Commonality and Requirements Analysis for Mesh Generating Software

Spencer Smith and Chien-Hsien Chen
Computing and Software Department, McMaster University
E-mail: smiths@mcmaster.ca

Abstract

Mesh generation has been a mature subject for over a decade, and the procedure of generating meshes has long been automated to become mainly the responsibility of software. However, like other scientific software, mesh generators are mostly developed by domain experts who are lacking the training of proper software development methodologies. As a result, this type of software is often observed to be difficult to verify, hard to maintain, inflexible, inextensible, and consequently hard to reuse.

Presented in this work is our proposed solution to improve the software quality by applying software engineering methodologies, in particular commonality and requirements analysis. The significance of a useful documentation system is also emphasized in our work. Hence, the designs of each documentation, produced during a design stage, will also be presented.

1. Introduction

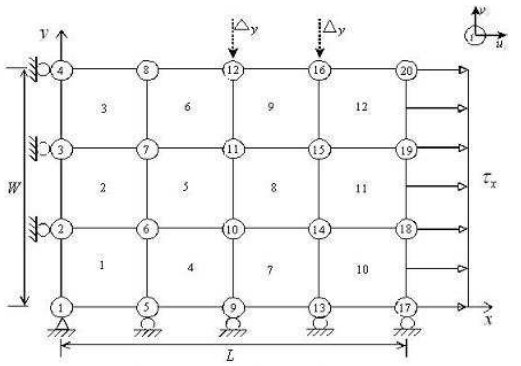
Software engineering methodologies have been gaining acceptance for many different types of software, such as business applications, real-time systems and safety critical systems. Unfortunately, scientific computing software has not yet received all of the benefits from the latests advances in software engineering. This paper will show that the development of scientific software, in particular mesh generating software, can greatly benefit from the use of such software engineering methodologies as commonality analysis and requirements analysis. Moreover, research in the field of software engineering will also benefit by tackling some of the unique challenges that come up during an analysis of scientific software.

Section 2 provides an overview of mesh generating systems that provides the background for later discussion of the system. In Section 3 the question of why commonality and requirements analysis benefit mesh generators is addressed. Research in the field of software engineering benefits as well because, as Section 4 discusses, there are several interesting challenges during the predesign step for mesh generators that make this analysis different from the analysis of other types of software. The details of the commonality and requirements analyzes for mesh generators are presented in Sections 5 and 6, respectively. The final section, Section 7, contains concluding remarks and mentions directions for future work.

2. Mesh Generating Software

A mesh is a discretization of a geometric domain into small simple shapes, such as line segments in 1D, triangles or quadrilaterals in 2D, tetrahedral or hexahedra in 3D. Meshes are popular in many application areas. For instance, in geography and cartography, meshes are used to give compact and precise representations of terrain data [4]. In computer graphics, most objects are first reduced to meshes before being rendered to the screen. The principal application of interest for the current study is the finite element method, where meshes are essential in the numerical solution of partial differential equation arising in physical simulation [4].

A simple 2D mesh of quadrilateral elements, which could be used for analysis of a solid mechanics problem, is illustrated in Figure 1. The files created by a mesh generator must describe the following: how the domain is decomposed into quadrilateral cells; the boundary conditions on the domain, with respect to applied tractions ($e.g.\ \tau_x$), prescribed displacements ($e.g.\ \Delta_y$) and fixity ($e.g.$ roller versus pinned versus free); and the material properties. The mesh shown in Figure 1 is an example of a structured mesh for a rectangular domain. In many practical problems the domain does not have such a simple geometry and the cell topology is no longer regular, which means an unstructured mesh must be adopted. More detail on meshes can be found elsewhere [4].



In the majority of cases, the tedious preparation and checking of a mesh are too demanding to do manually, especially when the model description contains several thousand or more elements. Therefore, automatic generation of meshes, using a mesh generator, is of obvious practical value for reducing the workload. As a result, the user will only need to concentrate on a few input parameters and rely on a mesh generator to produce a corresponding mesh. The occurrence of human errors can thus be greatly diminished [11]. Given the importance of mesh generators, careful thought should occur before one proceeds to the implementation of the system.

3. Why is Predesign Analysis Necessary?

The predesign analysis that is advocated here consists of a commonality analysis and a requirements analysis. A commonality analysis is conducted to answer the question of whether the software should be designed as a program family. A program family is defined in [8] as a "set of programs whose common properties are so extensive that it is advantageous to study the common properties of the programs before analyzing individual members". Most useful software exists in many versions. The contributing factors may be the variation in application demands, the continuing improvement of technologies, the varieties of different algorithms, and so on. Instead of building those similar programs in the ignorance of their existence of one another, one should take the advantage of developing them as a family.

The detailed analysis of why mesh generating software is well suited to development as a program family is postponed until Section 5.1. However, the idea has intuitive appeal when one considers the proliferation of mesh generating software. The software systems have much in common as they all produce mesh data, but they differ in the shape of the original domain, in the types of elements used, in the field of application of the resulting mesh, *etc.* Rather than have many independent programmers working on very similar problems, it makes sense to investigate where the commonalities lie, extract that information, and then do the job once, do it well, and let future developers benefit from the expertise of their predecessors.

Although there are many examples of mesh generating systems, following different design paradigms, there is little indication of anyone using standard and systematic methods to elicit, document, classify and analyze requirements. Software engineers generally advocate gathering and analyzing requirements in advance of building any software system because it is much easier and cheaper to correct mistakes and misconceptions at the beginning of the process than it is to try and fix things during implementation and maintenance. The ultimate success of any project depends on the quality of the requirements. During the process of requirement gathering, the requirements identified need to be specified in a document, called a software requirement specification (SRS). In terms of contents, a SRS should include the external behavior of the system, the constraints placed on the implementation, the forethought about the life cycle of the system, and the acceptable response of the undesired events [7]. In terms of quality, a SRS should be correct, unambiguous, complete, consistent, modifiable, verifiable, and traceable [3]. In the case of mesh generating systems there is apparently no examples of SRS documentation. The absence of such documentation has the following consequences:

- Practitioners argue over the relative merits of different designs based on their own implicit requirements. A developer may criticize a design because it is not efficient, but this criticism would not be justified if the designer clearly started out with requirements that clearly stated that precision, maintainability and portability are more important than efficiency. Design inevitably involves tradeoffs between different requirements, but because the requirements are not documented, anyone can criticize any design by "picking on" a requirement that it was not designed to address.
- Verification and validation (V & V) are important topics in scientific computing, but how can an implementation be verified or validated without knowing what it is being verified and validated against? Clear requirements are necessary to know what the system should be inspected and tested for. Moreover, current V & V efforts focus on functional requirements, but it is the nonfunctional requirements, like accuracy, efficiency, portability $etc.$, that often separate one design from another.

4. Challenges for Predesign Analysis of Mesh Generators

The previous section explained how mesh generating systems can benefit from software engineering methodolo-

gies. This benefit actually goes in both directions, as research in software engineering can benefit by tackling the challenges presented by mesh generating systems that are less present in other types of software systems. Some examples of these challenges include the following:

- Software engineers often advocate the use of formal methods, which consist of using mathematical techniques and notations to specify qualities and attributes of the software. Mesh generators are based on the field of computational geometry, which also advocates the use of mathematics. The challenge is to bridge the gap of communication between these fields, but the exciting thing is that both disciplines already "talk" in a common language, the language of mathematics.

- A goal that many software engineers hope to reach is the ability to proceed directly from specification to code in an automatic manner. In general this is an exceedingly difficult problem to solve, but if the domain of application is restricted, the problem may become feasible. The field of scientific computation is a good test bed or these kind of theories, because it consists of well-defined problems.

- Most software engineering methodologies rely on the use of discrete mathematics, but scientific computing, including mesh generating, are problems that are described using continuous mathematics. The challenge of adopting methodologies to handle continuous data is an interesting one.

5. Commonality Analysis

The first section below considers whether mesh generators should be developed as program families, by considering three hypotheses. The second section describes the actual commonality analysis document and provides examples.

5.1. Program Family Hypotheses

As indicated in [10], there are three basic assumptions underlying the production strategies of program families. In other words, to see if we should adopt the strategy of developing a program family for this type of systems, we need to see if what we are building meet the three hypotheses:

- The Redevelopement Hypothesis
  This hypothesis requires that most software development involved in producing the family should be redevelopement, which means that there should be a significant portion of the requirements, design, and codes in common among the family members. For mesh generators, some members vary in their user interfaces or output formats, while other parts of their systems are very much alike. In some other cases when two mesh generators have less in common, for example, when they are used to generate different types of meshes (*e.g.* structured mesh, unstructured mesh, or block-structured mesh), one can still find common requirements, designs, and codes between them.

- The Oracle Hypothesis
  This hypothesis requires that the types of changes that are likely to occur during the system's lifetime be predictable. This is certainly the case for a mesh generator. The topic of mesh generation has long been explored, so there now exist many different variations for this type of systems. Domain experts in this field can help contribute in predicting the user's requirements, the types of the physical problems that we wish the system to handle, the types of meshes to be produced, and so on. One can also learn from other previously implemented mesh generators to discover other possible variabilities.

- The Organizational Hypothesis
  According to the organizational hypothesis, the program to be developed using the program family production approach should be one that allow designers and developers to organize the software, as well as the development effort, in a way that the predicted changes can be made independently. If this assumption holds, then a predicted change will require changing only a few modules in the system. Consequently, the effort in producing the next version of the software can be minimized, which is the central goal of the program family concept. This hypothesis, however, is more challenging for our system.

  For some of the likely changes, such as the changes in user interface, visualization, and output format, the changes can be dealt with in an elegant way. However, for some other type of the changes, like the use of different mesh generating algorithms and the use of different optimization algorithms, the goal of restricting the change within one module would be difficult to meet. A mesh-generating algorithm creates the mesh, and the optimization algorithms modify the mesh. They all assume, to an extensive level, the type of mesh they create/modify. If we have the mesh information stored in a module, many aspects of the module, for example the mesh information it contains or even the abstract data type it represents, are inevitably assumed by these algorithms. So when a different algorithm is used, it is often the case that the data storage module will be the next immediate thing to change.

Beside the mesh data, different algorithms may require different user input information. So, a replacement of an algorithm, in our case, can result in massive changes in many other modules.

The difficulty in organizing the system so that the predicated changes can be made independently will be one of the major obstacles in applying software engineering principles to mesh generating systems. The research on this topic has begun [5].

5.2. The Document

The organization of the document basically follows the template proposed in [**?**]. We will look closely into some of the more interesting sections: commonality, variability, and parameter of variations.

Each one of the commonalities, the variabilities, and the parameters of variation has its unique item identification number, so cross-referencing is possible. Also, the item numbers were made purposely disperse to make easy future addition or deletion of any item.

Another important point to note: the commonalities, the variabilities, and the parameters of the variations were all stated in terms of external behavior instead of the internal design and implementation, as stressed in [10].

Commonality Section

This section lists the assumptions that are true for all the members in the family. To make the list accessible, every commonality is organized into its relevant category, and to make the category easy to find, we list all the categories of the commonalities at the beginning of the section.

Each commonality is presented in a uniform table as shown in Figure 2. Some commonalities have corresponding variability while some others do not.

Item #	160
Description	A mesh should have only one element type throughout its entire domain.
Related Variability	320,
History	Created – Oct. 21, 2002

Fig 3.1.1a -- a Commonality

Variability Section

This section contains all the changes expected in the capability of the family. This section also has structure, as in the commonality section, and since most variabilities are derived from certain commonalities, the structure found here is usually very similar to the commonality section.

Each variability is presented in a standard form, as shown in Figure 3. If we take a closer look at this particular variability in Figure 3, we can see that it was discovered by considering the proposed commonality in Figure 2.

Item #	320
Description	Different M.G. may generate meshes of different elements.
Related Commonality	160,
Related Parameter	460
History	Created – Oct. 22, 2002

Fig 3.1.1b – a Variability

Parameter of Variations

This section further specifies variabilities by quantifying them. Each item in this section specifies the range of the values for one variability, and is presented in the standard form shown in Figure 4. As an example, the item in Figure 4 gives the concrete information on the possible variation for its corresponding variability in Figure 3. The binding time marks the date when the decision is made.

The structure in this section is identical with the one in the variability section.

Item #	460
Corresponding Variability	320
Range of Parameters	Triangles, Quadrilateral
Binding Time	Oct. 23, 2002

Fig 3.1.1c – a Parameter of Variation

Relationship among Items from Different Sections

The relationship between commonality, variability, and the parameter of variation are:

1. Commonality vs. Variability - Some commonalities have their variabilities, while some do not. Some variabilities have corresponding commonalities and some other do not. However, while a commonality may have more than one variability, one variability can have only one commonality.

2. Variability vs. Parameter of variation - The relationship between variability and parameter of variation is one-to-one. That is, given a variability, one parameter of variation can be found, and vice versa.

6. Requirements Analysis

The first section below summarizes the characteristics of the system. Following this section the design decisions or the example SRS for mesh generators are discussed.

6.1. Characteristics of the system

The purpose of a SRS is to describe a proposed system appropriately. Every system has its characteristics. When designing a SRS, these system-specific characteristics should be taken into account so that the resulting SRS is better suited for the system it describes. The characteristics listed below are found for most of mesh generators and would be taken into consideration in making design decisions for our SRS.

Provision of the background concept/information is important

A mesh generator is a type of system that is not commonly seen by majority of program developers whose technical backgrounds are within Computer Science. Unless the domain experts are hired to do the programming jobs for this type of system, the provision of its background information is essential to facilitate the readers' understanding about the system requirements. Therefore for a mesh generator, a section that introduces this information should constitute a major part of the SRS.

There are many rules and conventions

For the type of system like a mesh generator, there are many rules and conventions involved in preparing a correct mesh. So a section including this information should be made explicit and placed appropriately in the SRS to facilitate a readers' frequent access.

System features can be used to categorize system's functionalities

If well designed, a mesh generator can have each of its system features in a well-defined boundary that does not overlap with other features. That is, each system feature can be viewed as a process, since only one feature can be executed at a time. So, every one of a mesh generator's functionalities can easily fall into only one system feature. For this reason, all of its functional requirements can be naturally organized into the system features that they belong to.

User interfaces are major system capacities/features

For a mesh generator, its user interfaces such as user's specification of input parameters and the generation of the output files are major parts of the system features. So, it follows that user interfaces for this type of system can be discussed in terms of functional requirements.

6.2. Design Decisions of the SRS

In this section, major design decisions made in creating the SRS will be discussed. We will first discuss the structure of the SRS for our mesh generator. Detail design decisions will be discussed in the subsequent subsection.

Structural Design Decisions

The structure of the SRS will be discussed section by section below.

- Section One - Introduction
  Most formal documents begin with an introduction. But what should the introduction of the SRS discuss? In [1] and [9], the approach was taken to introduce the system right from the beginning in a SRS, without mentioning the purpose of the documents. Whereas in [2] and [3], both the document and the system are introduced together in the first section.

  As for the SRS of the mesh generator, the general aspect of the system information would definitely be found in the later section. To keep the principle of separation of concerns, it was decided that the introduction on the system should be delayed to the section on the general system information, and the first section would only introduce the document.

- Section Two - General System Description
  After the introduction on the document, the next sensible step is to introduce the system. The purpose of the section is to provide background knowledge about the system to help readers understand specific system requirements. The information contained in this section is meant to be stated at high enough level so that other similar system can borrow this section with only minor changes.

  In [9], the general information about the system is scattered in various different sections, which offers a less compact organization and violates the separation of concern principle. In [2] and [3], their overall system description sections have confusing and/or redundant subsections so that it is hard to distinguish between subsections in terms of their contents.

  Also, in many templates ([1, 2, 3]), major system constraints are discussed in this section. Constraints are one type of requirement, and would be address later in system requirement section. To avoid redundancy, constraints should be stated once in the SRS, and should be in the system requirement section.

  As the result, the SRS for the mesh generator takes the skeleton of the general system description section

from [1] and removes the subsection where constraints are discussed.

- Section Three - Specific System Requirements
  After the general system information section comes the specific system requirement section. This is the major section of the SRS, and all system requirements should be here and they should be easily found under their relevant subsection. There are three types of requirements, system constraints, functional requirement, and non-functional requirements. System constraints are more restrictive and are more stable in comparison with other types of requirements. Rules and conventions for preparing a mesh should fall into this category.

  The SRS of the mesh generator takes some of ideas from [2] and [3]. In addition to system constraints, there is one subsection for all the functional requirements and one for non-functional requirements. The functional requirements section has its template from [3] as those requirements are grouped into system features. Note that the user interfaces are discussed as system features.

- Section Four - Other System Issues
  The idea of including some other supporting information relevant to the system development in a SRS is proposed in [9]. Examples of the materials included in this section are open issue, off-the-shelf solution, project risk, project cost and so on.

  The SRS of the mesh generator adopts this practice but only chooses some of the subsections that are closely related to the process of developing the system. To be consistent with the program family developing methodology, we have a subsection named "Our Program Family". It was added here to provide a blueprint of how the system will be extended. It delineates the subset of the current requirements, and it also contains projections on the future versions of the system.

7. Conclusions Remarks and Future Work

This paper presented a case study where software engineering methodologies were applied to a mesh generating system. The results of the commonality and requirements analyzes highlight how valuable these exercises are to mesh generating software. By identifying commonalities, much of the redundant effort that occurs today could by eliminated. Moreover, an understanding of the requirements for the system would end many arguments about the relative merits of any one system. In the future, any mesh generator can be considered to be of high quality, if it meets its stated requirements. The usefulness of software engineering methodologies does not end at the predesign stages. A mesh generator can also benefit from methodologies for decomposing the system into modules and for documenting the interfaces of these modules [6].

8. Acknowledgements

The financial support of the Natural Sciences and Engineering Research Council (NSERC) and of Material and Manufacturing Ontario (MMO) are gratefully acknowledged.

References

[1] IEEE Std. 1233, *IEEE guide for developing system requirements specifications*, 1996.

[2] ESA PSS-05-0 Issue 2, *ESA software engineering standards issue 2*, 1991.

[3] IEEE Std. 830, *IEEE recommended practice for software requirements specifications*, 1998.

[4] Marshall Bern and Paul Plassmann, *Mesh generation*, Handbook of Computational Geometry, Elsevier Science, 2000.

[5] Guntram Berti and Georg Bader, *Design principles of reusable software components for the numerical solution of pde problems*, presented at the the 14th GAMM-Seminar Kiel on Concepts of Numerical Software (January 23rd to 25th, 1998).

[6] Chien-Hsien Chen, *A software engineering approach to developing mesh generators*, Masters thesis, McMaster University, Hamilton, Ontario, Canada, 2003.

[7] Kathryn L. Heninger, *Specifying software requirement for complex system: New techniques and their application*, IEEE Transactions on Software Engineering, vol. 6, no. 1, pp. 2-13 (January 1980).

[8] David Parnas, *On the design and development of program families*, IEEE Transaction on Software Engineering, vol. 5, no. 2, pp. 1-9 (March 1976).

[9] James & Suzanne Robertson, *Volere requirements specification template edition 6.1*, Atlantic Systems Guild, 2000.

[10] David M. Weiss and M. Ardis, *Defining families: The commonality analysis*, Proceedings of the Nineteenth International Conference on Software Engineering, pp. 649-650 (1997).