Towards a Documentation Maturity Model

Shihong Huang
Department of Computer Science
University of California, Riverside
shihong@cs.ucr.edu

Scott Tilley
Department of Computer Sciences
Florida Institute of Technology
stilley@cs.fit.edu

ABSTRACT
This paper presents preliminary work towards a maturity model for system documentation. The Documentation Maturity Model (DMM) is specifically targeted towards assessing the quality of documentation used in aiding program understanding. Software engineers and technical writers produce such documentation during regular product development lifecycles. The documentation can also be recreated after the fact via reverse engineering. The DMM has both process and product components; this paper focuses on the product quality aspects.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *documentation*.
D.2.9 [**Management**]: Software quality assurance (SQA*)*.

General Terms
Documentation, Standardization

Keywords
Documentation, maturity model, quality, reverse engineering

1. INTRODUCTION

Documentation has long played an important role in aiding program understanding to support software evolution. The format of the documentation (e.g., textual versus graphical), the manner in which the documentation is produced (e.g., manually versus automatically), and the value of the documentation (e.g., accuracy with respect to the subject system's source code) all directly affect a software engineer's ability to implement changes to complex applications in a disciplined manner.

Unfortunately, most documentation is of low quality relative to such attributes, making its use in program understanding problematic [13]. This is due in part to the fact that software system documentation is usually generated in an ad hoc manner. There is little objective guidance for judging documentation quality or for improving the documentation process. The result is documentation quality that is difficult to predict, challenging to assess, and that usually falls short of its potential.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGDOC'03, October 12–15, 2003, San Francisco, California, USA.
Copyright 2003 ACM 1-58113-696-X/03/0010…$5.00.

There are two aspects to documentation quality: the process and the product. Documentation process quality focuses on the manner in which the documentation is produced (for example, whether or not an organization follows a systematic approach to document development that is closely aligned with the rest of the software process). Documentation product quality is concerned with attributes of the final product (for example, whether or not it uses standardized representations like the Unified Modeling Language (UML) [16] in graphical documentation).

This paper presents preliminary work towards a unifying structure for assessing documentation quality. The Documentation Maturity Model (DMM) is specifically targeted towards assessing the quality of software system documentation used in aiding program understanding. Product developers and technical writers produce such documentation during regular development lifecycles. It can also be recreated after the fact via reverse engineering, a process of source code analysis and virtual subsystem synthesis [21].

The process component of the DMM is a five-level staged maturity model. It is based on the Software Capability Maturity Model (CMM) created by Carnegie Mellon University's Software Engineering Institute [3]. The product component of the DMM is centered on a set of key product attributes. It is inspired by the Reverse Engineering Environment Framework (REEF) [27], which was developed as an assessment instrument for reverse engineering tools and techniques. Although the process component of the DMM is very important, this paper focuses more on the product quality aspects.

The next section provides an overview of the challenges associated with program understanding, and the important role that system documentation can play in helping achieve this goal. Section 3 discusses process maturity, including related maturity models from application domains other than documentation, and provides an overview of the process component of the DMM. Section 4 focuses on product quality, and details the key product attributes used to assess documentation quality in the DMM. Section 5 summarizes the paper and outlines possible avenues of further work.

2. PROGRAM UNDERSTANDING

The goal of program understanding is to acquire sufficient knowledge about a software system so that it can evolve in a disciplined manner [26]. The essence of program understanding is identifying artifacts, discovering relationships, and generating abstractions. This process is essentially pattern matching at various abstraction levels. It involves the identification, manipulation, and exploration of artifacts in a particular representation of a subject system via mental pattern recognition

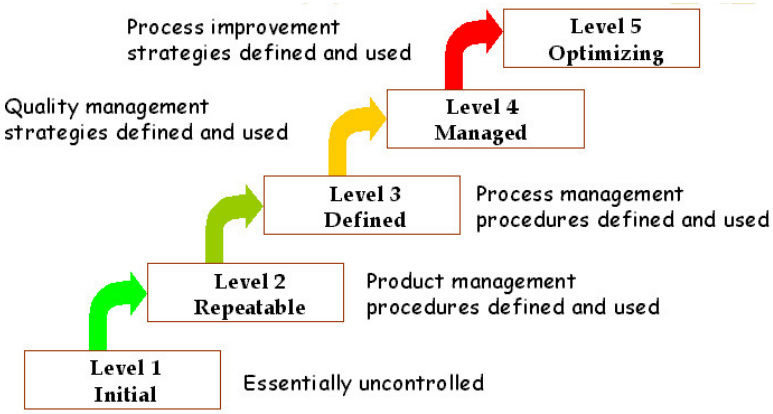

Figure 1: The Software CMM

by the software engineer and the aggregation of these artifacts to form more abstract system representations.

The program understanding process depends on several factors, including one's cognitive abilities and preferences, one's familiarity with the application domain, and the set of support facilities provided by the software engineering environment. These factors often determine the approach taken to understand a complex application. For example, someone who is familiar with the implementation domain, but unfamiliar with the application domain, may adopt a bottom-up approach to program understanding. This involves analyzing low-level code constructs and iteratively building higher-level mental models of the subject system.

There are a variety of support mechanisms for aiding program understanding. They can be grouped into three categories: unaided browsing, leverage corporate knowledge and experience, and using computer-aided techniques like reverse engineering. By far the most advantageous technique for large and complex software systems is reverse engineering, because it provides automated support for tedious and error-prone tasks such as code analysis, cross-referencing definitions and uses of software artifacts like types and variables, and the creation of abstract representations of the subject system that are closer to the application domain—a necessity when it comes to properly understanding change requests in support of evolution.

Reverse engineering is a process of analysis, not a process of change [7]. It relies on compilation technologies such as parsing and data flow to recreate abstract representations of the subject system. In the absence of original program documentation, the recreated abstraction representations serve as proxy system documentation for the software engineer. This documentation can take many forms. For example, inline text, hyperlinks to secondary sources of information, or a variety of graphical representations that rely on advanced visualization techniques.

Regardless of its format, high-quality system documentation plays a key role in aiding program understanding. Without such documentation, there is only source code to rely on. Since complex software systems can have several million lines of source code, it is beyond the capabilities of even the most talented software engineer to gain sufficient understanding of the entire application to make informed decisions. Instead, they must rely on a mature process that produces quality documentation to help them with their program understanding tasks.

3. PROCESS MATURITY

In the manufacturing domain, there is a direct link between process maturity and product quality. As various stages of the manufacturing process are automated, the final product is less likely to suffer from random faults injected into the process through human error. Indeed, several measures of productivity for many areas in the economy are based on assessing process maturity. These measures are based on the assumption that, as an organization matures, its ability to produce higher quality (and/or higher quantity) products improves accordingly.

There are several international standards related to documentation processes. For example, ISO/IEC 15910 provides a detailed process for the development of user documentation (paper and on-line help). A related standard, ISO/IEC 18019, offers more guidance on how to produce documentation that meets user needs.

For the DMM, the process portion of the model is currently based on the best-known maturity model used in a closely related activity: the engineering of software. For the moment, this part of the DMM is only loosely sketched out; the maturity levels are provided, but the key process areas have not yet been identified. As the DMM is refined over time, the process maturity indicators will be completed.

3.1 The Software CMM

In the software engineering domain, the inability of most organizations to produce quality products (in terms of attributes such as budget, schedule, and requirements) led to research into software process improvement. One of the best-known results of this research was the Software Engineering Institute's Capability Maturity Model (CMM) for Software. Underlying the CMM is the thesis that as an organization's software processes mature, the resultant software product's quality improves as well.

As shown in Figure 1, the CMM is a staged model of process maturity that reflects an organization's ability to produce expected results by developing a quantitative understanding of both the software process and the software product. The CMM's five levels of process maturity are (1) initial, (2) repeatable, (3)

defined, (4) managed, and (5) optimizing. For each of these levels, an organization is assessed according to its capabilities in several key process areas (KPA). For example, mastery of software configuration management is a KPA for Level 2 (repeatable) of the CMM.

The CMM has been extremely influential. However, it is not suitable for use by all organizations. It was developed to assess the software engineering maturity of military contractors. It focuses extensively on project management, somewhat to the detriment of product development (and by extension, underplaying the role of technology). Perhaps most importantly, it does not define its domain of applicability.

3.2 Related CMMs

Since the release of the original CMM over a decade ago, there have been numerous other maturity models related to the engineering of software that follow its structure. Two examples of such models are the Systems Engineering Capability Maturity Model (SE-CMM) [6] and the Integrated Product Development Capability Maturity Model (IPD-CMM) [4]. The current focus of the process community is the CMMI Integration (CMMI) product suite [8], which blend the software and systems engineering CMMs (among others).

Moreover, several derivative works have been developed and used by numerous institutions (not all of which are focused solely on software). These works include the Personal Software Process (PSP) [14] and the Team Software Process (TSP) [15]. The birth of the Extreme Programming [22] movement can also be partially explained as a reaction to CMM advocates.

The CMM has also been used as a framework for the development of maturity models that are not directly related to software engineering per se, but to ancillary activities that nevertheless have an impact on the quality of the final product. Two examples of such published models are the Software Acquisition Capability Maturity Model [5] and the People Capability Maturity Model [10]. There are research efforts that have created less widely known CMM-based models for other disciplines, such as the Testing Maturity Model [2] for program testing.

3.3 The DMM Process Maturity Levels

The DMM's process maturity levels are currently an exact copy of the CMM's levels. The five process maturity levels of the DMM provide a foundation for continuous documentation quality improvement. The five levels are as follows:

Level 1 - Initial: Essentially an uncontrolled environment. It is the baseline for all organizations; there is no level 0. The mentality is "write first, ask quality questions later." This method is sometimes successful, but its success depends on good people.

Level 2 - Repeatable: Product management procedures are defined and used. A disciplined, project-based, process maturity is evident. Organizations leverage previous experience from similar documentation projects.

Level 3 - Defined: Process management procedures are defined and used. Standard, consistent, organization-wide processes are in place. Processes are tailored per project as needed.

Level 4 - Managed: Quality management strategies are defined and used. Predictable processes are employed. Quantitative quality goals are set for products and processes, the latter of which are instrumented. Quality can be predicted and risks can be managed.

Level 5 - Optimizing: Process improvement strategies are defined and used. The key feature at this level is continuously improving processes. Defects are analyzed and root causes determined. Technology and process improvements managed as ordinary business activities.

As stated above, there are as of yet no KPAs identified for the DMM. There is a clear need for them, but the work remains to be done. However, as with the CMM's process maturity levels, the CMM's KPAs offer a good base upon which to build—as long as they are applicable to documentation, and not specifically to software.

4. PRODUCT QUALITY

The maturity levels discussed in the previous section form a preliminary framework upon which process quality can be assessed. As process maturity increases, the belief is that product quality will improve as well. However, the ability to independently assess product quality directly, irrespective of the process used to create it, would be very valuable in helping determining overall documentation quality.

However, product "quality" is an elusive concept that is sometimes difficult to quantify. This is particularly true for intangible products such as software and documentation. If quality is defined, the definition is usually something so vague that it is difficult to measure [19]. In an addition, the definition of quality is very contextual; users of software products may value certain quality attributes more in one context than in another, and likewise for program documentation.

The product quality component of the DMM is based on the notion of Key Product Attributes (KPA). The use of the acronym "KPA" in this context is not by accident. Although overloaded, the phrase KPA was selected because of the immediate notion of essential quality attributes related to maturity models that it connotes.

4.1 Software Quality

Using a traditional interpretation of quality, software quality should be evaluated according to its specification [9]. However, for software systems, there are some problems in realizing this definition [24]. For example, a software specification describes an engineer's perspective of a system's characteristics, based on customer requirements. Maintaining consistency and a common interpretation of the requirements and the specifications documents is one of the most vexing problems in software engineering. It is notoriously difficult to craft specifications that are both complete and consistent.

Moreover, some of the necessary requirements are not captured in the specifications documents. For example, internal attributes such as maintainability may not be of interest to the customer, but they may be of importance to the long-term success of the product. The result is that a software product may be assessed as "high quality" according to its specifications, even though the customer may be entirely displeased with the final result.

Table 1: DMM KPAs and Maturity Levels

KPA		Maturity Level				
		1	2	3	4	5
Format	Text	Inline & Informal	Inline & Standardized	Hyperlinked	Contextual	Personalized
	Graphics	Static & Informal	Static & Standardized	Animated	Interactive	Editable
Efficiency		Manual	Semi-automatic & Static	Semi-automatic & Dynamic	Automatic & Static	Automatic & Dynamic
Granularity		Source code	Design patterns	Software architecture	Requirements	Product lines

The ISO/IEC 9126 standard provides a framework for evaluating software quality [18]. This descriptive model defines six quality attributes (with sub-attributes) that are used to evaluate software quality. The six top-level quality attributes used in ISO/IEC 9126 to evaluate software quality are functionality, reliability, usability, efficiency, maintainability, and portability.

4.2 Documentation Quality

A significant amount of work has been done in defining and evaluating documentation quality. However, the focus of much of this work was on guidelines for technical writers and editors producing manuals targeted towards end users; less work has been done for technical writers and software engineers producing system documentation targeted towards product developers. Nevertheless, this previous work provides useful insight into the documentation creation processes, and can be used to create a documentation quality model whose main audience is product developers.

As with documentation processes, there are international standards related to documentation products. For example, ISO/IEC 18019 (2000) provides guidelines for the design and preparation of software user documentation. This standard describes how to establish what information users need, how to determine the way in which that information should be presented to the users, and how to prepare the information and make it available. It covers both on-line and printed documentation, and is based on two earlier British Standards. The first is BS 7649 (1993), a guide to the design and preparation of documentation for users of application software. The second is BS 7830 (1996), a guide to the design and preparation of on-screen documentation for users of application software. There is also an ANSI/IEEE standard 1063 (1987) on software user documentation. But it, like the ISO standard, is primarily focused on documentation for the consumer—not documentation for the developer.

Outside of the standards bodies, IBM did pioneering work on guidance for technical writers and editors producing quality documentation in the early 1980s. In the seminal report IBM produced in 1983, called "Producing Quality Technical Information" (PQTI), they wrote, "technical information that meets all the requirements is quality information" [17]. This concept of documentation quality is illustrated by showing both requirements and examples. At the end of the report, PQTI provides a checklist for reviewers to ascertain whether or not these requirements are met. The real innovation and impact of this work is not its format, but its "quasi-scientific, analytical approach to information development" [33]. This approach implies that engineering discipline can be applied to information development, with results are measurable and repeatable. This is a philosophy ascribed by the DMM.

In 1998, after several iterations within IBM, a new book called "Developing Quality Technical Information: A Handbook for Writers and Editors" was published [12]. This book expanded the original seven quality attributes into nine, which are grouped into three main categories. The categories are "Easy to Use," "Easy to Understand," and "Easy to Find." The book also provides a procedure for reviewing and evaluating technical information according to these criteria.

Determining and assessing quality is complex, and no single report and checklist can guarantee the development of quality document. PQTI remains one of the earliest and perhaps best attempt at laying out the characteristics that constitutes quality documents. A recent article by Karl Smart pointed out one of perceived limitations of the PQTI: its lack of "contextual framing of documents" [23]. The standard of quality is contextual in the sense that "the importance of individual dimensions of quality changes depending upon the audience, context, and purpose of the document." Smart classified dimensions of quality into three categories, which suggest their relative importance: essential, conventional, and attractive.

4.3 The DMM Key Product Attributes

The process portion of the DMM is structured as five maturity levels, each of which has associated key process areas. The product portion of the DMM is inverted: each key product attribute has five maturity levels. These levels are inclusive, i.e., the maturity level for a specific KPA implies that the document in question also has the desirable characteristics of the levels below it. This is in keeping with the general interpretation of staged maturity levels in other application domains.

As depicted in Table 1, the product-quality KPAs in the current version of the DMM are efficiency, format (textual and graphical), and granularity. There are other candidate KPAs that are currently under consideration for inclusion in the next draft of the DMM, such as Accuracy and Completeness. However, at this time these candidates have not been sufficiently explored to warrant full incorporation into the DMM.

4.3.1 Efficiency

Product efficiency refers to the level of direct support the documentation provides to the software engineer engaged in a program understanding task. There are several aspects to the definition of efficiency, such as accuracy between the documentation and the source code, the ease with which the documentation can be generated, and the completeness of the documentation with respect to the information required by the developer.

Level 1 – Manual: This is the most inefficient type of program documentation. Since the (usually textual) documentation is generated manually, it very often quickly becomes out of sync with respect to the source code it describes—unless it is maintained along with the rest of the system. Since developers are rewarded for coding, and not for writing, such synchronization rarely takes place in the real world. This may not be the desired situation, but it is an accurate reflection of the current state of affairs.

Level 2 – Semi-automatic & Static: This type of documentation is produced semi-automatically, using generative or reverse engineering tools, as directed by the developer or the technical writer. The documentation is static, reflecting system characteristics only at the time of generation or recreation. If the system changes after the documentation has been produced, the synchronization problem arises again.

Level 3 – Semi-automatic & Dynamic: The difference between Level 2 and Level 3 documentation efficiency is that Level 3 documentation is dynamic. Changes made to the system can be automatically and instantaneously reflected in the documentation—as long as the developer directs the tools to perform the generation or recreation.

Level 4 – Automatic & Static: This type of documentation is produced using automated mechanisms. It is still static in the sense that the documentation represents a snapshot of the system at the time the production occurred. However, the automated aspect means that there is no need for developer involvement.

Level 5 – Automatic & Dynamic: Level 5 documentation efficiency is fully automatic and completely dynamic. Tools within the software engineering environment can be programmed to produce the necessary documentation on demand. This ensures higher-quality documentation and removes a major impediment to adoption.

4.3.2 Format

Documentation product format refers to the types of documentation produced, and their characteristics. For the purposes of clarity, document format is further refined as being either "Textual" or "Graphical." Depending on the type of document artifact produced, only one part of this KPA is directly applicable to quality assessment.

Textual

Textual documentation ranges from inline prose written in an informal manner, to personalized views dynamically composed from a document database.

Level 1 – Inline & Informal: A developer usually writes inline and informal textual documentation while coding. The documentation is typically a brief explanation of low-level functionality, and is tightly connected to small code fragments. There is no standard format and style that the inline comments must follow. The quality of inline documentation at this level is highly dependent on the experience and efforts of the developer.

Level 2 – Inline & Standardized: Level two adds standardization to the inline documentation. The standard followed may be the developer's own convention that is used consistently across all components for which they are responsible. A team or an organization, requiring that all developers document their code using an agreed-upon template, may also impose standards. The presence of standards-compliant documentation can be automatically checked using third-party tools. Standardization removes the need for a developer to ask, "What should I document?"

Level 3 – Hyperlinked: Hyperlinked documentation adds a level of indirection to the text, permitting a far richer explanation of program behavior than is possible in source code. The target of the hyperlink could be more text, graphics, or multimedia commentary. An older example of hyperlinked documentation is the INFO system [30]; a more modern example is the Javadoc standard [25].

Level 4 – Contextual: Contextual documentation is made possible using tool support; it is not possible to achieve in printed text. Eliding irrelevant data and enhancing pertinent information provides context. This is somewhat similar to a collapsible outline in a word processor, or program slicing in a software engineering environment [35].

Level 5 – Personalized: Personalized documentation provides contextual perspectives that are under the control of the reader, not the writer. To thoroughly understand a software system, it is helpful for the software engineer to have access to multiple, complimentary views of the software system [31][32].

Graphical

The least mature type of graphical documentation is a static image, which may use non-standard representations of software artifacts and relationships. The most advanced graphical documents are editable by the user, better enabling them to create customized representations of the subject system.

Level 1 – Static & Informal: Static graphs are the oldest form of graphical documentation. They are relatively easy to produce and to integrate into existing software engineering process. Static graphs are hardcopy-like images that the user can view or print. The produced graph can be in common formats such as GIF or PDF. Other than possibility selecting the artifacts that appear in the visualization, the user does not interact in any way with the static image; it is included solely as read-only graphical documentation.

Level 2 – Static & Standardized: In contrast to Level 1 graphical documentation, Level 2 uses standard representations for software components. This means using templates or stencils that provide recognizable icons for software artifacts and relationships. The

best-known example of a standard used in graphical documentation is the UML [22].

Level 3 – Animated: Animated graphical documentation has the potential to improve program understanding by illustrating key algorithmic concepts or data structure interactions in a visual manner. The user may have very little interaction with the animation, but the benefits can still be realized.

Level 4 – Interactive: At this level, the generated graphical documentation is interactive, which permits the user to navigate the diagram much like a Web page. The user can traverse the graph by selecting an edge and then following the link to the next node. This traversal feature provides a clear path in a complex graph so that the software engineer can chase down the artifacts and relationships of interest. The interactive documentation can includes some type of animation with better response to user feedback than plain animated documentation.

Level 5 – Editable: At level 5, graphical documentation produced by software visualization tools is editable. Editable visualizations let the user actually change the generated graph itself, such as adding new nodes or edges. If the editable graph is connected to a backend database, it can be saved for future reference, completing the cycle from maintenance back to new development. Editable documentation is also referred to as "live documents" in the literature [34].

4.3.3 Granularity

Product granularity refers to the level of abstraction described by the documentation. The lower the granularity level is, the closer the documentation is to the implementation of the subject system. Conversely, the higher the granularity level is, the closer the documentation is to the business functionality realized by the subject system.

Level 1 – Source code: Documentation at the level of source code provides commentary on low-level implementation details of selected aspects of the subject system. It provides support for understanding the algorithms and data structures. The software engineer, using the basic information provided by source code documentation, must manually construct more abstract mental models.

Level 2 – Design patterns: One level above source code is design patterns [11]. These are common programming idioms that are language independent. Documenting the design patterns used in an application can help a developer understand the higher-level rationale behind implementation decisions.

Level 3 – Software architecture: High-level design, sometimes referred to as software architecture [1], captures the essential aspects of a system's structural characteristics. Understanding a system's architecture is essential to make significant changes to its functionality. UML diagrams are often used to represent software architecture.

Level 4 – Requirements: Software architecture captures the engineer's view of the system's high-level design. Requirements represent the system's intended purpose from the point of view of the user. Since maintenance requests are often couched in the terminology of the application, engineers are forced to map such changes through multiple levels of abstraction to identify the affected source code. Requirements-level documentation can greatly aid program understanding by illuminating original intention in an explicit manner.

Level 5 – Product lines: One step above single-product requirements and software architecture is product line documentation. This captures important information concerning the commonalities and variabilities in the product. Understanding how a specific product is related to similar products can greatly aid program understanding for domain experts with knowledge experience from past projects.

5. SUMMARY

Product quality has always been a somewhat ethereal concept that is difficult to quantify. In software engineering, assessing quality attributes such as maintainability is usually accomplished by measuring tangible attributes of the source code, such as cyclomatic complexity [20], and using the measurements as an indirect indicator of the quality attribute in question. However, quality attributes such as maintainability are primarily of interest to the developer; they are of less interest to the user of the final application.

Assessing documentation quality in the context of program understanding is equally challenging. Since system documentation quality is dependent on so many factors, such as the user's preferences for certain types of information (preferences which change during the program understanding task), it is difficult to define a unified framework of document quality that is suitable in all circumstances.

This paper presented preliminary work towards a documentation maturity model that attempts to capture the important factors of both the process and the product. The process component of the DMM currently consists of a skeletal structure that is based on the five stages of the CMM. The thesis underlying the process part of the DMM is that as an organization matures, its ability to produce quality documentation should improve accordingly. The five maturity levels of the DMM provide a foundation for continuous documentation quality improvement. As with the original CMM, moving up the DMM levels is not expected to be easy. By focusing on a limited set of activities and working aggressively, an organization should be able to steadily improve its organization-wide software documentation process, resulting in long-term benefits related to software evolution. However, this aspect of the DMM requires further research before these claims can be empirically validated.

The product component of the DMM uses a hierarchy of key product attributes, each of which has a five-level maturity index. The levels are inclusive, meaning that a higher level includes all the characteristics of the KPAs at lower levels. Taken together, the KPAs provide an objective indication of the quality of the program documentation.

The creation of the KPAs is based on several years of research into system redocumentation. A reverse engineering tool suite that produces documentation of high quality according to the DMM was developed and deployed in a commercial setting [13]. An ongoing project with an industrial collaborator helped classify the different formats of textual and graphical documentation [29]. Based on feedback from their developers, the relative maturity of each type of documentation was identified and formed the basis of the maturity levels of the related KPA. A series of experiments focused on assessing the qualitative efficacy of the UML as a form of graphical documentation in support of program understanding also helped shaped the DMM [28].

There are a number of avenues of future work in this area. The process component of the DMM needs to be completed, including a closer examination of the staged maturity levels and the development of the key process areas for each level. This part of the research would also benefit from more experimental trials, to ascertain the applicability, correctness, and completeness of the selected KPAs. The product component of the DMM should be further enhanced, with the candidate KPAs investigated for inclusion in the framework.

REFERENCES

[1] Bass, L.; Clements, P.; and Kazman, R. *Software Architecture in Practice* (2nd Edition). Addison-Wesley, 2003.

[2] Burnstein, I.; Suwannasart, T.; and Carlson, C. "Developing a Testing Maturity Model". *Crosstalk: The Journal of Defense Software Engineering*. Part I: 9(8):21-24, August 1996; Part II: 9(9):19-26, September 1996.

[3] Carnegie Mellon University, Software Engineering Institute (Paulk, M.; Weber, C.; and Curtis, B. contributors). *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, 1995.

[4] Carnegie Mellon University, Software Engineering Institute. *The Integrated Product Development Capability Maturity Model (IPD-CMM)*. Online at http://www.sei.cmu.edu/cmm/ipd-cmm.html. Accessed March 2003.

[5] Carnegie Mellon University, Software Engineering Institute. *The Software Acquisition Capability Maturity Model (SA-CMM)*. Online at http://www.sei.cmu.edu/arm/SA-CMM.html. Accessed March 2003.

[6] Carnegie Mellon University, Software Engineering Institute. *The Systems Engineering Capability Maturity Model (SE-CMM)*. Online at http://www.sei.cmu.edu/cmm/se-cmm.html. Accessed March 2003.

[7] Chikofsy, E.; and Cross, J. "Reverse Engineering and Design Recovery: A Taxonomy." IEEE Software 7(1):13-17, January 1990.

[8] Chrissis, M.; Konrad, M.; and Shrum, S. CMMI: Guidelines for Process Integration and Product Improvement. Addison-Wesley, 2003.

[9] Crosby, P. "Quality is Free." New York: McGraw-Hill 1979. (Ch. 24)

[10] Curtis, B.; Hefley, B.; and Miller, S. The People Capability Maturity Model: Guidelines for Improving the Workforce. Addison-Wesley, 2001.

[11] Gamma, E.; Helm, R.; Johnson, R.; and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[12] Hargis, G.; Hernandez, A.; Hughes, P.; Ramaker, J.; Rouiller, S.; Wilde, E. *Developing Quality Technical Information: A Handbook for Writers and Editors*. Prentice Hall PTR, 1998.

[13] Hartmann, J.; Huang, S.; and Tilley, S. "Documenting Software Systems with Views II: An Integrated Approach Based on XML." *Proceedings of the 19th Annual International Conference on Systems Documentation* (SIGDOC 2001: Santa Fe, NM; October 21-24, 2001), pp. 237-246. ACM Press: New York, NY, 2001.

[14] Humphrey, W. Introduction to the Personal Software Process. Addison-Wesley, 1996.

[15] Humphrey, W. Introduction to the Team Software Process. Addison-Wesley, 1999.

[16] IBM Corp. "UML Resource Center". Online at http://www.rational.com/uml.

[17] IBM Corporation (1983). *Producing Quality Technical Information*. Santa Teresa, CA.

[18] ISO/IEC 9126: Information technology – Software Product Evaluation – Quality characteristics and guidelines for their use, 1991.

[19] Mandel, T. "Quality Technical Information: Paving the Way for Usable Print and Web Interface Design". *ACM Journal of Computer Documentation*. 26(3):118-125, August 2002.

[20] McCabe, T. "A Complexity Measure." *IEEE Transactions on Software Engineering*, SE-7(4):308-320, September 1976.

[21] Pierce, R. and Tilley, S. "Automatically Connecting Documentation to Code with Rose". Proceedings of the *20th Annual International Conference on Systems Documentation* (SIGDOC 2002: October 20-23, 2002; Toronto, Canada), pp. 157-163. ACM Press: New York, NY, 2002.

[22] Scott, K. *UML Explained*. Addison-Wesley, 2001.

[23] Smart, K. "Assessing Quality Documents." *ACM Journal of Computer Documentation*. 26(3):130-140, August 2002.

[24] Sommerville, I. *Software Engineering* (6th Edition). Addison-Wesley, 2001

[25] Sun Microsystems. Javadoc. Online at http://java.sun.com/j2se/javadoc/.

[26] Tilley, S. "The Canonical Activities of Reverse Engineering." *Annals of Software Engineering* 9:249-271, 2000.

[27] Tilley, S. A Reverse-Engineering Environment Framework (CMU/SEI-98-TR-005). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998.

[28] Tilley, S. and Huang, S. "A Qualitative Assessment of the Efficacy of UML Diagrams as a Form of Graphical Documentation in Aiding Program Understanding." To appear in *Proceedings of the 21st Annual International Conference on Systems Documentation* (SIGDOC 2003: October 12-15, 2003; San Francisco, CA). ACM Press: New York, NY, 2003.

[29] Tilley, S. and Huang, S. "Documenting Software Systems with Views III: Towards a Task-Oriented Classification of Program Visualization Techniques". *Proceedings of the 20th Annual International Conference on Systems Documentation* (SIGDOC 2002: October 20-23, 2002; Toronto, Canada), pp. 226-233. ACM Press: New York, NY, 2002.

[30] Tilley, S. and Müller, H. "INFO: A Simple Document Annotation Facility." *Proceedings of the 9th Annual International Conference on System Documentation* (SIGDOC'91: October 10 -12 1991; Chicago, IL), pp. 30-36. New York, NY; ACM Press, 1991.

[31] Tilley, S.; Müller, H.; and Orgun, M. "Documenting Software Systems with Views." *Proceedings of the 10th Annual International Conference on Systems Documentation* (SIGDOC '92: Ottawa, ON; October 13-16, 1992), pp. 211-219. New York, NY: ACM Press, 1992.

[32] Tilley, S.; Whitney, M.; Müller, H.; and Storey, M.-A. "Personalized Information Structures." *Proceedings of the 11th Annual International Conference on Systems Documentation* (SIGDOC '93: Waterloo, ON; October 5-8, 1993), pp. 325-337. New York, NY: ACM Press, 1993.

[33] Waite, B. "Introduction". *ACM Journal of Computer Documentation*. 26(3):64-65, August 2002.

[34] Weber, A.; Kienle, H. and Müller, H. "Live Documents with Contextual, Data-Driven Information Components". *Proceedings of the 20th Annual International Conference on Systems Documentation* (SIGDOC 2002: October 20-23, 2002; Toronto, Canada), pp. 236-247. ACM Press: New York, NY, 2002.

[35] Weiser, M., "Program slicing." *IEEE Transactions on Software Engineering*, SE-10(4):352-357, July 1984.