). The success of **GRASS** should in part be attributed to its extensive documentation, which includes a full requirements specification, design documentation through a programmer's manual (written in the **doxygen** format), explicit code submission rules and coding standards. The **GRASS** website includes documentation to help new developers get started, including instructions on code compilation, how to get involved and a list of feature requests.

Another success story for scientific computing software is **VTK** (VTK Team, 2014) by KitWare. In a study comparing medical imaging software following the same methods as described in this paper, **VTK** was one of the top performers (Kapil et al., Submitted 2014?). As was the case for **GRASS**, **VTK** uses **doxygen** for automatic extraction of documentation from the code. **VTK** does particularly well at installability across a wide variety of platforms. This success is attributable to the number of different options for **VTK** installation. **VTK** can be installed using a package manager, such as **MacPorts** for **OS X**, or via freely downloading **Anaconda** (`https://store.continuum.io/cshop/anaconda/`), a scientific Python distribution that already includes **VTK** and other scientific software packages. In addition, **VTK** can be compiled from source using Kitware's **CMake**, a cross-platform, open-source build system. Another Kitware tool that improves the quality of **VTK** is **CDash**, which is an open source, web-based software testing server that aggregates, analyzes and displays the results of the submitted tests. Using **CDash**, **VTK** is automatically tested nightly with the results immediately available to the development community.

The relatively smaller size of the oceanography development community compared to **GRASS** and **VTK** means that emulating the success of these products will be challenging. **VTK** is sponsored by multiple national laboratories, including Los Alamos and Sandia National Laboratories and **GRASS** has been developed by a large number of federal US agencies, universities, private companies, and a worldwide network of developers. To achieve the same success it is likely necessary for oceanography developers to either increase the size of their community, or adopt strategies that facilitate a smaller community acting as a larger one. Some potential ideas, which are not mutually exclusive, are as follows:

1. Admittedly a naive suggestion, and one that only an outsider to the community would likely make, but on the surface it seems that a larger development community would be formed by combining some of the existing projects. Although functionality was outside of the scope of this study, a survey of the various software packages suggests repetition exists. To increase the size of the development community multiple projects could be consolidated into one, with portions of the "abandoned" projects reused in the consolidated one.

2. Another strategy would be to focus energy and time on developing information for new developers, including a programmer's guide, design documentation, submission

guidelines and coding standards. Along with the documentation would need to come a set of development tools for version control, document generation (like **doxygen**), submission verification and test automation. **GRASS** and **VTK** both have fully worked out software documentation and tool infrastructures. The presence of documentation and tools means that a smaller community can be effective. For example, this is the strategy adopted by the R community. As observed for psychometrics software (Smith et al., 2015), the CRAN policy, Rd files and Sweave, allow a conscientious individual developer (or small team) to produce software of the quality of much larger teams.

3. The size of the development community could be increased by reaching out to the broader open source community. For instance, if a **MacPorts** developer (or other package manager) were convinced to write a portfile for the oceanography software, this would greatly improve installability. Hosting more of the oceanography software on GitHub, as mentioned in the previous section, could also make a difference, since this is a popular site for all kinds of open source developers.

The processes and tools mentioned previously for software development are relatively mainstream. A less frequently used strategy for improving verifiability, understandability and reproducibility for scientific software that deserves attention is Literate Programming (LP) (Knuth, 1984). An LP tool (Sweave) is used by CRAN, but the aim is teaching the use of the code to new users, not to convincing other developers that the implementation is correct (Smith et al., 2015). Examples of LP to improve the quality of scientific computing software can be found in Nedialkov (2006) and Pharr and Humphreys (2004). Specific documentation and code examples that highlight the quality improvements possible using LP, together with a Software Requirements Specification (SRS) template, can be found in Smith et al. (Submitted 2014).

6 Conclusion

R packages and oceanographic model frameworks have advantages in qualities related to development. The quality of the R packages are consistently high because of the requirements set by CRAN. On the other hand, the quality of the oceanographic model frameworks varied according to the different expectations between different development communities. Matlab toolboxes have the best installability; Matlab toolboxes and R packages did well in showing correctness and verifiability. The R packages and oceanographic model frameworks seem to be better maintained by their communities than the Matlab toolboxes.

In our report we made recommendations including reducing the manual steps for installation of oceanographic model frameworks, and adding to CRAN a specific section with examples and their expected output. We suggested consolidating existing projects into fewer software products, adopting the process and tools used by **GRASS** and **VTK** and reaching out to the broader open source community for assistance. Our suggestions for

documentation included software requirements specification, design documentation, programmer's manual, submission guidelines, and a coding standard. For tools, we suggested the use of version control software, GitHub and **doxygen**. For those that want to move outside of mainstream for improving verifiability, understandabilty and reproducibility, we suggested considering LP.

We understand that development communities set their goals differently depending on the number of developers, or amount of funding they have. We do not want to leave the impression that the software that ranked lower is necessarily bad software. In some cases, the context in which the software will be used, and the type of users that will use it, mean that aiming for a "high" grade would be a waste of resources. Our goal was to summarize suggestions based on the common issues we observed in the experimental results. In doing this we implicitly assumed that all of the products are aiming for high results for all measures. Developers can take this information and decide for themselves which measures to target for improvements based on the priorities of the oceanographic community.

References

Jarle Berntsen and Helge Avlesen. *The Bergen Ocean Model*, 2011. URL `http://www.mi.uib.no/BOM/`.

Deirdre Byrne. *STAPLOT*, 1999. URL `http://gyre.umeoce.maine.edu/staplot/staintro.html`.

COSIM. *The Los Alamos sea ice model*, 2013. URL `http://oceans11.lanl.gov/drupal/CICE`.

Jessica M. Côté, Frances S. Hotchkiss, Marinna Martini, and Charles R. Denham. *Acoustic Doppler Current Profiler Data Processing System*, 2011. URL `http://woodshole.er.usgs.gov/operations/stg/pubs/ADCPtools/`.

CRAN. The comprehensive r archive network, 2014a. URL `http://cran.r-project.org/`.

CRAN. Cran task view: Analysis of ecological and environmental data, 2014b. URL `http://cran.r-project.org/web/views/Environmetrics.html`.

Andrew P. Davison. Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science and Engineering*, 14(4):48–56, July-Aug 2012.

Michael Eby. *Earth System Climate Model 2.8*, 2013. URL `http://climate.uvic.ca/model/2.8/`.

John Evans. *Rslice*, 2007. URL http://marine.rutgers.edu/~jevans/rslice/rslice/doc/html/index.html.

Oliver Fringer, Phil Wolfram, and Yun Zhang. *SUNTANS*, 2013. URL http://sourceforge.net/projects/suntans/?source=navbar.

FVCOM. *The Unstructured Grid Finite Volume Community Ocean Model*, 2013. URL http://fvcom.smast.umassd.edu/fvcom/.

Jean-Pierre Gattuso, Jean-Marie Epitalon, Heloise Lavigne, James Orr, Bernard Gentili, Andreas Hofmann, Aurélien Proye, Karline Soetaert, and James Rae. *seacarb: seawater carbonate chemistry with R*, 2014. URL http://cran.r-project.org/web/packages/seacarb/index.html.

Marc-Oliver Gewaltig and Robert Cannon. Quality and sustainability of software tools in neuroscience. *arXiv preprint arXiv:1205.3025*, 2012.

Marc-Oliver Gewaltig and Robert Cannon. Current practice in software development for computational neuroscience and how to improve it. *PLoS computational biology*, 10(1): 1003376, 2014.

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2002.

GRASS Development Team. *GRASS*, 2014. URL http://grass.osgeo.org/.

Norman Herr. Summary of don norman's design principles, 2002. URL http://www.csun.edu/science/courses/671/bibliography/preece.html.

HYCOM. *HYBRID COORDINATE OCEAN MODEL*, 2013. URL https://hycom.org/hycom.

IRD. *A tool for interactive quality control of sea surface temperature and salinity*, 2013. URL http://www.ird.fr/us191/spip.php?article63.

Benjamin Jones. *ocean: Biophysical Oceanography Tools*, 2014. URL http://cran.r-project.org/web/packages/ocean/index.html.

Antoni Jordi. *Stony Brook Parallel Ocean Model*, 2012. URL http://www.imedea.uib-csic.es/users/toni/sbpom/index.php.

Vasudha Kapil, Spencer Smith, and Jacques Carette. State of the practice for software tools in medical imaging. *Journal of Digital Imaging*, Submitted 2014?

Dan Kelley. *oce: Analysis of Oceanographic data*, 2014. URL http://cran.r-project.org/web/packages/oce/index.html.

Diane Kelly. Industrial scientific software: A set of interviews on software development. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '13, pages 299–310, Riverton, NJ, USA, 2013. IBM Corp. URL http://dl.acm.org/citation.cfm?id=2555523.2555555.

D. E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984. doi: 10.1093/comjnl/27.2.97. URL http://comjnl.oxfordjournals.org/content/27/2/97.abstract.

Adam Lazzarato, Spencer Smith, and Jacques Carette. State of the practice for remote sensing software. Technical Report CAS-15-03-SS, McMaster University, January 2015.

Friedrich Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In Wolfgang Härdle and Bernd Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg, 2002. URL http://www.stat.uni-muenchen.de/~leisch/Sweave. ISBN 3-7908-1517-9.

J. M. Lilly. *JLAB*, 2012. URL http://www.jmlilly.net/jmlsoft.html.

Rick Luettich and Joannes Westerink. *ADCIRC, A (Parallel) ADvanced CIRCulation Model for Oceanic, Coastal and Estuarine Waters*. University of North Carolina at Chapel Hill and University of Notre Dame, 2013. URL http://adcirc.org/.

Madagascar. Madagascar, 2014. URL http://www.ahay.org/wiki/Main_Page.

T.J̃. McDougall and P.M̃. Barker. *Getting Started with TEOS-10 and the Gibbs Seawater (GSW) Oceanographic Toolbox*. SCOR/IAPS WG127, 2013. URL http://www.teos-10.org/software.htm#1.

MITgcm. *MIT General Circulation Model*, 2013. URL http://mitgcm.org/.

MOHID. *MOHID Studio*, 2014. URL http://www.actionmodulers.pt/default.aspx?canal=33.

Nedialko S. Nedialkov. VNODE-LP — a validated solver for initial value problems in ordinary differential equations. Technical Report CAS-06-06-NN, Department of Computing and Software, McMaster University, 1280 Main Street West, Hamilton, Ontario, L8S 4K1, 2006.

Nedialko S. Nedialkov. Implementing a Rigorous ODE Solver through Literate Programming. Technical Report CAS-10-02-NN, Department of Computing and Software, McMaster University, 2010.

NEMO. *Nucleus for European Modelling of the Ocean*, 2014. URL http://www.nemo-ocean.eu/.

NOAA-GFDL. *Generalized Ocean Layer Dynamics*, 2012. URL https://code.google.com/p/gold-omod/.

NOAA-GFDL. *The Modular Ocean Model*, 2014. URL http://mdl-mom5.herokuapp.com/web.

ODV. *Ocean Data View*, 2014. URL http://odv.awi.de/en/home/.

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM, vol. 15, no. 2, pp. 1053-1058*, December 1972.

Rich Pawlowicz. *A mapping package for Matlab*, 2013. URL http://www2.ocgy.ubc.ca/~rich/map.html.

Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

R Core Team. Writing R extensions, 2014. URL http://cran.r-project.org/doc/manuals/r-release/R-exts.html.

ROMS. *Regional Ocean Modeling System*, 2014. URL http://www.myroms.org/.

T.L. Saaty. How to make a decision: The analytic hierarchy process. *Interfaces*, 24(6): 19–43, 1994.

Spencer Smith, Yue Sun, and Jacques Carette. Comparing psychometrics software development between CRAN and other communities. Technical Report CAS-15-01-SS, McMaster University, January 2015.

Spencer Smith, Nirmitha Koothoor, and Ned Nedialkov. A document driven method for facilitating certification of scientific computing software. *IEEE Transactions on Software Engineering*, Submitted 2014.

Karline Soetaert. *OceanView: Visualisation of Oceanographic Data and Model Output*, 2014. URL http://cran.r-project.org/web/packages/OceanView/index.html.

MPAS Development Team. *The Model for Prediction Across Scales*, 2013. URL http://mpas-dev.github.io/.

Tech.plym.ac.uk. Matlab toolbox, 2014. URL http://www.tech.plym.ac.uk/spmc/links/matlab/matlab_toolbox.html#Oceanography.

Stuart H. Mann Triantaphyllou. Using the analytic hierarchy process for decision making in engineering applications. *International Journal of Industrial Engineering: Applications and Practice*, 2(1):35–44, 1995.

Lars Umlauf, Hans Burchard, and Karsten Bolding. *General Ocean Turbulence Model*, 2013. URL http://www.gotm.net/index.php.

USGS. *SeaGrid Orthogonal Grid Maker For Matlab*, 2013. URL http://woodshole.er.usgs.gov/operations/modeling/seagrid/.

VTK Team. *VTK Visualization Toolkit*. Kitware, 2014. URL http://www.vtk.org/.

WAFO. *WAFO*, 2011. URL http://www.maths.lth.se/matstat/wafo/index.html.

Hadley Wickham. testthat: Get started with testing. *The R Journal*, 3:5–10, 2011. URL http://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf.

Greg Wilson, D.A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H.D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumblet, Ben Waugh, Ethan P. White, and Paul Wilson. Best practices for scientific computing. *CoRR*, abs/1210.0530, 2013.

Gregory Wilson. Where's the real bottleneck in scientific computing? *American Scientist*, 94(1):5+, 2006. ISSN 0003-0996. doi: 10.1511/2006.1.5. URL http://dx.doi.org/10.1511/2006.1.5.

Woodshole.er.usgs.gov. Sea-mat, 2014. URL http://woodshole.er.usgs.gov/operations/sea-mat/.

Appendices

A Full grading template

The table below lists the full set of measures that are assessed for each software product. The measures are grouped under headings for each quality, and one for summary information. Following each measure, the type for a valid result is given in brackets. Many of the types are given as enumerated sets. For instance, the response on many of the questions is one of "yes," "no," or "unclear." The type "number" means natural number, a positive integer. The types for date and url are not explicitly defined, but they are what one would expect from their names. In some cases the response for a given question is not necessarily limited to one answer, such as the question on what platforms are supported by the software product. Case like this are indicated by "set of" preceding the type of an individual answer. The type in these cases are then the power set of the individual response type. In some cases a superscript * is used to indicate that a response of this type should be accompanied by explanatory text. For instance, if problems were caused by uninstall, the reviewer should note what problems were caused.

Table 5: Grading Template

Summary Information
Software name? (string)
URL? (url)
Educational institution (string)
Software purpose (string)
Number of developers (number)
How is the project funded (string)
Number of downloads for current version (number)
Release date (date)
Last updated (date)
Status ({alive, dead, unclear})
License ({GNU GPL, BSD, MIT, terms of use, trial, none, unclear})
Platforms (set of {Windows, Linux, OS X, Android, Other OS})
Category ({concept, public, private})
Development model ({open source, freeware, commercial})
Publications using the software (set of url)
Publications about the software (set of url)
Is source code available? ({yes, no})
Programming language(s) (set of {FORTRAN, Matlab, C, C++, Java, R, Ruby, Python, Cython, BASIC, Pascal, IDL, unclear})
Installability (Measured via installation on a virtual machine.)

Are there installation instructions? ({yes, no})
Are the installation instructions linear? ({yes, no, n/a})
Is there something in place to automate the installation? ({yes*, no})
Is there a specified way to validate the installation, such as a test suite? ({yes*, no})
How many steps were involved in the installation? (number)
How many software packages need to be installed before or during installation? (number)
Run uninstall, if available. Were any obvious problems caused? ({unavail, yes*, no})
Overall impression? ({1 .. 10})

Correctness and Verifiability

Are external libraries used? ({yes*, no, unclear})
Does the community have confidence in this library? ({yes, no, unclear})
Any reference to the requirements specifications of the program? ({yes*, no, unclear})
What tools or techniques are used to build confidence of correctness? (string)
If there is a getting started tutorial, is the output as expected? ({yes, no*, n/a})
Overall impression? ({1 .. 10})

Surface Reliability

Did the software "break" during installation? ({yes*, no})
Did the software "break" during the initial tutorial testing? ({yes*, no, n/a})
Overall impression? ({1 .. 10})

Surface Robustness

Does the software handle garbage input reasonably? ({yes, no*})
For any plain text input files, if all new lines are replaced with new lines and carriage returns, will the software handle this gracefully? ({yes, no*, n/a})
Overall impression? ({1 .. 10})

Surface Performance

Is there evidence that performance was considered? ({yes*, no})
Overall impression? ({1 .. 10})

Surface Usability

Is there a getting started tutorial? ({yes, no})
Is there a standard example that is explained? ({yes, no})
Is there a user manual? ({yes, no})
Does the application have the usual "look and feel" for the platform it is on? ({yes, no*})
Are there any features that show a lack of visibility? ({yes, no*})
Are expected user characteristics documented? ({yes, no})
What is the user support model? (string)
Overall impression? ({1 .. 10})

Maintainability

Is there a history of multiple versions of the software? ({yes, no, unclear})

Is there any information on how code is reviewed, or how to contribute? ({yes*, no})
Is there a changelog? ({yes, no})
What is the maintenance type? (set of {corrective, adaptive, perfective, unclear})
What issue tracking tool is employed? (set of {Trac, JIRA, Redmine, e-mail, discussion board, sourceforge, google code, git, none, unclear})
Are the majority of identified bugs fixed? ({yes, no*, unclear})
Which version control system is in use? ({svn, cvs, git, github, unclear})
Is there evidence that maintainability was considered in the design? ({yes*, no})
Are there code clones? ({yes*, no, unclear})
Overall impression? ({1 .. 10})

Reusability

Are any portions of the software used by another package? ({yes*, no})
Is there evidence that reusability was considered in the design? (API documented, web service, command line tools, ...) ({yes*, no, unclear})
Overall impression? ({1 .. 10})

Portability

What platforms is the software advertised to work on? (set of {Windows, Linux, OS X, Android, Other OS})
Are special steps taken in the source code to handle portability? ({yes*, no, n/a})
Is portability explicitly identified as NOT being important? ({yes, no})
Convincing evidence that portability has been achieved? ({yes*, no})
Overall impression? ({1 .. 10})

Surface Understandability (Based on 10 random source files)

Consistent indentation and formatting style? ({yes, no, n/a})
Explicit identification of a coding standard? ({yes*, no, n/a})
Are the code identifiers consistent, distinctive, and meaningful? ({yes, no*, n/a})
Are constants (other than 0 and 1) hard coded into the program? ({yes, no*, n/a})
Comments are clear, indicate what is being done, not how? ({yes, no*, n/a})
Is the name/URL of any algorithms used mentioned? ({yes, no*, n/a})
Parameters are in the same order for all functions? ({yes, no*, n/a})
Is code modularized? ({yes, no*, n/a})
Descriptive names of source code files? ({yes, no*, n/a})
Is a design document provided? ({yes*, no, n/a})
Overall impression? ({1 .. 10})

Interoperability

Does the software interoperate with external systems? ({yes*, no})
Is there a workflow that uses other softwares? ({yes*, no})
If there are external interactions, is the API clearly defined? ({yes*, no, n/a})
Overall impression? ({1 .. 10})

Visibility/Transparency
Is the development process defined? If yes, what process is used. ({yes*, no, n/a})
Ease of external examination relative to other products considered? ({1 .. 10})
Overall impression? ({1 .. 10})
Reproducibility
Is there a record of the environment used for their development and testing? ({yes*, no})
Is test data available for verification? ({yes, no})
Are automated tools used to capture experimental context? ({yes*, no})
Overall impression? ({1 .. 10})

B Summary of Measurements

A question mark (?) is used to indicate where a measurement's value was unclear.

B.1 Installability

Name	II	II linear	AI	Inst valid	# of Steps	# S/w lib	Uninst
oce	Yes	Yes	Yes	No	2	4	No
seacarb	Yes (Cran)	Yes	Yes	No	2	1	No
ocean	Yes (Cran)	Yes	Yes	No	2	5	No
OceanView	Yes (Cran)	Yes	Yes	No	2	4	No
SeaGrid	Yes	Yes	Yes	No	2	1	No
GSW	Yes	Yes	Yes	Yes	5	1	No
STAPLOT	Yes	Yes	Yes	No	7	1	No
sbPOM	Yes	No	Yes	No	17 minimum	9	No
WAFO	Yes	Yes	Yes	No	5	0	No
ADCP	Yes	Yes	N/A	No	2	1	No
JLAB	Yes	Yes	N/A	Yes	2	0	No
RSLICE	Yes	N/A	N/A	No	2	1	No
TSG-QC	Yes	Yes	N/A	No	8	4	No
M_Map	Yes	Yes	N/A	No	2	1	No
*NEMO	Yes	Yes	Yes	Yes	7	6	?
*GOLD	Yes	No	Yes	No	22	1	?
*MOM	Yes	Yes	Yes	Yes	5	3	?
*MPAS	Yes	Yes	Yes	Yes	7	7	?
*GOTM	Yes	Yes	Yes	Yes	6	9	?
MITgcm	Yes	Yes	Yes	Yes	6	1	No
*CICE	Yes	No	Yes	Yes	?	0	?
*ADCIRC	Yes	No	Yes	No	8	?	?
*ROMS	Yes	Yes	Yes	No	9	7	?
BOM	Yes	Yes	Yes	No	3 minimum	1	No
MOHID	Yes	Yes	Yes	No	1	0	No
ODV	Yes	Yes	Yes	No	1	1	No
SUNTANS	Yes	Yes	Yes	No	7	2	No
ESCM	Yes	Yes	Yes	Yes	8	2	No
*HYCOM	Yes	Yes	Yes	No	about 22	1	?
*FVCOM	Yes	Yes	Yes	No	10 minimum	?	?

Table 6: Installability – II: Installation instructions available, II linear: Linear installation instructions, AI: Automated installation, Inst valid: Tests for installation validation, # S/W lib: Number of software/libraries required for installation, Uninst: Any uninstallation problem?, *: Software that was not installed successfully.

B.2 Correctness and Verifiability

Name	Library	SRS	Evidence	Example
oce	Yes (Cran)	Yes	Manual	Yes
seacarb	No	No	Manual	No output given
ocean	Yes (Cran)	Yes	Manual	No output given
OceanView	Yes (Cran)	No	Manual	Yes
SeaGrid	?	No	No	?
GSW	Yes (IAPWS-09)	Yes	Manual	Yes
STAPLOT	?	No	No	Yes
sbPOM	?	No	No	?
WAFO	?	Yes	Manual	Yes
ADCP	Yes	Yes	Manual	Yes, no output given
JLAB	?	No	Manual	Yes, no output given
RSLICE	?	No	No	Yes, no input data
TSG-QC	Yes (M_Map, seawater)	No	No	Yes
M_Map	?	No	No	Yes
NEMO	Yes (OASIS and IOIPSL)	No	NEMO Book	?
GOLD	?	No	No	?
MOM	?	Yes	Elements of MOM 5	?
MPAS	?	No	API	?
GOTM	?	Yes	Manual	?
MITgcm	?	Yes	Manual	?
CICE	?	No	Manual	No
ADCIRC	?	Yes	No	?
ROMS	Yes	Yes	Manual	?
BOM	?	No	Manual	No standard example
MOHID	?	No	Manual	Yes
ODV	?	No	API	General example
SUNTANS	?	Guide	Guide	?
ESCM	Yes, MOM	No	Manual	No
HYCOM	?	Yes	Manual	?
FVCOM	Yes (GOTM)	No	Manual	?

Table 7: Correctness and Verifiability – Library: Use of standard libraries, SRS: Software requirements specification, Evidence: Evidence to build confidence?, Example: Standard example explained.

B.3 Reliability, Robustness and Performance

Name	Install	Test	Wrong I/P handling	Format	Perf
oce	No	No	Yes	Yes	?
seacarb	No	No	No	N/A	?
ocean	No	Yes	Yes	N/A	?
OceanView	No	No	Yes	N/A	?
SeaGrid	Yes	?	?	N/A	Yes
GSW	No	No	No	N/A	?
STAPLOT	No	No	Yes	N/A	?
sbPOM	No	N/A	?	N/A	Yes
WAFO	No	No	Yes	N/A	?
ADCP	No	No	?	N/A	?
JLAB	No	No	Yes	N/A	?
RSLICE	No	N/A	Yes	N/A	?
TSG-QC	No	No	Yes	N/A	?
M_Map	No	No	No	N/A	?
NEMO	Yes	?	?	N/A	Yes
GOLD	Yes	?	?	N/A	?
MOM	Yes	?	?	N/A	Yes
MPAS	Yes	?	?	N/A	Yes
GOTM	Yes	?	?	N/A	Yes
MITgcm	No	Yes	?	N/A	Yes
CICE	Yes	?	?	N/A	Yes
ADCIRC	Yes	?	?	N/A	Yes
ROMS	Yes	?	?	N/A	Yes
BOM	No	N/A	?	N/A	Yes
MOHID	No	No	Yes	N/A	?
ODV	No	No	No	N/A	?
SUNTANS	No	No	?	N/A	?
ESCM	No	N/A	?	N/A	Yes
HYCOM	Yes	?	?	N/A	Yes
FVCOM	Yes	?	?	N/A	Yes

Table 8: Reliablity, Robustness, And Performance – Install: Software break during installation, Test: Software break during initial tutorial testing, Wrong I/P handling: Handling of wrong input by software, Format: Can the software gracefully handle a change of the format of text input files where the end of line follows a different convention?, Perf: Evidence that performance was considered?

B.4 Usability

Name	Tut	Ex	UM	Look/feel	Vis	UC	User support
oce	Yes	Yes	Yes	Yes	Yes	No	Email, github
seacarb	No	Yes	Yes	Yes	Yes	No	Email
ocean	No	Yes	Yes	Yes	Yes	Yes	Email, git
OceanView	Yes	Yes	Yes	Yes	Yes	No	Email
SeaGrid	Yes	Yes	No	?	?	No	Email
GSW	Yes	Yes	Yes	Yes	Yes	No	Email
STAPLOT	Yes	Yes	No	Yes	No	No	Email
sbPOM	Yes	Yes	No	Yes	Yes	No	Email, forum
WAFO	Yes	Yes	Yes	Yes	Yes	No	Email, FAQ, Google code
ADCP	Yes	Yes	Yes	Yes	Yes	No	Email
JLAB	No	Yes	Yes	Yes	Yes	No	Email
RSLICE	Yes	Yes	Yes	Yes	No	No	No
TSG-QC	Yes	Yes	Yes	Yes	No	No	Email, tech support
M_Map	Yes	Yes	Yes	Yes	Yes	No	Email
NEMO	Yes	Yes	Yes	?	?	No	FAQ, User meeting, forum
GOLD	Yes	Yes	Yes	?	?	No	Email,Googlecode
MOM	Yes	Yes	Yes	?	?	No	Email, git
MPAS	Yes	Yes	Yes	?	?	No	Email, git
GOTM	No	No	Yes	?	?	No	Email
MITgcm	Yes	Yes	Yes	Yes	Yes	No	Email
CICE	No	No	Yes	?	?	No	FAQ,Email
ADCIRC	Yes	Yes	Yes	?	?	No	User Meeting, Email
ROMS	Yes	Yes	Yes	?	?	No	Forum, Email, FAQ
BOM	No	No	Yes	Yes	Yes	No	Email
MOHID	Yes	Yes	Yes	Yes	No	No	Email, forum
ODV	Yes	Yes	Yes	Yes	No	No	Forum, Email
SUNTANS	Yes	Yes	Yes	Yes	Yes	No	SourceForge, Email
ESCM	Yes	No	Yes	Yes	Yes	No	Email
HYCOM	Yes	Yes	Yes	?	?	No	Email, forum
FVCOM	Yes	Yes	Yes	?	?	No	Email, forum

Table 9: Usability – Tut: Getting started tutorial, Ex: Standard example, UM: User manual, Look/feel: Usual look and feel of software, Vis: Lack of visibility (Norman's principle), UC: User characteristics documented.

B.5 Maintainability

Name	VH	RC	Log	MT	Issue	Bugs	VS	Evidence	Clone
oce	Yes	Yes	Yes	C/P	Github	Yes	Github	Yes	No
seacarb	Yes	No	Yes	P	?	?	?	Yes	No
ocean	Yes	No	No	?	git	?	git	Yes	No
OceanView	No	No	No	?	?	?	?	Yes	No
SeaGrid	No	No	No	?	?	?	Svn	No	No
GSW	Yes	No	Yes	C/P	?	?	?	No	No
STAPLOT	No	No	Yes	P	?	?	?	No	No
sbPOM	No	No	Yes	C/P	?	?	?	Yes	No
WAFO	Yes	No	Yes	P	Google	No	SVN	Yes	No
ADCP	No	No	No	?	?	?	?	Yes	No
JLAB	No	No	Yes	C/P	?	?	?	No	No
RSLICE	No	Yes	Yes	C/P	?	?	?	No	No
TSG-QC	Yes	No	No	?	?	?	SVN	Yes	No
M_Map	Yes	No	No	?	?	?	?	Yes	No
NEMO	Yes	Yes	Yes	C/P	Trac	Yes	SVN	Yes	No
GOLD	No	No	No	No	Google	Yes	Google	Yes	No
MOM	Yes	Yes	No	?	YouTrack, git	Yes	git	Yes	No
MPAS	Yes	Yes	No	?	git	No	git	Yes	No
GOTM	Yes	Yes	Yes	?	?	?	?	Yes	No
MITgcm	Yes	Yes	Yes	C/P	?	?	CVS	Yes	No
CICE	Yes	Yes	No	?	Trac	?	?	Yes	No
ADCIRC	No	Yes	Yes	C/P	?	?	SVN	Yes	No
ROMS	Yes	No	Yes	C/P	Trac	Yes	SVN	Yes	No
BOM	Yes	No	Yes	C/P	?	?	?	No	No
MOHID	Yes	Yes	Yes	C/P	CodePlex	?	svn	Yes	No
ODV	Yes	No	Yes	C/P	?	?	?	?	?
SUNTANS	No	No	No	?	Sourceforge	No	SVN,CVS	No	No
ESCM	Yes	No	Yes	C/P	?	?	?	Yes	No
HYCOM	Yes	Yes	No	?	Google	?	Google	Yes	No
FVCOM	No	No	No	?	?	?	?	Yes	No

Table 10: Maintainability – VH: Versions history available, RC: Information on reviewing and contributing, Log: Change log available, MT: Maintenance type, Issue: Issue tracking tool, Bugs: Majority of bugs fixed, VS: Versioning system used, Evidence: Any evidence that maintainability was considered in design, Clone: Evidence of code clones, C: Corrective, P: Perfective, A: Adaptive

B.6 Reusability

Name	S/W reused by any package	Any evidence of reusability
oce	Yes	Yes
seacarb	Yes	Yes
ocean	No	Yes
OceanView	No	Yes
SeaGrid	?	No
GSW	?	Yes, API
STAPLOT	?	No
sbPOM	?	No
WAFO	?	No
ADCP	?	No
JLAB	?	Yes
RSLICE	?	Yes
TSG-QC	?	No
M_Map	?	No
NEMO	Yes	Yes
GOLD	?	No
MOM	?	No
MPAS	?	Yes
GOTM	?	No
MITgcm	?	Yes
CICE	?	No
ADCIRC	?	No
ROMS	?	No
BOM	?	No
MOHID	?	No
ODV	?	Yes
SUNTANS	?	No
ESCM	?	No
HYCOM	?	No
FVCOM	?	No

Table 11: Reusability

B.7 Portability

Name	Platforms	Port in code	Port not imp	Evid in doc
oce	Win, Linux, Mac	Branch in repository	No	Yes
seacarb	Win, Linux, Mac	Branch in repository	No	Yes
ocean	Win, Linux, Mac	Branch in repository	No	Yes
OceanView	Win, Linux, Mac	Branch in repository	No	Yes
SeaGrid	Win, Linux	Branch in repository	No	Yes
GSW	Win, Linux, Mac	No	No	No
STAPLOT	Win, Linux, Mac	No	No	No
sbPOM	Linux	Make file variable	No	Yes
WAFO	Win, Linux, Mac	No	No	No
ADCP	Win, Linux, Mac	No	No	No
JLAB	Win, Linux, Mac	No	No	No
RSLICE	Win, Linux, Mac	No	No	No
TSG-QC	Win, Linux, Mac	Branch in different folder	No	No
M_Map	Win, Linux, Mac	No	No	No
NEMO	Linux	Make file variables	No	Yes
GOLD	Linux	No	No	No
MOM	Linux	Make file variables	No	Yes
MPAS	Linux	Make file variables	No	Yes
GOTM	Win, Linux, Mac	Make file variables	No	Yes
MITgcm	Linux	Make file variable	No	Yes
CICE	Linux	Different Make file	No	No
ADCIRC	Linux	No	No	Yes
ROMS	Linux	Make file variables	No	No
BOM	Linux	Make file variables	No	Yes
MOHID	Win, Linux, Mac	Not for current version	No	No
ODV	Win, Linux, Mac	Branch in repository	No	Yes
SUNTANS	Linux	No	No	No
ESCM	Linux	Make file variables	No	No
HYCOM	Linux	Make file variable	No	Yes
FVCOM	Linux	Make file variable	No	Yes

Table 12: Portability – Platforms: Platforms specified for the software to work on, Port in code: How portability is handled (if source code given), Port not imp: Portability explicitly identified as not important, Evid in doc: Convincing evidence present in documentation for portability.

B.8 Understandability (of code)

Name	Frmt	Std	Id	Constant	PC	Algo	Mod	Para	SC	DD
oce	Yes	No	Yes	No	No	Few	Yes	Yes	Yes	Yes
seacarb	No	No	Yes	No	No	No	Yes	Yes	Yes	Yes
ocean	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	Yes
OceanView	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes	No
SeaGrid	No	No	Yes	No	Yes	No	Yes	Yes	Yes	No
GSW	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
STAPLOT	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
sbPOM	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
WAFO	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	No
ADCP	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	No
JLAB	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	No
RSLICE	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
TSG-QC	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
M_Map	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	No
NEMO	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	No
GOLD	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
MOM	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
MPAS	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes
GOTM	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
MITgcm	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes
CICE	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No
ADCIRC	Yes	Yes	Yes	No	Some	No	Yes	Yes	Yes	No
ROMS	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
BOM	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
MOHID	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	No
ODV	Yes	No	Yes	No	Yes	No	Yes	?	Yes	Yes
SUNTANS	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
ESCM	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
HYCOM	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes
FVCOM	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No

Table 13: Understandability (of code) – Frmt: Consistent Identation and formatting, Std: Explicit coding standard, Id: Distinctive, meaningful identifiers name, Constant: Constants (other than 0 or 1) hard coded, PC: Proper comments, Algo: Reference to algorithm used, Mod: Code is modularised, Para: Parameters are in same order, SC: Descriptive names of source code files, DD: Design document present.

B.9 Interoperability

Name	External package	Workflow uses other s/w	API
oce	No	Yes	Yes
seacarb	No	No	Yes
ocean	No	Yes	Yes
OceanView	No	Yes	Yes
SeaGrid	No	No	No
GSW	No	No	Yes
STAPLOT	No	No	No
sbPOM	No	No	No
WAFO	No	No	No
ADCP	No	No	No
JLAB	No	No	No
RSLICE	No	No	No
TSG-QC	No	No	No
M_Map	No	No	No
NEMO	Yes	Yes	No
GOLD	No	No	No
MOM	?	Yes	No
MPAS	?	?	No
GOTM	No	Yes	No
MITgcm	No	No	No
CICE	?	?	No
ADCIRC	No	Yes	No
ROMS	No	No	No
BOM	No	No	No
MOHID	No	Yes, plugin software	No
ODV	Yes	No	No
SUNTANS	No	Yes	No
ESCM	No	Yes, MOM	No
HYCOM	No	No	No
FVCOM	No	Yes	No

Table 14: Interoperability – External package: Software communicates with external package, Workflow uses other s/w: Workflow uses other software, API: External interactions (API) defined.

B.10 Visibility/Transparency

Name	Dev process defined	Ease of ext exam
oce	No	10
seacarb	No	8
ocean	No	8
OceanView	No	8
SeaGrid	No	8
GSW	No	9
STAPLOT	No	8
sbPOM	No	8
WAFO	No	9
ADCP	No	8
JLAB	No	7
RSLICE	No	7
TSG-QC	No	8
M_Map	No	8
NEMO	No	8
GOLD	No	7
MOM	Yes, continuous integration	8
MPAS	Yes	9
GOTM	No	7
MITgcm	Yes	9
CICE	Yes	7
ADCIRC	No	8
ROMS	No	8
BOM	No	7
MOHID	No	8
ODV	No	7
SUNTANS	No	8
ESCM	No	7
HYCOM	No	8
FVCOM	No	8

Table 15: Visibility and Transparency – Dev process defined: Development process defined, Ease of ext exam: Ease of examination relative to other software (out of 10).

B.11 Reproducibility

Name	Dev env record	Test data for verification	Auto tools
oce	Only for testing	No	No
seacarb	Yes, testing	No	No
ocean	Yes, testing	No	No
OceanView	Yes, testing	No	No
SeaGrid	No	No	No
GSW	No	No	No
STAPLOT	No	No	No
sbPOM	No	No	No
WAFO	No	No	No
ADCP	No	No	No
JLAB	No	No	No
RSLICE	Yes, but not very detailed	No	No
TSG-QC	No	No	No
M_Map	No	No	No
NEMO	Only for testing	No	No
GOLD	No	No	No
MOM	No	No	No
MPAS	No	No	No
GOTM	No	No	No
MITgcm	Yes	Yes	Yes, test scripts
CICE	No	No	No
ADCIRC	No	No	No
ROMS	No	No	No
BOM	No	No	No
MOHID	No	No	No
ODV	No	No	No
SUNTANS	No	No	No
ESCM	No	No	No
HYCOM	No	No	No
FVCOM	No	No	No

Table 16: Reproducibility – Dev env record: Record of development environment, Test data for verification: Availability of test data for verification, Auto tools: Automated tools (like MADAGASCAR) used to capture experimental data.