AN ABSTRACT OF THE THESIS OF

Marcello Visconti for the degree of Doctor of Philosophy in Computer Science presented on December 2, 1993.

Title: Software System Documentation Process Maturity Model

Abstract approved:

Curtis R. Cook

One major goal of software engineering is to produce the best possible working software along with the best possible supporting documentation. Empirical data shows that software documentation process and products 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 These studies show that poor quality, out of date, or missing documentation is a major cause of errors in software development and maintenance. Virtually everyone agrees that documentation is important; however, in spite of these studies, they do not realize that documentation is a critical contributor to software quality. A solution to the problem of poor quality, out of date or missing documentation is to improve the documentation process.

This dissertation proposes a Software System Documentation Process Maturity model that provides the basis for an assessment of the current documentation process and identifies key practices and challenges to improve the process. The focus is on the documentation used in software development and maintenance, not end-user documentation. The approach has been influenced by Carnegie Mellon University's Software Engineering Institute (SEI) Software Process and Capability Maturity models. A four-level documentation maturity model has been designed. An assessment procedure (an assessment questionnaire and a scoring method) has been developed to determine where an organization's documentation process stands relative to the model. From the responses to the questionnaire it is possible to map

an organization's experience and past performance to a documentation maturity level and generate a documentation process profile. The profile indicates key practices for that level and identifies areas of improvement and challenges to move to the next higher level.

The software documentation maturity model and assessment procedure have been used to assess a number of software organizations and projects, and a cost-benefit analysis of achieving documentation maturity levels has been performed using CoCoMo, yielding an estimated return on investment of about 6:1 when moving from the least mature level to the next, and estimating positive returns, though decreasing, for the higher levels. These results support the main claim of this research: software organizations that are at a higher documentation process maturity level also produce higher quality software, resulting in reduced software testing and maintenance effort.

Software System Documentation Process Maturity Model

By

Marcello Visconti

A THESIS

submitted to

Oregon State University

in partial fulfillment of

the requirements for the

degree of

Doctor of Philosophy

Completed December 2, 1993

Commencement June 1994

Approved:

Redacted for Privacy

Professor of Computer Science in charge of major

Redacted for Privacy

Head of Department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis presented December 2, 1993

Typed by Marcello Visconti for Marcello Visconti

Acknowledgments

Dr. Curtis Cook, my Major Professor, was the biggest influence during my tenure as a graduate student at Oregon State University. As a teacher first and then as a research advisor he provided the guidance, advise, insight and encouragement without which it would not have been possible for me to complete this Doctoral degree. I am grateful to him for his support.

I thank the faculty on my committee for their guidance, insight and encouragement, and the many anonymous people who contributed to this research, by either reviewing early drafts or taking part in the experimental assessments. My sincere thanks to all of them.

I extend my gratitude to the Oregon State University Computer Science Department, to Universidad Santa Maria in Chile and to the Government of Chile for providing the financial support.

My wife Silvana and my daughter Bianca made all this effort worthwhile. This achievement is as theirs as it is mine. Without their love and support I could not have done it.

Table of Contents

List of Tables

Chapter 1

Introduction

1.1 Overview

One basic goal of software engineering is to produce the best possible working software along with the best possible supporting documentation. And yet, documentation seems to be considered a *second class* object and not as important as the software itself. However, empirical data shows that low quality or missing system documentation is a major cause of errors in software development and maintenance. Often documentation does not exist, or if it does, it is incomplete, inaccurate or out of date. This research addresses the problem of software documentation by focusing on the software documentation process; and the development of a 4-level Documentation Process Maturity model and assessment procedure whose objective is to assess an organization's documentation level and to identify potential areas for improvement. This approach is based on the principle of improving the quality of the software product by improving the quality of the software process. In particular, this thesis focuses on improving the quality of the documentation process.

1.2 Importance of documentation

Software quality has been identified as **the** goal of the 90s in the software engineering field [Basi91]. This section shows that software documentation process and products are key components of software quality and that poor or missing documentation is a major contributor to the *software crisis*, namely low product quality and high development and maintenance costs.

Documentation is the written record of what the software is supposed to do, what it does, how it does it and how to use it. Virtually everyone agrees that good documentation is important to the analysis, development and maintenance phases of the software process and is an important software product. However, not everyone acknowledges how important documentation is.

Documentation is probably most crucial to the maintenance phase, which accounts for 60%-75% of the total cost of the software [Hager91, Kell89]. Osborne [Osborne87] reports that documentation accounts for more than 60% of maintenance costs, and is involved in about one-third of maintenance tasks. A quick understanding of the existing software is a key activity of the maintenance process [Chapin88]. Chapin [Chapin87] asserts that maintenance people spend 40% of their time dealing with documentation. Fjelstad and Hamlen [Fjel79] showed that when making a program modification 47% of a maintenance programmer's time is spent studying the program source code and associated documentation. They also found that when correcting errors, the time increases to 62%.

Documentation has appropriately been called the *castor oil* of the software process—it is beneficial but no one likes to do it. Far too often documentation may not exist, or if it does exist, it may be incomplete, inaccurate, or out of date. Basili and Rombach [Romb87] studied an industrial maintenance environment and found that 20% of the maintenance problems were due to bad documentation, with the most frequent problems being documentation faults and documentation clarifications. They claim that better documentation can solve a big percentage of maintenance problems. According to Chapin [Chapin88], maintenance programmers report that for most maintenance tasks the source code is the only available and dependable documentation. Buckley [Buck89] claims that in most cases maintainers discover that the available documentation is not current. Poston [Post84] asserts that flawed or outdated documentation is more costly than no documentation.

Poor quality documentation is a major problem. In a survey of 487 data processing organizations, Lientz and Swanson [Lien81] found documentation quality ranked 3rd in the list of 26 maintenance problem items. They identify documentation quality and adequacy of design specs as accounting for 70% of product quality. Guimaraes [Guim83] claims that the documentation rating has an inverse relationship with the average yearly maintenance expenditures and that maintenance programmers felt that the most important document was an *"English narrative describing what the programs and modules are supposed to do"*.

Documentation impacts the analysis and development phases as well. Boehm [Boehm75] estimated that documentation costs run about 10% of total Software

Development costs. Scheff and Georgon [Scheff91] found that 85% of all software development errors are introduced during requirements, analysis or design. Ramamoorthy [Ramam88] asserts that 80% of software errors in large real-time systems are requirements and design errors due to ambiguity, incompleteness, or faulty assumptions. Basili and Perricone [Basi84] conducted a study to analyze the factors that cause program errors and found that misunderstanding of a module's specifications or requirements constituted the majority of detected errors. Card, Garry and Page [Card87] studied a production environment to evaluate the effectiveness of different technologies and their impact on productivity and reliability. They found that high-use of documentation improves productivity by 11% and reliability by 27% compared to low-use of it. To improve quality, they suggest effective documentation of each phase of development. Fagan [Fagan76] claims that documentation quality inspections are as important as program inspections when the goal is to increase productivity and final software quality. In one experiment he found that 82% of the total number of errors discovered during development were found during formal design and code inspections. A study at GTE by Howden [How78] compared the efficiency of design review and testing. Design reviews uncovered 45% of the errors taking 17% of the development time, whereas testing took up to 75% of the time to uncover just 10% of the errors.

Development impacts maintenance. Hager [Hager89] claims that 60% of the software costs associated with development occur during maintenance. This is because development methodologies do not provide adequate visibility for maintenance concerns. Thus ease and cost of maintenance are greatly affected by what

takes place during development. He believes that the documentation process plays a key role during development by acting as a design medium that helps organize the information, provides a framework to produce the necessary documents to develop the product, and addresses the need for continuity between the engineering phases through clearly defining document relationships.

1.3 Proposed solution: a maturity model

The Software System Documentation Process Maturity model proposed in this thesis has a structure based upon the structure of the Software Engineering Institute (SEI) Software Process Maturity and Capability Maturity models. The problem of poor quality documentation can be regarded as the inability to manage the software documentation process. The focus is on the documentation used in software development and maintenance and does not consider end-user documentation. In that perspective, the proposed documentation process maturity model along with an assessment procedure provides the infrastructure and support to move towards a solution. The model provides the basis for an assessment of the current documentation process and identifies practices and challenges to improve the process. This research shows that achieving a high documentation maturity level means producing higher quality software.

The documentation process model consists of 4 maturity levels, with **Level 1** representing the most immature and **Level 4** representing the most mature. Level 1 represents an ad-hoc, chaotic situation regarding documentation. Level 2

recognizes that documentation is important and there is a type of check-off list to ensure that all documentation is done. However, there is no consistent assessment of the quality of the documentation. Level 3 incorporates quality assessment of the documentation. Level 4 is attained when measurements of the quality of the documentation are fed into a process of continual improvement.

The model describes an evolutionary improvement path from an ad hoc, chaotic process to a mature, disciplined one. The model establishes some standard features against which it is possible to formally judge the maturity of an organization's software documentation process. This forms the basis for improvement plans for its documentation process.

The model is to be used along with an assessment procedure to map an organization's documentation status to one of the maturity levels, generating a documentation process maturity profile. The profile indicates the challenges and areas of improvement to move to next level.

The model was to be validated by showing that for a software project developed with a higher documentation process maturity level, the fraction of the total errors discovered during testing that are requirements and design errors was less than for projects developed with a lower documentation process maturity level. To get around the difficulties encountered with that approach, software cost estimation model CoCoMo was used to estimate cost-benefits of achieving each documentation maturity level. The results support this research's claim that investing in improving the quality of the documentation process can improve the quality of the software product, and significantly reduce testing and maintenance effort.

1.4 Organization of dissertation

The remainder of this dissertation is organized as follows. Chapter 2 describes the SEI's Software Process and Capability Maturity models. The documentation process maturity model was highly influenced by the SEI's models. Chapter 3 discusses important issues regarding software process models assessment procedures and validation. Chapter 4 describes in detail the Software System Documentation Process Maturity model and its associated assessment procedure. Chapter 5 describes the validation of the model based on software development model CoCoMo. Chapter 6 addresses the initial validation approach, its difficulties and the results from the assessments conducted. Chapter 7 addresses the main contributions of this work and identifies some open issues and areas for further research. Appendix A contains a sample assessment questionnaire used in the assessment efforts. Appendix B shows the assessment questionnaire questions sorted by subject area. Appendix C contains the definitions used to build the documentation assessment report after conducting a project assessment.

Chapter 2

SEI's Process Maturity and Capability Models

2.1 Overview

This chapter presents a summarized vision of the SEI's Process Maturity model, and its transition to the SEI Capability Maturity model. The chapter is organized as follows: the first section describes the purpose and objectives of the SEI and its software process models; the second section introduces the SEI's Software Process Maturity model; the third section addresses important criticisms generated in the software community about the model; and the fourth section presents the new version of the model, the SEI Capability Maturity model (CMM). This work by the SEI is the basis for the approach used and the documentation process model proposed in this research.

2.2 SEI background

The Software Engineering Institute (SEI) was formed by the Defense Department at Carnegie Mellon University in 1984 to establish standards of excellence for software engineering and to accelerate the transition of advanced technology and

methods into practice. One of its results was the Software Process Maturity Model, which was developed to assess software organizations' capabilities and to identify the most important areas of improvement [Humph88].

An important first step in addressing software problems is to treat the entire software development task as a process that can be controlled, measured, and improved. A process is a sequence of tasks that, when properly performed, produces the desired result. A fully effective software process must consider the relationships of all required tasks, the tools and methods used, and the skill, training, and motivation of the people involved.

To improve their software capabilities, organizations must take five steps:

- understand the current status of their development process or processes;
- develop a vision of the desired process;
- establish a list of required process improvement actions in order of priority;
- produce a plan to accomplish these actions; and
- commit the resources to execute the plan.

The maturity framework developed at the SEI addresses these five steps by placing a software process into one of five maturity levels given below and by identifying those areas where improvement actions are most likely to move the process to the next level.

2.3 SEI's software process maturity model

The model is presented in Table 2.1 [Humph91]. Each level establishes an intermediate set of goals to reach higher levels of process maturity.

Humphreys [Humph91] summarizes the model as follows : The five maturity levels reasonably represent the historical phases of evolutionary improvement of actual software organizations, represent a measure of improvement that is reasonable to achieve from the prior level, suggest interim improvement goals and progress measures, and make obvious a set of immediate improvement priorities once an organization's status in the framework is known. While there are many aspects to these transitions from one maturity level to another, the overall objective is to achieve a controlled and measured process as the foundation for continuous improvement.

The SEI developed an assessment questionnaire with the purpose of assessing software organizations' capabilities and identifying the most important areas of improvement [Humph88, Humph89]. The questionnaire consists of 101 Yes/No questions and determines the maturity level. The SEI Software Process Maturity Model Assessment Questionnaire and the validation efforts are further addressed in Chapter 3.

2.4 SEI's maturity model criticisms

SEI assessments have been somewhat controversial because they are closely related to a program called Software Capability Evaluations (SCE), which is used by US government agencies to judge how capable companies are at developing software.

Level	Characteristics	Key Challenges
1 Initial	*(Ad hoc/chaotic process)* No formal procedures, cost estimates, project plans No management mechanisms to ensure procedures are followed, tools not well integrated, and change control is lax Senior management does not understand key issues	Project management Project planning Configuration management Software quality assurance
2 Repeatable	*(Intuitive)* Process dependent on individuals Established basic project controls Strength in doing similar work, but faces major risks when presented with new challenges Lacks orderly framework for improvement	Training Technical practices (reviews, testing) Process focus (standards, process groups)
3 Defined	*(Qualitative)* Process defined and institutionalized Software Engineering Process Group established to lead process improvement	Process measurement Process analysis Quantitative quality plans
4 Managed	*(Quantitative)* Measured process Minimum set of quality and productivity measurement established Process database established with resources to analyze its data and maintain it	Changing technology Problem analysis Problem prevention
5 Optimizing	Improvement fed back into process Data gathering is automated and used to identify weakest process elements Numerical evidence used to justify application of technology to critical tasks Rigorous defect-cause analysis and defect prevention	Still human-intensive process Maintain organization at optimizing level

Table 2.1: SEI's software process maturity model

Bollinger and McGowan [Boll91] have criticized the SEI's process maturity model as a preparatory test for the SCE's. Humphreys and Curtis [Humph91a] have defended the use of such methods as effective tools to foster the improvement of US industrial software capability. Details of the controversy can be found in [Boll91, Humph91a].

The main criticisms in [Boll91] can be summarized as follows:

- Statistical and methodological problems determining the assessment scores.
- Final reports are not sufficiently detailed.
- Maturity model favors maintenance process rather than development process.
- When grading, Level 1 means *failed* so no effort needed to attain it.
- Artificial ordering due to multihurdle grading. Missing a few questions in a level does not allow answering questions in the next higher level, even though the answers could be positive.
- Grading algorithm too complex given the few questions involved.
- Inflexibility in scoring: a few questions missed may mean failing a given level when the rest of the practices may be quite strong. A profile is more useful.
- Specific problems with Yes/No questions: hard to generalize, subject to rapid obsolescence, encourage fixes of limited scope, overlook complex problems, unfairness to innovative approaches, ambiguity due to oversimplification, encourage myopic views, use of jargon.

- Yes/No questions cover a broad range of issues that are not clearly correlated.
- While levels 1–3 are clearly defined, that's not the case with levels 4–5.

They suggest modifying the assessment method, particularly the grading system of assigning one of five process-maturity level numbers to software organizations. They go as far as to say that "the SEI's process-maturity model is too incomplete to be viewed as the improvement goal for the software industry", although they acknowledge its merits as a good initial attempt.

The new version of the SEI model, described in the next section, addresses most of these criticisms.

2.5 New version of the model: the SEI's capability maturity model

The SEI made the transition from the current Maturity Model to the new Capability Maturity Model (CMM) in 1991 [Baum91].

The new model decomposes each maturity level -initial, repeatable, defined, managed and optimizing- into *key process areas* which identify goals to be reached before an organization can say it has achieved a particular maturity level. Each key process area is further defined to contain *key practices* which are procedures and activities that contribute most to achieving the goals of the key process area. Each key practice has one or more *key indicators* which offer the greatest insight into whether goals have been satisfied. These indicators form the basis for the SEI's maturity assessment material.

SEI took steps to deemphasize the score of an assessment or evaluation. The new model's end product is now a key process area profile instead of a single number indicating maturity level. The profile template lists each key process area as not satisfied, partially satisfied or fully satisfied. An organization's maturity level is set at the highest level at which it satisfies all key process areas on a continuing basis.

The CMM (Version 1.0) is fully described in [Paulk91, Weber91]. This initial release was reviewed and used by the software community during 1991 and 1992.

CMM Version 1.1 is the result of that feedback and the feedback from a workshop held in April 1992. The main differences between CMM Version 1.0 and Version 1.1 are changes made to improve the consistency of the key-practices structure, clarify concepts, and provide consistent wording. The maturity framework is left unchanged [Paulk93].

The following are the key process areas that the CMM maturity framework defines at each level:

- **Level 1** No key process areas.
- **Level 2**
  - Requirements Management
  - Software Project Planning
  - Software Project Tracking and Oversight
  - Software Subcontract Management

  - Software Quality Assurance
  - Software Configuration Management

- **Level 3**
  - Organization Process Focus
  - Organization Process Definition
  - Training Program
  - Integrated Software Management
  - Software Product Engineering
  - Intergroup Coordination
  - Peer Reviews

- **Level 4**
  - Quantitative Process Management
  - Software Quality Management

- **Level 5**
  - Defect Prevention
  - Technology Change Management
  - Process Change Management

A detailed description of the CMM Version 1.1 and its key practices can be found in [Paulk93a, Paulk93b].

2.6 Summary

The SEI Capability Maturity model (initially SEI Software Process Maturity model) has become a *de facto* standard in the software industry. It has generated a lot of attention given its importance as an evaluation tool for the U.S. Department of Defense's software contractors. The model has moved away from the initial version that stressed the numeric maturity level to a version that is much more concerned with identifying problem areas and potential for improvement. In spite of that evolution, for many software organizations and managers the main purpose of undergoing a software process assessment is to achieve a given *grade* rather than to determine how to improve their software practices. The SEI has also failed to conclusively validate its claims that moving to the next level in their model actually improves software productivity and quality. They have not provided hard data to support their claims, but have rather reported on a few successful experiences. Chapter 3 gives a more detailed description on the SEI assessment procedures and its efforts to validate their models.

Chapter 3

Software Process Assessment and Validation

3.1 Overview

This chapter discusses relevant issues of software process models assessment and validation. It takes a detailed look at different approaches (SEI, ISO-9000, others) for software process assessments and improvement, and addresses the main issues on software model validation. The first section addresses relevant issues regarding software process assessment: the SEI software process assessment procedure, the ISO-9000 standard, and some variations of them as software process improvement tools. The second section concentrates on validation of software models; it discusses the SEI approach to validate its model, and describes other experiences. The issues discussed in this chapter form the basis of the assessment and validation effort of this research.

3.2 Assessment

The purpose of the software process assessment procedure for a model is to determine where an organization stands relative to that model. According to

Pressman [Press93], process assessment refers to both qualitative and quantitative information gathering about the software process. The determination of current state of process practices lays the foundation for the creation of a transition plan that should lead to improvement in software engineering procedures, methods, and use of tools. The primary objective is to improve the quality of the product and the productivity of the people who build it.

A process assessment helps an organization characterize the current state of its software process and provides findings and recommendations to facilitate improvement. The intent of process assessments is to understand the state of software engineering practices in the organization, to identify key areas for improvement, and to initiate actions that will lead to improvements. An assessment instrument should aid in determining the state of software engineering practices in an organization. The assessment instrument can be an assessment questionnaire; an expert (or team of experts) who assess the state of software engineering practices according to some ad-hoc ranking scheme by using a system of interviews, observations, or some other subjective method; or a mixture of them. The assessment instrument must be applied in the proper context and used skillfully.

There are several contexts in which software process assessments can be conducted [Humph87]:

- Externally-assisted assessments of an organization in which an assessment team conducts in-depth interviews with project teams and formulates a composite profile of the state of the practice in the organization.

- Self-assessments conducted by a project or organization in order to determine the state of its software engineering practices.

- Workshop assessments conducted in conjunction with a conference or other tutorials. This type of assessment can be useful in quickly gathering industry profile data, generating a high volume of feedback on the quality of the assessment instrument, and providing broader awareness of the assessment process and its benefits.

- Contractor evaluation assessment.

Humphrey et al.[Humph87] define the following key principles in conducting a successful assessment:

- Confidentiality: A software assessment is done for the benefit of the organization and must not be used as a means of evaluating individuals. This is essential, so individual answers to the assessments must be kept confidential.

- Action orientation: The entire motivation of an assessment must be directed toward improvement. The orientation is action so the assessment instrument must focus on defining those problems that need to be addressed right away.

- Senior management involvement: Senior managers must be convinced of the action's importance if anything is to happen.

- Non-adversarial attitude: Focus on learning and understanding. Convince people to contribute, so assessment is the catalyst to motivate self improvement.

The following section gives an overview of the assessment procedure devised by the SEI to complement its process maturity models.

3.2.1 SEI models assessment procedure

The SEI process-maturity structure is intended to be used with an assessment methodology and a management system. The assessment methodology identifies the organization's current software process maturity status and the most critical issues to improve their process. A management system establishes a structure for actually implementing the priority actions necessary to improve the process. Once its level in this maturity structure is defined, the organization can concentrate on those items that will let it advance to the next level [Humph88, Humph89]. A complete description of the SEI assessment methodology can be found in [Humph87].

Assessment phases

SEI-assisted assessments are typically conducted in five phases:

1. During the first phase, an organization is identified as a desirable candidate for assessment. After the initial contact, the SEI obtains organizational commitment to the full assessment process. This commitment includes the personal participation of the senior site manager, site representation on the assessment team, and agreement to follow up on recommended actions. An assessment agreement, which includes these and other elements of the joint agreement, is signed by the organization and the SEI.

2. The second phase is devoted to preparing for the on site assessment. An assessment team is selected and trained, and an assessment plan is formulated.

3. In the third phase, the on-site assessment is conducted. On-site presentations are made to orient site personnel who will be participating in the assessment. The assessment instrument is applied, and the resulting data and information is reviewed and evaluated by an assessment team. The final on-site activity is to present preliminary findings to assessment participants and senior site management.

4. The fourth phase is concerned with formulation and communication of final assessment findings and recommended actions. The assessment team prepares a formal written report which, along with a formal on-site briefing, is presented to the organization.

5. The final phase consists of post-assessment follow-up activities. An action team composed entirely of professionals from the organization is assembled and charged with planning and implementing the recommended actions. Typically, there is also some continuing support and guidance by the SEI, as well as a subsequent follow-up assessment to determine the effectiveness of changes.

Assessment questionnaire

The SEI developed an assessment questionnaire with the purpose of assessing a software organization's capabilities and identifying the most important areas of improvement [Humph88, Humph89]. The SEI Software Process Maturity Model

Assessment Questionnaire is fully described in [Humph87a], along with descriptions of the technical approach, self-assessment usage guide and guidelines for evaluation of results. The SEI questionnaire consists of 101 questions organized in the following way:

- Organization and Resource Management (17 questions): Organizational Structure (7 questions); Resources, Personnel and Training (5 questions); and Technology Management (5 questions). Questions in this section deal with functional responsibilities, personnel, and other resources and facilities.

- Software Engineering Process and its Management (68 questions): Documented Standards and Procedures (18 questions); Process Metrics (19 questions); Data Management and Analysis (9 questions); and Process Control (22 questions). Questions in this section are concerned with the scope, depth, and completeness of the software engineering process and the way in which the process is measured, managed, and controlled.

- Tools and Technology (16 questions). This section deals with the tools and technologies used in the software engineering process. It helps determine the effectiveness with which the organization employs basic tools and methodologies.

The assessment instrument is a structured set of yes/no questions organized by software process areas. It has also been designed to assist U.S. Department of Defense acquisition organizations in identifying software contractors with acceptable software engineering capabilities. Since the instrument and method for applying it

are publicly available, software contractors can use it to identify areas for improvement within their organizations. Besides training its own teams, the SEI provides training on how to conduct effective assessments for organizations interested in conducting their own assessments. The result of the assessment is the software process maturity level and a list of recommended actions for improvement.

Evolution of the assessment procedure

The assessment procedure has been the target of criticism [Boll91, Humph91a]. The main criticism of the assessment procedure is the excessive role assigned to the numeric maturity level as there is little inherent value in the number itself. The assessment does not explicitly identify areas of improvement. These problems have been addressed in the new SEI Capability Maturity model [Paulk91, Weber91], which deemphasizes the numerical score and emphasizes the identification of key process areas, key practices and key indicators. It produces a key-process-area profile, rather than a single numeric maturity level. This modification of the assessment procedure supports a more explicit identification of areas of improvement, since each process area is now judged according to the level of satisfaction.

The SEI Capability Maturity Model uses the same questionnaire, but the questions were reordered to simplify the identification of key process areas, practices and indicators introduced in the model, and the transitions among them. Every process area is decomposed into practices, practices into indicators, and indicators into questions in a straightforward manner.

Composition and training of the assessment team

The assessment team is composed of experienced software developers and whenever possible at least one member with experience in each phase of the software development and maintenance process. Three to five professionals are typical, although more may be used if desired. A team with more than ten members can be expensive or hard to manage. The SEI team leader must have had prior assessment experience and be familiar with the software development and maintenance process. All the team members, however, should be open-minded and capable of making presentations and gathering information in a non-threatening manner. At least one member from the organization being assessed should participate as a full member of the team to facilitate the planning process, to provide a means for the team to learn about the organization, and to establish a focal point for later action. Since the local member is so crucial to the success of the entire effort, the senior site manager should personally make the selection.

The training of the assessment team is intended to familiarize them with the assessment activity and to build a coherent team. This training consists of the following sessions:

- SEI Overview.
- Process Management Overview.
- Site Overview.
- Assessment Introduction and Guidelines.

- Assessment Questionnaire Review.
- Supporting Material Discussion.
- Assessment Evaluation and Findings.
- Conclusions and Recommendations.
- Planning for site visits.

Conducting the assessment

The assessment is composed of the following activities:

- Introductory presentation: Brief statement of the agenda, introduction of the assessment team members, review of assessment agreement, addressing of questions or concerns.
- Applying the assessment instrument: Detailed discussion of the assessment process, formulation of initial responses to the questions, initial determination of maturity level, review progress of assessment.
- Functional areas interviews: Determination of actual details of the software process.
- Project feedback reviews: Review of findings for projects, gathering of additional data, identification of other problem areas.
- Presentation of findings: Preparation and presentation of individual projects and composite organization findings.

Final assessment findings and recommendations

The final report should contain the following information:

1. Summary and conclusion: an executive level summary of the assessment team's findings.

2. Software assessment: a brief description of the context of the assessment and a chronology of key events.

3. Site status: a description of the software site under software process assessments in terms of composite maturity levels and technology state.

4. Key findings: a list with the most important findings regarding the status of the software process practices.

5. Recommendations: the assessment team's recommended actions to improve the practices as identified above.

6. Assessment agreement: a copy of the signed assessment agreement as an appendix (agreed upon at the beginning of the assessment).

Post assessment follow-up

The goal of the initial assessment is to characterize the current state of software engineering practice in the organization and identify key next steps for improvement, but the ultimate intent is to be the catalyst for improvement. The following activities can increase the likelihood that improvements will occur:

- Action plans.
- Periodic site-management review.
- Process group formation.
- SEI continuing support and guidance.
- Follow-up reassessment (after a year or so).

3.2.2 ISO-9000 quality standard

Many nations (particularly in Europe) are making the International Standards Organizations (ISO) certification a prerequisite for doing business. Their intention is to exclude suppliers who aren't serious about quality. As a result, organizations must have explicit ISO certification in order to prove their capabilities.

The general purpose of ISO 9000 is to improve quality by assuring a contractual basis for defining it. The ISO 9000 series consists of five documents, two guidelines (ISO 9000 Quality Management and Quality Assurance Standards and Guidelines for Selection and Use, and ISO 9004 Quality Management and Quality System Elements) and three models of quality assurance systems (9001, 9002, and 9003). Software and information systems developers are instructed to use 9003 "Guidelines for the application of ISO 9001 to the Development, Supply and Maintenance of Software", published in 1991. Strictly speaking, the software developer's process must conform to requirements specified in ISO 9001. Certification of conformance is actually a registration (accreditation) of an audited and accepted quality management system. The certification is performed by trained certifiers. ISO 9001

and the accompanying, interpretation in, 9003 guidelines, represent an international standard but compared to government standards like the mandatory DoD 2167A and 2168 and the voluntary industry standards like IEEE's software engineering series, ISO 9003 is more like a complete quality system framework rather than an explicit directive.

The intention of ISO 9000 was to bring the various competing standards under a single standard. It actually originated in 1959, with DoD's Quality Management Program (MIL-Q-9858). This standard itemized a set of quality system requirements which was adopted with slight changes by NATO (AQAP-1) and used by both through the 1970s. In 1979, the British Standard Institute (BSI) developed the first quality management standard for industry (BS 5750) from the AQAP-1 standard. The general intent of BS 5750 was to define the requirements for a quality system that would fit most businesses. None of this was specifically software related. In 1987, ISO began publishing the ISO 9000 series (developed directly from BS 5750). The U.S. adopted ISO 9000 through the Q-90 standards published by the American National Standards Institute (ANSI) and the American Society for Quality Control (ASQC). Under normal circumstances, a company can use preparation for ISO 900x audit to assess, consolidate and improve their quality assurance systems as a means of quality process management [Jovan93].

Both SEI and ISO deal with the promotion of quality within a conventional life-cycle. Each extensively covers management of development, training, testing and reviewing, change control, standards and metrics. As for the differences, the SEI emphasizes the whole process of software production. Accordingly, it stresses

such factors as productivity, sizing and costing and the transfer of experience across and within projects, e.g. reuse of designs, prototyping, process improvement and the use of a metrics database, whereas ISO does not greatly concern itself with these.

However, ISO has emphases which SEI omits. SEI is not particularly concerned with project control when the development process is virtually complete. Hence, large sections of ISO (acceptance, replication, delivery and installation) receive little coverage in SEI. Nor, surprisingly, does maintenance, being presumably treated as an extension of the original contract. The BSI rightly recognizes the unique problems of maintenance by supporting ISO with a separate Quality Assurance Schedule for maintenance.

SEI does not reflect, as ISO does, the immense problem of documentation control on projects, and some aspects of project control are not found in SEI, e.g. "included software product" and "product requirement specification", if done by the supplier. ISO places more emphasis on interfaces, both managerial and technical [Gilch92].

3.2.3 Other assessments

Gilchrist [Gilch92] adapted the SEI Process Maturity model for individual projects and assessed a set of software projects of all sizes and over an extensive period. He proposed using a spreadsheet to store and process the information, and analyzed some merits and difficulties with the approach. While Gilchrist downplays

the maturity level concept, he does stress the importance of identifying a project's deficiencies and a prioritized action list to remove them.

Seddio [Seddio92] describes the application of review and product metrics to the software engineering process at Eastman Kodak. Reviews were held for software requirements specification and high level design, and for software programs, subsystem test plans and code coverage results. The assessment tools employed were a software requirements specifications checklist to determine documentation, completeness and testability ratings, and a code review measurement matrix. Seddio claims that the use of this simple software assessment approach has resulted in enhanced understanding of the software process, more objective assessments of the quality of the software products, and more objective software products reviews.

Henry and Henry [Henry92] propose a methodology for assessing the software process used by an organization, integrating the principles of Total Quality Management (TQM) and the work of the SEI. The basis for integrating these 2 approaches is that the SEI assessment specifies the activities comprising the software process, and the TQM implementation phase evaluates the effectiveness of the activities. The 4 steps of their assessment methodology are: investigate the process, model the process, gather data, and analyze data. They claim their approach provides insight into both the content and the effectiveness of an organization's software process.

Other efforts to drive company-wide or industry-wide software process improvement have been reported in [Foster93, Strig93, King93, Cousin93].

3.2.4 Costs and difficulties of assessments

The costs for process assessments are the time and effort involved to prepare the assessment instrument, to prepare for the assessment, to conduct it, to analyze its results, and to draw the conclusions. This is highly variable depending on the assessment instrument used and the extent of the process under assessment [Press93].

Some of the difficulties of process assessments are:

- The creation of an assessment instrument that will properly cover the topics under investigation and that will gather the information sought. This means to determine which questions to ask, how many, in which order, the type of answers, among others.
- The potential variability of the answers that the respondents in a given organization may provide. A thorough pre-assessment analysis of the questions with the potential answerers can be very helpful in achieving consistency in the interpretation of the questions.
- The training of assessment experts, i.e. people who will make up the assessment teams used to assess organizations.
- The limited scope of the assessment: the assessment instrument cannot cover all the issues regarding the process, but should cover the most critical ones.
- The analysis of answers, determination of findings and recommended actions, and the implementation of improvement plans.

3.3 Validation

Validation of a software model is the process of developing convincing evidence that the model works, i.e. the model does explain the software phenomenon of interest. Ejiogu [Ejiogu93] stresses the difference between validation and correlation analysis: while the latter has the goal of calculating the degree of correlation or agreement between two or more target models, the former has the goal of finding any logical grounds to accept or reject a given model based on some admissible properties it possesses, so that the validation of a model means *proving* that its definitions, target attributes of software behavior, and mathematical function postulates are correct. By necessity a model only incorporates what is believed to be important and excludes what is not considered important. Achieving proof of correctness for software models is almost impossible, for more than one model may represent the same software phenomenon and it is not possible to prove one as *the* correct model over the others, and because many software factors modeled cannot be precisely expressed in unequivocal mathematical terms. This is reflected on an ever-present inability to prove correctness of software models.

Traditionally, software models have been validated by demonstrating they produce the results claimed, rather than by proving their correctness. One way to do that involves the collection of data to be statistically interpreted (i.e. correlation analysis). Analyzing historical records or performing controlled experiments are two ways to gather data [Conte86]. Data can also be gathered through software analyzers, report forms, and interviews. The goal of statistical interpretation is

the identification of significant relationships between and among software features, so that the variation on one software feature can be explained by the variation of another. Many software estimation models have been validated that way. CoCoMo is a good example: a widely known software cost estimation model proposed by Boehm [Boehm81], this model was validated against a database of 63 projects completed during a 15-year period. The projects were written in different languages, vary in size and type. Cost driver attributes ratings were assigned to each project using *expert judgment.* To derive the equations, the author used a combination of experience, results of other cost estimation models, the subjective opinion of experienced software managers, and trial-and-error to arrive at initial model parameters that were further refined and calibrated. The Intermediate version of the model produced excellent results, at least when applied to its own database. Other software models and their validation approaches are described in [Conte86].

3.3.1 Role of measurement in software engineering

Without measurement it is impossible to determine whether any improvement is made. By evaluating productivity and quality measures, senior management can establish meaningful goals for improvement of the software engineering process. To establish goals for improvements, the current status of software development must be understood. Hence, measurement is used to establish a process baseline from which improvements can be assessed. The collection of software quality metrics (e.g. number and distribution of software defects by phase) enables an organization

to tune its software engineering process to remove the vital few causes of defects that have the greatest impact on software development [Press93].

Metrics should be an integral part of all transition plans. Grady and Caswell [Grady87] define 10 steps of a successful transition strategy for software metrics:

1. Define a set of objectives for your measurement program.
2. Assign responsibility.
3. Do research.
4. Select the metrics that you plan to collect during the early stages of your measurement work.
5. Sell the metrics program to management and technical people.
6. Obtain tools that enable automatic collection and analysis of metrics data.
7. Establish a metrics education program.
8. Encourage participation by publicizing success and soliciting feedback.
9. Create a metrics database.
10. Remain flexible.

Measuring a software process is not simple. Before one can use the measured data with confidence, one needs to be aware of many potential issues [Yeh93]:

- Calibration of measurement. Lack of calibration will introduce a large component of variability into the measured result.

- Measurement vs. interpretation. What is measured and what is wished to be measured may not be the same thing.
- Measurement errors. Watch out for potential errors introduced through misunderstanding of the metrics or counting rules or errors in the measurement tools, which cause the errors to show up in a systematic pattern.
- Be aware of assumptions. Interpretation of measurements results through certain models or assumptions.
- Lack of data issue. Many of the software processes generate only one or two data points over a long period of time. This makes doing meaningful data analysis very difficult. Even in that case it is useful to take measurements to have a benchmark to detect improvement.
- The after the fact issue. Some of the data are not complete until after the fact. It is important to collect these data in order to complete the feedback loop to improve the process.
- Comparing measures. Quality and productivity data from different projects must be compared very carefully, to draw a meaningful conclusion from the data set. To do that, set up precise definitions and counting rules, and apply them uniformly and rigorously.

3.3.2 SEI models "validation procedure"

As part of its effort to validate the initial version of its process maturity model and assessment procedure, the SEI conducted 2 process assessments for

Hughes Aircraft's Software Engineering Division (SED) : one in 1987 and the other one in 1990 [Humph91]. According to the SEI, between 1987 and 1990 Hughes' SED moved from level 2 to level 3 in its 5-level maturity model. The 1987 assessment identified the strengths and weaknesses of the SED, and proposed some actions for process improvements. The 1990 assessment found a strong level 3 organization. SEI claims that the process improvement was tremendously cost-effective. The assessment itself cost Hughes $45,000 and the two-year improvement plan cost about $400,000. The revenues (represented by better working conditions, employee morale, scheduling, and costs) were estimated at $2 million annual savings, by comparing the *Cost Performance Index* variation in the period.

A detailed description of the assessment procedure can be found in [Humph87, Humph87a, Humph91], although the SEI does not provide details about how they arrived at the costs and estimated savings figures they claim. For a true validation of its model, the SEI needs to obtain quantitative estimations of costs and benefits associated to achieving each software process maturity level, and show those estimations are actually supported by real-world data by applying its assessment procedure and process improvement plan to a number of companies/projects at all maturity levels. Without that, the SEI has not actually *validated* its maturity model and associated assessment procedure, but rather reported on a successful experience. Hence, the data presented in these validation studies is mostly *anecdotal.*

The fact that the SEI has not provided a quantitative estimation of the savings and costs of moving from one level to the next for all 5 levels shows how difficult it is to validate maturity models like the one proposed in this thesis.

Software Process Maturity Level	% of Sites
1	81
2	12
3	7
4	0
5	0

Table 3.1: SEI assessment results

3.3.3 SEI assessments: state of the practice

The results of SEI software process assessments of 296 projects at 59 sites indicate that many organizations have low maturity software process [Over93]. Table 3.1 presents a summary with the percentage of sites found for each software process maturity level. The small number of organizations at SEI maturity level 2 and 3 suggest that, in general, the process used in most organizations lacks the controls, precision, and accuracy required to achieve predictable, repeatable results. Further, the lack of organizations at SEI maturity level 4 and 5 indicates that process measurement and evolution are nonexistent at the organization level. Based on this data, the software engineering state-of-the-practice can be characterized as largely crisis-driven, or ad-hoc [Humph89a].

3.3.4 Other process improvement success stories

Raytheon's Equipment Division started a process-improvement initiative in 1988, and has improved its bottom line, increased productivity, and changed the corporate culture. They have been able to convince top-level management that software process improvements within that division have yielded a $7.7 return on

every dollar invested, a two-fold increase in productivity, and an evolution from level 1 (Initial) to level 2 (Repeatable) to level 3 (Defined) process maturity according to the SEI's Capability Maturity Model [Dion93]. Their software improvement effort generated an $8.2 million savings from reduced cost of non-conformance (costs savings generated by reduction in rework). Their investment in software process improvement totaled $1.1 million [Over93].

The Aircraft Software Division (LAS) of the Oklahoma City Air Logistics Center assessed their Test Program Set and Industrial Automation processes using the SEI process assessment methodologies. They implemented 44 improvements, and gathered return-on-investment data on 18 of them. They concluded that the 18 improvements cost $462,000 and generated $2.935 million for a return-on-investment of 6.35 to 1 [Lipke92].

3.3.5 Costs and difficulties of validation

Validation by correlation analysis has the problem that it requires many data points to be statistically significant. Moreover, when dealing with process improvement there are some situations that create problems that cannot be easily solved, if at all. Some of these problems are:

- Difficulty of gathering data. Many organizations do not keep historical records of their developments, and when these are kept they are not always accurate.

- In the case of a software process maturity model it may be the case that there are no organizations at the higher levels, as happens in the SEI models. Is

it worth the extra effort to achieve the higher levels? That question is very difficult to answer without actual supporting data.

- It may be very difficult to associate quantitative improvements with a single cause. It is perfectly possible that a given improvement in the software process comes about as a result of implementing concurrently two or more recommended actions. One action alone may not produce the desired effect unless other changes are also made. How much improvement can be credited to each action?

- It is even more difficult to completely determine the costs and benefits associated with any improvement for many of these are non-tangible ones.

One way to try and get around some of those problems is by using generally accepted software development models that describe similar software issues. A study done at Lockheed [Over93] predicts some substantial improvements in quality and productivity using the SEI's model. Based on a comparative study of several projects at Lockheed, the data (normalized for a 500 KSLOC (thousand source lines of code) project and extrapolated for level 4 and 5) predicts that the increases in quality and productivity between a level 1 organization and a level 5 organization are on the order of 100-fold for quality and 10-fold for productivity. Unfortunately, the actual confirmation of these claims is impossible given the lack of organizations at the levels 4 and 5. Therefore, increases in quality and productivity for the two highest levels are just good *guesstimates*.

3.4 Summary

Assessments are unavoidable in any improvement plan. It is absolutely necessary to know the current status of the assessed organization to identify potential problems, to be able to propose improvement actions and to measure their effects. The validation of assessment procedures that complement process models like the SEI's or the one proposed in this thesis is a very difficult task, because they need to be proved as reasonable predictors of the software process. By themselves, assessments are not a solution to the software process problems. The actions taken to improve the key practices as indicated by the assessments are the solution.

The following chapters describe the documentation process maturity model, its assessment procedure and its validation.

Chapter 4

System Documentation Process Maturity Model

4.1 Overview

This chapter describes in detail the 4-level System Documentation Process Maturity model introduced in this thesis, which is based on both the SEI Software Process Maturity model and the Capability Maturity model. The first section presents the System Documentation Process Maturity model; the second section concentrates on the assessment procedure devised to complement the model; and the third section addresses the process of determining documentation process maturity and the improvement actions associated with each maturity level.

4.2 The model

The System Documentation Process Maturity Model consists of 4 maturity levels. Each proposed maturity level is described as follows:

- Name
- Keywords

	Ad-hoc	Inconsistent	Defined	Controlled
Keywords	Chaos Variability	Standards Check-off list Inconsistency	Product assessment Process definition	Process assessment Measurement, Control Feedback Improvement
Succinct Description	Documentation not a high priority	Documentation recognized as important and must be done	Documentation recognized as important and must be done well	Documentation recognized as important and must be done well consistently
Key Process Areas	Ad-hoc process Not important	Inconsistent application of standards	Documentation quality assessment Documentation usefulness assurance Process definition	Process quality assessment and measures
Key Practices	Documentation not used	Check-off list Variable content	SQA-like teams for documentation quality and usefulness Consistent use of documentation tools	Minimum process measures Data collection and analysis Extensive use of documentation tools and integration with CASE tools
Key Indicators	Documentation missing or out of date	Standards established	SQA-like practices Consistent use of documentation tools	Data analysis and improvement mechanisms
Key Challenges	Establish documentation standards	Exercise quality control over content Assess documentation usefulness Specify process	Establish process measurement Incorporate control over process	Automate data collection and analysis Continual striving for optimization

Table 4.1: Documentation process maturity model—summary table

- Key Process Areas (to define the basic characteristics)
- Key Practices (procedures, activities and typical scenarios)
- Key Indicators (needed in the assessment)
- Key Challenges (to move to the next level)
- Key Significance (to software development and maintenance)

The concepts *key process areas, key practices and key indicators* were taken from the SEI's Capability Maturity model whereas the concept *key challenge* was taken from the SEI's Software Process Maturity model. The concept *key significance* has been added to stress the impact of each level of documentation on software development and maintenance.

A summary of the model is presented in Table 4.1.

- **Level 1 : Ad-hoc**
  - Keywords: Chaos, variability.
  - Key Process Areas : Non-dependable documentation. System documentation process non-existent, ad-hoc or chaotic. Importance of system documentation not understood.
  - Key Practices : Since documentation may not exist or may be out of date, documentation is not used. No company standards about types of documentation that should or must be created.
  - Key Indicators : Missing documentation or out of date documentation. Use of code rather than documentation during maintenance.

- Key Challenges : Realization of importance of documentation. Establish required documentation standards. Define what documentation must be created.
- Key Significance :
  * Development : The non-dependability of the documentation creates heavy reliance on informal communication creating the conditions for ambiguities, inconsistencies, incompleteness, etc. This results in an environment conducive to errors.
  * Maintenance : The only dependable documentation is the source code, so program modifications are costly, time-consuming and error-prone.

- **Level 2 : Inconsistent**
  - Keywords: Documentation standards, required documentation, inconsistency
  - Key Process Areas : Inconsistent application of documentation standards. Poorly controlled documentation process. No consistent use of documentation tools. No attempt to gauge usefulness of documentation or how frequently it is used.
  - Key Practices : Documentation exists, but varies in content and completeness. Check-off list of required documentation. Use of simple documentation tools in creation and maintenance of documentation. No quality control over content.

- Key Indicators : Documentation standards (IEEE, DoD, local). Documentation tools used.
- Key Challenges : Institute quality control over content of documentation. Keep documentation up to date. Assess quality of documentation products. Assess usefulness of documentation products (Are they used? If not used, why not?). More effective use of documentation tools.
- Key Significance :
  * Development : Standards for which documents must be generated. Since no quality assessment, documentation may be incomplete and may not be up to date. Much informal communication.
  * Maintenance : Both the source code and associated documentation are available. However, the utility of the documents is suspect, given the lack of quality control.

- **Level 3 : Defined**
  - Keywords: Documentation process, product assessment, documentation team.
  - Key Process Areas : Documentation products quality assessment and usefulness assurance. Documentation process definition. System documentation teams established, to monitor for quality and usefulness.
  - Key Practices : Independent units established to monitor quality and usefulness of documentation (like SQA). Consistent use of documentation

tools, through a defined documentation process. Documentation updated after each change.

- Key Indicators : Documentation team goals and procedures. Consistent use of documentation tools. Documentation quality and usefulness assessment methods.
- Key Challenges : Establish measures of documentation process quality. Case tool integration. Establish mechanism for user feedback concerning usefulness of documentation.
- Key Significance :
  * Development : Existence of higher quality documentation will improve the communication process, and attack the causes of errors. This will affect maintenance, by improving visibility for maintenance concerns.
  * Maintenance : Extensive use of documentation, allowing maintenance programmers to work at a higher level of abstraction, reducing complexity, time and costs.

- **Level 4 : Controlled**
  - Keywords: Process assessment, measurement, control, feedback, improvement, optimization
  - Key Process Areas : Documentation process quality assessment and measurement. Data analysis used as feedback into process improvement loop.

- Key Practices : Establishment of minimum measures for documentation process quality. Data collection and analysis. Mechanism to feed measurements into process to identify areas of improvement. Extensive use of documentation tools. Documentation tools integrated with Software CASE tools. Mechanism for incorporation of feedback, from users of the documentation.
- Key Indicators : Data analysis mechanism used to assess the process. Improvement-feedback mechanisms.
- Key Challenges : Automate collection and analysis of process data. Maintain a continual optimization of documentation process. Continual striving for improvement incorporated into process.
- Key Significance :
  * Development : Existence of a measurable documentation process provides specific indication as to how to proceed making the process more formal and encouraging its reusability.
  * Maintenance : Overall maintenance costs and time decreased with a well-defined and measurable documentation process. A key aspect in maintenance is having quick access to and understanding of the supporting documentation items relevant to the task at hand.

Table 4.2 shows how the challenges identified at one level are specifically addressed at the next level. The table also helps in understanding how the model progresses from one level to the next.

Level	Challenges	Practices at next level
Ad-hoc	Realize importance of documentation Establish required documentation Define documentation to be created	Documentation exists, but varies in content and completeness Documentation standards Check-off list of required documents
Inconsistent	Institute control over content Keep documentation up to date Assess quality of documentation products Assess usefulness of documentation products More effective use of documentation tools	Independent unit established to monitor quality Documentation updated after each change Documentation quality assessment methods Documentation usefulness assessment methods Consistent use of documentation tools
Defined	Establish measures of documentation process quality Further tool integration	Establishment of minimum measures for documentation process quality Data collection and analysis Extensive use of documentation tools and integration with CASE tools
Controlled	Continual optimization of documentation process	Mechanism to feed improvements into process to identify areas of improvement

Table 4.2: Level-to-level transitions table

The model is to be used along with the assessment procedure to map an organization's documentation status to one of the maturity levels, generating a documentation process maturity profile. Then, the organization can work on identified areas of improvement to move on to the next level.

4.3 Assessment procedure

The purpose of the assessment procedure for a model is determine where an organization stands relative to that model. For this model, the assessment maps an organization's experience and past performance to a documentation maturity level and generates a documentation process profile. The profile indicates key practices for that level, what practices the organization is doing well, what practices need improvement, and challenges to move to the next level.

The assessment questionnaire is presented in this section. A sample of the actual questionnaire used in the assessments is included in Appendix A.

4.3.1 Design of questionnaire

The questions have been derived directly from the model and the practices determined for each maturity level. Each practice defines one or more questions whose answer determines the degree of satisfaction of the practice. A total of 18 practices for the 4 maturity levels have been defined, and 56 questions are used to determine their degree of satisfaction. Table 4.3 maps each maturity level's practices into a number of questions.

4.3.2 Basic definitions

Definitions of basic terms that are extensively used:

- Software Documentation: Includes all documents generated as part of software development, i.e. software requirements specifications, design documents, code, test plans and history (test cases). Software documents refer to either hard copy or electronic form. It excludes end-user documentation, i.e. user manuals and operations manuals; and it excludes managerial documentation, i.e. project plans, schedule and staff plans, etc.

- Software CASE tools: Software designed to assist software engineers and programmers cope with the complexity of the process and the artifacts of Software Engineering [Lewis91].

Level	Key Practices	Number of Questions
1	Creation of basic software development documents	5
	Documentation generally recognized as important	1
2	Written statement about importance of documentation	1
	Adequate time and resources for documentation	3
	Adherence to documentation standards	4
	Use of a check-off list of required documentation	1
	Use of simple documentation tools	1
3	Use of software documentation generated	5
	Mechanisms to update documentation	7
	Mechanisms to monitor quality of documentation	5
	Methods to assess usefulness of documentation	1
	Use of common sets of documentation tools	1
	Use of advanced documentation tools	1
	Documentation-related technology and training	3
4	Measures of documentation process quality	7
	Analysis of documentation usage and usefulness	5
	Process improvement feedback loop	4
	Integrate CASE and documentation tools	1

Table 4.3: Documentation process practices and assessment questions

- Documentation tools: Software designed to aid software engineers to cope with the complexity of the process and artifacts of documentation.

  Types of documentation tools [STSC92]:

  - Simple:

    1. Text processors: word processors, desktop publishers, editors, spelling checkers, electronic mail.
    2. Graphics: flowcharting, technical drawing.

  - Advanced:

    1. Document management systems: storage, retrieval, browsing, distribution, sharing, consistency checking.
    2. Integrated documentation/CASE tools.

- Quality of documentation: Quality includes the following characteristics: adequacy, completeness, usability, consistency, currency, readability, ease of use, ease of modification, tracebility [Humph89, Lee87, Oman91, Post85, Stev88].

- Usefulness of documentation: Extent to which the documentation products are used by software developers and maintainers, the documentation users.

4.3.3 Assessment questionnaire

The assessment questionnaire consists of 56 questions, decomposed as follows:

- By maturity levels:
  - Level 1 : 6 questions
  - Level 2 : 10 questions
  - Level 3 : 23 questions
  - Level 4 : 17 questions
- By subject area:
  - Lifecycle documentation : 10 questions
  - Documentation standards : 4 questions
  - Documentation tools : 4 questions
  - Documentation quality control : 16 questions
  - Documentation usefulness determination and assurance : 4 questions
  - Documentation error analysis and improvement feedback : 13 questions
  - Documentation-related training : 3 questions
  - Management attitude towards documentation : 2 questions

A list of questions sorted by maturity level is presented below. A list of questions sorted by subject area is presented in Appendix B.

1	never
2	seldom
3	sometimes
4	usually
5	always

Table 4.4: Meaning of answers in range of 1-5

Types of answers

35 of the questions in the assessment questionnaire are to be answered in the range of 1-5, and 21 are yes/no questions. The numbers in the range of 1-5 represent *how often* a given action is performed. Table 4.4 matches a number with its meaning.

Questions sorted by maturity level

The questions that are to be answered in the yes/no mode are highlighted.

- Level 1: Ad-hoc

  1. Are software documents other than code created during development?
  2. Are software requirements specifications (including prototyping documents) generated?
  3. Are design documents (including prototyping documents) generated?
  4. Are test plan Documents generated?
  5. Are test cases used in testing recorded in a document?
  6. **Does management have a policy (not necessarily written) supporting the importance of software documentation?**

- Level 2: Inconsistent

1. Are there check-off lists that indicate which software documents must be created?
2. Are there standards indicating what must be included in each software document?
3. Is there any procedure or form used to specify how and when to write each software document?
4. Is there a formal procedure used for checking that document contents satisfy standards?
5. Is adequate time allocated to develop software documentation during software development?
6. Are software documents checked to see that they have been done?
7. For each software project, is there a person responsible for collecting the documentation?
8. For each software project, is there a person responsible for maintaining the documentation?
9. Are simple documentation tools (text processors, graphics) used to create and maintain software documentation?
10. **Does management view software documentation as of major importance and have written policies to this effect?**

- Level 3: Defined

1. Are all software documents generated in the development phase used? (SRS in design, design document in coding, SRS and design in maintenance, etc)
2. **Are software documents other than code used during development?**
3. **Are software documents other than code used during maintenance?**
4. **When software documents are not used, is it because they are unreliable, incomplete, or out of date?**
5. **When software documents are not used, is it because they are not easily accessible?**
6. **Is there a mechanism for checking that a software document has been completed satisfactorily?**
7. If so, how frequently is it used?
8. **After a change has been made to the code, is there a mechanism for checking that all related documentation is updated?**
9. If so, how frequently is it used?
10. **After a change has been made to the design, is there a mechanism for checking that all related documentation is updated?**
11. If so, how frequently is it used?

12. **After a change has been made to the requirements, is there a mechanism for checking that all related documentation is updated?**

13. If so, how frequently is it used?

14. Is affected documentation updated after each maintenance change?

15. **Is there a mechanism to monitor the quality of the documentation?**

16. If so, how frequently is it used?

17. **Is there an independent group whose function is to assess the quality of the documentation generated in a project?**

18. **Is there a mechanism to assess the usefulness of the documentation generated in a project?**

19. Is formal training available for the use of documentation standards?

20. Is formal training available for the use of documentation tools?

21. Are advanced documentation tools (integrated, documentation management) used during development and maintenance?

22. Are common sets of documentation tools used in the different development environments throughout the organization?

23. **Is there a mechanism to foster the incorporation of advances in documentation technology across the organization?**

- Level 4: Controlled

1. Are documentation errors and trouble reports tracked to the solution?
2. Are documentation process data and error data for software projects recorded in a database?
3. Are statistics gathered on documentation errors?
4. **Is documentation error data analyzed to determine distribution and characteristics of errors?**
5. If so, how frequently is this done?
6. **Is there a mechanism to analyze documentation error root causes?**
7. If so, how frequently is it used?
8. **Is there a documentation usage profile generated for software development?**
9. If so, how frequently is this done?
10. **Is there a documentation usage profile generated for software maintenance?**
11. If so, how frequently is this done?
12. **Is there a mechanism for users of the documentation (software developers and maintainers) to provide feedback to improve documentation usefulness?**
13. If so, how frequently is it used?

14. Are measures of usefulness of documentation collected?

15. **Is there a mechanism to feedback improvements to documentation practices or standards?**

16. If so, how frequently is it used?

17. **Are advanced documentation tools integrated with software CASE tools?**

4.3.4 Administering the questionnaire

The assessment is to be conducted as follows:

- Determine the company and project(s) that will undergo a documentation process maturity assessment.
- Gather the project team(s) under assessment, explain to them the goals and procedures of the assessment and clarify any unresolved issue.
- Have the project team(s) complete the questionnaire. This phase should not exceed 30 minutes.

4.4 Determination of documentation process maturity

This section describes the algorithm to determine maturity levels, profiles and challenges. The determination of maturity levels is based upon the team answers for all questions. To satisfy a given maturity level, the team needs to answer positively certain questions. Positive answers of just a partial set of questions for

each level can determine a maturity level to be either weak, solid or strong. This distinction of solidity for maturity levels is useful to determine practices that must be improved in the current level before incorporating improvements to move to higher maturity levels. The answers to each question are then used to determine the degree of satisfaction of each practice (each practice is addressed by one or more questions) which is in turn used to determine a documentation process maturity profile and an action plan of improvements. The next subsections describe the method to determine team's answers, to determine maturity levels, to determine degree of satisfaction for the practices and to generate a documentation assessment report (documentation maturity profile, and improvement plan). The questions have been designed to facilitate the determination of maturity levels and documentation practices and their degree of satisfaction.

4.4.1 Determining the team's answer

The maturity level is based on the answers that the project team provides to each question. Some of the questions are answered in the range of 1-5, and some are yes/no questions. The questions must be analyzed individually and the team answer to a question is determined as follows:

- For answers in range of 1-5, an average of the scores remaining after removing the highest score and the lowest score is computed.
- For yes/no answers, the mode (i.e. the most frequent answer) is taken as the team answer.

4.4.2 Determining the maturity levels

The maturity levels are determined according to the following rules. The following question numbers are those used in the previous section where the questions were sorted by maturity level.

- Level 1: Since this is the first level in our model, no effort is needed to attain it. So, the main interest at this level is to determine whether the organization shows a strong performance.

  - Strong: The practices that are relevant at this level are: *Creation of software development documents* and *Documentation generally recognized as important.*

    Answers needed: Questions 1 through 5 receive a score of 4 or 5. Question 6 (yes/no) is answered positively.

- Level 2: At this level, the organization could be in either one of three situations: weak, solid, or strong performance.

  - Weak: The practices that are relevant at this level are: *Use of a check-off list of required documentation* and *Use of simple documentation tools.*

    Answers needed: Questions 1 and 9 receive a score of 4 or 5.

  - Solid: The practices that are relevant at this level are: *Use of a check-off list of required documentation, Use of simple documentation tools* and *Adherence to documentation standards.*

Answers needed: Besides requirements for weak performance, questions 2,3,4 and 6 receive a score of 4 or 5.

- Strong: The practices that are relevant at this level are: *Use of a check-off list of required documentation*, *Use of simple documentation tools*, *Adherence to documentation standards*, *Written statement about importance of documentation* and *Adequate time and resources for documentation.*

  Answers needed: Besides requirements for a solid performance, questions 5,7,8 receive a score of 4 or 5. Question 10 (yes/no) is answered positively.

- Level 3: This level is defined in terms of a solid or a strong performance.

  - Solid: The practices that are relevant at this level are: *Use of software documentation generated*, *Mechanisms to update documentation*, *Mechanisms to monitor quality of documentation* and *Methods to assess usefulness of documentation.*

    Answers needed: Questions 1,2,3 and 6 through 18 receive a score of 4 or 5, or are answered positively if they are (yes/no) questions.

  - Strong: The practices that are relevant at this level are: *Use of software documentation generated*, *Mechanisms to update documentation*, *Mechanisms to monitor quality of documentation*, *Methods to assess usefulness of documentation*, *Use of common sets of documentation tools*, *Use of advanced documentation tools* and *Documentation-related technology and training.*

Answers needed: Besides requirements for a solid performance, questions 19 through 23 receive a score of 4 or 5, or are answered positively if they are (yes/no) questions.

- Level 4: This level is defined only in terms of a solid performance, given its optimizing nature.
  - Solid: The practices that are relevant at this level are: *Measures of documentation process quality*, *Analysis of documentation usage and usefulness*, *Process improvement feedback loop* and *integrate CASE and documentation tools.*

    Answers needed: All questions 1 through 17 receive a score of 4 or 5, or are answered positively if they are (yes/no) questions.

4.4.3 Beyond maturity levels: determining profiles and challenges

One of the goals of the assessment procedure is to determine where the organization stands and what are the challenges to move up to the next maturity level. Determining a numeric documentation process maturity level will be of little use if it is not complemented with a description of what's being done well and what isn't (a documentation process maturity profile), a list of satisfactory practices, practices needing improvements, missing practices and challenges to move to the next level. This section describes how to map answers into maturity profiles and challenges for improvement. The improvements associated with the partially satisfied practices and the actions associated with the missing practices define the challenges.

Key practices and their satisfaction

This section describes the key practices for each level and which answers contribute to the satisfaction of them. The degree of satisfaction may be either one of: *none*, *partial* and *full*. Since each practice's degree of satisfaction is defined directly by the answers to a set of questions each degree of satisfaction is determined in terms of the proportion of relevant answers that are either positive or negative. A fully satisfied practice requires that all relevant questions be answered positively; a not satisfied practice occurs when more than half of the relevant answers are negative (threshold defined arbitrarily), and a partially satisfied practice is determined when it is neither fully nor not satisfied. The questions numbers are those used in section of questions sorted by maturity level.

- Level 1
  - Creation of basic software development documents

    Not satisfied: If more than 3 of the 5 answers (1-5) are negative (below 3)

    Fully satisfied: If all 5 answers (1-5) are positive (above 3)

    Partially satisfied: Any other case for answers (1-5)

  - Documentation generally recognized as important

    Not satisfied: Answer 6 is negative (*no*)

    Fully satisfied: Answer 6 is positive (*yes*)

    Partially satisfied: Any other case for answer 6

- Level 2
  - Written statement about importance of documentation

    Not satisfied: Answer 10 is negative (*no*)

    Fully satisfied: Answer 10 is positive (*yes*)

    Partially satisfied: Any other case for answer 10

  - Adequate time and resources for documentation

    Not satisfied: If more than 1 of the 3 answers (5,7,8) are negative (below 3)

Fully satisfied: If all 3 answers (5,7,8) are positive (above 3)

Partially satisfied: Any other case for answers (5,7,8)

- Adherence to documentation standards

  Not satisfied: If more than 2 of the 4 answers (2,3,4,6) are negative (below 3)

  Fully satisfied: If all 4 answers (2,3,4,6) are positive (above 3)

  Partially satisfied: Any other case for answers (2,3,4,6)

- Use of a check-off list of required documentation

  Not satisfied: If answer 1 is negative (below 3)

  Fully satisfied: If answer 1 is positive (above 3)

  Partially satisfied: Any other case for answer 1

- Use of simple documentation tools

  Not satisfied: If answers 9 is negative (below 3)

  Fully satisfied: If answer 9 is positive (above 3)

  Partially satisfied: Any other case for answer 9

- Level 3

  - Use of software documentation generated

    Not satisfied: If more than 1 of the 3 answers (1-3) are negative (below 3 or a *no*)

    Fully satisfied: If 3 answers (1-3) are positive (above 3 or a *yes*)

    Partially satisfied: Any other case for answers (1-3)

  - Mechanisms to update documentation

    Not satisfied: If more than 2 of the 4 answers (8,10,12,14) are negative (*no*)

    Fully satisfied: If all 7 answers (8-14) are positive (above 3 or a *yes*)

    Partially satisfied: Any other case for answers (8-14)

  - Methods to monitor quality of documentation

    Not satisfied: If more than 1 of the 3 answers (6,15,17) are negative (*no*)

    Fully satisfied: If 5 answers (6,7,15,16,17) are positive (above 3 or a *yes*)

    Partially satisfied: Any other case for answers (6,7,15,16,17)

  - Methods to assess usefulness of documentation

    Not satisfied: If answer 18 is negative (*no*)

    Fully satisfied: If answer 18 is positive (*yes*)

    Partially satisfied: Any other case for answer 18

- Use of common sets of documentation tools

  Not satisfied: If answer 22 is negative (below 3)

  Fully satisfied: If answer 22 is positive (above 3)

  Partially satisfied: Any other case for answer 22

- Use of advanced documentation tools

  Not satisfied: If answer 21 is negative (below 3)

  Fully satisfied: If answer 21 is positive (above 3)

  Partially satisfied: Any other case for answer 21

- Documentation-related technology and training

  Not satisfied: If more than 1 of the 3 answers (19,20,23) are negative (below 3 or a *no*)

  Fully satisfied: If 3 answers (19,20,23) are positive (above 3 or a *yes*)

  Partially satisfied: Any other case for answers (19,20,23)

- Level 4

  - Measures of documentation process quality

    Not satisfied: If more than 3 of the 5 answers (1-4,6) are negative (below 3 or a *no*)

    Fully satisfied: If all 7 answers (1-7) are positive (above 3 or a *yes*)

    Partially satisfied: Any other case for answers (1-7)

  - Analysis of documentation usage and usefulness

    Not satisfied: If 3 answers (8,10,14) are negative (below 3 or a *no*)

    Fully satisfied: If 5 answers (8-11,14) are positive (above 3 or a *yes*)

    Partially satisfied: Any other case for answers (8-11,14)

  - Process improvement feedback loop

    Not satisfied: If 2 answers (12,15) are negative (*no*)

    Fully satisfied: If 4 answers (12,13,15,16) are positive (above 3 or a *yes*)

    Partially satisfied: Any other case for answers (12,13,15,16)

  - Integrate CASE and documentation tools

    Not satisfied: If answer 17 is negative (*no*)

    Fully satisfied: If answer 17 is positive (*yes*)

    Partially satisfied: Any other case for answer 17

4.4.4 Documentation assessment report

Based on the responses to the questionnaire a Documentation Assessment Report is generated. It contains an executive summary with the maturity level number, a documentation process maturity profile and an action plan. The profile indicates satisfactory practices, practices needing improvement, missing practices, and challenges to move to the next level. The action plan describes specific actions to improve existing practices and to address missing practices to enable the organization to move to the next higher maturity level. The following is an example of an actual report generated after conducting an assessment for a software project at a software company identified as project Y and company X. A complete description of the actions defined for each practice that is either partially satisfied or not satisfied is given in Appendix C.

Example Documentation Assessment Report

Contents of the Report

This report presents an executive summary of the main findings of the assessment conducted or project *Y* at company *X* on *month day, year.* It includes the process maturity level, a documentation process maturity profile table, and an action plan of needed improvements and challenges.

Executive Summary

Maturity Level

The maturity level for this project is: **Strong Level 1**

Documentation Process Profile

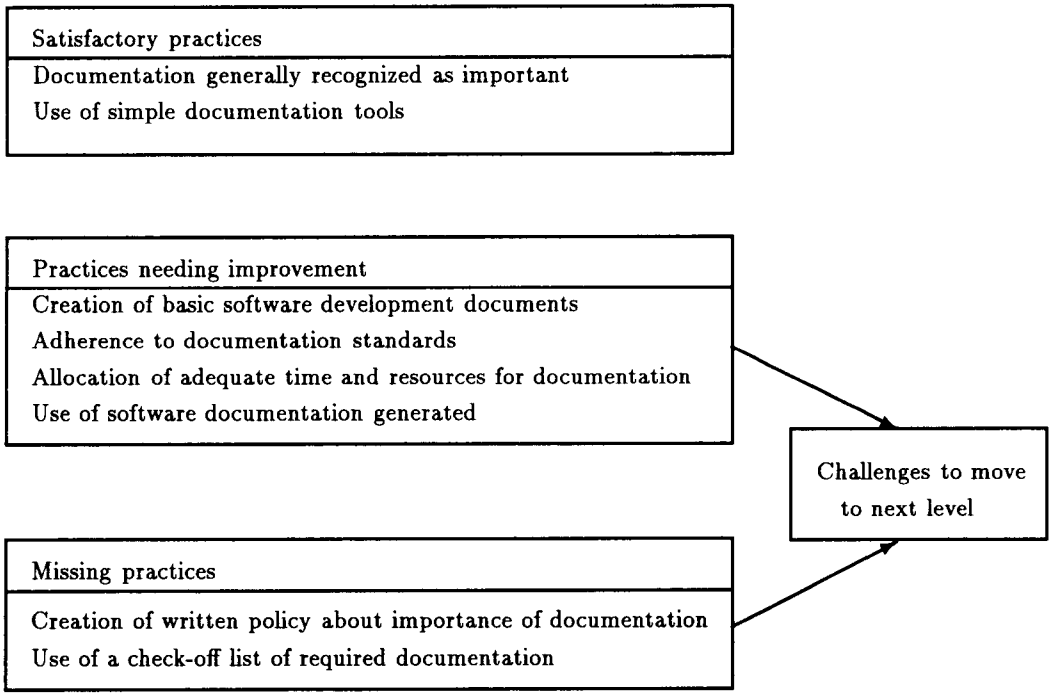

Figure 4.1: Documentation Process Profile

Action Plan

The assessment has defined the project to be at a *Strong Level 1* maturity level, which means that documentation is not given a high priority.
The followings lists describe the actions needed to solidify the current practices and the actions needed to move to the next maturity level.

Satisfactory current practices

- Documentation is generally recognized as important.
- Use of a check-off list of required documentation.
- Use of simple documentation tools

Current documentation practices needing improvement

- All basic software documents must be created for all phases of software development. This includes software requirements specification, design documents, test plans, and test implementations.
- Documentation standards must be followed in the creation of all software documents.
- Software documentation generated must be used in all subsequent phases of the software process.
- Interview software developers and maintainers to assess usefulness of all documents.
- Use of a common set of tools by the software developers when working in a particular software development environment.

Documentation practices that need to be addressed

- Create written policy on importance of documentation and make that policy an important part of each software development project.
- Allocate sufficient time and resources to create software development documentation that meets the standards for each phase of the development.

Level	Key Practices	Degree of Satisfaction
1	Creation of basic software development documents	Partial
	Documentation generally recognized as important	Full
2	Written statement about importance of documentation	None
	Adequate time and resources for documentation	None
	Adherence to documentation standards	Partial
	Use of a check-off list of required documentation	Full
	Use of simple documentation tools	Full
3	Use of software documentation generated	Partial
	Mechanisms to update documentation	None
	Mechanisms to monitor quality of documentation	None
	Methods to assess usefulness of documentation	Partial
	Use of common sets of documentation tools	Partial
	Use of advanced documentation tools	None
	Documentation-related technology and training	None
4	Measures of documentation process quality	None
	Analysis of documentation usage and usefulness	None
	Process improvement feedback loop	None
	Integrate CASE and documentation tools	None

Table 4.5: Maturity profile of documentation process practices

4.5 Summary

The problem of low quality of system documentation has been long neglected despite its importance. This research proposes a system documentation process maturity model as a means to manage the complexity of the documentation process. The process maturity approach is a novel one, but has gained widespread attention due to the SEI's maturity models and their impact in the software community. Chapters 5 and 6 show that the documentation process maturity model proposed is indeed a valuable tool to improve the quality of software.

Chapter 5

Validation of Model

5.1 Overview

The goal of the validation of a software model is to show that it is accurate, i.e. it does represent the reality claimed and that the estimations it provides fit the actual phenomenon. There are different ways to do validation: expert opinions, statistical analysis of data gathered (from historical records or through controlled experiments), or related software models and their estimations.

The ultimate goal of the validation of the proposed documentation process maturity model is to show that organizations at a higher documentation maturity level produce higher quality software (i.e. organizations at a high maturity level catch a larger proportion of errors earlier in the development phase than organizations at a lower maturity level). Initially, it was decided to concentrate on software errors because errors were thought to be a commonly accepted measure of quality and that *organizations were most likely to collect error data.* This last assumption proved wrong, given the fact that only a small proportion of the organization assessed actually kept records of their error detection and correction activities. De-

Marco [DeMarc82] points out that software defects are simply out of control in the software industry because many software companies *avoid* collecting defect data. Given this unpleasant scenario and the difficulties in gathering sufficient empirical data to perform significant statistical analysis, it was decided to use software development models to show the impact of documentation maturity on software quality and software testing and maintenance effort. The SEI has *validated* their software process maturity models using software development model estimated results and a few actual data points that do not cover the whole maturity level spectrum, but that are consistent with the model's claims. A similar approach is used in this research.

This chapter discusses causes of errors in software development, analyzes software development models and their relation to software documentation, presents the results from applying CoCoMo model as a validation tool, and estimates costs and benefits associated with each maturity level in the documentation process model, i.e. the bottom-line justification behind documentation process improvement.

5.2 Errors in software development

Many empirical studies of errors discovered during integration testing and maintenance have shown the following:

- The majority of errors were introduced before any code was written.
- Most errors are removed long after they were introduced, resulting in very expensive testing and maintenance processes.

Studies have consistently shown that the majority of errors were introduced before coding, but discovered and removed after coding. Boehm [Boehm81] estimates that 42% of the errors discovered during testing were introduced during the program design phase. Alberts [Alb76] found that 46-64% of all errors were design errors and Grady and Caswell [Grady87] found that 50% of errors are design errors. In Basili and Perricone [Basi84] 48% of errors were attributed to misinterpreted functional requirements or specifications. Endres [Endres75] analyzed software errors and their causes, and concluded that 46% of them relate to poor communications and problem understanding. Ramamoorthy [Ramam88] states that the majority of requirements and design errors are caused by ambiguity, incompleteness, or faulty assumptions in the specifications. Grady and Caswell [Grady87] claim that documentation is responsible for 50% of design errors. Seddio [Seddio92] describes an application of review and product metrics to the software process at Eastman Kodak. He found that 63% of errors in software specification documentation are caused by incomplete specifications and that 24% violate documentation standards. Boehm [Boehm81] believes that typically there are twice as many design errors as coding errors. All of these studies clearly indicate that most errors are design and requirements errors and in most instances these errors are discovered and corrected during testing and maintenance.

It is much less expensive to repair an error early in the software life cycle than later in the life cycle. Boehm [Boehm81] estimates that it may cost 50 times more to discover and repair a design error during testing than during design and it may cost 100 times more if the error is repaired during maintenance. Alberts [Alb76]

estimated that up to 70% of design errors are not found during development and the cost to remove these errors is almost half (47%) of the development costs. His analysis of the costs associated with errors revealed that 80% of the cost can be related to design errors.

5.2.1 Role of reviews and software documentation

What can be done to reduce the number of errors in the early stages of software development, most of which are system documentation errors - inaccurate, incomplete, or missing information? The most effective techniques would seem to be ones aimed at improving the quality of system documentation and the documentation process. This has been confirmed by several studies. In one experiment Fagan [Fagan76] found that 82% of the total number of errors discovered during development were found during formal design and code inspections. A study at GTE by Howden [How78] compared the efficiency of design review and testing. Design reviews uncovered 45% of the errors taking 17% of the development time, whereas testing took up to 75% of the time to uncover just 10% of the errors. Software reviews are a filter for the software engineering process [Press92]. A formal technical review (inspection or *walkthrough*) is an effective means for improving software quality. Properly conducted reviews are the single most effective way to uncover and correct errors while they are still inexpensive to find and fix[Press93]. The obvious benefit of formal technical reviews is the early discovery of software defects so that each defect may be corrected prior to the next step in the software engineering process [Press92]. A number of industry studies (TRW, Nippon Electric,

Mitre Corp., among others) indicate that design activities introduce between 50 and 65 percent of all errors (defects) during the development phase of the software engineering process. However, formal reviews techniques have been shown to be up to 75 percent effective in uncovering design flaws. By detecting and removing a large percentage of these errors, the review process substantially reduces the cost of subsequent steps in the development and maintenance phases.

A major threat to software quality comes from a seemingly benign source: changes. Every change to software has the potential for introducing error or creating side effects that propagate errors. The change control process contributes directly to software quality by formalizing requests for change, evaluating the nature of change, and controlling the impact of change. Change control is applied during software development and later, during the software maintenance phase. Software documentation plays a key role in these activities, because it contains the information about the changes and must be easily accessible and usable, and it needs to be updated properly after each change to ensure it will continue to be useful during the remainder of the software product lifetime.

Documentation and technical reviews go hand in hand. It has been shown that technical reviews of design and code are more cost-effective than testing the code. These studies suggest that high quality documentation is required to review effectively, to avoid a costly effect later on especially during maintenance. Software documentation whose quality is periodically and formally reviewed requires a software organization to be at a documentation maturity level of 3 or higher, because at that level is when quality control is instituted in the process.

5.3 Validation

The documentation process maturity model proposed in this thesis has been subjected to a number of reviews by many developers and managers in the software industry. It has been generally regarded as a valuable and useful tool to assess the state of software documentation practices and to move towards their improvement. Many assessments have been conducted and the feedback received on their results and recommendations has been very positive.

This section is devoted to provide supporting evidence and demonstrate the validity of this model as a process improvement tool.

As part of the validation effort, 14 software projects on 5 different companies were assessed. Chapter 6 presents the results obtained from those assessments, as well as an analysis of the trends about early error detection capabilities drawn from the (insufficient) defect data actually collected. While many companies were willing to participate as subject of an assessment, most of them did not have defect data available.

The collection of software development defect data (and other kinds of data) require a fair amount of effort that competes with other pressing necessities. Unfortunately, many software managers and practitioners are not convinced that collecting data is beneficial, so they ignore the subject altogether. This situation generates a *data vacuum* that prevents finding a solid answer to questions like *what's the return on investment for process improvement?*. This data vacuum poses one of the biggest problems the software engineering community must address [Hersh93].

The lack of sufficient data to validate the documentation maturity model prompted a change in the approach, and it was decided to consider software documentation models that involve documentation as one of their factors to see what kind of an impact software documentation maturity had on software quality and software testing and maintenance effort according to them. The SEI used a similar approach to gauge cost and benefits of its models.

5.3.1 Software development models and the role of documentation

There are many software development models that estimate and quantify the software process: time of development, effort invested, productivity, etc. Unfortunately most of them do not consider the quality of the documentation produced as a distinct factor in their models, or group documentation together with many factors as one without providing guidelines to isolate the effect of a single factor on software productivity or software quality. Putnam's model and its *technology factor* is a good example: the Putnam Estimation Model [Press92, Conte86, Boehm81] is a dynamic multi-variable model that assumes a specific distribution of effort over the life of a software development project, called the *Rayleigh-Norden curve.* This curve is used to derive the equation that relates the number of delivered source code to effort and development time:

$$L = C_k K^{1/3} t_d^{4/3} \qquad \text{or} \qquad K = \frac{L^3}{C_k^3 t_d^4}$$

where K is the effort expended (in person-years) over the entire life cycle for software development and maintenance, L is the number of delivered lines of code, t_d is the development time in years, and C_k is a *state of technology* constant and reflects "throughput constraints that impede the progress of the programmer". Typical values might be: $C_k = 2,000$ for a poor software development environment (e.g. no methodology, *poor documentation and reviews*, a batch execution mode); $C_k = 8,000$ for a good software development environment (e.g. methodology in place, *adequate documentation/reviews*, interactive execution mode); $C_k = 11,000$ for an *excellent* environment (e.g. automated tools and techniques). The constant C_k can be derived for local conditions using historical data collected from past development efforts. Because of the high-order of the factors in the equation for K, relatively small variations on some of the factors can impact greatly the predicted effort to the project.

Even though this model includes documentation in the *state of technology* constant, it does not provide enough information to determine the weight of documentation as a factor of C_k. Neither does it allow the determination of documentation improvement trade-offs. So, it is not possible to determine the impact of documentation quality improvement using Putnam's model.

The most noted software development model that does consider software documentation as a *cost driver* is Barry Boehm's CoCoMo [Boehm81], albeit the consideration is implicit rather than explicit.

CoCoMo (Cost Construction Model) was developed by Boehm based on a set of observed factors which influence development time and effort. These factors are referred to as *drivers* in the model.

CoCoMo model is broken down in 3 levels or sub-models:

- Basic CoCoMo
- Intermediate CoCoMo
- Detailed CoCoMo

CoCoMo estimates development time and effort, including management time and effort during development, documentation time and effort. It excludes user training, installation of the finished product, conversion to new product. CoCoMo estimates cover the period starting after approval of software requirements and ending when the product is released.

Maintenance issues are not included in these CoCoMo estimates, but Boehm provides a similar separate model whose structure is as follows:

$$(MM)_{AM} = (1.0)(ACT)(MM)_{DEV}$$

where ACT stands for *annual traffic change*, $(MM)_{AM}$ stands for *annual maintenance effort* and $(MM)_{DEV}$ stands for *development effort.*

The basic CoCoMo model for software development computes development effort and time, so it consists of 2 equations of the following form:

$$K_d = MM = a(KDSI)^b$$

$$t_d = TDEV = c(MM)^d$$

K_d corresponds to the *development effort* in man-month (MM).

t_d corresponds to the *development time* in months ($TDEV$).

$KDSI$ are the *thousands lines of delivered instructions.*

a, b, c, d are parameters that depend on the kind of product being developed and the development mode (or level of difficulty).

Development modes:

- Organic—relatively small software development team on a well-understood application, in a highly familiar, in-house environment, usually small programs.
- Semidetached—more complex problems, more rigorous demands on communication, time and size.
- Embedded—demanding time constraints, interactions between hardware and software, high degree of interaction (real-time transaction system or concurrent processing).

Development effort and time needed are derived from estimated size in lines of code.

The intermediate CoCoMo model adds 15 cost drivers to compute the effort and time:

- Product Factors
  - required software reliability (RELY)
  - size of the database (DATA)

  - product complexity (CPLX)

- Computer Factors
  - execution time constraints (TIME)
  - main storage constraints (STOR)
  - machine stability (VIRT)
  - computer turnaround time (TURN)

- Personnel Factors
  - programmer analyst capability (ACAP)
  - programmer familiarity with the application (AEXP)
  - programmer capability (PCAP)
  - machine familiarity and experience (VEXP)
  - programming language experience (LEXP)

- Project Factors
  - modern programming practices (MODP)
  - use of software tools (TOOL)
  - development schedule (SCED)

Each of these 15 factors are estimated, converted into a number and used as a multiplier to adjust the effort and time provided by the basic CoCoMo formulae. Each factor is assigned 1 of 6 values (Very low, Low, Nominal, High, Very High

and Extra High) that in turn corresponds to a numeric value. Boehm also provides sample phase distributions that can be used to estimate development effort and time for the different activities on each phase.

The intermediate CoCoMo model for maintenance has the following structure:

$$(MM)_{AM} = (1.0)(ACT)(MM)_{NOM}(EAF)$$

where $(MM)_{NOM}$ represents the *nominal development effort* (i.e. without considering the 15 cost drivers), and EAF represents the *effort adjustment factor*, computed as the product of the 15 factors but using the maintenance multipliers. These may be different from the multipliers used for development because the ratings in maintenance may be different, and the factors RELY and MODP have different effort multipliers for maintenance.

Documentation does not have a role as a distinct cost driver in CoCoMo model, but it is included in the cost driver *modern programming practices (MODP).* The specific practices defined as modern are:

- Top-down requirements analysis and design. Developing the software requirements and design as a sequence of hierarchical elaborations of the users' information processing needs and objectives. This practice is extended to include the appropriate use of incremental development, prototyping, and anticipatory documentation.

- Structured design notation. Use of modular, hierarchical design notation (PDL, structure charts, HIPO) consistent with structured code constructs.

- Top-down incremental development. Performing detailed design, code, and integration as a sequence of hierarchical elaborations of the software structure.
- Design and code walkthroughs or inspections. Performing preplanned peer reviews of the detailed design and the code of each software unit.
- Structured code. Use of modular, hierarchical control structures based on a small number of elementary control structures, each having only one flow of control in and out.
- Program librarian. A project participant who operates an organized repository and control system for software components.

The ratings are defined in terms of the range of modern programming practices used in software development:

- Very Low: no use of modern programming practices.
- Low: beginning, experimental use of some modern programming practices.
- Nominal: reasonably experienced in use of some modern programming practices.
- High: reasonably experienced in use of most modern programming practices.
- Very High: routine use of all modern programming practices.

Table 5.1 shows MODP effort multipliers broken down by development phase. These effort multipliers reflect the impact on effort (and cost) on the different phases

MODP Rating	Requirements and Product Design	Detailed Design	Code and Unit Test	Integration and Test	Overall
Very low	1.05	1.10	1.25	1.50	1.24
Low	1.00	1.05	1.10	1.20	1.10
Nominal	1.00	1.00	1.00	1.00	1.00
High	1.00	0.95	0.90	0.83	0.91
Very high	1.00	0.90	0.80	0.65	0.82

Table 5.1: MODP effort multipliers by development phase

MODP Rating	Maintenance
Very low	1.40
Low	1.18
Nominal	1.00
High	0.85
Very high	0.72

Table 5.2: MODP effort multipliers for maintenance, product size 100k

when the particular *cost driver* shows a poor or good rating. A multiplier of 1.50 means a 50% increase on effort. A multiplier of 0.90 means a 10% decrease.

Table 5.2 shows the MODP effort multipliers for the maintenance phase, assuming a product size of 100,000 deliverable source instructions (other multipliers are provided for other product sizes too).

Table 5.1 shows that the biggest impact of MODP occurs on the integration testing phase, with respect to the *nominal* rating, where a *very low* rating translates

into a 50% increase in testing effort and a *very high* rating represents a 35% decrease in testing effort.

How is this related to the documentation maturity model? An analysis of the practices that compose the cost driver MODP shows that documentation impacts and influences many of them, most notably the quality and effectiveness of design walkthroughs. Tables 5.3 and 5.4 show the percent of reduction in testing and maintenance effort due to improvements in documentation maturity. The entries in the tables represent cumulative percentages of effort reduction assuming the impact of documentation in MODP ranging from 5% to 50%.

The values have been computed using the effort multipliers identified in Tables 5.1 and 5.2 and using Level 1 as the baseline. The effort multipliers are adjusted properly with the documentation impact assumed and the percentage effort reductions from level 1 to each of the next levels are computed. For example, when documentation impact is assumed at 25% the effort multipliers are adjusted from (1.50, 1.20, 1.00, 0.83, and 0.65) to (1.125, 1.05, 1.00, 0.9575 and 0.9125), i.e. the increases and decreases from the nominal rating are reduced to 25% of the initial value. The entries in the 25% column in Table 5.3 are the percent changes due to 25% documentation impact. Moving from a rating of 1.125 to a rating of 1.05 means a reduction of 6.7%; moving from a rating of 1.125 to a rating of 1.00 means a reduction of 11.1%; moving from a rating of 1.125 to a rating of 0.9575 means a reduction of 14.9%; and finally moving from a rating of 1.125 to a rating of 0.9125 means a reduction of 18.9%. The rest of the table for testing effort reduction and the table for maintenance effort reduction are completed in the same fashion.

MODP Rating	Documentation Maturity Level	Testing Effort Reduction by % of Documentation Impact on MODP									
		5%	10%	15%	20%	25%	30%	35%	40%	45%	50%
Very low	Level 1 Strong Level 1 Weak Level 2	-	-	-	-	-	-	-	-	-	-
Low	Level 2	1.5	2.9	4.2	5.5	6.7	7.8	8.9	10.0	11.0	12.0
Nominal	Strong Level 2	2.4	4.8	7.0	9.1	11.1	13.0	14.9	16.7	18.4	20.0
High	Level 3	3.3	6.4	9.3	12.2	14.9	17.5	20.0	22.3	24.6	26.8
Very high	Strong Level 3 Level 4	4.1	8.1	11.9	15.5	18.9	22.2	25.3	28.3	31.2	34.0

Table 5.3: Percentage effort reduction on testing

Each documentation maturity level has been mapped to one of the MODP ratings, using the definitions provided in CoCoMo for cost driver MODP, and the definitions provided in Chapter 4 for the documentation maturity levels. Given the mismatch between the number of MODP ratings and documentation maturity levels, some of the documentation maturity levels (Strong Level 1, Weak Level 2 and Level 4) do not have an associated effort reduction value directly from CoCoMo's MODP multipliers. These 3 values can be inter- or extrapolated if needed.

One of this research's main claims is that as the documentation maturity level increases, the proportion of errors caught early in the development cycle increases also, resulting in decreasing integration testing and maintenance effort. Tables 5.3 and 5.4 support this claim, suggesting that even a small improvement in documentation (and in MODP) will result in savings in the form of effort reduction in testing and maintenance phases.

A cost-benefit analysis of documentation maturity levels is explained in the next section. A very conservative assumption is made for cost-benefit analysis purposes: documentation *determines* 25% of MODP, and so it is responsible for 25% of the effort reduction due to improvements in MODP. The assumption is conserva-

MODP Rating	Documentation Maturity Level	Maintenance Effort Reduction by % of Documentation Impact on MODP									
		5%	10%	15%	20%	25%	30%	35%	40%	45%	50%
Very low	Level 1 Strong Level 1 Weak Level 2	-	-	-	-	-	-	-	-	-	-
Low	Level 2	1.1	2.1	3.1	4.1	5.0	5.9	6.8	7.6	8.4	9.2
Nominal	Strong Level 2	2.0	3.8	5.7	7.4	9.1	10.7	12.3	13.8	15.3	16.7
High	Level 3	2.7	5.3	7.8	10.2	12.5	14.7	16.9	19.0	21.0	22.9
Very high	Strong Level 3 Level 4	3.3	6.5	9.6	12.6	15.5	18.2	20.9	23.4	25.9	28.3

Table 5.4: Percentage effort reduction on maintenance

tive because documentation is directly involved in half of the modern programming practices that define MODP (namely the requirement of design notation, use of reviews and existence of central repository for software components) so 25% is most likely an under-estimation.

5.4 Bottom line: cost-benefit analysis of maturity levels

In order to convince decision-makers of the benefits of adopting new technologies they have to be presented with hard and convincing evidence that the benefits of the change will outweigh the costs. This is also known as bottom-line cost justification or return on investment. For instance, Pressman [Press93] shows that 3 to 5 percent improvement in software development productivity can be achieved simply by installing state-of-the-art document production capability.

Cost-benefit analysis is a useful tool as a method of decision making that involves looking for solutions that maximize the difference in the expected values of benefits and costs. Yeh [Yeh93] describes a procedure for cost-benefit analysis as follows:

- Identify all the important sources of costs and benefits.
- Estimate the values of the costs and benefits.
- When there are uncertainties, estimate the probabilities of obtaining the costs and benefits.
- Compute the expected values of the costs and benefits.

An unfortunate drawback in cost-benefit analysis is that some of the costs and benefits are intangible (e.g. improved staff morale), and are all but impossible to quantify. So the analysis must concentrate in showing that the tangible benefits have greater value than the tangible costs, and in producing convincing evidence that the intangible benefits and costs do not reverse that situation.

Another problem stems from the fact that people resist change. Yeh [Yeh93] presents some reasons for that resistance: they are busy, probably as a result of poor process; the priority is getting the product out; there is little interest in changing the process; they fear that quality metrics will be used to punish them, and they may be concerned with job security; they may not be used to a disciplined approach to producing software; they may have the misconception that standards and procedures will stifle creativity.

Pressman [Press93] lists the following common managers' objections about new technologies:

- *We don't have the money this year; we are going to have to put this off.* Need justification that any expenditure will pay itself back in a relatively short time period.

- *We can't free up the people to do this; they are all tied up with on-going critical projects.*

- *But we already have standards for software works.* Standards by themselves offer little benefit. Practice is improved by implementing new technology in the context of a disciplined approach.

- *This will put us on a learning curve that will really slow things down; we can't afford the time lag for ongoing projects.*

- *Our people are happy with things as they are. It is likely that they'll resist this stuff.*

- *Show me scientific data that proves that software engineering and CASE will give us the benefits you suggest.* Sadly, most data published on these subjects are anecdotal; few scientific experiments have been conducted.

5.4.1 Associated costs

The costs associated with the levels as specified in the Documentation Process Maturity model are of two kinds: investment costs and continuing costs. Investment costs are those associated with attaining the level. Continuing costs are those associated with maintaining it.

At level 1 there is virtually no cost needed to attain it. It is the baseline scenario.

At level 2, the investment costs are establishing check-off lists of required documents, documentation standards, and to adopt simple documentation tools. The

continuing costs are the on-going utilization of documentation standards and tools. An intangible cost is associated with recognizing the importance of documentation as a legitimate software aspect, particularly at the managerial level.

At level 3, the investment costs are establishing documentation quality monitoring and usefulness assessment procedures and control units, and to adopt more advanced documentation tools. The continuing costs are the permanent activities required for documentation quality and usefulness assessments, and the utilization of advanced documentation tools.

At level 4, the investment costs are establishing documentation error tracking and process measurement methods, as well as an effective process improvement feedback loop, and to integrate CASE tools and documentation tools. The continuing costs are associated to using those methods effectively and routinely.

An exact determination of costs of achieving and maintaining each documentation maturity level is impossible. But a reasonable estimation can be made based on Boehm's estimate [Boehm81] that the activities comprising the tasks of Configuration Management and Quality Assurance (CM/QA) represent about 5% of the total effort spent in software development and maintenance. Since quality of the documentation process and products are only one of the many factors that determine the tasks of CM/QA, it seems reasonable to estimate the cost of achieving each documentation process maturity level as a percentage of CM/QA effort, and then as a percentage of the total effort. Therefore, if the cost of achieving each documentation maturity level is assumed as 10% of the cost of CM/QA, that represents 0.5% of total costs; if the assumed cost is 25% of CM/QA, then the

cost of achieving each documentation maturity level is 1.25% of total costs. If the assumed cost is 50% of CM/QA, then achieving each documentation maturity level will represent 2.5% of total costs.

5.4.2 Return on investment

In order to establish the return on investment ratio for each documentation maturity level, all the benefits must be identified.

In this research, and given the particular approach used in its validation, the only tangible benefit considered will be the reduced software testing and maintenance effort as a result of higher quality software and higher early error detection capabilities as maturity levels increase, assuming a very conservative impact of documentation on MODP of 25%, as explained before. In that way, testing and maintenance reduction effort percentages due to documentation are summarized in Table 5.5 which shows the estimated cumulative proportion of software testing and maintenance effort reduction as documentation maturity levels increase. The entries in the table are those corresponding to a 25% documentation impact rate taken from Tables 5.3 and 5.4.

The values on Table 5.5 are absolutely consistent with this research's initial belief that the documentation process improvement biggest payoff is obtained when a software organization moves from a state of total disarray (level 1) to a more structured one (level 2). The payoffs of moving from level 3 to level 4 (and for intermediate cases between levels 1 and 2) can't be determined given the lack of relevant data. The values could be inter- or extrapolated, according to each case.

Documentation Maturity Level	Integration Testing Effort Reduction	Maintenance Effort Reduction
Level 1	–	–
Strong Level 1		
Weak Level 2		
Level 2	6.7%	5%
Strong Level 2	11.1%	9.1%
Level 3	14.9%	12.5%
Strong Level 3	18.9%	15.5%
Level 4		

Table 5.5: Cumulative effort reduction assuming documentation 25% of MODP

Table 5.6 shows the total cumulative effort savings associated to achieving each documentation maturity level, assuming that documentation has a participation of 25% on modern programming practices (MODP) as explained in CoCoMo model, and assuming the following *typical* software lifecycle phase distribution: development 40% (analysis and design 40%, coding 20%, testing 40%) and maintenance 60% [Press92]. The entries have been computed by weighing testing effort reduction with a factor of 0.16 (development 40%, testing 40% of development) and maintenance effort reduction with a factor of 0.6 (maintenance 60%).

Since the costs of achieving each documentation maturity level and maintaining it were assumed to be between 0.5% and 2.5% of total costs, each documentation maturity level defines a range of positive returns on investment as shown in Table 5.7. The entries for Levels 2 and 3 have been computed by dividing the

Documentation Maturity Level	Estimated Total Effort Reduction
Level 1	-
Strong Level 2	7.5%
Strong Level 3	12.1%
Level 4	

Table 5.6: Cumulative percentage reduction on total effort

Documentation Process Maturity Level	Ratio of Total Effort Reduction to Cost Cost of Documentation as % of CM/QA		
	Assuming 10%	Assuming 25%	Assuming 50%
Level 1	-	-	-
Strong Level 2	15:1	6:1	3:1
Strong Level 3	12:1	5:1	2.5:1
Level 4	9:1	4:1	2:1

Table 5.7: Cumulative documentation maturity return on investment

estimated total effort reduction from Table 5.6 by the estimated cost range, i.e. dividing the total percent of effort reduction by the percent cost of documentation assumed as 10%, 25% and 50% of CM/QA costs. The entries in Table 5.7 represent the cumulative benefit/cost ratio. The range of return on investment for Level 4 has been extrapolated considering the fact that return on investment decreases as documentation maturity level increases. The range of payoffs defined for Level 4 is consistent with the decreasing nature of the returns and the actual figures determined for the levels 2 and 3.

Intangible benefits are almost impossible to quantify. The following list shows some of the intangible benefits associated to each documentation maturity level above level 1.

- Level 2:
  - Higher likelihood that all projects will be developed in a more uniform fashion.
  - Easier for technical staff to move across projects.
  - More predictable deliverables and documentation.
  - A more predictable technical format that makes reviews easier.
- Level 3:
  - Improved ability to use automated tools to help manage projects.
  - Better morale among technical managers resulting from improved management oversight and control.
  - Ease in enforcing quality assurance activities.
  - Improved documentation quality.
  - Improved consistency in documentation, notation, and modeling.
  - Better enforcement of standards for documentation, notation, and modeling.

- Level 4:
  - Improved ability to predict and measure progress.
  - Creation of a central repository for information produced during the process.
  - Easier to assess the impact of change.
  - Better management control.

5.5 Summary

The documentation process maturity model has been shown to be an effective tool to assess the status of the software documentation practices with the purpose of improving the quality of the final software product. Reduced software testing and maintenance effort, as a result of a higher quality software product, has been shown to be correlated with moving up in the scale of documentation process maturity levels. The initial goal of this research was to show that organizations at a higher documentation process maturity also produce higher quality software than those that don't: the CoCoMo model-based software effort reduction analysis results confirm it with a positive return on investment for achieving higher documentation process maturity. The empirical evidence described in next chapter offers further support for the claim.

Chapter 6

Assessment Results

6.1 Overview

This chapter presents the documentation process assessment results for several companies obtained during this research. It describes the validation approach initially devised, presents the summarized assessment results and analyzes the relation between early error detection and documentation maturity level. 14 assessments were conducted during the period March - November 1993 for 5 different software organizations. In all 14 software projects assessed the answers were provided by the people directly involved in their development and maintenance. A **Documentation Assessment Report** was generated for each assessment according to the guidelines detailed in Chapter 4. This chapter presents a summary with the results from those reports and the breakdown of projects by documentation process maturity levels. For the projects for which defect data was available, the data supports the hypothesis that the fraction of the total errors discovered during testing that are requirements and design errors is less than for projects developed with a lower documentation process maturity level.

6.2 Initial validation plan

The main goal of the research is to show that an organization at a higher level of documentation process will produce a higher quality product. Since most errors are introduced before coding and high quality documentation should aid in the discovery of these errors, a higher level documentation process should lead to the discovery of more of these errors than a lower level documentation process. To validate the model the initial approach was to show that for a software project developed with a higher documentation process maturity level, the fraction of the total errors discovered during testing that are requirements and design errors will be less than for projects developed with a lower documentation process maturity level. The fraction of total errors that are requirements and design errors was chosen rather than the total number of errors as our measure because the total number of errors depends on the type of application, the size of the application, programmer training and experience, and a variety of other factors. We believe the fraction measure best reflects the influence of the quality of documentation on errors.

To gather the empirical data the plan was to select project teams and specific projects in an organization. Each team would complete the assessment questionnaire for a particular project and provide the error data for that project. The error data would include: number of errors introduced in each phase (requirements, design, coding, testing, maintenance), severity of each error (serious, moderate, minor), number of errors detected during each phase, and number of errors corrected during each phase. From the assessment questionnaire the team's documentation

maturity level was to be determined. The error data would be then correlated with maturity levels and errors to verify the hypothesis. It was expected that the data would show that the higher the project team documentation process level, the smaller the fraction of requirements and design to total errors. This would enable us to estimate the cost/benefits of achieving higher documentation maturity levels. Clearly it is far less expensive to discover and to correct errors in the phase in which they are introduced than in a later phase. We did not expect to be able to make any statements about the total number of errors being less for teams at higher maturity levels since the number of errors depends on the size of the project, application area, and so forth. However, we expected to see a different error distribution by phases for teams with higher maturity levels than for teams with lower maturity levels.

Next section presents the actual results and defect data gathered. Although the data gathered was not sufficient to allow us to perform a meaningful statistical analysis as described in this plan, it did support the claims from previous Chapter 5 which describes the change in validation approach.

6.3 Summary of assessment results

Fourteen assessments were conducted during the period March - November 1993 on 5 different software organizations. These organizations were a large hardware manufacturer's software lab, two medium-size health systems company's software divisions, a small software house and a small federal agency's software department. Table 6.1 summarizes the results obtained from the 14 assessments.

Company A	Project 1	Strong Level 1
Company B	Project 2	Level 1
Company C	Project 3	Strong Level 1
Company D	Project 4	Level 1
	Project 5	Level 1
	Project 6	Weak Level 2
	Project 7	Weak Level 2
	Project 8	Level 2
	Project 9	Level 2
	Project 10	Strong Level 2
Company E	Project 11	Strong Level 1
	Project 12	Weak Level 2
	Project 13	Level 2
	Project 14	Strong Level 2

Table 6.1: Assessment results by company/project

Software Documentation Maturity Level	Number of Projects
Level 1	3
Strong Level 1	3
Weak Level 2	3
Level 2	3
Strong Level 2	2
Level 3	0
Level 4	0

Table 6.2: Documentation assessment results

6.4 Project breakdown by documentation maturity level

Table 6.2 shows a breakdown of projects by software documentation process maturity level. The distribution of the projects by documentation maturity level appears quite predictable: no projects at the levels 3 and 4, and equal distribution in the first two levels.

6.5 Defect data analysis

Out of the 14 projects under assessment, only four of them had defect data available that is shown in Tables 6.3 and 6.4. The data corresponds to Projects 1, 11, 12 and 13 whose documentation maturity levels were determined as Strong Level 1, Strong Level 1, Weak Level 2 and Level 2 respectively. For Project 1 in Table 6.3, cumulative introduction and detection error percentages by phase are shown. Projects 11, 12 and 13 (all from same company) are shown in Table 6.4.

The error data for Project 1 in Table 6.3 shows a need for improved early error detection, especially during design: while 83% of the errors are introduced before testing, 64% of them are found during testing or after. 6% of all errors are introduced in design, but only 1% of them are found before implementation. This behavior is consistent with the thesis that improving the documentation process should significantly increase the proportion of errors that are caught in the same phase in which they are introduced.

Table 6.3 shows the cumulative percentages of errors introduced and found by the end of every phase. The data is presented for critical and serious errors and for all errors. The data shows that by the end of design only *low severity* errors have been found.

The error data for Projects 11,12 and 13 in Table 6.4 shows a clear match between documentation maturity level and percentage of errors introduced early during requirements and design. The percentage of serious errors is similarly high for all 3 Projects, but the ones at a higher documentation maturity level (Projects 12 and 13) show a smaller proportion of errors introduced early on in the development compared to Project 11. Projects 12 and 13 are assessed at documentation maturity level 2 and the percentage of errors introduced early is $\approx 25\%$. Project 11 is assessed at level 1 and its percentage of errors introduced early is $\approx 40\%$.

Lanphar [Lanph90] describes a project that was under way over the last decade at the Hughes Aircraft Company, Ground Systems Group, Software Engineering Division to improve the productivity and quality aspects of their software development process. Known as Quality Process Management (QPM), it was cre-

At the end of	Critical and Serious errors		All errors	
	% introduced	% found	% introduced	% found
Design	5	0	6	1
Implementation	83	39	83	36
Test	99	94	98	91
Post-release	100	100	100	100

Table 6.3: Project 1: cumulative error percentages

Project	Maturity Level	% Serious Errors	% Req&Des Errors
11	Strong Level 1	78	41
12	Weak Level 2	77	25
13	Level 2	85	27

Table 6.4: Projects 11, 12, 13: summary of errors introduced

At the end of	All errors	
	% introduced	% found
Design	37	30
Implementation	98	89
Test	100	95
Post-release	100	100

Table 6.5: QPM Project: cumulative error percentages

ated to provide management with a better understanding of the process and the needed changes by incorporating statistical process control to the software development activities to obtain a more predictable process change management system.

While this organization has not been assessed regarding documentation process maturity, it can be used as an example in this validation effort because its practices are consistent with a documentation process maturity level of 2 or higher. This is so because QPM incorporates check and balances, formal reviews, standards to create the project reports, documentation of the process, maintenance of a historical repository, among others features. A full description of QPM can be found in [Lanph90]

Lanphar also reports on historical defect detection capabilities since the adoption of QPM. The data has been consolidated and is summarized in Table 6.5.

The error data for QPM Project in Table 6.5 shows that while 98% of the errors are introduced before testing, only 11% of them are found during testing or after. QPM Project documentation maturity level is at least level 2.

6.6 Summary

The assessments performed and the actual, although limited, defect data gathered supports this research's documentation process maturity model as an effective tool to assess the status of the software documentation practices with the purpose of improving the quality of the final software product. The defect data gathered suggests that software development projects that show higher documentation maturity level also show higher early error detection capabilities.

Chapter 7

Conclusions

7.1 Contributions of this research

- This research addressed an important, but neglected problem: the low quality of the documentation produced during software development. Even though everyone in the software community agrees that documentation is an important factor in the quality of their development and maintenance efforts, most organizations don't invest enough in resources (time, personnel, tools) to improve the quality of their documentation process. This thesis work is a vivid quantitative demonstration of the importance of documentation.

- The approach used in this thesis is a novel one: development of a process maturity model. The principle behind this approach is to improve the software product quality by improving the documentation process quality. The model and the assessment procedure are the tools that permit the assessment of the software documentation practices and the identification of an action plan for improvement.

- This research and its products (the documentation maturity model and its assessment procedure) is not a *silver bullet*, i.e. it does not solve all the problems regarding development and maintenance of software. It is just a step in the right direction, because it focuses on an important problem and provides some guidance for its solution.

- This research has demonstrated the successful extension of the concepts of SEI's process maturity models to the documentation process, a specific software development process.

- The documentation maturity model emerges as a valid and useful tool to manage complexity of the process. It allows the determination of the state of the software documentation practices and the generation of action plans to improve them.

- This research quantifies the return on investment in terms of reduced software testing and maintenance effort as a result of higher quality software products associated with higher maturity documentation processes. This research's estimations are a return of investment of 3:1 to 15:1 for achieving a Strong Level 2, 2.5 to 12:1 for attaining a Strong Level 3. The extrapolation for Level 4 yields a 2 to 9:1 return on investment. The values are cumulative, using Level 1 as the baseline. Each level defines a range of return on investment rather than a single value because different percentages of total effort are assumed as the cost associated with improving documentation. The biggest return on investment occurs when a software development organization moves

from Level 1 to a Strong Level 2, i.e. when the organization replaces its obscure and chaotic documentation process for a structured one. Attaining the other higher levels is also very cost-effective, but not as much.

7.2 Open issues and areas of further research

- Extend the validation study to do more assessments and collect hard defect data to produce stronger support for the results obtained in this thesis.
- Study the impact of documentation maturity on software productivity. While it has been shown that the quality of the software process and products improves as the documentation process matures, the issue of software productivity during development is less understood.
- Development of a Documentation Maturity CASE tool, to be used as a facilitator to self-assessments and *automatic* generation of actions plans. While decision-making on software processes still remains a human activity, such a tool could be very useful for software managers dealing with software documentation process improvement issues.
- Extend process maturity model ideas to other software development processes, e.g. testing, design, maintenance.

Bibliography

[Alb76] Alberts, David S. The economics of software quality assurance. Conference Proceedings, NCC, New York, 1976.

[Basi84] Basili, Victor R. and Perricone, Barry T. Software errors and complexity: An empirical investigation. Communications of the ACM, Vol. 27, No. 1, 1984.

[Basi91] Basili, Victor R. and Musa, John D. The future engineering of software: a management perspective. IEEE Computer, September, 1991.

[Baum91] Baumert, John. New SEI maturity model targets key practices. In the news, IEEE Software, November 1991.

[Boehm75] Boehm, Barry W. The high cost of software. From Horowitz, Practical Strategies for Developing Large Software Systems, Addison-Wesley, Reading, Mass, 1975.

[Boehm81] Boehm, Barry W. Software engineering economics, Prentice-Hall, 1981.

[Boll91] Bollinger, Terry B. and McGowan, Clement. A critical look at software capability evaluations. IEEE Software, July 1991.

[Buck89] Buckley, J. Some standards for software maintenance. Standards, IEEE Computer, November 1989.

[Card87] Card, David N., Mc Garry, Frank E. and Page, Gerald T. Evaluating software engineering technologies. IEEE Transactions on software engineering, Vol. SE-13, No. 7, 1987.

[Cousin93] Cousin, Larry. Improving software quality through practical CMM level progression. Proceedings Pacific Northwest Software Quality Conference, Portland, Oregon, October 1993.

[Chapin87] Chapin, Ned. The job of software maintenance. Proceedings Conference on Software Maintenance, IEEE, 1987.

[Chapin88] Chapin, Ned. Software maintenance life cycle. Proceedings Conference on Software Maintenance, IEEE, 1988.

[Conte86] Conte, S.D., Dunsmore, H.E., Shen, V.Y. Software Engineering metrics and models. Benjamin/Cummings, 1986.

[DeMarc82] DeMarco, Tom. Controlling software projects: management, measurement and estimation. Yourdon Press, 1982.

[Dion93] Dion, Raymond. Process Improvement and the Corporate Balance Sheet. IEEE Software, July 1993.

[Ejiogu93] Ejiogu, Lem O. Five principles for the formal validation of models of software metrics. ACM SIGPLAN Notices, Vol. 28, No. 8, 1993.

[Endres75] Endres, Albert. An analysis of errors and their causes in system programs. IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, 1975.

[Fagan76] Fagan, M.E. Design and code inspections to reduce errors in program development. IBM Systems Journal, Vol. 15, No. 3, 1976.

[Fjel79] Fjelstad, R.K and Hamlen, W.T. Application program maintenance study report to our respondents. Proceedings GUIDE 48, Philadelphia, PA, 1979.

[Foster93] Foster, Lynne. Using the Motorola Quality System Review (QSR) to drive software improvement. Proceedings Pacific Northwest Software Quality Conference, Portland, Oregon, October 1993.

[Gilch92] Gilchrist, J.M. Project evaluation using the SEI method. Software Quality Journal 1, 37-44, 1992.

[Guim83] Guimaraes, Tor. Managing application program maintenance expenditures. Communications of the ACM, Vol. 26, No. 10, 1983.

[Grady87] Grady, Robert B. and Caswell, Deborah L. Software metrics: establishing a company-wide program. Prentice-Hall, 1987.

[Hager89] Hager, James A. Software cost reduction methods in practice. IEEE Transactions on software engineering, Vol. SE-15, No. 12, 1989.

[Hager91] Hager, James A. Software cost reduction methods in practice: a post mortem analysis. Journal of systems and software, Vol. 14, No. 2, 1991.

[Henry92] Henry, Joel and Henry, Sallie. An integrated approach to software process assessment. Proceedings of the Pacific Northwest Software Quality Conference, Portland, Oregon, October 1992.

[Hersh93] Hersh, Art. Where's the return on process improvement? IEEE Software, July, 1993.

[How78] Howden, William E. Empirical studies of software validation. Tutorial: Software Testing & validation Techniques, IEEE Computer Society, 1978.

[Humph87] Humphrey, Watt S. and Kitson, David. Preliminary report on conducting SEI-assisted assessments of software engineering capability. CMU/SEI-87-TR-16, ESD-TR-87-117, July 1987.

[Humph87a] Humphrey, Watt S. and Sweet, W. L. A method for assessing the software engineering capability of contractors. CMU/SEI-87-TR-23, ESD-TR-87-186, September 1987.

[Humph88] Humphrey, Watt S. Characterizing the software process: a maturity framework. IEEE Software, March 1988.

[Humph89] Humphrey, Watt S. Managing the software process. Addison-Wesley, Reading, Mass, 1989.

[Humph89a] Humphrey, Watt S., Kitson, David H., Kasse, Tim C. The state of software engineering practice: a preliminary report. CMU/SEI-89-TR-1, ESD-TR-89-01, February 1989.

[Humph91] Humphrey, Watt S., Snyder, Terry R. and Willis, Ronald R. Software process improvement at Hughes Aircraft. IEEE Software, July 1991.

[Humph91a] Humphrey, Watt S. and Curtis, Bill. Comments on "A critical look". IEEE Software, July 1991.

[Jovan93] Jovanovic, Vladan and Shoemaker, Dan. ISO 9000-3 self-auditing using the SDI method. Proceedings Pacific Northwest Software Quality Conference, Portland, Oregon, October 1993.

[Kell89] Kellner, Marc I. Non-traditional perspectives on software maintenance. Panel, Proceedings Conference on Software Maintenance, IEEE, 1989.

[King93] King, Karen S. Implementing software process improvement. Proceedings Pacific Northwest Software Quality Conference, Portland, Oregon, October 1993.

[Lanph90] Lanphar, Robert. Quantitative Process Management in Software Engineering, a reconciliation between process and product views. Journal of Systems and Software, December 1990.

[Lee87] Lee, Joan. To See Ourselves as Other See Us: Evaluating Software Documentation. ACM SIGDOC * Asterisk, Vol. 13, No. 1, 1987.

[Lewis91] Lewis, Theodore G. CASE: Computer-Aided Software Engineering. Van Nostrand Reinhold, 1991.

[Lien81] Lientz, Bennet P. and Swanson, E. Burton. Problems in application software maintenance. Communications of the ACM, Vol. 24, No. 11, 1981.

[Lipke92] Lipke, Walter H. and Butler, Kelley L. Software process improvement: a success story. Field Reports, CrossTalk, November 1992.

[Oman91] Oman, P., Hagemeister, J., Ash, D. A Definition and Taxonomy for Software Maintainability. Technical Report 91-08 (revised version), Computer Science Department, U. of Idaho, January 1992.

[Osborne87] Osborne, Wilma M. Building and Sustaining Software Maintainability. Proceedings Conference on Software Maintenance, IEEE, 1987.

[Over93] Over, James W. Motivation for process-driven development. Software Engineering Technology, CrossTalk, January 1993.

[Paulk91] Paulk, M. C., Curtis, B., Chrissis, M. B., et al. Capability Maturity model for software. CMU/SEI-91-TR-24, August 1991.

[Paulk93] Paulk, M. C., Curtis, B., Chrissis, M. B., and Weber, C. V. Capability Maturity Model, Version 1.1. IEEE Software, July 1993.

[Paulk93a] Paulk, M. C. et al. Capability Maturity model for software, version 1.1. CMU/SEI-93-TR-24, 1993.

[Paulk93b] Paulk, M. C. et al. Key Practices of the Capability Maturity model for software, version 1.1. CMU/SEI-93-TR-25, 1993.

[Post84] Poston, Robert M. When does more documentation mean less work?. Software Standards, IEEE Software, October 1984.

[Post85] Poston, Robert M. Selecting software documentation standards. Software Standards, IEEE Software, May 1985.

[Press92] Pressman, Roger S. Software engineering: a practicioner's approach. Third Edition. McGraw-Hill, 1992.

[Press93] Pressman, Roger S. A manager's guide to software engineering. McGraw-Hill, 1993.

[Ramam88] Ramamoorthy, C.V. Our job is to reduce the errors. From Myers, Ware. Can software for the strategic defense initiative ever be error free? IEEE Computer, Vol. 21, No. 11, 1988.

[Romb87] Rombach, H. Dieter and Basili, Victor R. Quantitative assessment of maintenance: an industrial case study. Proceedings Conference on Software Maintenance, IEEE, 1987.

[Scheff91] Scheff, Benson H. and Georgon, Thomas. Using documentation blueprints to produce mandated DoD data items. Journal of Systems and Software, Vol. 14, No. 2, 1991.

[Seddio92] Seddio, Carl. Applying review and product metrics to the software engineering process: a case study. Software Quality Journal 1, pp. 133-145, 1992.

[Stev88] Stevens, K. Todd, Arthur, James D. and Nance, Richard E. A Taxonomy for the Evaluation of Computer Documentation. SRC-88-008, Virginia Polytechnic Institute and State University, June 1988.

[Strig93] Strigel, Wolfgang B. Industry-wide software process improvements. Proceedings Pacific Northwest Software Quality Conference, Portland, Oregon, October 1993.

[STSC92] Crosby, D., Petersen, G., Sorensen, R. Documentation Tools Report, Software Technology Support Center (STSC), March 1992.

[Weber91] Weber, C.V., Paulk, M.C., Wise, C.J., Withey, J.V. Key practices of the Capability Maturity model. CMU/SEI-91-TR-25, August 1991.

[Yeh93] Yeh, Hsiang-Tao. Software Process Quality. McGraw-Hill, 1993.

Appendices

Appendix A

Assessment Questionnaire

Instructions to Administer Questionnaire

This questionnaire should take about 30 minutes to complete.

- Only identification needed is the software project.
- The questions must be answered regarding the project only, and not the organization as a whole.
- The questions refer to software documentation produced during development intended to be used by software developers and maintainers, not by end users.
- When referring to users, it means users of the documentation (software developers and maintainers), not end users.
- Simple documentation tools refer to text processors and graphic tools.
- Advanced documentation tools refer to integrated documentation/CASE tools or document management systems.

Software System Documentation Process Maturity

Sample Questionnaire Form

Project Identification:_____

Please read carefully the following term definitions

- Software Documentation: Includes all documents generated as part of software development, i.e. software requirements specifications, design documents, code, test plans and history (test cases). Software documents refer to either hard copy or electronic form. It excludes end-user documentation, i.e. user manuals and operations manuals; and it excludes managerial documentation, i.e. project plans, schedule and staff plans, etc.
- Software CASE tools: Software designed to assist software engineers and programmers cope with the complexity of the process and the artifacts of Software Engineering.
- Documentation tools: Software designed to aid software engineers to cope with the complexity of the process and artifacts of documentation.

  Types of documentation tools:

  - Simple:
    1. Text processors: word processors, desktop publishers, editors, spelling checkers, electronic mail.
    2. Graphics: flowcharting, technical drawing.
  - Advanced:
    1. Document management systems: storage, retrieval, browsing, distribution, sharing, consistency checking.
    2. Integrated documentation/CASE tools.
- Quality of documentation: Quality includes the following characteristics: adequacy, completeness, usability, consistency, currency, readability, ease of use, ease of modification, tracebility,
- Usefulness of documentation: Extent to which the documentation products are used by software developers and maintainers, the documentation users.

For each question, circle answer that best fits project

Each question is to be answered with either

- a yes/no answer, or
- an integer in range of 1-5 with the following meanings:
  - 1: never
  - 2: seldom
  - 3: sometimes
  - 4: usually
  - 5: always

1. Are software documents other than code created during development?

   1 2 3 4 5

2. Are software requirements specifications (including prototyping documents) generated?

   1 2 3 4 5

3. Are design documents (including prototyping documents) generated?

   1 2 3 4 5

4. Are test plan documents generated?

   1 2 3 4 5

5. Are test cases used in testing recorded in a document?

   1 2 3 4 5

6. Are all software documents generated in the development phase used? (SRS used in design, design document used in coding, etc)

   1 2 3 4 5

7. Are software documents other than code used during development?

   Yes No

8. Are software documents other than code used during maintenance?

Yes No

9. When software documents are not used, is it because they are unreliable, incomplete, or out of date?

Yes No

10. When software documents are not used, is it because they are not easily accessible?

Yes No

11. Are there check-off lists that indicate which software documents must be created?

1 2 3 4 5

12. Are there standards indicating what must be included in each software document?

1 2 3 4 5

13. Is there any procedure or form used to specify how and when to write each software document?

1 2 3 4 5

14. Is there a formal procedure used for checking that document contents satisfy standards?

1 2 3 4 5

15. Are simple documentation tools (text processors, graphics) used to create and maintain software documentation?

1 2 3 4 5

16. Are advanced documentation tools (integrated, documentation management) used during development and maintenance?

1 2 3 4 5

17. Are common sets of documentation tools used in the different development environments throughout the organization?

1 2 3 4 5

18. Are advanced documentation tools integrated with software CASE tools?

Yes No

19. Is adequate time allocated to develop software documentation during software development?

1 2 3 4 5

20. Are software documents checked to see that they have been done?

1 2 3 4 5

21. For each software project, is there a person responsible for collecting the documentation?

1 2 3 4 5

22. For each software project, is there a person responsible for maintaining the documentation?

1 2 3 4 5

23. Is there a mechanism for checking that a software document has been completed satisfactorily?

Yes No

24. If so, how frequently is it used?

1 2 3 4 5

25. After a change has been made to the code, is there a mechanism for checking that all related documentation is updated?

Yes No

26. If so, how frequently is it used?

1 2 3 4 5

27. After a change has been made to the design, is there a mechanism for checking that all related documentation is updated?

Yes No

28. If so, how frequently is it used?

1 2 3 4 5

29. After a change has been made to the requirements, is there a mechanism for checking that all related documentation is updated?

Yes No

30. If so, how frequently is it used?

1 2 3 4 5

31. Is affected documentation updated after each maintenance change?

1 2 3 4 5

32. Is there a mechanism to monitor the quality of the documentation?

Yes No

33. If so, how frequently is it used?

1 2 3 4 5

34. Is there an independent group whose function is to assess the quality of the documentation generated in a project?

Yes No

35. Is there a mechanism to assess the usefulness of the documentation generated in a project?

Yes No

36. Is there a mechanism for users of the documentation (software developers and maintainers) to provide feedback to improve documentation usefulness?

Yes No

37. If so, how frequently is it used?

1 2 3 4 5

38. Are measures of usefulness of documentation collected?

1 2 3 4 5

39. Are documentation errors and trouble reports tracked to solution?

1 2 3 4 5

40. Are documentation process data and error data for software projects recorded in a database?

1 2 3 4 5

41. Are statistics gathered on documentation errors?

1 2 3 4 5

42. Is documentation error data analyzed to determine distribution and characteristics of errors?

Yes No

43. If so, how frequently is this done?

1 2 3 4 5

44. Is there a mechanism to analyze documentation error root causes?

Yes No

45. If so, how frequently is it used?

1 2 3 4 5

46. Is there a documentation usage profile generated for software development?

Yes No

47. If so, how frequently is this done?

1 2 3 4 5

48. Is there a documentation usage profile generated for software maintenance?

Yes No

49. If so, how frequently is this done?

1 2 3 4 5

50. Is there a mechanism to feedback improvements to documentation practices or standards?

Yes No

51. If so, how frequently is it used?

1 2 3 4 5

52. Is formal training available for the use of documentation standards?

1 2 3 4 5

53. Is formal training available for the use of documentation tools?

1 2 3 4 5

54. Is there a mechanism to foster the incorporation of advances in documentation technology across the organization?

Yes No

55. Does management have a policy (not necessarily written) supporting the importance of software documentation?

Yes No

56. Does management view software documentation as of major importance and have written policies to this effect?

Yes No

End of questionnaire

Appendix B

Assessment Questionnaire: Questions by Subject Area

The questions that are to be answered in the yes/no mode are **highlighted.**

- Lifecycle documentation

1. Are software documents other than code created during development?
2. Are software requirements specifications (including prototyping documents) generated?
3. Are design documents (including prototyping documents) generated?
4. Are test plan documents generated?
5. Are test cases used in testing recorded in a document?
6. Are all software documents generated in the development phase used? (SRS in design, design document in coding, SRS and design in maintenance, etc)
7. **Are software documents other than code used during development?**
8. **Are software documents other than code used during maintenance?**
9. **When software documents are not used, is it because they are unreliable, incomplete, or out of date?**

10. **When software documents are not used, is it because they are not easily accessible?**

- Documentation standards

1. Are there check-off lists that indicate which software documents must be created?
2. Are there standards indicating what must be included in each software document?
3. Is there any procedure or form used to specify how and when to write each software document?
4. Is there a formal procedure used for checking that document contents satisfy standards?

- Documentation tools

1. Are simple documentation tools (text processors, graphics) used to create and maintain software documentation?
2. Are advanced documentation tools (integrated, documentation management) used during development or maintenance?
3. Are common sets of documentation tools used in the different development environments throughout the organization?
4. **Are advanced documentation tools integrated with software CASE tools?**

- Documentation quality control

1. Is adequate time allocated to develop software documentation during software development?
2. Are software documents checked to see that they have been done?
3. For each software project, is there a person responsible for collecting the documentation?
4. For each software project, is there a person responsible for maintaining the documentation?
5. **Is there a mechanism for checking that a software document has been completed satisfactorily?**
6. If so, how frequently is it used?
7. **After a change has been made to the code, is there a mechanism for checking that all related documentation is updated?**
8. If so, how frequently is it used?
9. **After a change has been made to the design, is there a mechanism for checking that all related documentation is updated?**
10. If so, how frequently is it used?
11. **After a change has been made to the requirements, is there a mechanism for checking that all related documentation is updated?**
12. If so, how frequently is it used?

13. Is affected documentation updated after each maintenance change?

14. **Is there a mechanism to monitor the quality of the documentation?**

15. If so, how frequently is it used?

16. **Is there an independent group whose function is to assess the quality of the documentation generated in a project?**

- Documentation usefulness determination and assurance.

1. **Is there a mechanism to assess the usefulness of the documentation generated in a project?**

2. **Is there a mechanism for users of the documentation (software developers and maintainers) to provide feedback to improve documentation usefulness?**

3. If so, how frequently is it used?

4. Are measures of usefulness of documentation collected?

- Documentation error analysis and improvement feedback

1. Are documentation errors and trouble reports tracked to the solution?

2. Are documentation process data and error data for software projects recorded in a database?

3. Are statistics gathered on documentation errors?

4. **Is documentation error data analyzed to determine distribution and characteristics of errors?**
5. If so, how frequently is this done?
6. **Is there a mechanism to analyze documentation error root causes?**
7. If so, how frequently is it used?
8. **Is there a documentation usage profile generated for software development?**
9. If so, how frequently is this done?
10. **Is there a documentation usage profile generated for software maintenance?**
11. If so, how frequently is this done?
12. **Is there a mechanism to feedback improvements to documentation practices or standards?**
13. If so, how frequently is it used?

- Documentation-related training.

1. Is formal training available for the use of documentation standards?
2. Is formal training available for the use of documentation tools?
3. **Is there a mechanism to foster the incorporation of advances in documentation technology across the organization?**

- Management attitude towards documentation.

    1. **Does management have a policy (not necessarily written) supporting the importance of software documentation?**

    2. **Does management view software documentation as of major importance and have written policies to this effect?**

Appendix C

Documentation Assessment Report: Action List

The following lists the practices and the improvement actions needed when it is either unsatisfactory or missing.

- Level 1
  - Creation of basic software development documents
    * Partial satisfaction: All basic software documents must be created for all phases of software development. This includes software requirements specification, design document, test plans and test implementations.
    * No satisfaction: Incorporate software documentation as a legitimate part of software development. Software requirements specification, design document, test plans and test implementations must be created besides the code.
  - Documentation generally recognized as important
    * Partial satisfaction: Increase awareness about importance of documentation in software development and maintenance.
    * No satisfaction: Recognize documentation as an important part of software development and maintenance.

- Level 2
  - Written statement about importance of documentation
    * Partial satisfaction: Written policy on importance of documentation must be available and be a part of all software development projects.
    * No satisfaction: Create written policy on importance of documentation and make that policy an important part of each software development project. This translates into allocating sufficient resources for documentation.
  - Adequate time and resources for documentation
    * Partial satisfaction: Increase the time allocated to create software development documentation. Appoint a person to be responsible for collecting and maintaining the documentation.
    * No satisfaction: Allocate sufficient time and resources to create software development documentation that meets the standard for each phase of the development.
  - Adherence to documentation standards
    * Partial satisfaction: Documentation standards must be followed in the creation of all software documents.
    * No satisfaction: Adopt documentation standards and a plan to satisfy them in all software development projects.

- Use of a check-off list of required documentation
  - * Partial satisfaction: A check-off list of required software development documents for all software development projects.
  - * No satisfaction: Define a specific check-off list of required software development documents for all software development projects.
- Use of simple documentation tools
  - * Partial satisfaction: Simple documentation tools (text processors, graphics) must be used by all software development projects and maintenance activities.
  - * No satisfaction: Define a set of simple documentation tools (text processors, graphics) to be used in software development and maintenance.

• Level 3

- Use of software documentation generated
  - * Partial Satisfaction: Software documentation generated should be used in subsequent phases of the software process. Investigate why documentation is not used in subsequent phases of the software process.
  - * No satisfaction: Define a mechanism to ensure that all software documentation generated is used throughout the software lifecycle.
- Mechanisms to update documentation

    * Partial Satisfaction: Documentation (requirements, design, code) must be updated after each change.

    * No satisfaction: Define a mechanism to update affected documentation after each change in code, requirements or design.

- Mechanisms to monitor quality of documentation

    * Partial Satisfaction: All software development documents must be checked for satisfactory completion, and their quality monitored and assessed by an independent group.

    * No satisfaction: Define a mechanism to check that software development documents have been completed satisfactorily, and to monitor quality of the documents.

- Methods to assess usefulness of documentation

    * Partial Satisfaction: Interview software developers and maintainers to assess usefulness of all documents.

    * No satisfaction: Establish a mechanism to assess usefulness of all documentation, by getting feedback from the users of the documentation, i.e. software developers and maintainers.

- Use of common sets of documentation tools

    * Partial Satisfaction: Use of a common set of documentation tools by the software developers when working in a particular software development environment.

    * No satisfaction: Define a common set of documentation tools to be used on each software development environment.

- Use of advanced documentation tools

    * Partial Satisfaction: Advanced documentation tools (integrated, documentation management) must be used by all software development projects and maintenance activities.

    * No satisfaction: Define a set of advanced documentation tools (integrated, documentation management) to be used in software development and maintenance.

- Documentation-related technology and training

    * Partial Satisfaction: Every software developer and maintainer must be trained in the use of documentation tools, satisfaction of documentation standards and advances in software documentation technology.

    * No satisfaction: Establish a program to train software developers and maintainers in the use of documentation tools and about satisfaction of documentation standards, and to transfer documentation technology to the software personnel.

- Level 4
  - Measures of documentation process quality
    * Partial Satisfaction: Collect and analyze measures of errors for all software development projects.
    * No satisfaction: Establish a measurement program to gather error data, analyze their causes, distribution and characteristics.
  - Analysis of documentation usage and usefulness
    * Partial Satisfaction: Analyze documentation usage and usefulness for all software development projects.
    * No satisfaction: Establish a mechanism to analyze documentation usage in software development and maintenance, and to measure documentation usefulness.
  - Process improvement feedback loop
    * Partial Satisfaction: Further incorporate improvements to the documentation process, based on feedback obtained through error analysis.
    * No satisfaction: Establish a mechanism to feedback improvements to the documentation process.
  - Integrate CASE and documentation tools
    * Partial Satisfaction: Use integrated software CASE tools and software documentation tools in all software projects.

* No satisfaction: Integrate software CASE tools and software documentation tools.