Significance of Software Documentation in Software Development Process

Mr. Vikas S. Chomal
Assistant Professor
The Mandvi Education Society Institute of Computer Studies, Mandvi, Gujarat, India
Email: vikschomal80@gmail.com, vikschomal@yahoo.co.in

Dr. Jatinderkumar R. Saini
Director (I/C) & Associate Professor
Narmada College of Computer Application, Bharuch, Gujarat, India
Email: saini_expert@yahoo.com

Abstract – Incorrect, missing, brief, or out of data documentation are considered some of major attributes which can be responsible for poor software quality. Therefore, software development associations need to have appropriate document control strategies. For every software irrespective of its size and complexity software documentation is generated. Inappropriate generation or omissions of documentation sometimes even leads to software failure. Consequently, software engineers should pay substantial amount of time concentrating on generation of software documentation. Hence, through this paper our aim is to focus on the importance of software project documentation. The paper is divided into four sections; first section is introductory, followed by literature review. Section three represents finding and analysis from literature review, followed by concluding section.

Keywords – Software Documentation, Software Development, Software Projects.

I. INTRODUCTION

In broad requisites, documentation is any communicable matter such as text, video, audio, CD, DVD etc., or combinations thereof used to describe some characteristics of an object, system or procedure. It is frequently used in today’s information period to mean engineering or software documentation, which is typically paper books or computer readable files that portray the structure and components, or on the other hand, operation, of a system/product.

Software documentation is an essential feature of both software projects and software engineering in common. In piece of evidence, documentation engineering has become an accepted sub-domain in the software engineering society. The task of documentation in a software engineering milieu is to commune information to its spectators and instils knowledge of the system it describes [3].

Documentation is requisite in software development. Even though every software development project is exclusive and produces diverse categories of documents, different amount of documentation, and may employ different documentation methods and notations, we need to be able to control the documentation produced in software development projects in a uniform manner [2][16].

According to Cock & Visconti [9] a fundamental goal of software engineering is to produce the best possible working software along with the best supporting documentation.

Software development is partly a learning and communication process. Software developers need to communicate with each other and also with various interest groups of the system to be developed, such as customers, marketing people, end users, service personnel, and authorities. Documentation is the basis for communication in software development organizations as well as between development organizations and the interest groups of the system to be developed. To ensure efficient communication, all communicating parties need to be able to identify various software documents, and, to ensure that the right information is found, all communicating parties should be able to anticipate what information is in each document [15][22].

According to Sommerville [12], documents associated with a software project and the systems being developed have a number of associated requirements:
1. They should act as a communication medium between members of the development team.
2. They should be a system information repository to be used by maintenance engineers.
3. They should provide information for management to help them plan, budget and schedule the software development process.
4. Some of the documents should tell users how to use and administer the system.

Documentation produced falls into two classes:
1. Process documentation: These documents record the process of development and maintenance. Plans, schedules, process quality documents and organizational and project standards are process documentation.
2. Product documentation: This documentation describes the product that is being developed. System documentation describes the product from the point of view of the engineers developing and maintaining the system; user documentation provides a product description that is oriented towards system users.

II. RELATED LITERATURE REVIEW

Laitinen [15] puts forward that software development is supposed to be documentation-oriented, which means that documents are considered to be the most essential and valuable products of the development process. Documentation-orientedness involves considering such computer-process able products as source program modules and batch-files as documents. On the other hand, a product such as executable machine code is regarded as
a by-product in the development process, because in a development environment we can always derive correct executable machine code when we have the correct documents. The executable machine code is essential to the user of a computer system, but it is considered less important to the software developers. Further Laitinen [15] classifies software documentation into five types of classes, which are illustrated in Table – 1. There are five documents classes: 1) Software Descriptions describe a system “as it is” and the executable machine code is derived according to those, 2) Utilization Documents are needed in order that the system derived from Software Descriptions might be used, 3) Development Plans are needed to carry out the development of Software Descriptions and Utilization Documents in a disciplined manner, 4) Quality Control Documents are needed to control the quality of the Software Descriptions and Utilization Documents, and 5) Administrative Documents are needed to establish the financial basis for the software development and to control how well the development proceeds as planned.

Table 1: Classes of Software Documents [15]

<table>
<thead>
<tr>
<th>Software Descriptions</th>
<th>Utilization Documents</th>
<th>Development Plans</th>
<th>Quality Control Documents</th>
<th>Administrative Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Software Description</td>
<td>User's Manual</td>
<td>Responsibility Plan</td>
<td>Analysis Request</td>
<td>Development</td>
</tr>
<tr>
<td>Short System Description</td>
<td>Operator's Manual</td>
<td>Work Breakdown Plan</td>
<td>Information Request</td>
<td>Contract</td>
</tr>
<tr>
<td>Requirement Description</td>
<td>Installation Manual</td>
<td>Schedule Plan</td>
<td>Reader's Report</td>
<td>Extended</td>
</tr>
<tr>
<td>Implementation Description</td>
<td>User's Help</td>
<td>Phase Plan</td>
<td>Inspection Report</td>
<td>Maintenance</td>
</tr>
<tr>
<td>Configuration Description</td>
<td>Operator's Help</td>
<td>Risk Plan</td>
<td>Test Report</td>
<td>Contract</td>
</tr>
<tr>
<td>Software Description Appendices</td>
<td>Installation Help</td>
<td>Test Plan</td>
<td>Review Call</td>
<td>Review</td>
</tr>
<tr>
<td>Terminology Description</td>
<td>Service Help</td>
<td>Acceptance Plan</td>
<td>Inspection Call</td>
<td>Minutes</td>
</tr>
<tr>
<td>Internal Message Description</td>
<td></td>
<td>Manual Plan</td>
<td>Test Call</td>
<td>Project Meeting</td>
</tr>
<tr>
<td>External Message Description</td>
<td></td>
<td>Method Plan</td>
<td></td>
<td>Minutes</td>
</tr>
<tr>
<td>Record Description</td>
<td></td>
<td>Tool Plan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interface Description</td>
<td></td>
<td>Reporting Plan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Description</td>
<td></td>
<td>Quality Plan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initialization Description</td>
<td></td>
<td>Documentation Plan</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Version control</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chomal and Saini [26, 30] in their recent works listed the causes that lead to software project failures. Chomal and Saini [28] in another work considered documentation of software projects prepared by students as a source for data collection and analyzed it. Chomal and Saini [27, 29] in still another work had also described how software documentation can be used as a knowledge management model for improving quality of software as well as database.

Cock & Visconti [9] elucidate that empirical data shows that software documentation products and processes are key components of software quality. These studies show that poor quality, out of date, or missing documentation is a major cause of errors in software development and maintenance. For example, the majority of defects discovered during integration testing are design and requirements defects, e.g. defects in documentation that were introduced before any code was written.

Hopkins & Jeroow [14] describes documentation as, “Documentation is the castor oil of programming. Managers know it must be a good because the programmers hate it so much”.

Hunt & Thomas [11] explain that communication is achieved only when information is being conveyed. Based on the assumption that documentation is communication, the goal of a particular software document is to convey information. This information may not necessarily be completely accurate or consistent. Chapin [8] states that the value of software can be determined by many factors. One of the four main issues related to software value is its documentation. Chapin describes several main concerns regarding documentation: quality, obsolescence or missing content. To preserve and enhance software value, Chapin recommends to “keeping the documentation current and trustworthy” [8]. It is important to note that merely being current and trustworthy do not necessarily imply applicable or useful.

Forward [3] describes the relationship between models, documents and documentation below in Figure – 1.

Fig.1. The relationship between models, documents, source code, and documentation [3].

Curtis et al [10] believe that documentation should focus on how requirements and design decisions were made, represented, communicated and changed over the lifespan.
of a software system. As well, documentation should describe the impact of the current system on future development processes. Their study involved interviewing personnel from seventeen large software projects. Their analysis focused on the problems of designing large software systems; but many results report directly about the use (and misuse) of documentation in a software project. Forward & Lethbridge [1] expresses documentation as; the software documentation includes any artefact whose purpose is to communicate information about the software system to which it belongs. These artefacts include requirement, specification, and architectural and detailed design documents. These documents are geared to individuals involved in the production of that software, such as managers, project leaders, developers and customers.

III. FINDINGS AND ANALYSIS

In this section we present the findings from the literature reviewed in Table 2 about the relevant importance of software project documentation.

Table 2: Analysis of Literature

<table>
<thead>
<tr>
<th>Sr No.</th>
<th>Author(s)</th>
<th>Major Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Thomas, Bill, Dennis Smith and Scott Tilley [24]</td>
<td>They put forward that, software engineers rely on program documentation as an aid in understanding the functional nature, high-level design, and implementation details of complex applications. However, no one really knows what types of documentation are truly useful to software engineers to aid system understanding. This workshop focuses on issues related to this fundamental problem, such as what formats the documentation should take, who should produce it, and when. The juxtaposition of a technical communication audience with software engineering researchers and practitioners will provide new insights into the problem.</td>
</tr>
<tr>
<td>2.</td>
<td>Andrew Forward, Timothy Lethbridge [1]</td>
<td>The goals of their survey was to uncover the perceived relevance (or lack thereof) of software documentation, and the tools and technologies used to maintain, verify and validate such documents. The survey results highlight the preferences for and aversions against software documentation tools. Participants agree that documentation tools should seek to better extract knowledge from core resources. These resources include the system's source code, test code and changes to both. Resulting technologies could then help reduce the effort required for documentation maintenance, something that is shown to rarely occur. Their data reports compelling evidence that software professionals value technologies that improve automation of the documentation process, as well as facilitating its maintenance.</td>
</tr>
<tr>
<td>3.</td>
<td>Antonial, G., Canfora, G., De Lucia, A. and Merlo, E. [4]</td>
<td>Their studies revealed that software system documentation is almost always expressed informally, in natural language and free text. Examples include requirement specifications, design documents, user manual pages, system development journals, error logs and related maintenance reports. They propose an approach to establish and maintain traceability links between the source code and free-text documents. A premise of our work is that programmers use meaningful names for program's items, such as functions, variables, types, classes and methods. They believe that the application domain knowledge that programmers process when writing the code is often captured by the mnemonics for identifiers; therefore, the analysis of these mnemonics can help to associate high-level concepts with program concepts, and vice versa. In this paper, the approach is applied to software written in an object-oriented (OO) language, namely C++, to trace classes to manual sections.</td>
</tr>
<tr>
<td>4.</td>
<td>Nasution, M.F.F, Weistroffer, H.R [18]</td>
<td>They proposed that agile software development methods seem inherently suitable for today's quick-paced business environment as they shorten the time to develop new systems and typically incur lower development costs compared to the conventional systems development life cycle (SDLC) approach. Software development project failures using conventional SDLC are often attributed to project delays, resulting in budget overruns. On the other hand, a well planned and documented systems development project is more likely to result in a system that meets the expectations of both the intended users and the software engineers. This paper takes another look at conventional SDLC methodology by focusing on an aspect that is often overlooked in systems development practice, namely the significance of good documentation.</td>
</tr>
</tbody>
</table>
5. Sulaiman, S, Sahibudding, S [23]  
They explained the importance of documentation stating that, System documentation (SD) is undoubtedly vital as one of the sources in software understanding. Despite its importance, practitioners are often confronted with the problems related to SD. A number of tools have been introduced in order to assist documenting activities. However such tools are still not widely used because they generally fail to meet users' needs. Hence we have conducted a survey in Malaysia with the main goal to study software engineers’ current practice during software development and maintenance in relation with SD based on four types of data elements: characteristic, behaviour, belief and attitude. At the very outset, we need to establish what kind of tools should be introduced, why is it introduced and when or how should it be introduced to meet their needs in documenting activities. The findings of the study will argue whether it is relevant to introduce reverse engineering or a document generator tool to serve required information early in the development stage.

He indicated that, the field of software documentation is reviewed by examining manual writing before and after 1985. Changes in the field include an increased emphasis on satisfaction of users, improved management strategies and improved design techniques. Three books on software documentation published since 1988 are surveyed, and it is argued that the trends after 1985 reinforce a social constructionist view of documentation.

Briand states that, it is a well-known fact that software documentation is, in practice, poor and incomplete. Though specification, design, and test documents—among other things—are required by standards and capability maturity models (e.g., SEI CMM), such documentation does not exist in a complete and consistent form in most organizations. When documents are produced, they tend to follow no defined standard and lack information that is crucial to make them understandable and usable by developers and maintainers. Then a fundamental practical question, which motivated this keynote address, is to better understand what type of documentation is required, what is needed to support its completeness and consistency, and what is the level of precision required for each type of document. These questions cannot be investigated at that level of generality though. Answers are likely to be very context-dependent if they are to be precise. They focus our attention here on object-oriented development and the Unified Modelling Language (UML).

8. Walters, N.J, Beck, C.E [33]  
To discover the similarities and differences between primary and secondary computer manuals, and to account for the popularity of the secondary texts, two best-selling books for word processing and spreadsheet programs are compared to documentation supplied by the manufacturer. A heuristic for analyzing software documentation based on cognitive and rhetorical principles is developed and applied to the corporate documentation for (WordPerfect 5.0) in contrast to Stewart's Using WordPerfect 5 from Que, and the corporate documentation from 'Lotus 1-2-3' in contrast to Gilbert and William's 'The ABC's of 1-2-3 from Sybex.' It is shown that the trade texts from Que and Sybex contain more conceptual background information than the corporate documentation and differ in their rhetorical stance: the writers provide a richer context by giving more examples for applying the software; the writers provide global and structural frameworks; the writers use persuasive marketing techniques to ease the reader's anxieties and remind them of the software's benefits; and the writers identify themselves.

There experience and research based paper discusses the reasons that software cannot be trusted and then explains how the use of greatly improved documentation can make software more trustworthy. It shows how tabular expressions can be used to prepare software documents that are both precise and easily used by developers, inspectors, and testers. The paper reviews a number of "tried and true" ideas and illustrates some new refinements in the methods that resulted from recent research. It is intended both to tell developers of techniques available to them and to suggest new research areas.

10. Remo, C. Boer, Hans van Vliet [21]  
In their work, they focus on the effectiveness of documentation within a development process is determined by the way in which the intentions of the
authors correspond to the expectations of the potential readers. Ideally, the members of a development team share a certain understanding of (the role of) the different types of documentation. However, since one's expectations of a document are personal, and part of a tacitly formed mental model, we can expect different levels of shared understanding between different development team members. They elicited and analyzed the mental models of software documentation from eight members of a single development team. They found indeed different levels of shared understanding between different people. To their surprise, the levels of shared understanding within the team appear closely tied to the development process employed. From Conway's law we know that an organization's structure is mirrored in the structure of the software that the organization produces. Their findings suggest that the organization's development process may likewise be mirrored in the extent to which a development team shares a common frame of reference. Hence, the development process followed may have implications for the effectiveness with which development knowledge can be shared through software documentation.

11. Visconti. M, Cook C. R [32] A system documentation process maturity model and assessment procedure were developed and used to assess 91 projects at 41 different companies over a seven year period. During this time the original version evolved into a total of four versions based on feedback from industry and the experience gained from the assessments. This paper reports the overall results obtained from the assessments which strongly suggest that the practice of documentation is not getting a passing grade in the software industry. The results show a clear maturity gap between documentation practices concerned with defining policy and practices concerned with adherence to those policies. The results further illustrate the need to recognize the importance of improving the documentation process, and to transform the good intentions into explicit policies and actions.

12. Young F. H [34] The author describes the development of a course at Rose-Hulman Institute of Technology that integrates software engineering and software documentation. A brief evaluation is followed by a discussion of future plans. It is pointed out that the most significant benefit of the combined course is the improved software engineering abilities of the students. The students see the importance of presentation and writing skills in the software development process. They apply these skills in ways that improve their ability to develop large software systems.

13. Lethbridge, T.C., Singer, J. and Forward, A [17] They delineates Software Engineering and focuses on the importance of Software Documentation by stating that, Software engineering is a human task, and as such we must study what software engineers do and think. Understanding the normative practice of software engineering is the first step toward developing realistic solutions to better facilitate the engineering process. They conducted three studies using several data-gathering approaches to elucidate the patterns by which software engineers (SEs) use and update documentation. Their objective is to more accurately comprehend and model documentation use, usefulness, and maintenance, thus enabling better decision making and tool design by developers and project managers. Their results confirm the widely held belief that SEs typically does not update documentation as timely or completely as software process personnel and managers advocate. However, the results also reveal that out-of-date software documentation remains useful in many circumstances.

14. Qian Hu [20] According to author, the software documentation is unpopular among many developers at present while the documents are important for the staffs who work for secondary development and software maintenance. For this phenomenon, a teaching method of writing software documentation is proposed in this paper, in which the software maintenance is the driving force to make students fully understand and grasp the method of software documentation writing through an upgrade and maintenance software project. And students learn to write effective software documentation to establish the level and structure of the document.
15. Vir Phoha [31] While there is no universally recognized standard for software documentation, there is a standard for documenting engineering and scientific software. Developed by the American National Standards Institute (ANSI) and the American Nuclear Society (ANS) in 1995, it is called the ANSI/ANS 10.3-1995 Standard for Documentation of Computer Software. The standard provides a flexible, robust framework for documentation needs. One of its goals is to encourage better communication between developer and user and to facilitate effective selection, usage, transfer, conversion and modification of computer software. The standard is not a rigid set of specifications but a guide that can apply to most software projects intended for internal or external use. While the standard cannot cover all documentation problems, it is a good starting point, even for the most complex software. Similarly, while the standard provides recommendations for documenting scientific and engineering software, it doesn't offer guidance for online monitoring, control or safety systems, and doesn't specifically address the unique requirements of consumer-oriented software.

16. Bill Brykczynski [6] According to author software documentation can be considered as one of the checklist for inspection of software.

Janicki, Ryszard, David L. Parnas, and Jeffery Zucker [13] They describe the use of relations (represented as tables) as a means of documenting the requirements and behaviour of software. Their work highlights two key limitations to current documentation practices. They also state, that documentation continues to be inadequate due to the vagueness and imprecision of natural languages. These individuals believe that even the best software documentation is unclear. Further they, suggest that the limitation in natural languages is a primary cause of informal documentation. They further add that this informality of the documentation in turn cannot be systematically analysed, resulting in inconsistent and incomplete information. They also present a solution to building consistent and complete documentation by describing software using tabular mathematical relationships.

17. Tilley Scott [25] The author comments on the deficiencies of traditional software documentation techniques. The author also notes that it is extremely inefficient to break down communication into units so that anyone can understand them.

IV. CONCLUSION Thus we conclude that documentation is considered as an essential instrument since it conveys meaningful and functional information related to software.

REFERENCES
