Advances in Engineering Software 100 (2016) 53–71

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

State of the practice for mesh generation and mesh processing software

W. Spencer Smith*, D. Adam Lazzarato, Jacques Carette

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

ARTICLE INFO

Article history:
Received 24 October 2015
Revised 2 May 2016
Accepted 12 June 2016

Keywords:
Mesh generation
Scientific computing
Software engineering
Software quality
Analytic hierarchy process

ABSTRACT

We analyze the state of development practices for Mesh Generation and Mesh Processing (MGMP) software by comparing 27 MGMP projects. The analysis employs a reproducible method based on a grading template of 56 questions covering 13 software qualities. The software is ranked using the Analytic Hierarchy Process (AHP), a multicriteria decision making method appropriate for cases with a mix of qualitative and quantitative factors. The results reveal concerns regarding the maintainability, usability, reusability and performance of some MGMP software. Five recommendations are presented as feedback to the MGMP community: (i) Use an issue tracker for bug management and support requests. (ii) Document performance measures. (iii) Increase the use of libraries to promote software re-use to avoid "re-inventing the wheel." (iv) Improve reproducibility by recording the set up details for the development and testing environments. (v) Improve confidence in correctness through requirements specification, formal specification languages, and automated testing.

© 2016 Elsevier Ltd. All rights reserved.

1. Introduction

This paper analyzes the state of development practice for Mesh Generation and Mesh Processing (MGMP) software. MGMP is used to discretize a given geometric domain into a mesh consisting of a set of smaller simpler shapes, such as triangles, quadrilaterals, or tetrahedrals. Meshes are used in such areas as finite element computations and computational graphics. The data structures and algorithms for MGMP can get complicated. The complexity of MGMP software raises concerns regarding correctness, reliability and performance. Moreover, the utility and importance of MGMP justifies concerns about the maintainability and reusability of this software. To address these concerns requires systematic and rigorous software development practices.

In the analysis that follows, we have aimed to be objective. Although two of the authors of this paper have some experience in the domain of mesh generation and processing, MGMP is not their primary research area, and they are not part of the MGMP community. Instead, we consider ourselves as experts in Software Engineering (SE) applied to Scientific Computation (SC) software. We have no prior attachment to any of the software examined in this paper. To keep the evaluations fair, the sole source of the information for each product is what is available in the product itself as well as what is obtainable from searching the Internet.

We evaluated 27 products using a grading template based on 13 different criteria (called "qualities" in the software engineering literature). Given our role as outsiders to the MGMP community, and to keep the focus on software engineering issues, the software is not graded based on functionality. (An older review of MGMP software based on functionality can be found in [37].) We graded the available software artifacts and the development processes against standard SE principles and practices. To select the software for grading, we used a list produced by a domain expert, as discussed in Section 3.1. The grading consists of pairwise comparisons between each of the software products using a multicriteria decision analysis process. The rankings from the decision analysis were then used to find trends between the software products.

The methods we used are an expanded and refined version of those presented by Gewaltig and Cannon [19,20] for computational neuroscience. In their work, Gewaltig and Cannon frequently found a gap between developers and users, with respect to their expectations for software quality. We looked at what kind of quality gap exists within the MGMP domain. The gap in computational neuroscience, where the majority of software is created by professional end user developers [51], may be due to the developers emphasis on their science, instead of on software development best practices.

In general, software developers who write scientific computation software do not follow the practices advocated by software

* Corresponding author.
E-mail address: smiths@mcmaster.ca (W. Spencer Smith).

http://dx.doi.org/10.1016/j.advengsoft.2016.06.008
0965-9978/© 2016 Elsevier Ltd. All rights reserved.

engineers [28,29,64]. As observed by Segal [52], "there is no discrete phase of requirements gathering or of software evaluation. Testing is of the cursory nature which would enrage a software engineer." Segals description is reinforced by a survey of approximately 160 SC developers [61], which showed that only 12% of development time is devoted to requirements specification and that only 10% of SC developers employ formal specification techniques.

Not only are SE methods not used in SC, they are often perceived as not useful. For instance, for SC software, Roache [43, p. 373] considers as counterproductive the writing of documentation at each stage of software development, although this is often advocated in SE. As the field studies of Segal [52] show, interaction between SE and SC can be problematic because each side fails to meet the expectations of the other. For instance, communication between SC developers and SE practitioners is challenging when it comes to requirements. A communication barrier exists as the scientists cannot precisely convey how the requirements will evolve. Not correctly articulating requirements, or changing requirements midway through a project, greatly impacts the productivity of the development team [50]. When engineers create software, the resulting development artifacts, such as user manuals and introductory examples, are not sufficient for the scientists to understand the product [53]. When end users (scientists) develop the software product, the situation is not better, since their training in science has not prepared them to consider important software qualities. In this paper we evaluate the current use of SE techniques and tools in MGMP. Moreover, for any recommendations that are made, we aim to make them useful by considering the needs of SC practitioners.

This paper has been written as part of a larger project analyzing the state of practice in multiple SC domains. Throughout this paper, some results for MGMP software will be contrasted with the results from the Remote Sensing (RS) domain [32]. RS and MGMP software are not obviously related based on their purpose, but they are similar in that each community produces SC software to solve problems. Since these two scientific domains are subject to the same gradings, contrasting the results gives a basic sense of the differences in practices between domains. Other domains that have been analyzed using the methods shown in this paper include psychometrics software [58] and oceanography software [57].

The remainder of this article is organized as follows: Section 2 provides some background information and mentions previous work. Our method is explained in Section 3. A summary of our results is presented in Section 4 and our recommendations are detailed in Section 5. Concluding thoughts are found in Section 6.

2. Background

Our grading template is based on 13 software qualities, which are summarized below, followed by an overview of the Analytic Hierarchy Process (AHP).

2.1. Software qualities

Our analysis is centered around a set of what software engineers call *software qualities*. These qualities highlight the desirable nonfunctional properties for software artifacts, which include both documentation and code. Some qualities, such as visibility, apply to the process used for developing the software. The following list of qualities is based on Ghezzi et al. [21], with the terms defined in the same order as in the source document. Excluded from this list are qualities that cannot be measured within the scope of the current study, such as productivity and timeliness. To the list from Ghezzi et al. [21], we have added two qualities important for SC: installability and reproducibility.

Installability A measure of the ease of installation.

Correctness and verifiability[1] Software is correct if the specification is perfectly adhered to. Software is not correct if it deviates from the specification. Verifiability is the ease with which properties of the software can be ascertained.

Reliability The probability that the software will meet its requirements under a given usage profile.

Robustness A measure of whether the software behaves "gracefully" during unexpected situations, such as when invalid data is input.

Performance A measure of the storage necessary and time required for the software to solve large problems.

Usability A measure of user-friendliness.

Maintainability The effort necessary to find and repair errors and to add features to an operational program.

Reusability The ease with which one program can be used to create another.

Portability The effort needed to run the software in a new environment.

Understandability The ease with which a programmer can understand the code.

Interoperability A measure of how smoothly a software product can work with external products or systems.

Visibility The ease of determining the current status of a project's development.

Reproducibility The ease of recreating software results in the future. SC code results should meet the scientific method requirement of repeatability. Scientific code must be robust to changing implementation details [12].

The above software qualities come from SE; they apply to any class of software. Wilson et al. [63] instead focus on issues specific to SC software. They provide a list of eight best practices for developers of SC software. Ideas from this list were used in the creation of our grading template. For instance, part of our measure for maintainability is looking for utilization of an issue tracker, as advocated by Wilson et al. [63].

2.2. Analytic hierarchy process

The objective of the Analytic Hierarchy Process (AHP) is decision making when comparing multiple options based on multiple criteria [47]. In the current work, AHP is used for comparing software products based on each of the identified qualities. AHP works well for this, since it focuses on relative comparisons, rather than requiring an unattainable unified scale for measuring quality. AHP starts with sets of n *options* and m *criteria*. In our project there are 27 software products ($n = 27$) and 13 criteria ($m = 13$). Selection of a specific software product requires prioritizing the criteria, but we do not emphasize this, since priorities are project specific. Instead, we focus on the next step in the AHP, which will give us a ranking of the software options for each criterion (quality). In this step, for each of the criterion, a pairwise analysis is performed between each of the options, in the form of an nxn matrix a. The value of a_{jk} ranges from 1, when options j and k are equally successful at achieving the criterion, to 9, when option j is extremely (maximally) more successful at achieving the criterion than option k. Saaty [47] shows the interpretation of the other values, between 1 and 9.

In our work, $a_{kj} = 1/a_{jk}$. Matrix a is then used to create matrix b, where $b_{jk} = a_{jk}/\sum(a_{.k})$. The dot notation ($\cdot$) stands for the entire row. The entries in b are then averaged to determine the overall score for each option for the given criterion. This information can be combined with the priorities to select an option, or the

[1] [21] separates these two, but external evidence for these are the same, so we have joined them here.

final scores can be used to create a ranking for each of the options against each of the criteria.

3. Method

After an overview of the method used to select the software for analysis, we summarize the grading template used for the analysis. Finally we highlight of the process used to grade each product.

3.1. Software product selection

We used Robert Schneiders' list of "Mesh generation and Grid Generation Software" [49] as a starting point. This list consists of free, open source and commercial software. Not all of the links were used: codes last updated in 2010, or earlier, were excluded. This left 25 public domain software products on the list. We removed OpenVolumeMesh [30] from the public domain list, since it is a mesh data structure, and we added OpenFlipper [34], which processes geometric data, using the OpenVolumeMesh data structure. For budgetary reasons, commercial products without free trial periods were also excluded. Of the 59 commercial projects, only 2 provided trial licenses. Therefore, 27 software products were selected. The full list is given in Section 4.

This selection process unfortunately means that our results are not necessarily representative of what all users of MGMP software will experience. First, because the list is not completely current, and secondly because of the under-representation of commercial software.

Another aspect is that some MGMP software explicitly hides the mesh, as they really are about solving a larger problem (PDEs, etc.). We have made sure that the products we have selected are explicitly MGMP softwares that allow the user fine control over the meshing process.

The sample size is large enough that some reliable conclusions can still be drawn, at least for the non-commercial products. The commercial products were kept, to see if there was a large difference in the results.

3.2. Grading template

Fig. 1 shows an excerpt from the grading template. The full template, consisting of 56 questions, can be found in Appendix A and at https://github.com/adamlazz/DomainX. The questions in the template were designed to be unambiguous, quantifiable and measurable with limited time and domain knowledge. The measures are grouped under headings for each quality, and one for summary information. 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 for many of the questions is one of "yes," "no," or "unclear." The type "number" means a natural number. 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. 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 the problems were.

The first section of the template summarizes general information, such as the software name, number of developers, etc. A project is defined as *alive* if it has been updated in the last 18 months, and *dead* otherwise. This time frame was selected because it coincides with the usual time for operating system updates. We follow the definitions given by Gewaltig and Cannon [19] for the categories of *public*, for software intended for public use, *private*, for software aimed only at a specific group, and *concept*, for software written simply to demonstrate algorithms or concepts. The three categories of development models are: *open source*, where source code is freely available under an open source license; *freeware*, where a binary or executable is provided for free; and, *commercial*, where the user must pay for the software product.

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

3.3. Grading process

As part of the grading process, each product is assigned a number from 1 to 10 for each quality. This number represents the grader's overall impression on how well this quality was achieved based on the measurements, past experiences and the other MGMP products. The following guidelines are used to help the reviewer keep their grading uniform:

- The grader should spend from 1 to 3 h with each product.
- When no source code is available 1 is awarded for understandability (of the code).
- For the qualities of performance, portability and reusability, a grade of 5 is assessed if the developer has not explicitly mentioned means to deal with these qualities. With the time available for grading, these qualities cannot be fully assessed; therefore it would be unfair to dock marks for poor performance. However, given no evidence to the contrary, we cannot award marks either.

The overall impression grades for each quality are the basis for the AHP results. The AHP ranking is calculated based on the differences between the overall impression grades. If the grades are equal, then the AHP result is 1, to represent that the products have the same performance for this quality. The use of AHP allows us to smooth out differences between different graders, as long as the relative trends between reviewers are the same. The absolute value of the overall impression grade is not relevant, only how it compares to the other grades.

To demonstrate that the grading process is reasonably reproducible, grading of 5 products was done by a second reviewer. The ranking via this independent review was almost identical to the original ranking. The main source of difference between the two gradings was the interpretation of the definition of correctness, and specifically what a requirements specification document entails. As long as each grader uses consistent definitions, the relative comparisons in the AHP results will be consistent between graders. Changes in perceived visibility of the software product also played a part in differences between grades. If information is hard to find this can hurt a product's grades, since not all reviewers will be able to find it.

4. Summary of results

A summary of the measurements for the 27 mesh generators can be found in Appendix B, with the complete data available at https://github.com/adamlazz/DomainX. General information about the software is reproduced in Table 1. Of the 27 software products,

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})
...

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 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})
..

Fig. 1. Excerpt from grading template

Table 1
General information regarding mesh generation software.

Name	Mesh types	Stat.	Lic.	Language	Type
ADMesh [26]	T, S	√	OS	C	T
CGAL [11]	T, S, Te	√	OS	C++	L
CGM [17]	–	√	OS	C++	L
Discretizer [7]	3	√	OS	Ruby	T
DistMesh [40]	T, Te	✗	OS	MATLAB	T
enGrid [15]	Te	√	OS	C++	T
EZ4U [22]	Q	√	F	N/A	T
Geompack++ [27]	T, Q, S, Te, H	√	F	Fortran, C++	T
GMSH [18]	T, Te	√	OS	C++	T
iso2mesh [16]	S, Te	✗	OS	MATLAB	T
Mefisto [39]	T, Te, 2, 3	✗	OS	Fortran, C	T
MeshGenC++ [45]	T, Te	√	OS	C++	T
MeshLab [41]	T	√	OS	C++	T
MMesh3D [33]	T, Q, H, 2, 3	√	OS	C	T
OpenFlipper [34]	T, Q, H, 2, 3	√	OS	C++	T
Overture [25]	2, 3	√	OS	C++	L
Pamgen [62]	Q, H	√	OS	C	L
Qhull [4]	T	✗	OS	C, IDL	B
Seagrid [55]	2	✗	OS	MATLAB	T
snappyHexMesh [36]	Te	√	OS	C++	T
TetGen [54]	Te	✗	OS	C++	T
TriGrid [13]	T	✗	OS	Fortran, C, C++	T
UGRID [24]	T, 2	✗	F	C++	T
UNAMalla [48]	2	√	F		T
ViennaGrid [46]	–	√	OS	C++	L
Algor [3]	T, Q, S, Te, H	√	C	N/A	T
Argus ONE [2]	T, Q, 2	✗	C	N/A	T

15 are associated with an educational institution. These institutions are the workplaces of the developers, or they are the source of the financial support for the software product. Twenty one (21) of the software products are open source (OS), 4 are freeware (F), and 2 are commercial (C). Unlike software in other domains, such as remote sensing [32], MGMP software cannot be split into distinct sets of products with different scopes and purposes. All mesh generation software follow a similar pattern, in which the user utilizes a data structure or graphical user interface to create a representation of a mesh. The mesh can then be examined, or used in other applications, such as computational fluid dynamics. The main differences between meshing software products is which types of meshes the software supports. Schneiders [49] presents this information on the source list, noting when a software supports triangular (T), quadrilateral (Q), tetrahedral (Te), unstructured hexahedral (H), 2D structured (2), 3D structured (3) and/or surface meshes (S). The packages that do not show an entry for the mesh type are not mesh generators, but rather mesh generator related geometry function libraries and generic data structures. For space reasons, in the table we use √ to mean *alive* and ✗ to mean *dead*; the last column indicates whether this is a Tool (T), a Library (L) or both (B).

In general, we have observed that

- Mesh generation software is often created in small teams. 18 of the 27 projects have 5 or fewer developers. 16 projects have only one or two developers.
- The GNU GPL is the most popular license, in use by 12 open source products. Three open source projects use the BSD license, and one uses the MIT license. Other projects, namely Geompack++, Mefisto, Qhull, Seagrid, and UGRID provide their own licenses as an End User License Agreement (EULA) or terms of use.
- Windows is supported natively in 22 projects, with the other projects (CGM, MMesh3D, Overture, Pamgen, snappyHexMesh) supporting Linux or Unix environments only.
- 6 are libraries, 22 are tools, with one (Qhull) being both.

In what follows, we outline the main results for each software quality. When the results could easily be improved, we make some recommendations, which will also be gathered at the end for easy reference (and give us a chance to expand on what we mean).

4.1. Installability

The installability of MGMP software varies. 24 of the 27 graded MGMP software products contained installation instructions, with 18 of the 24 presenting the instructions in a linear sequence of steps, a frequency much higher than for remote sensing [32]. Some of the software, such as Discretizer, MeshLab and TriGrid had a relatively simple automated installation process. Makefiles, packages and other build scripts automated the installation process for 20 software products. Alternatively, when pre-packaged executables were shipped, little installation was necessary. Such was the case for UGRID and DistMesh, which contain no installation instructions because the steps are relatively few and simple.

After installation, there are just 3 software products that specify ways to ensure the installation is valid: MeshGenC++, Overture, and Pamgen. For each of these three cases, this validation involves running a trivial example. Uninstallation automation is not provided in 21 of the 27 graded products. In some cases this automation was not necessary, since executables or folders fully encapsulate the project and can be deleted to uninstall the product.

Unamalla is an example of poor installability. Like TetGen, Unamalla requires the user to fill in a form with personal information before the product was downloaded. This is within the developer's rights, but it also forms a hurdle for users to jump before installation. The user is also required to fill in a captcha to ensure the user is human and to stop automated abuse in the submission form. Too many hurdles, and the potential addition of the human element, has a negative impact on installability.

4.2. Correctness and verifiability

The AHP results for correctness and verifiability are given in Fig. 2. One measure for correctness and verifiability is whether there is a requirements specification. 11 of the 27 graded software products specified the behavior of the software product through a requirements specification. This number is much higher than was observed in the domain of remote sensing [32]. These documents were almost never referred to explicitly as "requirements specification documents," but in all cases, the authors were looking to adhere to a specific algorithm (e.g. the Quickhull algorithm in Qhull), file format (e.g. GMSH), or to fully explain the functionality of the software and underlying mathematical principles written in an academic fashion (e.g. DistMesh, snappyHexMesh, and TetGen).

Other evidence of correctness comes in the form of referenced research in user documentation, as seen in Overture and MeshGenC++. Citing peer-reviewed academic work increases confidence in correctness because, while the software is not guaranteed to fully adhere to the research, at the very least the output of the software that appears in the publications has been peer reviewed. SC methods of increasing reliability (accuracy) exist as well, such as in Qhull, which handles floating point roundoff errors during calculations. Or in CGAL, which uses the "exact computation paradigm" to compute numbers to arbitrary precision. These methods build confidence. The use of standard libraries, which the community has confidence in, are in use by 13 of the 27 software products. However, these libraries are often not MGMP domain-specific, since their purpose is to help with the UI, portability or graphics processing.

Whenever there was a getting started tutorial for the user (10/27 cases), our testing results always matched the developer's intended results. This fact increases confidence in correctness, but

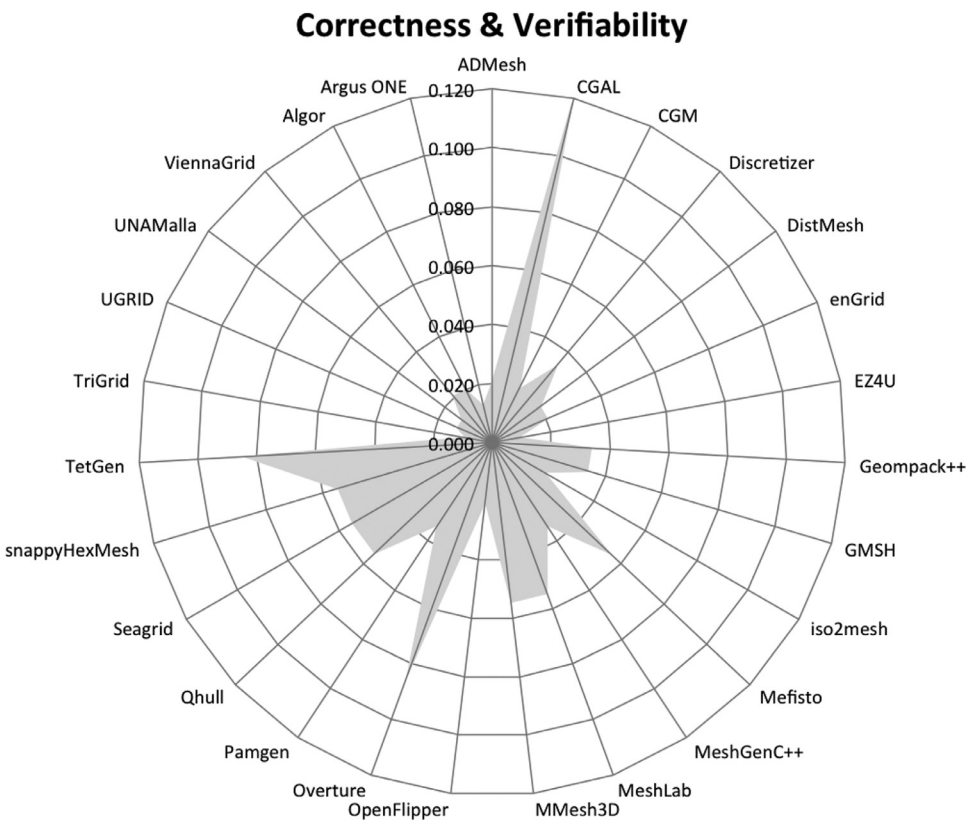

Fig. 2. AHP results for correctness

real confidence in correctness requires a comprehensive analysis of the product, not simply the results of one test case.

We thus recommend that more developers of MGMP software follow this trend and explicitly document their requirements. Furthermore, we also recommend that simple examples (such as those found in tutorials) should actually be made into an automated test suite, to verify that the software works properly on the current machine.

4.3. Reliability

Reliability is strong on the surface, during our short interaction with the graded MGMP software products. As mentioned above, there were several software products that "broke" during installation. Specifically, EZ4U, iso2mesh, Overture, TriGrid, UGRID and Unamalla. The other software products installed without any unexpected behavior. Once the software products were installed, they performed as expected, except for EZ4U, which crashed during initial testing.

4.4. Robustness

Surface robustness was achieved in all 27 of the software products graded. Purposely making typos and using broken input, we were able to cause error conditions in all 27 products. All of the errors were handled appropriately. MGMP software seems to do a good job of preparing for unexpected inputs and adverse conditions.

4.5. Performance

As seen in Fig. 3, performance is a quality that mesh generation software products rarely *explicitly* address. Out of the 27 graded software products, 18 did not contain any evidence that performance was considered during development. In some cases, like MeshGenC++, TetGen, UGRID, and Algor, the product is advertised as fast, but there is no explicit supporting quantitative evidence presented by the developers. Also, "fast" calculations are not necessarily indicative of good performance, if accuracy suffers. Pamgen and snappyHexMesh contain parallel implementations, which does increase performance. When parallelized appropriately, concurrent calculations increase speed (although we did not test this ourselves).

Since the developers of MGMP software often do extensive work regarding performance, they should document this, and make this available. Such quantitative information would be much more convincing that performance was a serious concern than simply stating that the product is "fast". Even better would be to embed explicit test cases, with accompanying measurements, as part of the complete product.

4.6. Usability

The results of surface usability are mixed. Only 23 projects have a user manual, which provides a detailed look at the function and purpose of the software product. This frequency is worse than what was observed in the domain of remote sensing [32]. EZ4U's user manual was written in Spanish. This cannot fairly be held against the developers, although the manual is of no use to English speakers. The 4 projects without a user manual are: MeshGenC++, Seagrid, Unamalla, Algor (commercial, the user manual likely comes with the purchased version). Even fewer (14/27) contained a simple getting started tutorial aimed at first-time users. Combined with the fact that there is almost never any information regarding expected user characteristics, it could be hard for some users to begin using some MGMP software.

Usability of complex software is significantly impact by how technical support is performed; a product is definitely more usable when the user can have questions answered. Every product has a support method, of varying degrees of usefulness. 8 soft-

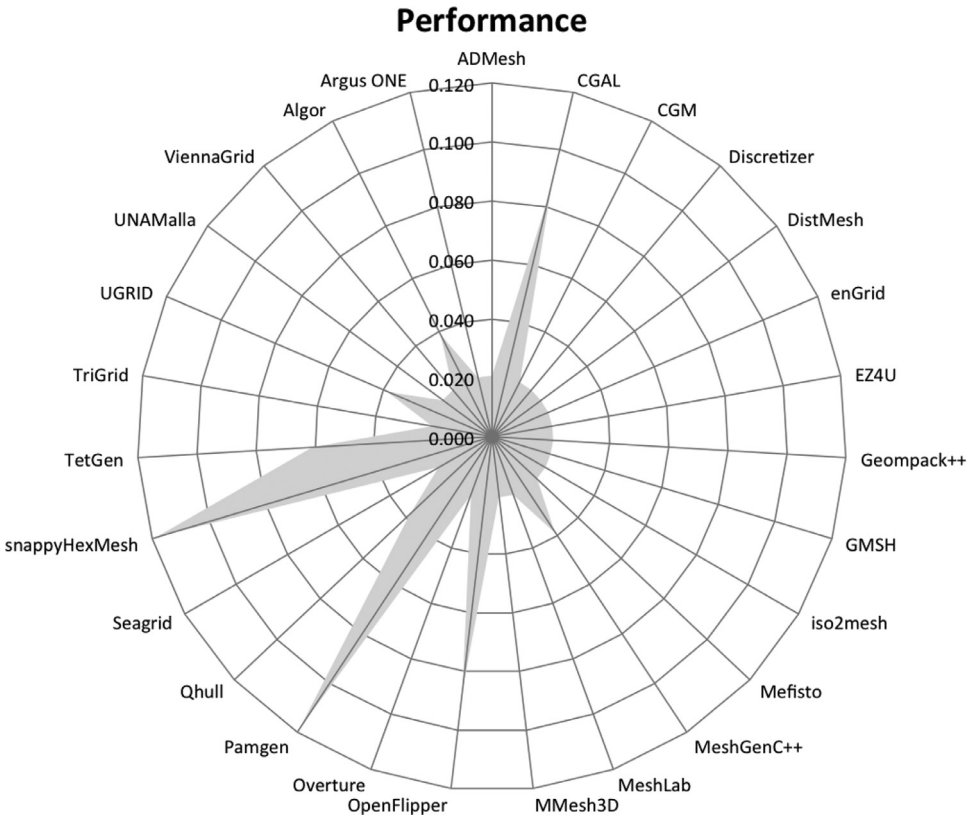

Fig. 3. AHP results for performance.

ware products use only email as a method of support (DistMesh, EZ4U, GeomPack++, Mefisto, MMesh3D, TriGrid, UGRID, UNAMalla). Email does not provide a public record of the conversations that take place. This hurts both usability and maintainability because users cannot see the questions asked by other users, and therefore developers may be tasked with replying to similar support requests multiple times. Other methods of support include mailing lists and FAQ pages, and even paid support and/or courses for snappyHexMesh, TetGen and Algor (commercial).

With a few minor exceptions, the graded MGMP software stuck to the "look and feel" of the platform that was being tested on. This goes for both GUI applications, like Mefisto or Unamalla, as well as for the programming libraries and/or command line applications, like ADMesh, or CGAL. There were a few minor issues with feature visibility, as defined by Norman [35]. The result is sensible, decently designed software. Products that suffer on usability do not suffer because of the design, rather the grades suffer for the lack of support methods and documentation material, as discussed previously.

When providing software for others to use, even for free, such software is not really *usable* if it does not have a user manual. Furthermore, especially for software as complex as that considered here, some kind of technical support mechanism is also needed.

4.7. Maintainability

Fig. 4 clearly shows that maintainability is very uneven amongst the products we measured. For example, of those products with multiple versions, 12 do not make the previous versions available to the public. This fact hurts reproducibility as well as maintainability because users cannot easily test the current version of the product against past releases. This can have a negative effect on the quality of bug reports.

Issue trackers are used by 11 of the 27 products. Some industry-standard issue trackers are in use, specifically GitHub issues (ADMesh, enGrid, Viennagrid), Trac (CGM, GMSH), and BitBucket (MeshGenC++). The issue trackers are most often used for corrective and perfective maintenance. Generally, externally visible signs of use of issue trackers correlate well with maintainability.

Sixteen (16) projects did not use issue trackers. In some cases, this information was completely absent, though QHull and TriGrid presented all known issues on self-made, static web pages. These systems involve more initial setup than using, say, GitHub issues, and users or other developers cannot add to this list without contacting the developer. Therefore, these systems are not as interactive as full-fledged issue trackers, to the detriment of the project's maintainability.

The ability to review and revert changes is made simple by the use of a version control system. Git and SVN are in use by 8 projects each; eleven projects do not use a version control system. It is possible that these systems are in use privately, and the released software is simply a snapshot of a revision in the system. However, there is no explicit evidence of this from the developers.

As the use of both issue trackers and version control systems is so easy, it is really inexcusable to not use them. Anyone considering using some software product for more than a simple experiment should eschew products which show no sign of having used these basic tools during development.

4.8. Reusability

Reusability is considered in a few packages. Plugin functionality is available in MeshLab, as well as the two commercial products: Algor and Argus ONE. Users may reuse existing portions of the software product to create their own custom functionality. There also exist frameworks, such as CGAL, CGM, Overture framework and Pamgen (a part of Trilinos) that allow users to create their

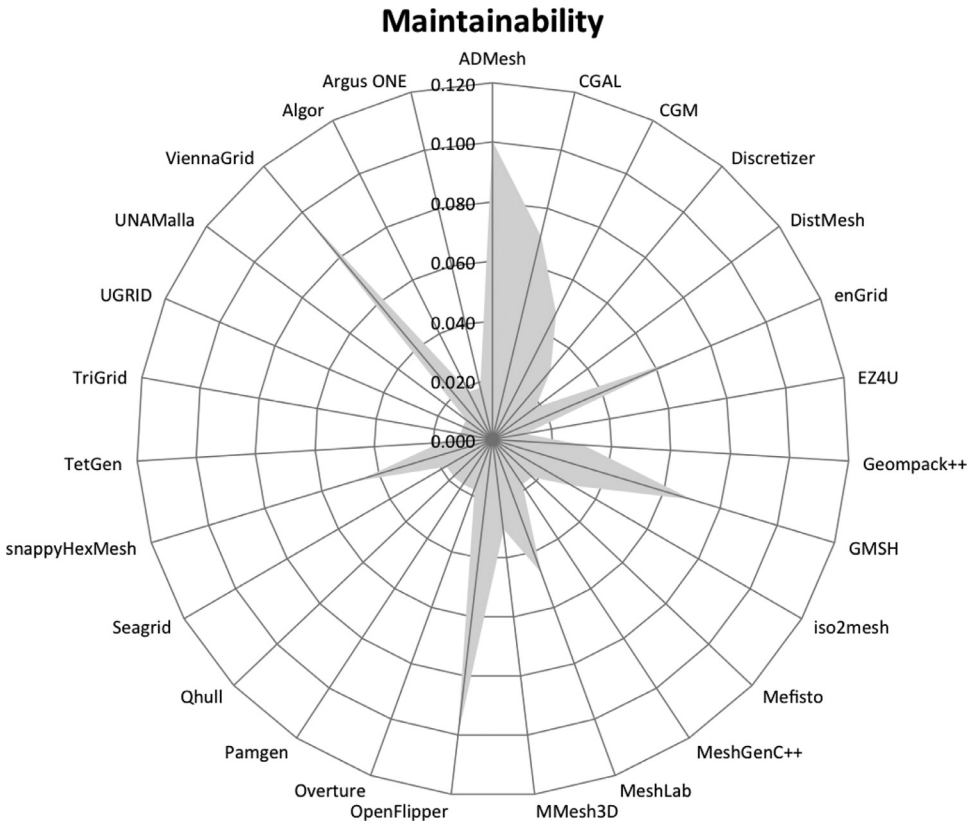

Fig. 4. AHP results for maintainability.

application-specific software. The scope of these frameworks extends beyond mesh generation, to physical simulation, geometry, linear and nonlinear solvers, differentiation and differential equations solvers. By their nature, these frameworks are extensible by users, to create meaningful working applications. Reverse dependencies exist in software products, such as Qhull, snappyHexMesh and TetGen, so at least portions of these products are being reused. However, there is no convincing evidence that reusability was explicitly considered for these projects.

Given the complexity of MGMP software, it is surprising how few reusable components exist – and how little evidence there was that large packages were built from pre-existing components. This is unlike other domains, like linear algebra say, where considerably more standardization has happened. We definitely recommend that MGMP developers pay more attention to reusability.

4.9. Portability

Out of the 27 products, 22 have achieved some sort of portability. Most often, Windows and Linux are supported. 20 of the projects support Windows, and the remaining 7 support a combination of Linux, OS X, and other Unix variations. 4 projects support only Windows, (EZ4U, UGRID, and the two commercial projects, Algor and Argus ONE). An Interesting case in portability is Mefisto, which supports Windows by using cygwin, which provides a way to use the Linux filesystem to build and run some Linux tools on Windows. Mefisto therefore does not support Windows natively. Another is MeshLab, which has ported its model viewing capabilities to both Android and iOS, the only MGMP software product to support these platforms. Cross platform build systems such as cmake have been employed to facilitate building on different platforms. Also, since the programming languages in use are highly portable (C++, C, MATLAB, Fortran, Ruby), the barrier to portability is low. In the cases of C++ and C, in use by at least 19 products, portability is handled with the use of a Makefile.

4.10. Understandability

After examining the source code of 21 products, we found that understandability of MGMP software is strong. We did not grade understandability for freeware or commercial projects, as their source was not accessible. The developers of all of the open source MGMP software products have released consistent code with respect to formatting, identifier and file naming, commenting and modularization. In some cases (CGAL, MeshLab, and snappyHexMesh), coding conventions and tips are provided for reference by the developers of the product. Comments are present in the code for all of the products. In Mefisto, the comments are in French; the corresponding website is written in English (with a French version). Only 3 products contained a software design document (CGAL, CGM, MeshLab), outlining the architecture of the software product.

4.11. Interoperability

Interoperability varies among the MGMP software products. 15 of the graded software show no evidence of using any libraries that the community has confidence in. Other software products such as CGAL, CGM, Discretizer, enGrid and snappyHexMesh provide lists of external software that is used to achieve their product's purpose. The external systems in use are varied, and not necessarily domain-specific. Helper libraries for graphics processing (OpenGL), UI frameworks (Qt) are used, as well as some of the graded software products; Tetgen is used in GMSH, CGAL is used in iso2mesh, GMSH is used in TetGen. As seen above, there are few products that support reusability, though when reusability is supported, these APIs are well documented in user manuals or developer-specific documentation.

4.12. Visibility

The visibility of the development process of MGMP software varies. In most cases, the development process is not defined. Information regarding the development process and contributing new code is given in CGAL and snappyHexMesh developer's guides, but this information is not available for the other products. Projects hosted on GitHub, like ADMesh or enGrid, follow the contribution and code review processes of GitHub. However, this process is not explicitly defined or referenced by the developers. Not having any development process-specific documentation hurts the visibility of the product because new developers will be unsure of the software life cycle, and how to go about contributing new code.

Strong design for an MGMP product web site is best demonstrated by CGAL, Gmsh, MeshLab, and Tetgen. These web sites provide a single destination for information on all aspects of the product. The design of these web sites vary, but there is always information regarding the product itself, download and installation options, tutorials, support, and other important documentation. Having a single website with logical sections to separate information helps the product's visibility because users or developers can find their information all in one place. An example that scored lower is OpenFlipper, since it has multiple web-sites, a Facebook page, a gitHub repository, and the occasional reference in the documentation to a past svn repository.

In 9 of the MGMP software products, multiple web sites are used, which decreases visibility. Though, as long as there is a highly visible link to and from each site, multiple sites can be manageable. The purpose of these sites are often separate; for example, a code hosting site to host the source code repository and a wiki for information about the project. Examples that use this model are ADMesh, Seagrid, and Pamgen. When multiple product web sites duplicate information (for example: Discretizer's old, fully populated website, and the new, sparse web site), the result severely hurts visibility because users and developers may be misled regarding which information is more accurate and up to date.

4.13. Reproducibility

MGMP software does not always address reproducibility – see Fig. 5. Automated tests exist in only 5 software products (CGAL, CGM, Pamgen, Qhull, ViennaGrid), which build confidence in the correctness of the software.

Sample data exists in 7 products, which is normally available from the getting started tutorial, or to demonstrate valid input data before the user creates their own. When used in the context of a getting started tutorial, sample data can demonstrate some level of reproducibility. However, this sample data does not comprehensively test every function of the software, so ensuring full reproducibility is not possible using only this data.

Only MMesh3D provides development and testing setup information. The developer indicates that MMesh3D has been installed without problems on Mac 10.5.4 and Ubuntu 8.10 with gcc version 4.3.2. The other software products do not provide this information, so other developers or potential users cannot be certain that their results are reproductions of the creator's original results. Furthermore, there is no evidence of automated tools to capture the experimental context so "works-on-my-computer" issues cannot be easily diagnosed.

4.14. Overall quality ranking

Once the grading has been finished, the overall impression of the product's performance on all software qualities is evaluated using AHP with equal weights between qualities, as shown in Fig. 6. As mentioned previously, the weights between qualities will actually vary depending on the needs of a specific project.

5. Recommendations

The full grading template in Appendix A provides a set of criteria for developers of SC software to consider to ensure best development practices and product quality. Based on the results from the previous sections, we have the following specific recommendations for MGMP software:

1. **Use an issue tracker for bug management and support requests.** To improve maintainability (and usability), the project should have an issue tracker. 16 of the graded products are not using an issue tracker, which seriously impacts their maintainability grades. By their nature, issue trackers are not normally used for support requests. However, using an issue tracker for support requests, as well as a bug tracking system, can help both maintainability and usability. Bugs can be reported and support questions can be asked in a unified, public manner. There are a number of free, open source issue tracking systems that are simple to set up, such as GitHub issues, SourceForge, Trac, and BugZilla.
   If the developer has the means, a discussion/mailing list can be set up to separate their support and bug tracking concerns. Visibility of these pages would have to be considered appropriately; the new services will need to be visible from the main product page, and vice versa. Since the maintainability and usability of MGMP software are generally the weakest qualities, using an issue tracker to track bugs and support requests may be the easiest way to help both of these qualities at once.
2. **Document performance measures.** Convincing evidence of performance measures in the graded software products was rare. The implication is not necessarily that MGMP software does not perform well. However, if the software has undergone performance optimizations, the developers should note these measures prominently in the user or developer documentation. As problem size grows, speed and efficiency of computations come into question, so any documentation or quantitative evidence (such as speedup limits as determined by Amdahl's Law) of the software's performance is of interest to prospective users. This is especially true for software products with parallel implementations (Pamgen, snappyHexMesh), as documenting parallelization schemes can show performance benefits over serial implementations. For instance, the section on parallel computation in Pamgen describes what the user can do to distribute data amongst different disks and running decomposed cases of problems. This creates a "domain decomposition" parallelization scheme, and performs better than a serial implementation.
3. **Use (and create) libraries to promote software re-use and avoid "re-inventing the wheel."** More than half of the software products do not seem to be using any external libraries. To reduce the amount of original code, which the developer is tasked with maintaining, the use of libraries can improve confidence in correctness, interoperability and reliability. If these libraries are in use by multiple products (and therefore by more developers and end users), bugs and reliability issues can be more easily found (and fixed). Responsiveness and speed in issue tracking can build a community's trust in a library. If the library's developers are not responsive or fast in issue tracking procedures, perhaps a contribution could be made, or a forked project could be developed, for the benefit of the entire community.
   These trusted libraries are not necessarily created by external sources. Any software products that can facilitate reuse, should do so, in the hope that developers of other projects

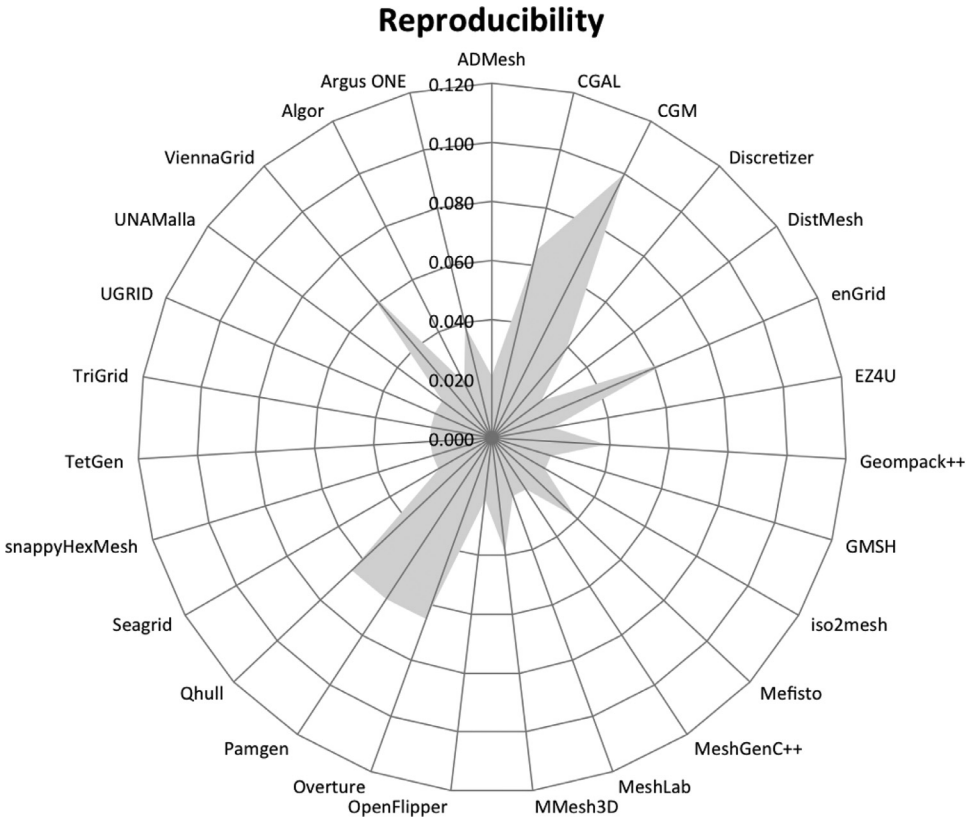

Fig. 5. AHP results for reproducibility

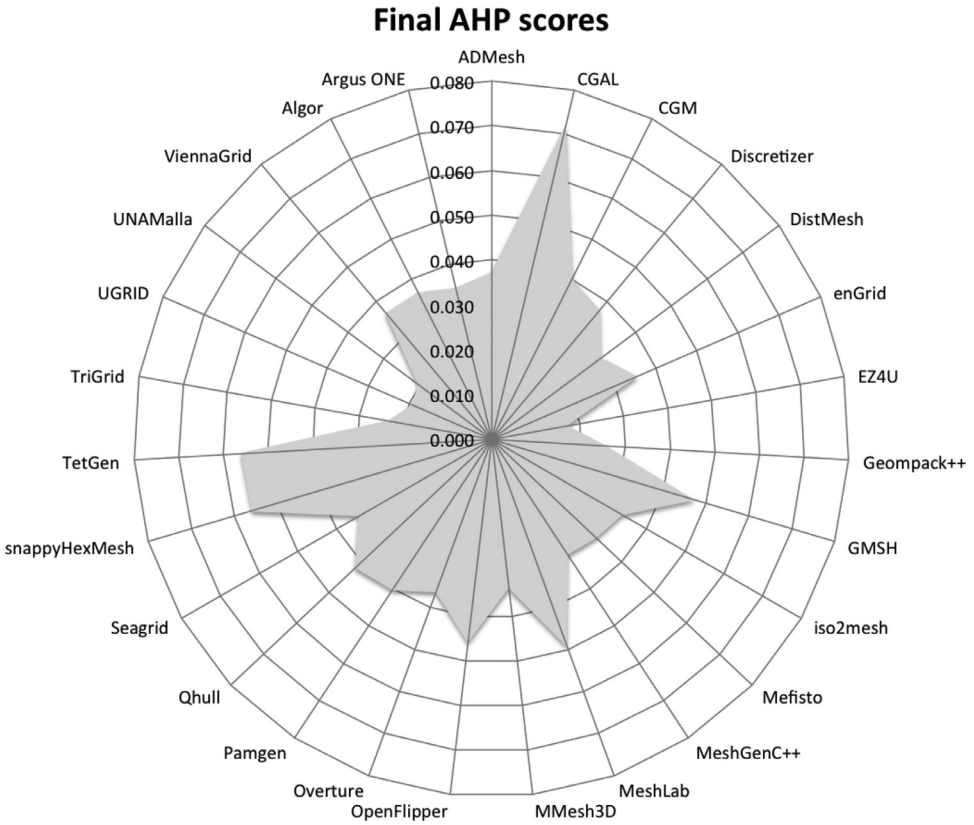

Fig. 6. Final AHP results.

will reuse their functionality. Although functionality is outside of the scope of this paper, we observed that manual reimplementing of meshing types seems to be a common practice. For instance, of the 27 graded software products, there are 12 products that generate triangular meshes. This trend is evident from larger samples of MGMP software. The survey by Owen [37] identifies 81 products, 52 of which generate triangular meshes, with 37 of these using some form of Delaunay triangulation. If a given triangular mesh generator became popular and trusted by the community, it could become a de-facto standard.

An object-oriented approach to design has been used in the past to improve the reusability of mesh generation tools. For instance, Bastarrica and Hitschfeld-Kahler [5] promote reuse by encapsultating processes as objects. Berti [8] considers the use of generic programming for improving the reusability of mesh data structures and algorithms. The Algorithm Oriented Mesh Database (AOMD) [42] in a sense combines the ideas mentioned in the previous two papers. AOMD uses object oriented programming for building the hierarchy of classes for the mesh entities and it uses generic programming by employing Standard Template Library (STL) algorithms and iterators.

The opportunity for reuse makes MGMP software a good candidate for development using the program family, or product line, approach. “Program families are defined ... as sets of programs whose common properties are so extensive that it is advantageous to study the common properties of the programs before analyzing individual members” [38]. An example of a specification for the requirements for a family of mesh generators, in the form of a commonality analysis, can be found in [59]. Bastarrica and Hischfeld-Kahler [6] provide a design for a family of meshing tools and Rossel et al. [44] apply domain modeling to the MGMP domain. A program family approach is facilitated by code generation, which is applied to a generative geometric kernel for MGMP in [9].

4. **Improve reproducibility through testing and set-up documentation.** It is very hard to ascertain if a local installation of a piece of software works properly. The first step would be to supply a test suite (with known results) that users can run post-installation to verify this.

   Another aspect would be for developpers to document their testing environments, i.e. where they can guarantee that their software worked properly. Knowing which configurations are known to work can help eliminate “works-on-my-computer” errors, and ensure that the software results are correct and reproduced as the developer intended, a key property of SC software. When external dependencies and complex setup become difficult or unmanageable, developers may use a virtual development environment to isolate these concerns on a VM for the purposes of development and testing. Software products such as Vagrant [23] exist to automatically configure and create VMs.
5. **Improve confidence in correctness through requirements specification, formal specification languages, and/or automated testing.** While requirements specifications appear more frequently in MGMP software (11/27) than in other scientific domains, specifically Remote Sensing (RS) (3/30) [32], automated testing occurs less frequently (MGMP: 6/27 RS: 17/30). To build confidence in correctness (specification and adherence to the specification), both should be present. Writing a proper requirements specification is difficult, but necessary to judge correctness. The specified behavior can be verified through testing. Templates exist to assist in the creation of SC requirements [56]. Smith and Yu [60] provide a sample specification for a parallel mesh generation toolbox and ElSheikh et al. [14] provide an example of a formal specification for MGMP design.

   An alternative way to specify and test code at the same time is by using formal specification languages such as ACSL for C in Frama-C [10]. These languages are written as code in the source files, and verified by testing the written specification against the behavior of the actual source code. Another method of building confidence is writing test cases for every function of the software product. Writing test cases has the benefit of implicitly providing a simple specification, since correct software needs to pass the tests. Although not as complete as a full requirements specification, test cases can be a good starting point, since the typically have the lowest barrier to entry for practitioners. Several popular unit testing frameworks exist, such as CUnit [31] or Check [1].
6. **Always provide a User Manual.** We were surprised that four products did not.

6. Conclusions

To provide feedback to the MGMP software community, we systematically graded 27 software products within the domain, starting from a list of software products from a domain expert. Using AHP, a multicriteria decision making method, we preformed pairwise comparisons between each of the software products. The results were summarized and interpreted for trends. Due to budgeting constraints, non-commercial products form the majority of our sample, so our conclusions mainly apply to them.

For the state of practice in mesh generation software, we found the following positive trends:

- Products seem to have good backing in academia and/or government funding. More than half of the products are associated with an educational institution. There also exists government funding among the graded products. This is positive because development of the product is easier to justify if there is a group or employer that is willing to assist.
- Less than half of the products use a specification, but the requirements specifications documents that do exist are well written, based on mathematics and comprehensively cover the product’s functionality.
- Installation and reliability of products during setup and initial testing were strong.
- Surface robustness was strong when testing products with bad input. The products all have error-handling capabilities.
- Most products support multiple platforms. Automation is generally used to elegantly handle portability.
- For open source projects, code understandability is high, and while there often is not an explicit coding standard, formatting, identifiers, and comments are generally consistent and easy to follow when examining the source code.

Our survey also found some negative trends. With the goal of providing useful feedback to the community, we presented recommendations to help fix these problems:

- Use an issue tracker for bug management and support requests.
- Improve confidence in correctness through requirements specification, formal specification languages, or automated testing.
- Document performance measures.
- Use (and create) libraries to promote software re-use and avoid “re- inventing the wheel.”
- Improve reproducibility by providing post-installation test suites, and recording the set up details for the development and testing environments.
- Provide a User Manual.

Regarding commercial products, our sample size is too low to provide meaningful conclusions. However, one could imagine that these would score higher, because of marketplace pressure for

high quality. This does not appear to be the case, but perhaps for non-obvious reasons: commercial software development correlates highly with secretive development practices, which means that there are few externally visible signs of these. Thus while the products *may* be very good, it is actually *harder* to tell that this is so.

Future work relating to the methods developed, or the measurements made, may include a re-grading of Schneiders's [49] list to determine changes in software quality over time. The methods outlined in Section 3 can be used with an alternate or expanded grading template. Using a different template could enable grading based on other aspects of the software, like functionality. AHP will be a key part in this analysis, since, as always, software qualities are difficult to quantify, but easier to compare relatively. Additionally, statistical tests, such as the Chi-squared test, could be used with our data to determine statistical dependence between multiple aspects or qualities of the software.

Acknowledgments

The authors acknowledge the time and effort of fellow team members Vasudha Kapil, Sun Yue and Zheng Zeng for their assistance in the project. In particular, Sun Yue is acknowledged for developing a Java program to automate pairwise comparisons for AHP using the overall impression grading scores. The feedback of the anonymous referees is also gratefully acknowledged.

Appendix 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. An (I) precedes the question description when its measurement requires a successful installation.

Table A.3
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)
(I) 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)
(I) 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})
(I) Did the software "break" during the initial tutorial testing? ({yes*, no, n/a})
Overall impression? ({1 .. 10})
Surface Robustness
(I) Does the software handle garbage input reasonably? ({yes, no*})
(I) 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})
(I) Does the application have the usual "look and feel" for the platform it is on? ({yes, no*})
(I) 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})

(*continued on next page*)

Table A.3 (*continued*)

Summary information
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})

Appendix B. Summary of measurements

The full gradings of the 27 mesh generation software products are below. The most recent gradings are available at: https://github.com/adamlazz/DomainX. The column headings correspond with the above questions from the grading template.

Table B.4
Installability, Val means tests for installation validation.

Name	Ins	Lin	Auto	Val	Steps	Pkgs	Uninstall
ADMesh	Yes	Yes	Makefile	No	4	0	not avail
CGAL	Yes	Yes	Makefile, packages	No	18	6	not avail
CGM	Yes	Yes	Makefile scripts	No	4	1	not avail
Discretizer	Yes	No	Scripts and binary release	No	4	1	not avail
DistMesh	Yes	No	No	No	2	0	not avail
enGrid	Yes	Yes	Makefile (Linux) Installer (Windows)	No	13	3	not avail
EZ4U	Yes	Yes	Installer (WIN)	No	4	0	not avail

Table B.4 (*continued*)

Name	Ins	Lin	Auto	Val	Steps	Pkgs	Uninstall
Geompack++	No	N/A	No	No	1	0	not avail
GMSH	Yes	Yes	Makefile, packages	No	4	0	not avail
iso2mesh	Yes	Yes	No	No	2	0	not avail
Mefisto	Yes	Yes	Script	No	2	0	not avail
MeshGenC++	Yes	Yes	Script	Yes	4	6	not avail
MeshLab	Yes	No	Makefile, packages	No	2	7	Yes
MMesh3D	Yes	Yes	Makefile	No	7	0	not avail
OpenFlipper	Yes	No	Cmake	No	20	15	not avail
Overture	Yes	Yes	Yes	Yes	70+	6	not avail
Pamgen	Yes	Yes	Cmake	Yes	6	0	not avail
Qhull	Yes	Yes	Makefile, .exe	No	3	0	Yes
Seagrid	Yes	No	No	No	3	0	not avail
snappyHexMesh	Yes	Yes	apt-get package	No	4	12	not avail
TetGen	Yes	Yes	make cmake	No	5	1	not avail
TriGrid	Yes	No	Makefile	No	4	0	not avail
UGRID	No	No	just run .exe	No	1	0	Yes
UNAMalla	Yes	Yes	Packages dmg	No	3	0	not avail
ViennaGrid	Yes	Yes	cmake	No	6	0	Yes
Algor	Yes	Yes	installer	No	3	0	Yes
Argus ONE	Yes	Yes	installer	No	3	0	not avail

Table B.5
Correctness grading results.

Name	Std Lib	Req Doc	Evidence	Std Ex
ADMesh	No	Yes	None	Yes
CGAL	Yes	Yes	Exact computation paradigm	Yes
CGM	Yes	No	None	Yes
Discretizer	Yes	No	Rdoc	Yes
DistMesh	No	Yes	None	Yes
enGrid	Yes	No	Doxygen	N/A
EZ4U	No	No	None	N/A
Geompack++	No	Yes	Research	N/A
GMSH	Yes	Yes	Research	Yes
iso2mesh	No	No	Research	Yes
Mefisto	No	Yes	None	Yes
MeshGenC++	Yes	No	Research	Yes
MeshLab	Yes	Yes	None	Yes
MMesh3D	No	Yes	None	Yes
OpenFlipper	Yes	No	Doxygen	No
Overture	Yes	No	Doxygen and Research	N/A
Pamgen	Yes	No	None	N/A
Qhull	No	Yes	Handles roundoff errors	N/A
Seagrid	No	No	None	Yes
snappyHexMesh	No	Yes	Doxygen	Yes
TetGen	Yes	Yes	None	Yes
TriGrid	No	No	None	N/A
UGRID	No	No	None	Yes
UNAMalla	No	No	None	Yes
ViennaGrid	No	No	No	N/A
Algor	Yes	No	Advertised as accurate	N/A
Argus ONE	No	No	No	Yes

Table B.6
Reliability grading results.

Name	Break during install	Break during initial test
ADMesh	No	No
CGAL	No	No
CGM	No	No
Discretizer	No	No
DistMesh	No	No
enGrid	No	No
EZ4U	Yes[2]	Yes
Geompack++	No	No
GMSH	No	N/A
iso2mesh	Yes[3]	No
Mefisto	No	No
MeshGenC++	No	No
MeshLab	No	No
MMesh3D	No	No
OpenFlipper	Yes	No
Overture	Yes[4]	N/A
Pamgen	No	No
Qhull	No	No
Seagrid	No	No
snappyHexMesh	No	No
TetGen	No	No
TriGrid	Yes[5]	N/A
UGRID	Yes[6]	No
UNAMalla	No	N/A
ViennaGrid	No	N/A
Algor	No	N/A
Argus ONE	No	No

[2] Windows error “This program might not have been installed properly”.
[3] Needed to add /bin to path.
[4] Difficult to install.
[5] No password for .exe and src installation failed many times.
[6] Chrome detected malicious code during download.

Table B.7
Robustness grading results.

Name	Handle garbage input	Handle line ending change
ADMesh	Yes	N/A
CGAL	Yes	N/A
CGM	No	No
Discretizer	Yes	N/A
DistMesh	No[7]	N/A
enGrid	Yes	N/A
EZ4U	No	N/A
Geompack++	Yes	N/A
GMSH	Yes	N/A
iso2mesh	Yes	N/A
Mefisto	Yes	N/A
MeshGenC++	Yes	N/A
MeshLab	Yes	N/A
MMesh3D	Yes	N/A
OpenFlipper	Yes	Yes
Overture	Yes	Yes
Pamgen	Yes	N/A
Qhull	Yes	N/A
Seagrid	Yes	N/A
snappyHexMesh	Yes	N/A
TetGen	Yes[8]	N/A
TriGrid	Yes	N/A
UGRID	Yes	N/A
UNAMalla	Yes	N/A
ViennaGrid	Yes	N/A
Algor	Yes	N/A
Argus ONE	Yes	N/A

[7] Admittedly not robust.
[8] Advertised as robust.

Table B.8
Performance grading results.

Name	Evidence of performance considerations
ADMesh	No
CGAL	Yes. Ensuring correctness = high overhead
CGM	No
Discretizer	No
DistMesh	No
enGrid	No
EZ4U	No
Geompack++	No
GMSH	No
iso2mesh	No
Mefisto	None
MeshGenC++	Yes. Advertised as fast
MeshLab	No
MMesh3D	No
OpenFlipper	Yes, Valgrind
Overture	No
Pamgen	Yes. Parallel with MPI
Qhull	Choice of language
Seagrid	No
snappyHexMesh	Yes. Parallel
TetGen	Yes. Advertised as fast
TriGrid	No
UGRID	Yes. Advertised as fast
UNAMalla	No
ViennaGrid	No
Algor	Yes. Advertised as fast
Argus ONE	No

Table B.9
Usability grading results.

Name	GS tutorial	Std Ex	User Man	Look and feel	Visib Prob?	User char	Support
ADMesh	No	No	Yes	Yes	No	No	GitHub issues email
CGAL	Yes	Yes	Yes	Yes	No	No	Mailing list forum
CGM	No	No	Yes	Yes	No	No	Mailing lists
Discretizer	Yes	Yes	Yes	Yes	No	No	Issue tracker, email
DistMesh	No	Yes	Yes	Yes	No	No	Email
enGrid	Yes	Yes	Yes	Yes	No	No	GitHub issues forum
EZ4U	No	No	Yes[9]	Yes	No	Yes[10]	Email
Geompack++	No	No	Yes	Yes	No	Yes[11]	Email
GMSH	No	Yes	Yes	No	No	No	Mailing list bug tracker email
iso2mesh	Yes	Yes	Yes	Yes	No	No	Mailing lists FAQ
Mefisto	No	Yes	Yes	Yes	No	No	Email
MeshGenC++	Yes	Yes	No	Yes	No	No	Mailing list email
MeshLab	No	Yes	Yes	No	No	No	Forums bug reporter
MMesh3D	No	No	Yes	No	No	No	Email
OpenFlipper	No	No	Yes	Yes	No	No	Bug tracker, email
Overture	No	Yes	Yes	Yes	No	No	Mailing list
Pamgen	No	Yes	Yes	Yes	No	No	Mailing list FAQ contact
Qhull	Yes	Yes	Yes	Yes	No	No	Email FAQ
Seagrid	Yes	Yes	No	Yes	No	No	Built-in email
snappyHexMesh	Yes	Yes	Yes	Yes	No	No	Courses paid bug tracker
TetGen	Yes	Yes	Yes	Yes	No	No	Paid support
TriGrid	No	No	Yes	Yes	No	No	Email
UGRID	No	No	Yes	Yes	No	No	Email
UNAMalla	Yes	Yes	No	Yes	No	No	Email
ViennaGrid	No	No	Yes	Yes	No	No	Mailing list forum Twitter
Algor	No[12]	No	No	Yes	No	No	Support DB commerical
Argus ONE	Yes	Yes	Yes	Yes	No	No	FAQ email docs

[9] Written in Spanish.
[10] Students from upc.edu.
[11] Knowledge geometric terms and NURBS curves and surfaces.
[12] Not for trial.

Table B.10
Maintainability, no evidence of code clones, C means Corrective, P means Perfective, BB means Bitbucket, ? means unclear, *Not for download.

Name	Mul Ver	Code rvw	Chlog	Type	Issue track	Bug fix	CVS	Evid
ADMesh	Yes	GH	Yes	C	GH	Yes	Git	No
CGAL	Yes*	DG	Yes	C	FusionForge	No	SVN	No
CGM	Yes*	No	No	C	Trac	No	Git	No
Discretizer	Yes*	No	No	C	SourceForge	No	SVN	No
DistMesh	Yes	No	Yes	?	?	?	?	No
enGrid	Yes	GH	Yes	C P	GH	No	Git	No
EZ4U	No	No	No	?	?	?	?	No
Geompack++	Yes	No	Yes	?	?	?	?	No
GMSH	Yes	No	Yes	P	Trac	Yes	SVN	No
iso2mesh	Yes	No	Yes	C	Habitat	No	SVN	No
Mefisto	Yes*	No	No	?	?	?	?	No
MeshGenC++	No	BB	No	?	?[13]	?	Git	No[14]
MeshLab	Yes	No	Yes	C	SourceForge	No	SVN	No
MMesh3D	Yes	No	Yes	?	?	?	Git, BB	No
OpenFlipper	Yes	GH	Yes	C P	GitLab	Yes	Git	Yes[15]
Overture	Yes*	No	Yes	?	?[16]	?	?	No
Pamgen	Yes[17]	No	No	?	?	?	Git	No
Qhull	Yes	No	Yes	C	HTML web page	?	?	No
Seagrid	No	No	No	?	?	?	SVN	No
snappyHexMesh	Yes	Yes	Yes	C	Self-made?	No	Git	No
TetGen	Yes*	No	Yes	?	?	?	?	No
TriGrid	Yes*	No	Yes	?	Self-made	?	?	No
UGRID	Yes*	No	No	?	?	?	?	No
UNAMalla	No	No	No	?	?	?	?	No
ViennaGrid	Yes	GH	Yes	P	GH	Yes	Git	No
Algor	Yes*	No	No	?	?	?	?	No
Argus ONE	Yes*	No	No	?	?	?	?	No

[13] Using Bitbucket, but 0 tickets.
[14] Advertised "easy extensibility" but no convincing evidence.
[15] Uses plugins for extensibility.
[16] Using SF, but 0 tickets.
[17] for download in different versions of trilinos.

Table B.11
Reusability grading results.

Name	Portions reused	Evidence
ADMesh	No	No
CGAL	Framework	No
CGM	Framework	No
Discretizer	No	No
DistMesh	No	No
enGrid	No	No
EZ4U	No	No
Geompack++	No	No
GMSH	No	No
iso2mesh	No	No
Mefisto	No	No
MeshGenC++	No	No
MeshLab	Yes plugins	No[18]
MMesh3D	No	No
OpenFlipper	No	Yes
Overture	Yes this is a framework of tools	Yes framework
Pamgen	This is a suite of software	No
Qhull	Yes this software is depended on	No
Seagrid	Yes possibly	No
snappyHexMesh	Yes, depended on by Discretizer etc	No
TetGen	Yes, depended on my GMSH etc	No
TriGrid	No	No
UGRID	No	No
UNAMalla	No	No
ViennaGrid	No	No
Algor	Plug-ins	No
Argus ONE	Plug-ins can be created	No

[18] plugin documentation is blank.

Table B.12
Portability grading results.

Name	Platform	Port in code	N imprtnt	Evid
ADMesh	WIN UNIX	Makefile	N/A	N/A
CGAL	WIN LIN OSX	Makefile cross platform build	N/A	N/A
CGM	LIN OSX	Cross platform code	No	No
Discretizer	WIN LIN	Cross platform code GUI	N/A	N/A
DistMesh	WIN LIN OSX	Cross platform code (MATLAB C++)	N/A	N/A
enGrid	WIN LIN	Bundled with specific windows tools	No	No
EZ4U	WIN	N/A	No	No
Geompack++	WIN LIN	N/A for exe cross platform code	No	N/A
GMSH	WIN LIN OSX	cmake (cross platform building)	N/A	N/A
iso2mesh	WIN LIN OSX	MATLAB/Octave code	N/A	N/A
Mefisto	WIN (cygwin) LIN OSX	Makefile	No	No
MeshGenC++	WIN LIN OSX	Python environment setup	N/A	N/A
MeshLab	WIN LIN OSX IOS ANDROID	Makefile IOS ANDROID separate	N/A	N/A
MMesh3D	UNIX	N/A	No	N/A
OpenFlipper	WIN LIN OSX	cmake Cross platform build system	No	Yes
Overture	LIN OSX	Different install processes.	No	No
Pamgen	UNIX	cmake	No	No
Qhull	WIN UNIX	cmake	N/A	N/A
Seagrid	WIN LIN OSX	Subdirectories in source files	N/A	N/A
snappyHexMesh	LIN	N/A	No	N/A
TetGen	WIN LIN OSX	cmake	N/A	N/A
TriGrid	WIN UNIX	Makefiles packages	No	N/A
UGRID	WIN	N/A	No	N/A
UNAMalla	WIN LIN OSX	Unclear	N/A	N/A
ViennaGrid	WIN UNIX	Makefile	N/A	N/A
Algor	WIN	N/A	No	N/A
Argus ONE	WIN	N/A	No	N/A

Table B.13
Understandability grading results.

Name	Indent	Std	Cons Id	Cnst	Cmnts	URL	Param	Mdlr	File name	Des doc
ADMesh	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
CGAL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
CGM	Yes	No	No	No	Yes	No	Yes	Yes	Yes	Yes
Discretizer	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	No
DistMesh	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
enGrid	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
EZ4U	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Geompack++	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
GMSH	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	No
iso2mesh	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
Mefisto	Yes	No	Yes	No	Yes[19]	No	Yes	Yes	No	No
MeshGenC++	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
MeshLab	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	Yes
MMesh3D	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	No
OpenFlipper	Yes	No	Yes	No	No	No	Yes	Yes	No	Yes
Overture	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	No
Pamgen	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
Qhull	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
Seagrid	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
snappyHexMesh	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes
TetGen	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	No
TriGrid	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes	No
UGRID	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
UNAMalla	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
ViennaGrid	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No
Algor	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Argus ONE	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

[19] Comments are in French.

Table B.14
Interoperability grading results.

Name	Ext systems	Workflow	API def
ADMesh	None	No	N/A
CGAL	Yes GMP OpenGL Qt etc	No	N/A
CGM	ACIS Cubit or Open.Cascade	No	Yes
Discretizer	OpenFOAM Paraview fxruby (GUI)	No	N/A
DistMesh	None	No	N/A
enGrid	VTK Qt Netgen (tetrahedral)	No	N/A
EZ4U	None	No	N/A
Geompack++	Author's polygonView libraries	Yes, Geomview	N/A
GMSH	FLTK OpenGL Netgen TetGen more	No	N/A
iso2mesh	CGAL binary included	No	N/A
Mefisto	None	No	N/A
MeshGenC++	MATLAB Python	No	N/A
MeshLab	VCG Library	No	No
MMesh3D	None	No	N/A
OpenFlipper	Plugins	Yes	Yes
Overture	Overture framework A++ P++ OpenGL	No	Yes
Pamgen	Many	No	No
Qhull	In octave mathematica and geomview	No	Yes
Seagrid	None	No	No
snappyHexMesh	12 (build-essential flex bison cmake zlib1g-dev qt4-dev-tools libqt4-dev gnuplot libreadline-dev libncurses-dev libxt-dev)	No	Yes
TetGen	GMSH and more	No	Yes
TriGrid	None	No	N/A
UGRID	None	No	N/A
UNAMalla	None	No	N/A
ViennaGrid	None	No	N/A
Algor	Microsoft .NET framework	Yes, Autodesk	Yes
Argus ONE	None	No	N/A

Table B.15
Visibility grading results.

Name	Dev process	External exam
ADMesh	Yes, GitHub	6 (two sites, need look for info in package)
CGAL	Yes, Dev manual	8 (one main site, all information available)
CGM	No	6 (TRAC and BitBucket site)
Discretizer	No	4 (new site, old site, and SF)
DistMesh	No	6 (one site, one page)
enGrid	Yes, GitHub	7 (two sites)
EZ4U	No	5 (one site, spanish manual)
Geompack++	No	7 (one page)
GMSH	No	8 (one site)
iso2mesh	No	7 (one site)
Mefisto	No	7 (one site, lots of information)
MeshGenC++	No	7 (one site, decent amount of information)
MeshLab	Yes, students graded on new plugins	8 (one site)
MMesh3D	No	7 (wiki, bitbucket)
OpenFlipper	No	5 (multiple sites, inconsistencies between them)
Overture	No	6 (one site, external tutorials)
Pamgen	No	5 (pamgen, trilinos, git, google code)
Qhull	No	6 (one site, good info, external tuts poor)
Seagrid	No	7 (two sites, main and bad repo site)
snappyHexMesh	Yes, code dev section	7 (one site)
TetGen	No	8 (one site)
TriGrid	No	8 (one site)
UGRID	No	4 (one site
UNAMalla	No	5 (one site)
ViennaGrid	Yes, GitHub	5 (three sites)
Algor	No	5 (one site)
Argus ONE	No	7 (one site)

Table B.16
Reproducibility, *Sample data, but not for verification.

Name	Dev env	Ver test data	Tools to capture exp context
ADMesh	No	No	None
CGAL	No	Yes tests	None
CGM	No	Yes tests	None
Discretizer	No	No*	None
DistMesh	No	No	None
enGrid	Yes	No	None
EZ4U	No	No	None
Geompack++	No	No*	None
GMSH	No	No	None
iso2mesh	No	No	None
Mefisto	No	No*	None
MeshGenC++	No	No	None
MeshLab	No	No	None
MMesh3D	Yes[20]	No*	None
OpenFlipper	No	No	None
Overture	No	Yes	None
Pamgen	No	Yes test suite	None
Qhull	No	Yes test suite	None
Seagrid	No	No*	None
snappyHexMesh	No	No	None
TetGen	No	No	None
TriGrid	No	No	None
UGRID	No	No	None
UNAMalla	No	No	None
ViennaGrid	No	Yes test suite	None
Algor	No	No	None
Argus ONE	No	No*	None

[20] OS X 10.5.4 Ubuntu 8.10.

References

[1] Archer B., Prickett C., Hugosson F.. Check; 2014. http://sourceforge.net/projects/check/.
[2] Argus ONE. Argus ONE; 2013. http://www.argusone.com/.
[3] Autodesk Inc.. Algor; 2014. http://www.autodesk.com/products/simulation/features/simulation-mechanical/all/gallery-view.
[4] Barber C.B., Dobkin D.P., Huhdanpaa H.. Qhull; 2012. http://www.qhull.org/.
[5] Bastarrica MC, Hitschfeld-Kahler N. An evolvable meshing tool through a flexible object-oriented design.. In: International meshing roundtable. Citeseer; 2004. p. 203–12.
[6] Bastarrica MC, Hischfeld-Kahler N. Designing a product family of meshing tools. Adv Eng Softw 2006;37(1):1–10.
[7] Bergqvist B.. Discretizer; 2013. http://www.discretizer.org/.
[8] Berti G. GrAL-the grid algorithms library. Future Gener Comput Syst 2006;22(1-2):110–22. http://dx.doi.org/10.1016/j.future.2003.09.002.
[9] Carette J, ElSheikh M, Smith S. A generative geometric kernel. In: ACM SIGPLAN 2011 workshop on partial evaluation and program manipulation (PEPM'11); 2011. p. 53–62.
[10] CEA-LIST. Frama-C; 2014. http://frama-c.com/.
[11] CGAL People. CGAL; 2014. http://www.cgal.org/.
[12] Davison AP. Automated capture of experiment context for easier reproducibility in computational research. Comput Sci Eng 2012;14(4):48–56.
[13] Dolling A.. TriGrid; 2013. http://trigrid.sourceforge.net/.
[14] ElSheikh AH, Smith WS, Chidiac SE. Semi-formal design of reliable mesh generation systems. Adv Eng Softw 2004;35(12):827–41.
[15] enGits. enGrid; 2014. http://engits.eu/en/engrid.
[16] Fang Q. iso2mesh; 2013. http://iso2mesh.sourceforge.net/cgi-bin/index.cgi.
[17] Fathom Team. CGM; 2013. http://trac.mcs.anl.gov/projects/ITAPS/wiki/CGM.
[18] Geuzaine C., Remacle J.-F. GMSH; 2014. http://www.geuz.org/gmsh/.
[19] Gewaltig M-O, Cannon R. Quality and sustainability of software tools in neuroscience. Cornell University Library; 2012. p. 1–20.
[20] Gewaltig M-O, Cannon R. Current practice in software development for computational neuroscience and how to improve it. PLOS Comput Biol 2014:1–9.
[21] Ghezzi C, Jazayeri M, Mandrioli D. Fundamentals of software engineering. 2. Prentice Hall; 2002.
[22] Girona J. EZ4U; Unclear. http://www.lacan.upc.edu/ez4u.htm.
[23] Hashimoto M., Bender J. Vagrant; 2014. https://github.com/mitchellh/vagrant.
[24] Hawken D. UGRID; 2012. http://www.telusplanet.net/public/djhawken/ugrid.htm.
[25] Henshaw W.D., Schwendeman D.W. Overture; 2014. http://www.overtureframework.org/index.html.
[26] Hrončok M. ADMesh; 2014. http://www.varlog.com/admesh-htm.
[27] Joe B. Geompack++; 2012. http://members.shaw.ca/bjoe/.
[28] Kelly D. 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. Riverton, NJ, USA: IBM Corp.; 2013. p. 299–310. http://dl.acm.org/citation.cfm?id=2555523.2555555.
[29] Kelly DF. A software chasm: software engineering and scientific computing. IEEE Softw 2007;24(6). http://dx.doi.org/10.1109/MS.2007.155. 120–119.
[30] Kobbelt L. OpenVolumeMesh; 2013. http://www.openvolumemesh.org/.
[31] Kumar A., Pye J., Gerhardy M. CUnit; 2014. http://sourceforge.net/projects/cunit/.
[32] Lazzarato A, Smith S, Carette J. State of the practice for remote sensing software. Technical Report CAS-15-03-SS. McMaster University; 2015.
[33] Marras S. MMesh3D; 2014. http://mmesh3d.wikispaces.com/.
[34] Möbius J, Kobbelt L. Openflipper: an open source geometry processing and rendering framework. In: Proceedings of the 7th international conference on curves and surfaces. Berlin, Heidelberg: Springer-Verlag; 2012. p. 488–500. doi:10.1007/978-3-642-27413-8_31. 978-3-642-27412-1.
[35] Norman DA. The design of everyday things. reprint paperback. New York: Basic Books; 2002. 0-465-06710-7.
[36] OpenFOAM Foundation. snappyHexMesh; 2014. http://www.openfoam.org/docs/user/snappyHexMesh.php.
[37] Owen SJ. A survey of unstructured mesh generation technology. In: International meshing roundtable; 1998. p. 239–67.
[38] Parnas D. On the design and development of program families. IEEE Trans Softw Eng 1976;SE-2(1):1–9.
[39] Perronnet A. Mefisto; 2013. http://www.ann.jussieu.fr/~perronnet/mefistoa.gene.html.
[40] Persson P.-O. DistMesh; 2012. http://persson.berkeley.edu/distmesh/.
[41] Ranzuglia G., Cignoni P. MeshLab; 2014. http://meshlab.sourceforge.net/.
[42] Remacle J-F, Shephard MS. An algorithm oriented mesh database. Int J Numer Methods Eng 2003;58:349–74.
[43] Roache PJ. Verification and validation in computational science and engineering. Albuquerque, New Mexico: Hermosa Publishers; 1998.
[44] Rossel PO, Bastarrica MC, Hitschfeld-Kahler N, Díaz V, Medina M. Domain modeling as a basis for building a meshing tool software product line. Adv Eng Softw 2014;70:77–89. doi:10.1016/j.advengsoft.2014.01.011.
[45] Rossmanith J.A., DoGPack Team. MeshGenC++; 2014. http://www.dogpack-code.org/MeshGenC++/.
[46] Rupp K.. ViennaGrid; 2013. http://viennagrid.sourceforge.net/.
[47] Saaty TL. How to make a decision: The analytic hierarchy process. Eur J Oper Res 1990;48(1):9–26.
[48] Sánchez P.B.. UNAMalla; 2013. http://lya.fciencias.unam.mx/unamalla/home_i.html.
[49] Schneiders R.. Software; 1998. http://www.robertschneiders.de/meshgeneration/software.html.
[50] Segal J. When software engineers met research scientists: a case study. Emp Softw Eng 2005;10(4):517–36.
[51] Segal J. Some problems of professional end user developers. In: VLHCC '07: proceedings of the IEEE symposium on visual languages and human-centric computing. Washington, DC, USA: IEEE Computer Society; 2007. p. 111–18. 0-7695-2987-9.
[52] Segal J. Models of scientific software development. In: Proceedings of the first international workshop on software engineering for computational science and engineering (SECSE 2008). Leipzig, Germany: In conjunction with the 30th International Conference on Software Engineering (ICSE); 2008. p. 1–6. http://www.cse.msstate.edu/~SECSE08/schedule.htm.
[53] Segal J. Developing scientific software. IEEE Softw 2008;25(4):18–20.
[54] Si H.. TetGen; 2013. http://wias-berlin.de/software/tetgen/.
[55] Signell R.. Seagrid; 2014. http://woodshole.er.usgs.gov/operations/modeling/seagrid/index.html.
[56] Smith S, Lai L. A new requirements template for scientific computing. In: Proceedings of SREP'05; 2005. p. 1–15.
[57] Smith S, Sun Y, Carette J. State of the practice for developing oceanographic software. Technical Report CAS-15-02-SS. McMaster University, Department of Computing and Software; 2015.
[58] Smith S, Sun Y, Carette J. Comparing psychometrics software development between CRAN and other communities. Technical Report CAS-15-01-SS. McMaster University; 2015.
[59] Smith WS, Chen C-H. Commonality analysis for mesh generating systems. Technical Report CAS-04-10-SS. McMaster University, Department of Computing and Software; 2004.
[60] Smith WS, Yu W. A document driven methodology for improving the quality of a parallel mesh generation toolbox. Adv Eng Softw 2009;40(11):1155–67. http://dx.doi.org/10.1016/j.advengsoft.2009.05.003.
[61] Tang J. Developing scientific computing software: current processes and future directions, Hamilton, ON: McMaster University; 2008. Master's thesis.
[62] The Trilinos Project. Pamgen; 2014. http://trilinos.sandia.gov/packages/pamgen/.
[63] Wilson G, Aruliah D, Brown CT, Hong NPC, Davis M, Guy RT, et al. Best practices for scientific computing. CoRR; 2013.
[64] Wilson GV. Where's the real bottleneck in scientific computing? Scientists would do well to pick some tools widely used in the software industry. Am Scientist 2006;94(1). http://www.americanscientist.org/issues/pub/wheres-the-real-bottleneck-in-scientific-computing.