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.

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 $b_{jk} = a_{jk}/\sum(a_{\cdot k})$. 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.