Software Engineering Practices In Lebanon And Suggestions For Lebanese Requirements Analysis And Software Testing Standards

By
Marianne F. Hajjar
B. Sc., Notre Dame University

PROJECT

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science at the Lebanese American University
August 1996

[redacted]
Dr. Nashat Mansour (Advisor)
Professor of Computer Science
Lebanese American University

[redacted]
Dr. Ali Ataya
Professor of Computer Science
Lebanese University

Acknowledgments

I would like to convey my special thanks to my dearest uncle and sponsor, Saeed Canaan, whose love and support kept me going on to complete what he had started, a higher education.

I also thank my advisor, Professor Nashat Mansour who urged me to feel proud of my work and keep the pressure up, in addition to my second reader, Professor Ali Ataya for taking the time to read this thesis, without forgetting all the software companies who filled my five page questionnaire with concern and enthusiasm.

Last, but not least, I thank my family, friends and colleagues at work who encouraged me at every step and cheered me up giving me the strength I needed.

Abstract

We analyze the Software Engineering practices in Lebanon and provide suggestions for local requirements analysis and software testing standards. The analysis is based on a questionnaire answered by local Software Engineering companies. The analysis results show great lack in documentation and standardization. Our suggested local standards are based on a selection of international standards namely the ISO, British and IEEE Software Engineering Standards. These standards would improve the local software quality and guide the local software engineers in producing internationally marketable software products.

Table of Contents

Chapter 1

Introduction

Software Engineering is the methodology that bridges the gap between the technology needs of a potential user and the delivery of a software system that satisfies those needs [Goldberg 1986]. It was first developed out of the necessity to develop large systems that greatly surpassed the ability of one or a few individuals to handle.

Software Engineering is an effective and efficient development process that is constantly improving through formal methods and standard guidelines to improve software quality. The life cycle of a software product is defined as the period of time that starts with concept exploration and ends when the product is finally retired. During this period, the product goes through a series of phases as follows:

Phase 1. *Requirements:* The concept is explored , refined and the clients requirements are ascertained and analyzed.

Phase 2. *Specifications:* The clients requirements are presented in the form of a document specifying what the product is supposed to do.

Phase 3. *Design:* The specifications undergo a design process following a design methodology. The resulting document specifies how the product is going to meet the requirements.

Phase 4. *Implementation:* The various components are coded and tested.

Phase 5. *Integration:* The product is tested as a whole. The phase ends when the client accepts the product.

Phase 6. *Maintenance:* It includes all changes to the product once the client has accepted it as satisfying the specifications.

Phase 7. *Retirement:* The product is removed from service, that is, retired.

Software Engineering practices refer to the methodologies used throughout the life cycle phases of a product. Software Engineering is a relatively new discipline in Lebanon. Local software applications have not reached a high degree of complexity. But, as software applications grow in number, size and complexity, Software Engineering techniques become necessary and urgent.

There are various international standards to guide the Software Engineering practices such as the IEEE and ISO standards. These standards are implemented in various ways in different countries. The Software Engineering industry in Lebanon lacks standard methods. This affects the software quality on the long run. Misinterpreted user and software requirements can lead to the production of a software that does not satisfy the customer's needs. Lack of test planning early in the software's life cycle will render testing a very difficult and complicated task since it is an inherent part of every phase in the Software Engineering process. Standardization is thus extremely important.

This thesis presents a study of the Software Engineering practices in Lebanon. It also provides suggestions to lebanese requirements analysis and software testing standards based on a collection of researches and international standards. The aim is to make a contribution to standardizing Software Engineering practices in Lebanon and to make these standards consistent with the international standards. This will greatly improve the marketability of locally developed software.

This thesis is organized at follows: Chapter 2 includes a questionnaire analysis that reflects the current local Software Engineering methodologies. This questionnaire was

answered by a number of software companies during Summer and Fall of 1995. Chapter 3 presents suggestions for requirements analysis standards followed by guidelines to implement them in chapter 4. Chapter 5 follows with suggestions to local software testing standards including planning and writing the test plan. Chapter 6 is the conclusion. Some relevant readings are presented under the appendices. Appendix A presents the International Standards Organization (ISO) emphasizing the ISO9000-3 which facilitates the application of the ISO9001 standard. Appendix B is an overview of the Institute of Electrical and Electronics Engineers (IEEE) standard dictionary concerning measures to produce reliable software. Appendix C summarizes the Australian Standards which are highly influenced by the IEEE Software Engineering Standards. Appendix D is the questionnaire format used in gathering information from local Software Engineering firms.

Chapter 2

Questionnaire Analysis

2.1 Content

Collecting questionnaires and analyzing the various answers concerning the different Software Engineering phases is not a straightforward process. While formulating the questionnaire, the main concern was to clarify the questions without wasting too much time.

Being concise meant that not all questions could have check answers and that implied spending more time on answering these specific questions. A compromise was reached and the questionnaire was achieved. There are 41 questions in all, 16 of which have check answers, 16 questions require detailed answers and there are 9 questions where a minimum Yes/No answer is accepted.

The questions are not randomly placed but rather follow a specific path. The questionnaire starts with the overall organization and operations of the firm such as the nature of software developed, the main phases it undergoes and the global planning and scheduling methodologies.

Questions related to the requirements analysis and how they are represented follow. Next comes the design methodology used in various applications with any possible graphical illustrations. Programming techniques, tools and languages are then discussed, followed by testing as in testing nature, methods, tools and documentation.

Maintainability is also portrayed, together with fault localization and tools used. Next, an attempt to figure out complexity analysis methods, if any. Last, but not least, management issues are discussed further through budgeting, various plans, and subjective opinions in assessing software quality and maintainability. Documentation and quality assurance are however the two main elements present all throughout the questionnaire.

2.2 Target Companies

First of all, an introductory letter was prepared through LAU university in order to encourage various firms to be honest and cooperative to obtain accurate results. A copy of this letter was given to every single company visited.

The companies chosen whose names are strictly confidential, were small and medium sized software houses, some of which are shops selling both hardware and software and others are very serious firms with abroad target markets. These reflect the majority of the software companies in Lebanon.

The primary contact is an introductory phone call introducing the purpose of the questionnaire and asking for an appointment. Not all interviews were conducted with the software development manager and not all companies interviewed had a specific manager but rather various people in charge. This minimized the accuracy of some answers.

Nevertheless, Most companies were very helpful and friendly. The questionnaire was filled on the spot either by the interviewer or the person questioned. The average time for every interview was thirty minutes. That is not much considering the time to introduce the topic, read the letter of introduction and understand the questions

requiring extra reflection to answer.

2.3 Detailed Analysis

2.3.1 Organization & Requirements analysis

Most companies interviewed have stock/accounting applications which reflect the market's demands for this type of software. Applications for banks and educational institutions are following up slowly. Systems and government software seem to be restricted to a few companies.

Lebanese firms tend to market one or two applications on hand rather than to invest in new ones. Around forty percent of the companies interviewed regard development phases as the time between interviews, proposal, and delivery after applying certain changes to the application in question, if any. Only one company claimed to follow the waterfall model.

Delivery deadlines are not strict nor are they well defined in some firms. Sometimes it is a standard three month open to change or a 50% increase to what the analyst or programmer decides.

A very interesting point is revealed in setting the price of the software. Apart from some standard listed prices, managers do find pleasure in competing in the market with studied prices which depreciate sometimes according to the time the software circulates. These prices do vary a whole lot according to time, money and effort spent. No well defined formula was given but rather hints. One company has abroad marketing and sales department which handles financial issues.

Planning and scheduling the software's development phases is unfortunately missing

from the agenda of Lebanese software companies. It might be due to the lack in investing in new products, fear of putting too much time and money on planning, rather than producing and selling.

This is an essential lack in local Software Engineering practices. Many do not even realize how important it is and consider it a waste of money. Others have tried to use Microsoft Project as a prospective tool to help them plan, design and maintain their software, but tend to neglect it.

2.3.2 Specification Representation & Design

Seventy percent of the companies offer the client a written document specifying what the software can and will do. This document is not however an obligation to the company nor is it represented in a semi-formal way. It is mostly written in plain English. Very few use graphical representations. The majority rely on demonstrations to intrigue and impress the client.

When asked if the demonstration is thrown away or the final product is built from it, fifty percent regarded it as an obvious question and did not even answer it. All companies use incremental design and never even consider building a prototype and then discarding it.

Modular and top-down designs are design methodologies mostly used locally. Their application is not however very clear. Some companies seemed to check some answers without fully understanding the design methodologies in question. The same goes for the graphical design representation. Out of seven companies using object oriented design, only three referred to Entity Relationship (ER) diagrams.

2.3.3 Implementation

The languages used in implementation vary a lot. Many are trying Foxpro in business applications. 35% of the applications are written in relational data bases. C is also a powerful language and tool appreciated by 25% . Only 15% still use COBOL extensively and one company relies solely on C. Old applications written in COBOL are still maintained, and converting written code is not an easy nor cheap task for some companies who wish to change but are not willing to at a high cost.

Proper documentation is a main concern in programming but is not applied fully except in comments written within programs. Coding is often done immediately on the screen without referring much to design graphs.

2.3.4 Testing

Testing is obviously misunderstood in local software companies. When asked if the best way to test software is to let the customer try for himself, 50% had affirmative answers!

Hardly any testing is done for the specifications documents neither by reviews nor walkthroughs. Errors are located by the majority at the testing phase and not prior to that. This phase is understood by all as testing after or within coding.

Only one company could locate 40% of the errors as early as in the requirements analysis phase. 30% of the persons interviewed regarded testing as a method to prove that there aren't any errors rather than a method to find those errors.

Also, since the majority do not use any complexity measures, fault localization can be a very expensive task at one point in the software's life cycle. Errors are located manually by 85% of the companies, though previous test cases are kept and reused

by 60% of those.

2.3.5 Maintainability & Quality Issues

Two questions requiring detailed answers and personal assessments of both the maintainability and quality of a software, were not easily nor quickly answered. "What do we mean by maintainability?", "How tangible is the quality?" were an instinctive first reaction to these very delicate couple of questions.

Most local companies do not debate such issues internally. Whenever the software works, free of any apparent errors, and whenever the customer is satisfied, that was field proof that the software is of good quality and can survive in the market, i.e. maintainable.

When discussed further with the interviewer, a software was maintainable by some when it worked, free of errors. 60% judged it as durability and modifiability. One company could not find a proper answer and another simply had no estimates locally.

Quality criteria was not much different. Client satisfaction of an operational software was a measure of quality by most. One company stated that having a unique software in the market surely meant that it was of excellent quality. There was one surprising answer by a company which passes Microsoft "Testing and Qualification" test to assure quality of its software locally and abroad.

2.3.6 Documentation

This apparently time consuming activity is an inherent part within every development phase and throughout the software's life cycle. Twelve questions are directly linked to documentation.

Unfortunately, this activity is not effectively practiced locally. Written specifications do not have a formal representation. Design documents are not updated, if they exist (45% do not keep them). 65% do not document their test cases and those who do, hardly keep them after delivery.

Whenever a client reports an error or a change, 30% discard any notes taken after charging him. A few keep them for a period of time. One company does actually use client reports for various statistics as for instance, frequency of calls, problems and their nature within every software, this is not solely for invoicing but also for improving a software. That is an exception.

Various changes are documented on paper mainly to circulate them to all clients. A few keep different software revisions and others add remarks within programs. Again how well updated are all these documents is a debatable issue.

2.3.7 Management Planning

Document plans either design, test or management plans, remain an ambiguous matter. 60% admitted not having any such plans. The honesty of the remaining 40% is subjectively doubted by the interviewer due to other answers regarding documentation. These questions were quickly checked without offering any details concerning them, so they might easily be misunderstood.

Regarding budgeting, 20% do not budget their software. The rest have special and sometimes secret ways, based on manpower and productivity assessments. No specific formula was given, but rather various hints. The same applies to setting price tags to each and every software which is a very flexible activity.

The relationship with the customer is a management concern. Personal contact is a priority in the early stages. Demonstrations are crucial to convince and sell the software. Afterwards, no specific rules are applied. Deadlines are set without any actual commitments due to the friendly relationship with the customer. 10% do not even set a deadline.

Upon delivery it is always the customer who causes delays, by "not knowing what he wants" or "wanting too much". One company realized that at times, rush of clients may affect the testing period, which can cause delays in delivering the software.

Twenty per cent of the software firms do not apply product/acceptance testing as a management policy. 40% ensure acceptance right upon signature or after seeing the demonstration. The remaining 40% give a certain testing period after delivery.

2.4 Summary

Table 2.1 gives a general overview of the specifications and gaps of the majority of Software Engineering companies in Lebanon. It does not in any way, portray all local practices, but merely gives a summary of those practices as a whole:

SPECIFICATIONS	GAPS
Management	
- Business Oriented Applications - Informal Contact With Client - Emphasis On Management Aspects Of Software Engineering - Extensive Budgeting Methods - Self Assurance In Product Capabilities	- Rely On Individual Assessment - Lack Of Scheduling - Unable To Define Maintainability & Quality - No Complexity Metrics - Frequent Delays In Delivery For Various Reasons
Methodology	
- Modular Design - Testing is Defined As Testing The Code - Considerable Perfective Maintenance Activities - Special Interest In User Manual	- Lack Of CASE Tools - No Acceptance Testing - Informal Specifications Requirements - No Specific Plans Used Or Updated

Table 2.1: Specifications And Gaps Of Software Engineering Companies In Lebanon

2.5 Personal Notes

Software Engineering practices in Lebanon are applied based on very subjective assessments and self-made rules for the majority. Today, software firms are operating successfully locally and some are trying to extend abroad.

Every company learns, as it operates in the market, how to create a certain software, sell it , and prolong its life cycle as much as possible. Details in methodologies, documentation and planning issues are not the main concern, but rather the software as a whole.

By next year, new companies may emerge with new enthusiastic methodologies. Now, there are some very serious companies in Lebanon. They do however, to this

day, form an exception. Software Engineering practices are a burden that can delay the product from emerging and increase its cost. Who needs tools, when everything is done manually so far? The question is : "Why invest in such detailed analysis, design and all that follows?"

We can not jump a big leap into practicing a set of international standards such as the ISO series and IEEE collection on Software Engineering. But, we can certainly try to meet Lebanese engineering practices half way and find a compromising alternative to increase software quality. This is what this research aims at accomplishing next.

Chapter 3

Suggestions For Lebanese Requirements Analysis Standards

In this chapter, we present suggestions for Lebanese Standards for the software requirements analysis phase. The suggested standards consist of a mixture of international standards namely of ISO [ISO 9000-3 1991], IEEE [IEEE Std 830 1984] and British Standards [British Standard 1986] . They are highly influenced by the ISO and the european perspectives of the user requirements phase[STARTS 1988]. The software requirements phase follows closely the British Standard which subdivides the requirements into functional and non-functional requirements. The IEEE requirements specifications include quality requirements such as reliability, maintainability and usability. In this standard, these are part of the non-functional requirements.

The software requirements document's outline is influenced by the IEEE and British standards. In other words, it adopts the european acceptance criteria, and following the IEEE standards, it contains only the technical requirements, whereas cost and schedule information is not tackled, (It could be part of the project management plan).

All the methods divide their requirements analysis phase into a set of tasks or steps to facilitate its use and to enable better management and control. The exact number of tasks varies considerably between methods. There is a tradeoff between productivity and methodology. IEEE standards emphasize productivity whereas ISO and British Standards emphasize methodology. Our suggestions for Lebanese standards are more influenced by the methodology emphasis of the european standards.

The objectives of these standards are:

1) Aid the customers to accurately state their needs
2) Aid the software engineer to understand the user's needs.
3) Provide an organized and detailed outline for a standard software requirement analysis specification and documentation.

This standard is not however intended to specify or discourage the use of any particular software development method or tool.

Table 3.1 presents the suggested standard outline:

1. Scope And Applicability

2. Analyzing Existing Operational System

3. Determining User's Needs
- 3.1 Proceeding With Interviews
- 3.2 Analyzing Existing Operations

4. Producing User Requirements Document (URD)
- 4.1 Reviewing Document
- 4.2 Obtaining Approval From Both Parties

5. Determining Software Requirements
- 5.1 Determining Functional Requirements
  - 5.1.1 Input Requirements
  - 5.1.2 Performance Requirements
  - 5.1.3 Output Requirements
- 5.2 Determining Non-Functional Requirements
  - 5.2.1 Reliability Requirements
  - 5.2.2 Interface Requirements
  - 5.2.3 Design Constraints

6. Producing Software Requirements Document (SRD)
- 6.1 Outline
- 6.2 Review Criteria
- 6.3 Approval

Table 3.1 Requirements Analysis Standard Outline

3.1 Scope And Applicability

The requirements phase is the problem definition stage in the life cycle. The principal aim is to refine an idea about the task to be performed by the software into some form of requirements analysis. The user and the developer have different views of a software. The user views the software from a business or operational viewpoint, while the developer seeks the technical point of view. Therefore, the software requirements phase tends to reconcile these different views by expressing the user requirements in terms more meaningful in a software context leading to software design. This is done by analyzing the statement of user requirements and producing a set of software requirements which are clear, explaining what the software is expected to do.

This standard focuses on the activities performed by both the customer and user resulting into the required product for the requirements analysis phase which is the software requirement specification document. It shall be applied and tailored according to the software demands in question. That is, certain activities and documents may have to be expanded or diminished according to the needs of every software development project.

3.2 Analyzing Existing Operational system

This is intended to determine how the existing system is operating in order to have a better understanding of the user's need and later determine any constraints to the suggested new system. This shall include determining all hardware components present, set of inputs and outputs produced, security levels of operation and all characteristics of the operational system.

3.3 Determining User's Needs

3.3.1 Proceeding with interviews

The user's requirements need to be grasped by interviews or surveys or even by building a prototype to illustrate and clarify any misunderstandings. Checklists are a good suggestion in order to avoid fuzzy and unclear requirements. In a step to clarify requirements, the analyst should:

- Express constraints with equations and logical expressions
- Illustrate when necessary
- Use pseudo code to express computational algorithms

3.3.2 Analyzing Existing Operations

The analyst shall use graphical aids such as data flow diagrams in analyzing customers requests. The analyst should analyze requested features and state any ambiguous constraints. It is important to distinguish between those features which are functions of the required system and those which are quality.

3.4 Producing User Requirements Document

This document gives a general description of what the user expects the software to do and defines all constraints that the user wishes to impose on any solution. It can have the following format [British Standards, 1986]:

Part 1: Summary of User Requirement

a) Introduction

b) Environment

c) User Interface

d) Essential Features

e) Desirable Features

f) Time and Cost Limits

Part 2: Detailed Description Of User Requirements

a) Information to be Provided to the Required System

b) Manipulation of the Information Provided

c) Resultant Actions or Information

d) Acceptance Criteria

e) Performance Monitoring

In part 1, the user should include the main purpose and scope of the required system and state clearly any peculiarities in the environment, particularly the physical environment in which the required system will operate. The category of requirement will vary in importance according to the type of system under consideration. Considerable work may be needed to define procedures for data input and system control as well as for data presentation and archiving within the user interface. This may include the definition of a command language and interactive dialogue. A distinction between essential and desirable features should be made. Finally, the prospective user should determine when the required system needs to be available to him and how much he is prepared to pay to obtain and operate the required system.

In part 2, the specification of user requirements should be a fully detailed description of the information to be given to the required system, what the system needs to do with this information for both valid and invalid cases and what the resultant actions of information need to be. The completeness of this detailed statement will guide the supplier to determining whether the solution will be an off-the-shelf package or not.

3.4.1 Reviewing User Requirement Document

Participation should include the users, the operational personnel, the developers (hardware and

software engineers) and the manager concerned.

3.4.2 Obtaining Approval From Both Parties

The User Requirements Document is the input document to the software requirements analysis. It should be approved by both the customer and software development manager.

3.5 Determining Software Requirements

The major output of this phase is the Software Requirements Document. The requirements definition process includes [Brackett 1990]:

- Requirements Identification
- Requirements Representation
- Requirements Communication
- Preparation For Validation Of Software Requirements

3.5.1 Determining Functional Requirements

Functionality can be considered as having three components [STARTS 1988]:

1) Transformations are carried out on inputs to produce outputs.
2) Dynamic requirements are concerned with requirement for parallelism.
3) Exception handling is concerned with those aspects of the requirements outside the mainstream of normal system behavior.

3.5.1.1 Input Requirements

A detailed description of all data inputs is required. The analyst should also define all of the operations to be performed on the input data and intermediate parameters to obtain the output.

3.5.1.2 Performance Requirements

The analyst should specify both the static and the dynamic numerical requirements placed on the software or on human interaction as a whole. Static requirements can include the number

of terminals and users to be supported, the number of records and files to be handled and the sizes of tables and files. Dynamic requirements may include the number of transactions and amount of data processed within certain time periods.

3.5.1.3 Output Requirements

A detailed description of all data outputs is required. Input/Output sequences will be needed when a system is required to remember its behavior so that it can respond to an input based on that input and past behavior, that is, behave like a finite state machine.

3.5.2 Determining Non Functional Requirements

Some constraints are essentially non functional but are an important aspect of software requirements. They must never be overlooked. The following are examples of non functional requirements:

3.5.2.1 Reliability Requirements

These are the system availability, the system fault recovery and exception handling.

3.5.2.2 Interface Requirements

The analyst should specify adequately the nature of an interface (human, hardware or software) without going into its full detail. This interface will be part of non functional requirements.

3.5.2.3 Design Constraints

Three design types are considered:

1) ***Software Design Constraints*** include [STARTS 1988]:

- Standards and languages including design standards, implementation language, coding standards, data naming, etc.

- Software interfaces which are described above
- The use of any specific packages which might be required
- Communication standards and interfaces
- The purchaser may require the system to use a specific database system with associated report generators
- Details need to be given if the system is to be implemented using a specific operating system or executive
- The specification should identify constraints on program size, etc... indicating margins to be allowed for expansion of data handling capabilities

2) ***Hardware Design Constraints*** include:

- Requirements to use specific types of hardware; details of the working environment, hardware reliability requirements, mechanical and physical constraints, also need to be specified
- A statement of required engineering standards
- Hardware interfaces which are described above
- Requirements for spare capacity or spare performance

3) ***User Design Constraints*** include:

- The features of the operator/user need to be identified together with details of the working environment and any special features that may be required

3.6 Producing Software Requirements Document

The Software Requirements Document (SRD) is a deliverable product that is considered the main input document in the design phase.

3.6.1 Outline

The Software Requirements Document shall have the following outline as seen in Table 3.2. The introduction of the Software Requirements Document shall contain the identification, overview and scope of the document along with all documents controlling or referenced by the User Requirements Document. Three priority levels are assigned to the criteria by which the requirements are assigned; critical, important and desirable.

The program set description specifies the overall function of the program set. It includes a functional overview without stating the requirements. The functional and non functional requirements should be elaborated to the level of detail needed by the assigned software design phase staff. Finally, the Software Requirements Document should specify the criteria by which the program set is to be judged acceptable and the methods by which each requirement or collection of requirements is to be qualified. The glossary and acronyms shall be part of the appendices.

1. **Introduction**
    - 1.1 Identification
    - 1.2 Overview
    - 1.3 Scope
    - 1.4 Notation Used
    - 1.5 Controlling and Applicable Documents
    - 1.6 Priorities
2. **Program Set Description**
3. **Functional Requirements**
4. **Non Functional Requirements**
5. **Acceptance Criteria**
6. **Appendices**
    - 6.1 Glossary
    - 6.2 Acronyms

Table 3.2 Software Requirements Document Outline

3.6.2 Review Criteria

The following criteria shall be used in reviewing the Software Requirements Document [Fouser 1988]:

a) The software that the requirements specify can be demonstrated, tested, analyzed, or inspected to show that it satisfies the requirements.

b) The individual requirements, where possible, have discrete, unambiguous, testable values. Accuracy, precision, type, rate, units, frequency, and volume of inputs and outputs have been specified.

c) The customers have agreed to the acceptance test criteria.

d) The requirements are consistent and complete. The specified objectives of one requirement do not conflict with those of another.

e) Design constraints dictating how the program set will be built are uniquely and precisely identified.

f) All inputs to a required function are necessary and sufficient for that function. All inputs are available from external interfaces or as outputs from other required function.

g) All outputs from a function are used by another function or transferred across an interface to hardware or operational procedures.

h) The requirements are stated in an implementation free manner (i.e., not stating how to perform the function).

i) All the requirements are agreed upon by users, operators, the system engineer and all managers concerned.

3.6.3 Approval

This section represents approval of the entire software requirements specification by all parties. Each individual's signature implies that they have read and understood the Software Requirements Specification and signifies their intent of mutual cooperation of the described Software project. Also, it will be understood that the software requirements specification is subject to change, renegotiation and modification.

3.7 Concluding Remarks

The suggestions for local standards are meant to be concise, serving to express the user's needs accurately through the User Requirements Document and to produce a solid Software Requirements Document, review it thoroughly and attain mutual agreement to move into the design phase.

Local Software Engineering practices have a gap in the customer/designer relationship. If the IEEE standard was to be applied literally, user requirements wouldn't have been emphasized and no clear acceptance criteria would have been stated to insure mutual understanding early is the product's life cycle. The ISO and British Standards are more closely related to local needs. Still, their application as a whole would have stressed applying a list of requirements to be met by a quality system as described in a framework dealing with management responsibility, internal quality system audits and correction actions, within a quality plan.

The suggested local trend is to abide by the given standards in order to concentrate on the requirements analysis activity by helping the customer state his needs and translating these needs into a solid input document to the design phase. This does not however mean that quality is neglected in any way. A specific review criteria is suggested to increase the quality of the document presented. The next chapter will also highlight some of the quality issues that need to be stressed in the locally suggested Software Requirements Analysis Standards.

Chapter 4

Guidelines For Implementing Local Requirements Analysis standards

4.1 Introduction

Though no specific techniques are specified in the suggested standards, the following sections stress quality and management issues along with review and inspection techniques that can be followed in checking the requirements. It is intended to aid the software engineer in the requirements analysis standards implementation and give him an insight to a few of many ways to ensure desired features of the requirements such as correctness, reliability and traceability...

4.2 Quality Considerations

It is very important to realize how tightly effective requirements analysis is linked to quality. The international standard quality vocabulary [ANSI/IEEE 983-1986] defines quality as: "The totality of features and characteristics of a product or service that bear on its ability to meet stated or implied needs". If seen from the customer's angle, the above definition will mean customer satisfaction. That is, when customer's needs are met, they are satisfied and when satisfied, the customer regards the software of good quality.

Quality is then determined by:

* How fully the need is understood and captured by a requirements definition.
* How well the definition is transformed into a software product.

* How well the product is supported.

All starts with the requirements that capture the customer needs and , if well translated, can lead into a software that accurately satisfies those needs. Therefore, quality should be the aim of every local software firm in the sense that it leads to client utmost satisfaction.

4.3 Resolving Fuzzy Requirements

Software requirements tend to be fuzzy especially if expressed in plain english. The problems are numerous and need specific methodologies to minimize them. Table 4.1 clarifies some of the problems encountered and their possible solution. [Gray and Rao 1993]

Problem Category	**Possible Methods**
The Essence	
How to match language capabilities with interface needs while reducing the scope for interpretation	Partitioning of requirements, multiple view approach, control the degree of misinterpretation
How to ensure that there are no 'gaps' in requirements	Improved judgement, prototyping, multiple views
How to ensure that the requirements describe the right system	Improved modeling via multiple techniques, prototyping
How to cope with size and complexity of requirements	Improved methods/techniques
The Accidents	
How to ensure correct use of requirements expression language	Better training in formalism
How to ensure consistency in requirements specifications	Use of review technique
How to ensure modifiability of requirements	Use of Modularity (in structured methods) intentional incremental delivery strategy

Table 4.1: Problems In Requirements Analysis & Suggested Solutions

4.4 Review Techniques

It is very important to perform regular reviews to ensure the adequacy of the requirements stated in the software requirements specifications. Software Requirements Review (SRR) is an evaluation of the Software Requirements Specifications (SRS) to ensure adequacy, technical feasibility and completeness. The review should be held with user participation to examine the SRS and verify that is unambiguous, verifiable, consistent, modifiable and traceable. A checklist is recommended including the following issues:

1) Traceability and completeness of the requirements from the next higher level specification (such as a system specification or user requirements specification)
2) Adequacy of rationale for any derived requirements
3) Compatibility of external hardware and software interfaces
4) Adequacy of requirements for the man-machine interface
5) Availability of constraints and tables for calculations
6) Testability of the functional requirements
7) Adequacy and completeness of the verification and acceptance requirements
8) Conformance to requirements specification standards
9) Adequacy and feasibility of performance requirements
10) Adequacy of consideration for human factors

Table 4.2 also provides sample questions in a checklist and their respective feature assessment. [Davis 1990]

Checklist Question	Feature Assessed
Are all requirements included?	Completeness
Is each requirement Correct?	Correctness
Is each requirement precise, clear and unambiguous?	Precision
Does any requirement conflict with any other?	Consistency
Is each requirement pertinent to the problem?	Relevance
During testing, will it be possible to check that every requirement is satisfied?	Testability
Does each requirement relate to a problem?	Traceability
Can each requirement be implemented, given the techniques, resources and organizations available to us?	Feasibility
Have any requirements been obscured by premature design detail?	Problem based
Can any changes to requirements be controlled, with minimal effect of other requirements?	Manageable

Table 4.2: Checklist Questions in Requirements Analysis

Making an extra effort to check the software requirements document will certainly reveal if any of the features mentioned above are missing, thus jeopardizing the software's quality early in the products life cycle.

4.5 Use Of Phased Inspections

While walkthroughs and formal reviews are generally biased toward error detecting, inspections are often used to establish additional properties such as portability and adherence

to standards. Error detection is important, but correctness is not the only desirable characteristic of software products. Maintainability and reusability are examples of other characteristics with which inspection might be concerned.

Knight and Meyers have proposed an improved inspection technique which can be very useful to local software designers. [Knight and Meyers 1993] This technique is designed to permit the inspection process to be consistently rigorous, tailorable and efficient in its use of resources. Phased inspections examine the work product in a series of small inspections termed Phases, each of which is designed to ascertain whether the work product possesses some desirable property.

Phases are designed so that the work product's compliance with associated properties is ensured, at least informally, with a high degree of confidence. To achieve this, two phase types are defined, Single Inspector and Multiple Inspector.

A single inspector phase is a rigidly formatted process driven by a list of unambiguous checks. For each check, the product either complies or does not comply. Single inspector phases might be used to check compliance with single formatting properties in design documents or compliance with simple programming practices in source code.

A multiple inspector phase is designed to check properties such as completeness or correctness issues for requirements properties such as completeness or correctness issues for requirements or functional correctness concerns for implementations. In a multiple inspector phase, several inspectors first examine the product independently in a highly structured way and then meet to compare findings. The staff used in the various phases can be chose so that their qualifications meet the needs of the phase. This helps address the goal of making efficient use of human resources.

These individual inspections are driven by checklists that are in part application specific and in part domain-specific. The goal of the checklists is to ensure that the inspectors focus on the work product in a systematic way and with complete coverage. Being unable to answer a question is an important outcome of a multiple-inspector phase since it indicates that the work product was not sufficiently documented or was not clearly written.

4.6 Management Commitment

The two main documents produced in the software requirements specification suggested earlier are the User Requirements Document and Software Requirements Document. The transition for local software engineers from informal interviews, non structured documents and casual deadlines to standardization and acceptance criteria will require management commitment to the standard presented.

Planning a Software Engineering project consists of all the management activities that lead to the selection of future courses of action for the project and the program for carrying the actions out. Local management involvement is extremely important to motivate software engineers into applying standard methodologies and deliver software product on time, within cost and to the customer's satisfaction.

Chapter 5

Suggestions For Lebanese Software Testing Standards

The following chapter presents suggestions for Lebanese Software Testing Standards. the main test document associated with software testing procedures is the test plan. It tightly follows the IEEE Standard for test documentation [ANSI/IEEE Std 829 1983], yet it includes quality issues raised by the ISO9000 Standards [Schmauch 1994], where the test plan itself describes the testing needed during the production of a quality product and is owned by the project manager.

In the ISO9000, the test plan incorporates much of the quality plan, verification plan and test plan. It is a comprehensive document which addresses a number of topics related to testing and quality activities. Our suggested standard follows the basic IEEE standard in the test plan procedure yet it refers to some quality issues and self assessments suggested by the International Standard Organization. The subsections of the standard are expanded to give a clearer view of the Test Plan demands.

The objectives of the Lebanese software testing standard are:

1) Aid the software engineer in planning the test process early in the software development life cycle.
2) Provide a test plan applicable to both initial testing and testing of subsequent software releases.
3) Provide an insight to quality issues related to testing management.

The standard does not however call for specific testing methodologies, approaches, techniques or tools and does not specify the documentation of their use.

Table 5.1 presents the suggested standard outline:

1. Scope And Applicability
2. Testing Considerations
3. Planning The Test Process
3.1 Analyzing The Requirements & Establishing Verification Methods
3.2 Establishing a Verification Sequence & Identifying Large Software Units
3.3 Identifying Supporting Material
4. Producing The Test Plan
5. Quality Assurance Checklist

Table 5.1 Test Documentation Standard Outline

5.1 Scope And Applicability

This standard is meant to stress the importance of test planning through producing the test plan document. It also exposes the local software engineer to the effects of proper testing on quality. In Lebanon, testing is still misinterpreted by software houses who refer to testing as in testing the code. This standard insures testing early in the software's life cycle. Though the test plan in the locally suggested standard follows the IEEE Standard, it does not apply it fully. Other deliverables in the standard are not included.

The methodology aspect is emphasized stressing solely the planning process and the test plan without attacking the test deliverable such as the test design specifications, test case specifications and test logs. The local software engineer shall realize the importance of test

planning and understand the solid connection of testing and Software Quality through the application of this standard.

5.2 Testing Considerations

Test documentation is a necesary and useful tool for managing and maintaining the software. The documents produced by the software engineers answer the questions:

* What to test?
* When to test?
* How to analyze test results?
* How can testing be improved?

Planning for testing is an activity that occurs throughout the software development life cycle. The sooner software engineers are involved, the more likely that the final product will achieve its quality objectives.

5.3 Planning The Test Process

The term planning is used here to mean the process of thinking about the *what* and the *how* of actually verifying that the software performs in accordance with imposed requirements. Before actually writing the Test Plan, the software engineer should do the following:

5.3.1 Analyzing the Requirements & Establishing Verification Methods

The first step in the test planning process is the identification of the specific requirements to be verified. A verification method must be associated with each requirement to be verified or validated either by inspections, analysis or even demonstration [DOD-Std 2167A]. Once

a complete list of requirements has been developed, it is necessary to identify a specific piece of software with each of the requirements to be verified.

5.3.2 Establishing a Verification Sequence & Identifying Large Software Units

The second step in the development of a workable test plan is the selection of the desired sequence for the verification of requirements, and determining the level of testing that is appropriate for each requirement. A sequence of verifcation steps always flows more smoothly and is more convincing when each verified requirement establishes a foundation for the ones that follow. The sequence of test levels must be from that requiring the smallest units to that requiring the largest ones.

5.3.3 Identifying Supporting Material

At this point in the test planning process, a candidate sequence of events has been developed. Still, software cannot be successfully tested unless the test organization has in its possession the necessary elements. Special support software such as test drivers, special analysis programs or debuggers are often required to run software verification. Checking the computer configuration is needed for testing consistency. Hardware too needs to be verified. Several categories of personnel may be needed: Test engineers to actually execute the tests, software engineers to help in the analysis of problems and hardware engineers to support, maintain and analyze hardware needs during software testing.

5.4 Producing The Test Plan

A test plan is given in Table 5.2:

1. Test-plan Identifier
2. Introduction
3. Test Items
4. Features to be tested
5. Features not to be tested
6. Testing phases and Approaches
7. Item pass/fail criteria
8. Suspension criteria and resumption requirements
9. Environmental Needs
10. Responsibilities
11. Staffing and training needs
12. Schedule
13. Risks and Contingencies
14. Approval

Table 5.2 Test Plan Outline

5.4.1 Test-plan Identifier

It specifies the unique identifier assigned to this test plan. On some projects, this can mean a simple title. On others, it will specify which of several tests the plan addresses or a specific subsystem of the complete product which may limit the scope of a test to a particualr subset of tests.

5.4.2 Introduction

It summarizes the software items and software features to be tested. The author of this section should consider briefly describing the product's purpose, its place in the grander scheme of things, the method of software development, and the need that caused the product to be developed.

5.4.3 Test Items

It identifies the test items including their version/revision level. This level will be known at the time of the initial release of the plan. If, on the other hand, the test manager and the approving organization wish that the internal version numbers for all the components

comprising the test items be stated, the initial release of the Test Plan will necessarily contain a void (usually filled by the letters "TBD" for "To Be Determined"). References to the following item documentations should be supplied if they exist:

- Requirements specification
- Design specification
- Users guide
- Operations guide
- Installation guide

5.4.4 Features to be Tested

It identifies all software features and combinations of software features to be tested [IEEE 829-1983] and defines software features as a distinguishing characteristic of a software item (for example, performance, portability, or functionality). The features defined must have definitions that are meaningful to all those interested in the test particularly the customer or buyer of the system.

5.4.5 Features Not to be Tested

It identifies all features and significant combinations of features which will not be tested and the reasons. The list of features will have a similar appearance as the previous section. In this list, it is a good idea to state why the features are omitted and when they will be tested.

5.4.6 Testing Phases and Approaches

It describes the overall approach to testing and specifies the major activities, techniques, and tools which are used to test the designated groups of features. It also identifies the techniques which will be used to judge the comprehensiveness of the testing effort and those to be used to trace the requirements. It is important that the readers of the Test Plan understand exactly what is being tested and how thouroughly.

The key point of the distinction between several testing tasks or phases is that the tasks are based on different levels of abstraction, i.e. each testing step emphasizes a different aspect. Each testing phase concentrates on testing those aspects which are emphasized in the corresponding construction document. An example is shown in Table 5.3 [Mullerburg 1990]. Different approaches can be used for testing using methods such as path, transaction flow, logic based or domain testing, etc., depending on the task to be performed. [Beizer 1990]

Task	Objects	Aims	Inputs	Test Strategies
Module Testing	Algorithms Variables	Correctness	1) Module Spec. 2) Source Code 3) Program	1) BBT 2) GBT
Integration Testing	Module Relations and Interfaces	Correctness	1) Architectural Design 2) Source Code 3) Tested Modules	1) BBT 2) GBT
System Testing	External Functions External Data	Functionality Reliability Performance ...	1) System Def. 2) Tested System	1) BBT 2) StT
Acceptance Testing	External Functions External Data Documentation	Functionality Reliability Performance	1) Req. Def. 2) User Doc. 3) System	1) BBT 2) StT

Table 5.3 An Example of Emphasized Aspects of Testing Phases
BBT refers to Black Box Testing, GBT refers to Glass Box Testing and StT refers to Statistical Testing.

5.4.7 Item Pass/Fail

It specifies the criteria used to determine whether each test item has passed or failed testing. The tester might consider defining several categories of "failures", including such things as catastrophic failure, serious failure, failure and inconvenience. The pass/fail criteria stated in

the Test Plan would then state the maximum number of each type of failure which are premitted while still considering the test a success.

5.4.8 Suspension Criteria and Resumption Requirements

It specifies the criteria used to suspend all or a portion of the testing activity on the test items associated with this plan. The tester must consider two situations: normal suspension of the test, as at the end of a day; and abnormal suspension, as would be necessary because of a hardware malfunction. If there is any possibility that a test could be suspended, the tester should plan to mark a number of possible termination points in the test procedure document. The tester shall take into consideration that when a test is resumed, a great deal of the system's data base and operating context may require restoration. This needs to be considered when selecting the suspension points.

5.4.9 Environmental Needs

It specifies both the necessary and desired propreties of the test environment. Physical characteristics include hardware, communications, the mode of usage and any other software of supplies needed to support the test. Levels of security should also be specified for the test facilities.

5.4.10 Responsibilities

It identifies the groups responsible for managing, designing, preparing, executing, witnessing, checking, and resolving. In addition, it identifies the groups responsible for providing the test items identified in 5.4.4.

5.4.11 Staffing and Training Needs

It specifies test staffing needs by skill level and identifies training options for providing necessary skills. Normally, each task is analyzed and the amount of effort determined. The task duration and number of people required is then calculated by the test manager.

5.4.12 Schedule

It includes test milestones identified in the software project schedule as well as all item transmittal events. This schedule should be consistent with the software development schedule presented in the Software Development Plan.

5.4.13 Risks and Contingencies

It identifies the high-risk assumptions of the test plan and specifies contingency plans for each. It is important to list the things most likely to go wrong, and the things that will have the biggest impact if they go wrong, regardless of the probability.

5.4.14 Approvals

It specifies the names and titles of all persons who must approve this plan. It also provides spaces for the signatures and dates.

5.5 Quality Assurance Checklist

In order to meet the ISO9000 quality system, the following checklist regarding testing is used by the International Standards Organization. The software engineer can achieve a self assessment of his testing activities. The checklist in Table 5.4 [Schmauch 1994] is very useful to the local standard and will thus be integrated in the suggested standard. The answer of the checklist should normally be "Yes" to all questions.

* Have you identified and documented the inspection and testing required for the project?
* Can you demonstrate that the testing that you identified will adequately validate the product?
* Do you have procedures for the required testing?
* Do you maintain records of test activities?
* Can you show evidence that the required testing has been done?
* Do you verify that inspection and test tools fit the purpose and are maintained in an agreed manner?
* Are test functions appropriately specified and controlled?
* Do procedures exist to ensure that test planning, execution, and recording is carried out appropriately on individual software items, integration of software and hardware system components, and the whole system?
* Is fault information recorded at all levels of testing?
* Are replication processes, if used, subject to inspection and test activities at appropriate stages?
* Are specific test responsibilities of the customer identified and documented?
* If a number of versions of a product have been released, are test records for each version retained?

Table 5.4 Self-assessment Checklist

Chapter 6

Conclusions

To this day, Software Engineering practices in Lebanon do not use any standard methods. The software development phases are not clearly subdivided in local Software Engineering firms. No actual plans are used and updated. Methodologies used in requirements analysis fail to present clear and proper documentation, though sincere efforts are made to understand the customer's needs through friendly contacts. Testing is misinterpreted. It is considered a method used only to test the code and eliminate encountered errors. Also, no measures are taken to assess the complexity and quality of the software developed.

This thesis has presented an objective assessment of Software Engineering practices in Lebanon. It has also provided suggestions for requirements analysis and software testing standards that are indispensable for local Software Engineering firms in order to attain an international level in the software production industry. Using these standards, the local software engineer should be able to refine the user requirements, write a Software Requirements Document and review it properly to minimize errors early in the software's life cycle. He should also be able to properly plan the test process and write a test plan for quality assurance.

Software Engineering firms in Lebanon should realize the importance of standardization. The suggestions presented in this thesis can evolve to become national Software Engineering Standards.

Appendix A

International Standards Organization (ISO)

A.1 Overview

ISO9000 is a series of quality systems standards associated guidance material first published in 1987 by the international standards organization (ISO). Appropriate standards from the series are selected to meet the needs of different industries. For the software industry, the appropriate standards are the following:

A.1.1 ISO9001

This standard applies to all products and services where meeting technical requirements involves design. It is used as the prime standard for assessment, supported by the ISO9000-3 and ISO9004-2 guidelines where appropriated.

It concisely lists requirements to be met by a quality system for use when conformance to specified requirements is to be assured during several stages which may include design/development, production, installation and servicing.

A.1.2 ISO9004-2

This standard gives guidance for establishing and implementing a quality system in an organization which provides services to customers, with or without a degree of product content. That is it is applicable to software organizations which support the products they supply. It identifies a number of quality system principles which must be built into a suitable quality system as shown in Table A.1:

Management Policies • Follows quality policy • Ensures defined responsibilities • Plans to provide sufficient resources and achieve quality objectives
Quality Assurance • Emphasize preventive action without sacrificing the ability to respond to and correct failures as they arise • Effective interaction between customers and the service organization's personnel

Table A.1: Quality System Principles

A.2 ISO9000-3

The ISO9000-3 sets out guidelines to facilitate the application of ISO9001 to organization developing, supplying and maintaining software. It splits the subject matter into three broad categories which need to be addressed in establishing a quality system:

1- *The overall framework:* Deals with questions of management responsibility, the quality system, internal quality system audits and corrective action.

2- *The life cycle activities:* Deals with all aspects of the life-cycle. In particular, the standard draws attention to contract reviews, development and quality planning, testing and validation, and maintenance procedures.

3- *The supporting activities:* These activities are not phase dependent. They deal with configuration management, document control, measurements and tools.

The following section will describe in detail the ISO9000-3 preparing for quality management and quality assurance and providing guidelines for the application of ISO9001 to the development, supply and maintenance of software.

A.2.1 Framework

A.2.1.1 Management Responsibility

As a quality policy, the supplier's management shall define and document its policy and objectives for quality and its commitment to it. The organization should give freedom and authority for personnel to identify and record any product quality problems, initiate and provide solutions through designated channels.

- Verification activities should include inspection, test and monitoring of the design, production, installation and servicing processes and/or product.

- The quality system adopted to satisfy the requirements of ISO9000 shall be reviewed at appropriate intervals by the supplier's management to ensure its continuing suitability and effectiveness. Records of such reviews shall be maintained.

- The purchaser should assign a representative with responsibility for dealing with the supplier on contractual matters such as approving the supplier's proposals, defining acceptance criteria and procedures.

- Regular joint reviews involving the supplier and purchaser should also be scheduled to cover verification and acceptance test results that need to be agreed and documented.

A.2.1.2 Quality System

The supplier should establish and maintain a documented quality system. This quality system should be an integrated process throughout the entire life cycle. All the quality system elements, requirements and provisions should be clearly documented in a systematic and orderly manner. A quality plan is needed to implement quality activities for each software development on the basis of the quality system.

A.2.1.3 Corrective Action

The supplier shall establish, document and maintain procedures to do the following:

- Investigate the cause of non conforming product and the corrective action needed to prevent recurrence.
- Analyze all processes and eliminate potential causes of non conforming product.
- Initiate preventive actions to deal with problems to a level corresponding to the risks encountered.
- Implement and record changes in procedures resulting from corrective action

A.2.2 Life Cycle Activities

A software development project should be organized according to a life-cycle model. Quality related activities should be planned and implemented with respect to the nature of the life-cycle model used.

A.2.2.1 Contract Review & Quality Items

Each contract should be reviewed by the supplier to ensure that the scope of the contract and requirements are defined and documented. Any possible contingencies or risks should be identified and requirements differing from these in the tender should be resolved.

The supplier is responsible to meet contractual requirements and so is the purchaser. Items such as acceptance criteria, handling of problems after acceptance, facilities tools and software items provided by the purchaser and procedures to be used should be included in the contract.

A.2.2.2 Requirements Specifications

To have a good start, the supplier should have a complete unambiguous set of functional requirements which should include all aspects necessary to satisfy the purchaser's need. The purchaser's requirements specification should be subject to documentation control and configuration management as part of the development documentation.

A.2.2.3 Development Planning

The development plan should include the definition of the project and its objectives, the organization of the project resources, including the team structure, a project schedule identifying the tasks to be performed and the identification of related plans.

The development plans should also include the development phases with required inputs and outputs for each phase. The supplier should draw up a plan for verification of all development phases outputs at the end of each phase. The verification results and any further actions required to ensure that the specified requirements are met should be recorded and checked when the actions are completed.

A.2.2.4 Quality Planning

The quality plan should be formally reviewed and agreed by all organizations concerned in its implementation. This could be an independent document or a part of another document. The quality plan contains quality objectives expressed in measurable terms whenever possible. It identifies the types of test, verification and validation activities to be carried out with their detailed planning including schedules,

resources and approval authorities.

A.2.2.5 Design & Implementation

Due to the complexity of software products, it is imperative that these activities be carried out in a disciplined manner, in order to produce according to specification. Design rules and internal interface definitions should be examined. A systematic design methodology appropriate to the type of software product being developed, should be used.

Testing and maintenance should be taken into consideration in the design approach. In each implementation activity, rules such as programming rules, languages, consistent naming conventions, coding and adequate commentary rules should be specified and observed.

A.2.2.6 Testing & Validation

There are different approaches to testing and integration. The test plan could also be an independent document. In test planning, considerations should be given to test cases, test data and expected results, types of test to be performed, test environment, tools and user documentation.

Special attention should be paid to some aspects of testing. Any discovered problems and their possible impacts to any other parts of software should be noted. Areas impacted by any modifications should be identified and retested. Test adequacy and relevancy should be evaluated.

A.2.2.7 Acceptance & Delivery

When the supplier is ready to deliver the validated product, the purchaser should judge whether or not the product is acceptable according to previously agreed criteria and in a manner specified in the contract. Before carrying out acceptance activities, the

supplier should assist the purchaser to identify time schedules, procedures for evaluation and acceptance criteria.

Upon delivery, roles, responsibilities and obligations of the supplier and purchaser should be clearly established, taking into account the schedule, including out-of-normal working hours, the availability of skilled personnel, the availability and access to the purchaser's system and equipment and a formal procedure for approval of each installation upon completion.

A.2.2.8 Maintenance

Maintenance activities for software products are typically classified into problem resolution (Corrective), interface modification (Adaptive) and functional expansion (Perfective). Programs, specifications ,data structures should be maintained. Maintenance should be carried out and managed in accordance with a maintenance plan containing the scope of maintenance, maintenance activities, records and reports.

Rules for the submission of maintenance reports should be established and agreed upon by the supplier and purchaser. The maintenance records should include the list of requests for assistance or problem reports that have been received and the current status of each, with priorities assigned to the corrective actions.

The record of the maintenance activities may be utilized for evaluation and enhancement of the software product and for improvement of the quality system.

A.2.3 Supporting Activities

A.2.3.1 Configuration Management

The configuration management system should identify uniquely the versions of each software item, control simultaneous updating of a given software item by more that one person and provide coordination of the updating of multiple products in one or

more locations as required.

Procedures should be applied to ensure that functions and technical specifications can be identified for each version of a software item, and that all development tools which affect the functional and technical specifications be known.

A.2.3.2 Document Control

The document control procedures should be applied to the relevant documents including procedural documents describing the quality system to be applied in the software life-cycle, planning documents and product documents such as verification, validation plans, and maintenance documentation.

A.2.3.3 Measurements

Metrics should be reported and used to manage the development and delivery process and should be relevant to the particular software product. There are currently no universally accepted measures of software quality. However, at a minimum, some metrics should be used which represent reported field failures and/or defects from the customer's view point.

Metrics for process measurement should reflect how well the development process is being carried out and how effective it is at reducing the probability that faults are introduced or that any faults introduced go undetected. The choice of metrics should fit the process being used and, if possible, have a direct impact on the quality of the delivered software.

Appendix B

IEEE Standards Dictionary Of Measures To Produce Reliable Software

B.1 Measures To Produce Reliable Software

This guide addresses two major topics. The first presents a measure selection criteria. It addresses the following question: "To whom is the standard directed?" "When should the standard be applied?" and "why was the standard created?"

The second topic is the application of these measures to product, project, and industry. The questions addressed include: "How can the standard be used effectively?", "How should results be interpreted?", and "How should objectives be set?"

The measures are selected to provide information throughout the life cycle of a product. The basic goal is to provide the elements of a measurement program that support a constructive approach for achieving optimum reliability of the end product.

The dictionary focuses on measures of the potential causes of failure throughout the software life cycle, rather than just on measures of the effect of poor reliability of nearly completed products. It calibrates the rulers, measurement tools, through the use of common units and a common language.

B.1.1 Constructive Approach to Reliability Software

Software reliability measurement is the determination of the probability that software will not cause the failure of a system for a specified time under specified conditions. A constructive approach to reliable software seeks to remove the root causes of this

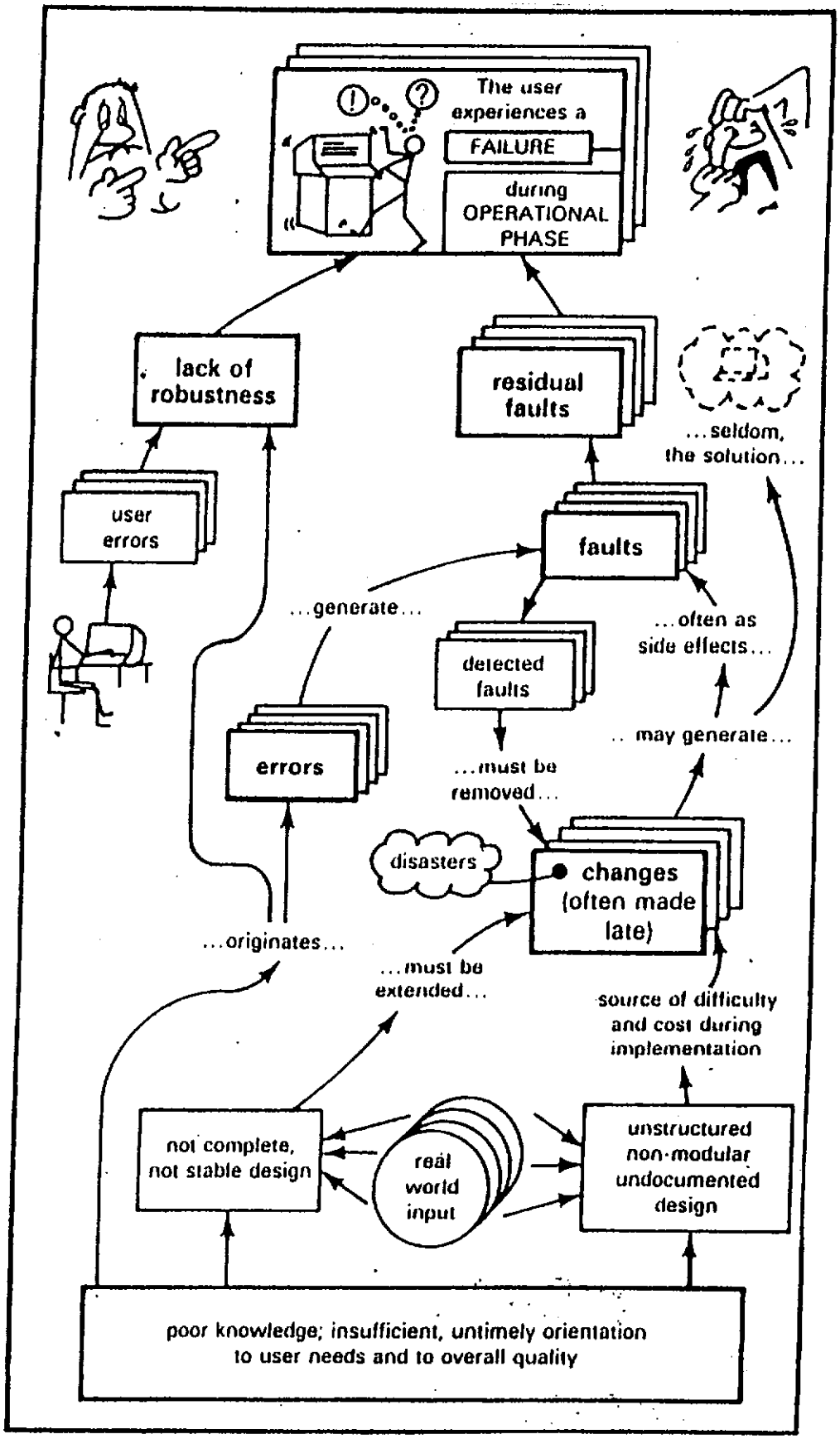


Figure B.1: Causes Of Failure

class of system failure (figure B.1) through software development and support processes that promote fault avoidance, early fault detection, appropriately prompt removal, and system-designated fault tolerance (figure B.2).

The driving criteria of this constructive process strategy are as follows:

- **Do it right the first time:** Stress error prevention and fault avoidance through employment of skilled personnel, early user involvement in the conception of the product and usage of modern methods, languages and automated tools.

- **Detect is Early, fix it as soon as practicable:** Detection and repair should be properly planned activities. Early detection of faults throughout the live cycle throgh reveiws, inspections, prototyping simulation, or proof of correctness of the product will improve reliability.

- **Monitor it:** Primitive data should be collected and evaluated throughout the life cycle. Measurments evaluation is iused not only for software quality evaluation, but also for fault detection.

A program to construct reliable software must consider not only reliability, but the other major project drivers: schedule and cost. Change most often occurs in response to unmet quality objectives, such as ease-to-use, maintainability or portability, that were not given proper attention during design. Therefore, there is a need for a quality model that defines user-oriented needs (figure B.3). This model must be evolved during the same time period as functions and performance objectives.

B.1.2 Measurement Environment

When measurements are used as process diagnostics, there must be a proper environment for data collection, analysis, and corrective action in a closed loop system. The building blocks for deriving more complex measures from collected data

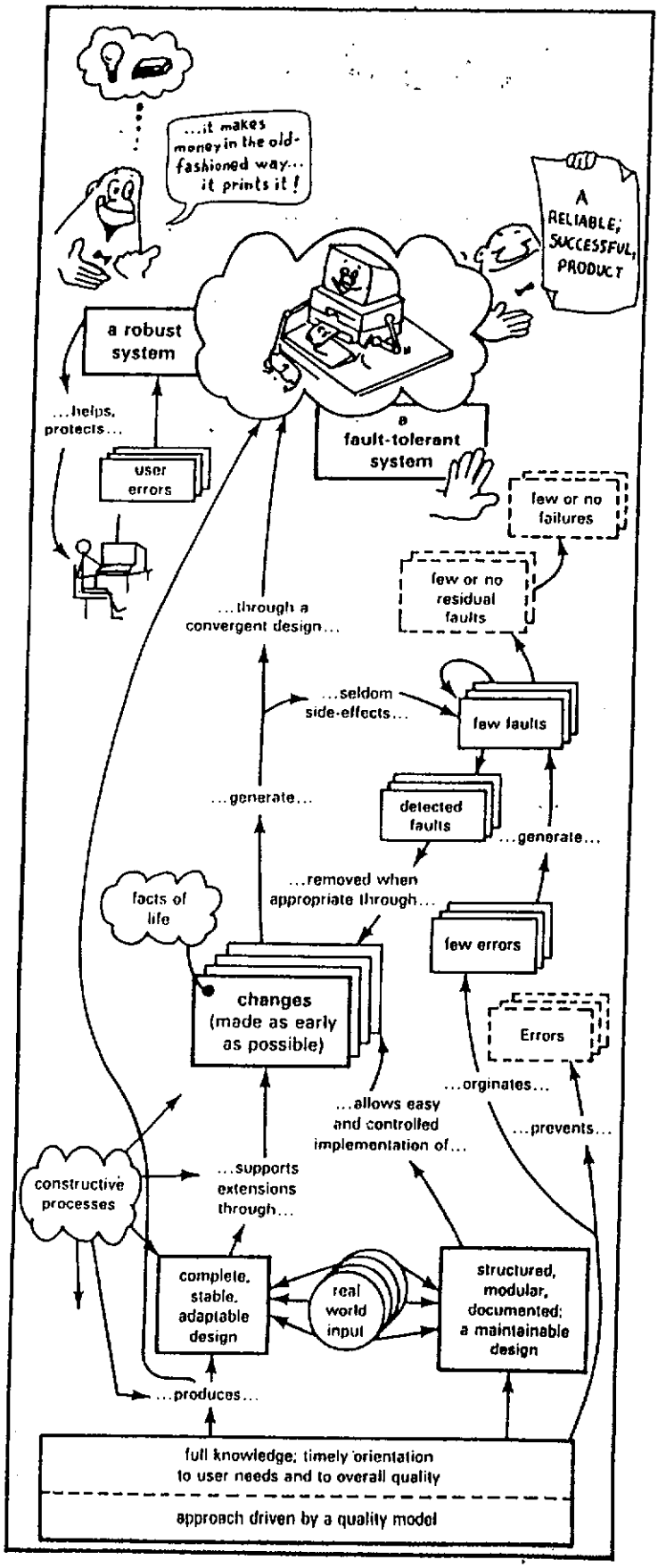


Figure B.2: Constructors Of A reliable Product

are called primitives.

An important outgrowth of implementing a measurement effort is the ability to evaluate the results of collected data, develop measures, and apply analysis of the results to the end product.

B.1.3 Measurement Selection Criteria

The standard includes product measures to determine how carefully a product is verified as for completeness and traceability. It also includes complexity measures to accommodate the possibility of change to the system. The process measures have the potential to simulate adoption of good engineering practices at the proper project phase, specifically encouraging early fault detection.

From the point of view of skill required, the measures are arranged in increasing order of difficulty. This allows individuals with little knowledge of computational methodologies to start making meaningful measurements.

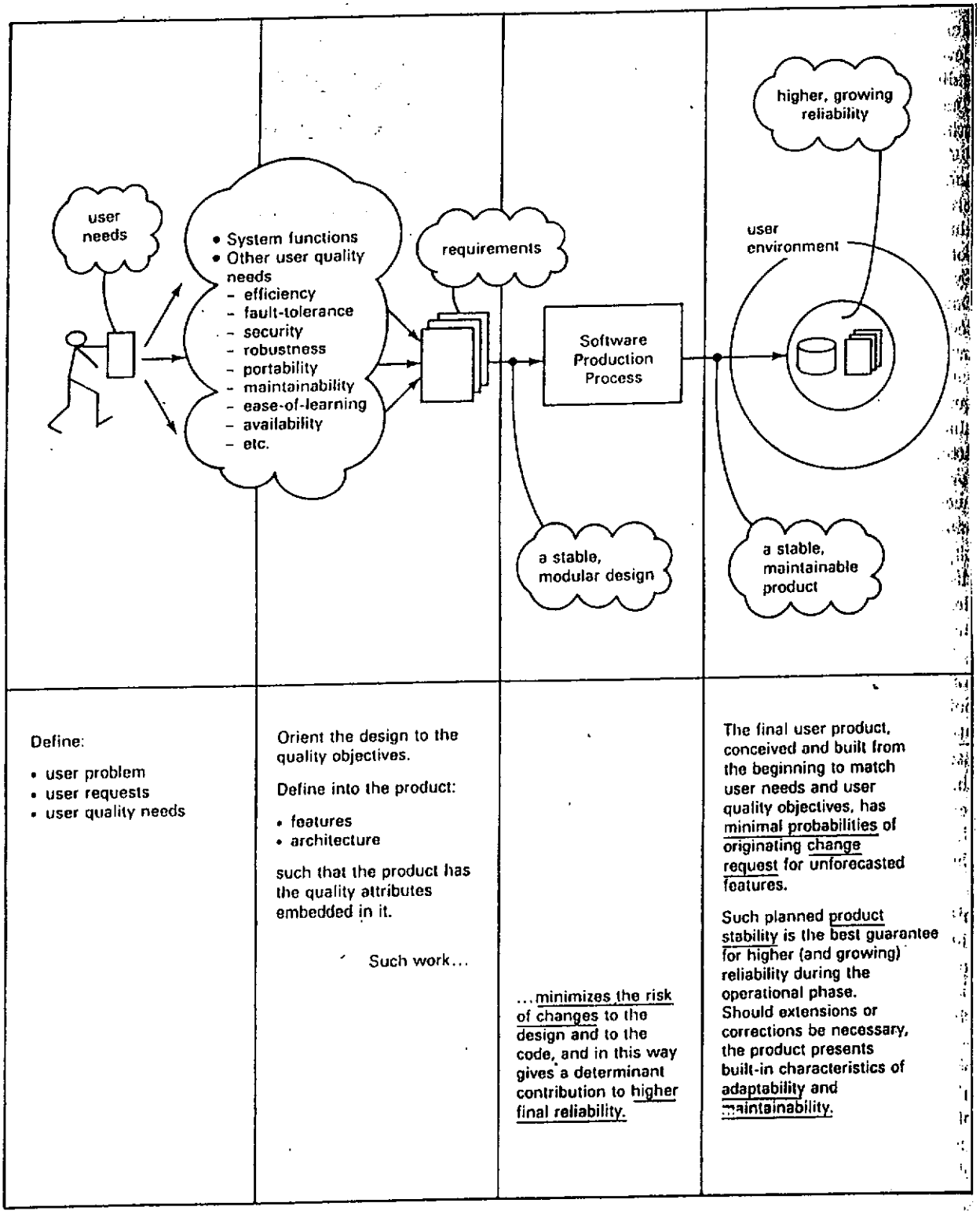
B.2 Framework For Measures

This measurement environment establishes a framework for determining and interpreting indicators of software reliability. A measurement process formalizes the data collection practices in both development and support. It provides for product evaluation at major milestones in the life cycle.

The software reliability measurement process can be described in nine stages where each influences the production of a delivered product with the potential for high reliability:

1. *Plan Organizational Strategy:* Initiate a planning process

2. *Determine software reliability goals:* in order to optimize reliability in light of



*During the operational phase, the quality model for software reliability is a function of product stability and adaptability.

Figure B.3: Quality Model For Software Reliability

realistic assessments of project constraints.

3. *Implement Measurement Process:* that best fits an organization's needs

4. *Select Potential Measures:* that would be helpful in achieving the reliability goals established in stage 2.

5. *Prepare Data Collection & Measurement Plan:* data should be organized so that information related to events during the development effort can be properly recorded in a data base and retained for historical purposes.

6. *Monitor Measurements:* the measurements assist in determining whether the intermediate reliability goals are achieved and whether the final goal is achievable.

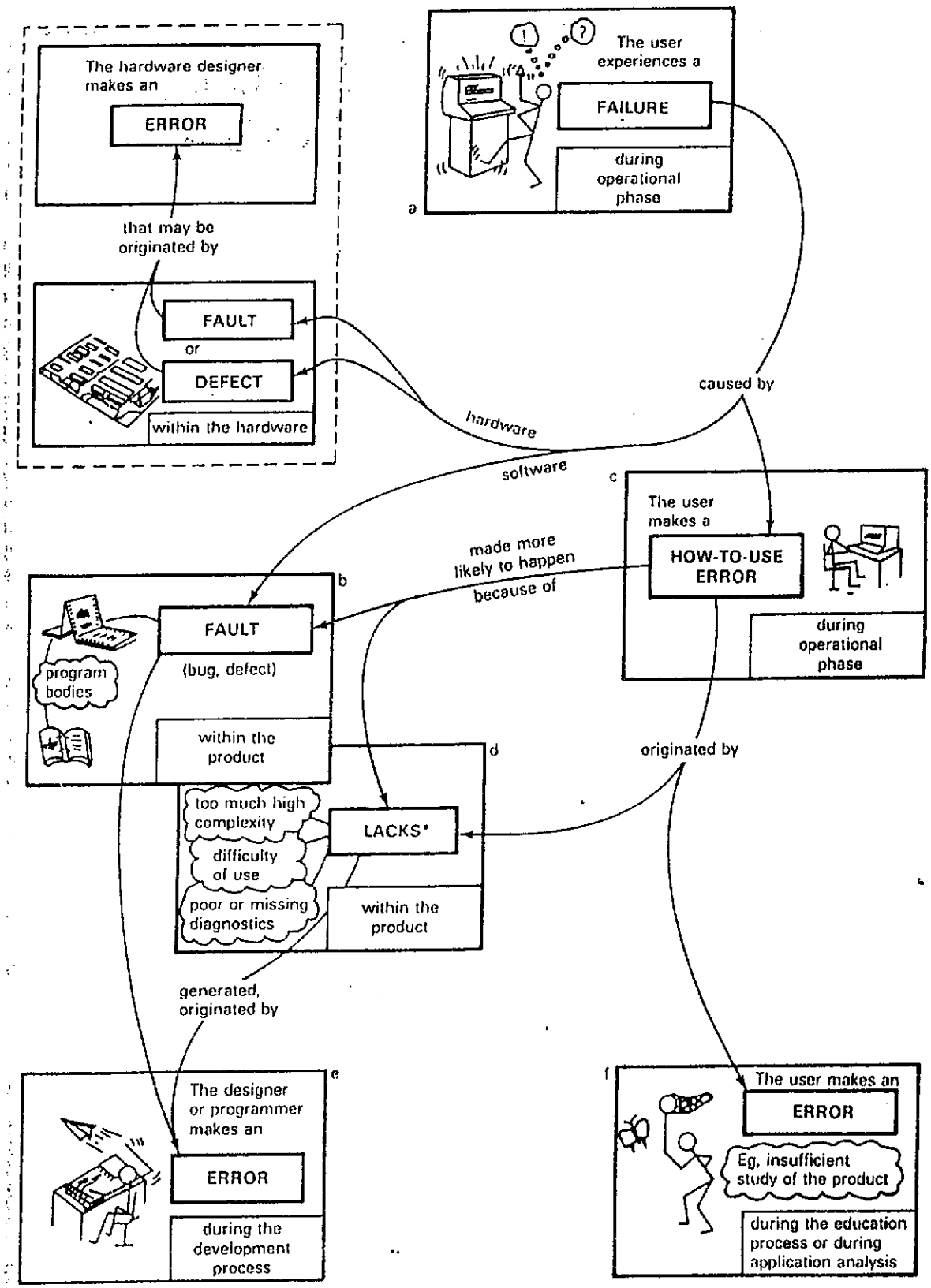
7. *Assess Reliability:* Analyze measurements to ensure that reliability of the delivered software satisfies the reliability objectives.

8. *Use Software:* assess the effectiveness of the measurement effort and perform necessary corrective action.

9. *Retain Software Measurement Data:* provides a baseline for reliability improvement and an opportunity to compare the same measures across completed projects.

B.3 Errors, Faults, & Failures Analysis for Reliability Improvement

A key to reliability improvements is the maintenance of an accurate history of errors and faults associated with failure events. Failures may occur when a fault is encountered or when a how-to-use error is made by the user (figure B.4). The purpose



*Product lacks (or weaknesses) could also be seen as a subclass of faults.

Figure B.4: The Error, Fault, Failure Chain

of failure analysis is to acquire knowledge of the most important "failure modes" of the product in the user environment.

Failures caused by user error can be due to lack of training, lack of support... An analysis of these failures can improve the human factors of the product based on a better understanding of the user's needs.

Failures due to developer or maintenance error are caused by product faults. Error analysis gathers historical information about error types in order to acquire insight into typical individual and organizational behavior in a Software Engineering environment. Surely, failures can be minimized by avoiding errors that generate the corresponding faults.

Appendix C

Australian Standards

The Australian standard (AS) [Australian Standards, 1991-1993] is a set of standards, technical reports, concerning documentation, requirements analysis design methodologies and other issuers. The main purpose is to present a solid guideline for Software Engineering methodologies.

C.1 Management of Software Documentation

The main purpose is to assist managers in ensuring that effective documentation is produced in their organization. The standard presented by AS is applicable to all types of software and relates to all stages of the software lifecycle. It is required during the early stages of a project and must be available and maintained throughout the software development process.

The six major functions of software documentation are:

1) *Communication to Management:* by periodic reports to follow schedule
2) *Task-to-Task Communication:* Between different teams such as analysts, programmers, Auditors
3) *Quality Assurance:* Responsibility for quality assurance by development & product documentation
4) *Instruction and Reference:* To aid usage of the software product
5) *Software Support:* To locate bugs and enhance software as required
6) *Historical Reference:* To assist in the transfer and conversion of software

to new environments

It is important to produce documents that describe the software development process, specify the requirement the software should fulfill, the design, how it should be tested and how to assure its quality.

C.1.1 Procedures

Documentation should be performed in sequences as defined below:

1- Planning
2- Preparation
3- Configuration Control
4- Review
5- Approval
6- Production
7- Storage
8- Backup
9- Distribution and Updating
10-Disposal

Finally, checklists must be answered for software documentation management to make documentation a company policy, ensure that procedures are followed and adopt a proper documentation plan.

C.2 User Documentation

This international standard is applicable to software packages sold off-the-shelf to consumers for business, scientific educational and home use. The main purpose of user documentation is to provide the end user with sufficient information for a clear understanding of the purpose, functions and characteristics of the software . It also explains how to install and use the software. It finally states the contractual rights and responsibilities.

C.2.1 Reference Documentation

This includes:

1) ***Identification of The package:*** such as title, version and release dates.

2) ***Package Components:*** including physical items supplied in the package and any associated software, hardware or documentation which may be relevant but are not included with the package.

3) ***Functional Description of the Software:*** This includes the purpose of the software and the functions it performs with performance characteristics and security facilities if any. Hardware and software requisites are also an asset.

4) ***Software Installation:*** Such as description of all actions necessary to introduce the software into its operation environments and how to create backup copies if allowed.

5) ***Using the Software:*** This covers loading the software, guidelines to functions of all control instructions or commands, list of output messages and their meaning, content , logical studies and format of the input detail and their restriction.

6) ***Software Technical Information:*** This is applicable to some types of packages, such as scientific software. The information can include the language used, software structures and algorithms used.

7) ***Testing:*** Including sample inputs and expected outputs

8) ***Contractual information:*** Such as the warranty, legal rights and responsibilities.

9) ***Glossary and Index:*** A glossary of technical or other specialist terminology with which the use is unlikely to be familiar and an index of the contents of the reference documentation can include this reference documentation

C.2.2 Training Documentation

The purpose is to give a step by step introduction to the use of the package for new inexperienced users. It is not however necessary to describe the user interface in complete detail. Also, as an alternative to , or in addition to printed documentation, training may be provided in an on-line form.

C.3 Requirements

This standard establishes requirements for a software developer's quality management system. It identifies each of the elements of a quality management system to be designed, developed and maintained by the developer with the objective of ensuring that the software will meet the requirements of a contract. The following explains the quality system requirements.

C.3.1 Management Responsibility

The developer's management shall define and document its policies and objectives for quality and its commitment to it. The developers should identify in-house verification and validation requirements.

Although developers should have quality assurance, quality control and inspection functions, these alone cannot satisfy all of the requirements of the customer. The management representative ensures that the requirements of the standard are implemented and maintained. He should act as a focal point for coordination and resolution of quality matter.

It is important that the quality system be subject to continual review and assessment of its appropriateness and effectiveness. These reviews need to be systematic and analyzed constantly to determine actions necessary to maintain the system's

effectiveness.

Finally, the developer should establish and maintain procedures for the review of quality costs as a means of identifying opportunities to be adopted for their optimization.

C.3.2 Quality System

The system should ensure that quality requirements are determined and that standards and procedures are established to satisfy such requirements, including development, acquisitions, inspection and testing, packaging, shipping, storage, installation and maintenance.

It is the developers responsibility to establish procedures appropriate to their own scale, methodologies and organization in order to achieve the requirements of the standard. Project management, design techniques and review procedures should also be covered.

C.3.3 Contract Review, Planning & Requirement Control

Each contract is reviewed by the developer to ensure that the requirements are properly defined and documented, contractual requirements are met, and responsibilities of the customer are defined.

Planning activities in early phases should be adequate in inspection, validation, verification, testing and reviews. A quality plan will identify standards and procedures to be used to achieve the software quality objectives specific to the contract. All deliverables and project dependencies should be documented in plans.

It is the customer's responsibility to ensure that the software product requirements reflect the real needs of the user. It is thus very important that control of

requirements, once established and approved, be maintained.

C.3.4 Design, Programming & User Documentation Control

In design control, the developer should use defined methodologies, establish design reviews, provide feedback from previous developments, and take into consideration. Safety, reliability and maintainability. Reviewing the design ensures that the software design is consistent with the contractual requirements. Such procedures include identification of review points and provision for the recording of analysis and recommendations of reviews.

Programming standards shall describe approved programming practices and list any prohibited practices appropriate to the programming language used. The employment of such standards should also aid in the reduction of errors in code.

C.3.5 Configuration Management

The developer establishes procedures for identifying and controlling software components, including changes and for maintaining the traceability of the configuration at all stages of contract performance.

Configuration management should be established at the beginning of the software development cycle. Configuration items include specifications, design documents, test documentation and all modules whether purchased or developed.

C.3.6 Inspection & Testing

Either the developer or customer shall be responsible for providing test data, facilities and documentation needed to demonstrate conformance with the software product requirements, to the extent stated in the contract.

The developer shall identify all relevant quality assurance procedures related to inspection and testing addressing, test planning, test specifications, controlling software test plans and procedures changes and appropriate inspection during all phases.

C.3.7 Handling, Storage, Packaging & Delivery

The developer shall develop, adopt and maintain procedures to ensure correct identification and certification of the deliverable configuration. Packaging should ensure safe arrival at the destination.

C.3.8 Internal Quality Audits

The quality system should be subject to continued improvement and assessment of its effectiveness. In order to ensure this, the supplier's management should periodically and systematically conduct management reviews and internal quality audits.

The internal quality audit should be a planned, purposeful and comprehensive examination of management objectives, assignment of duties, delegation of responsibilities and methods of operation. Audits serve as a check on management controls at all levels. They are also designed to uncover potential problems and to eliminate waste or unnecessary loss.

C.3.9 Software Maintenance

Software is subject to continual change throughout its life, both to rectify problems, i.e. maintenance and to provide additional features or capabilities i.e. enhancements. It is important that appropriate quality management procedures are applied throughout the change process and that such procedures are not of a lesser standard than those that would be applied during the development process.

C.4 Software Design Description (SDD)

A Software Design Description (SDD) is a representation of a software system that is used as a medium for communicating software design information. Applicability is not restricted by the size, complexity, or criticality of the software. It is assumed that the quality design information and changes to the design or description will be managed by other project activities.

C.4.1 Considerations

The life cycle approach is an effective engineering management tool and provides a model for a context within which to discuss the preparation and use of the SDD. For both new software systems and within systems under maintenance, it is important to ensure that the design and implementation used for a software system satisfy the requirements driving the system.

It is important to realize that the SDD shows how the software system will be structure to satisfy the requirements identified in the software requirements specification. In a complete SDD, each requirement must be traceable to one or more design entities.

C.4.2 Content

The SDD model should provide the precise design information needed for planning, analysis and implementation of the software system. The design description model used to represent a software system can be expressed as a collection of design entities, each possessing properties and relationships.

The objective is to divide the system into separate components that can be considered, implemented, changed and tested with minimal effect on other entities. Each design

entity will have a name, purpose, and function. The common characteristics of entities are described by design entity attributes.

Design entity attributes can be thought of as questions about design entities. The answers to those questions are the values of the attributes . The collection of answers provides a complete description of an entity. All attributes shall be specified for each entity. The attributes and associated information items are defined as follows:

1) *Identification:* The name of the entity. This will simplify referencing and tracking in addition to providing identification.
2) *Type:* A description of the kind of entity
3) *Purpose:* A description of why the entity exists
4) *Function:* A statement of what the entity does
5) *Subordinates:* The identification of all entities composing this entity. This information is used to trace requirements to design entities and to identify parent/child structural relationships through a software design decomposition.
6) *Dependencies:* A description of the relationships of this entity with other entities
7) *Interface:* A description of how other entities interact with this entity
8) *Resources:* A description of the elements used by the entity that are external to the design.
9) *Processing:* A description of the rules used by the entity to achieve its functions.
10) *Data:* A description of data elements internal to the entity. The description of data may be in the form of a data dictionary that describes the content structure and use of all data elements.

C.4.3 Organization

Each design description used may have a different view of what is considered the

essential aspects of software design. A recommended organization of design entities and their associated attributes are presented to facilitate the access of design information from various technical viewpoints.

Each design view represents a separate concern about a software system. Together these views provide a comprehensive description of the design in a concise and usable form that simplifies information access and assimilation. Table C.1 summarizes recommended design views:

Design View	Scope	Entity Attributes	Example Representation
Decomposition Description	Partition of The System into Design Entities	Identity, Type, Purpose, Function, Subordinate	Hierarchical Decomposition diagram, Natural lay
Dependency Description	Description of the relationship among entities & system resources	Identity type, Purpose, Dependency, Resources	Structure Charts, Data Flow Diagrams, Transaction diagrams
Interface Description	List of everything a designer Programmer or tester needs to know to use the design entities that make up the system	Identity, Function, Interfaces	Interface Files, Parameter Tables
Detail Description	Description of The Internal Design Details of an Entity	Identity, Processing, Data	flowcharts, N-S Charts, PDL

Table C.1: Recommended Design Views

C.5 Software Test Documentation

This standard describes a set of basic test documents which are associated with the dynamic aspects of software testing. Several of the documents may be also applicable to other testing activities such as design and code reviews.

The standard addresses the documentation of both initial development testing and the testing of subsequent software releases. Each organization using the standard will need to specify the classes of software to which it applies and the specific documents requirements for a particular test phase.

The standard does not call specific testing techniques nor does it impose specific methodologies for documentation control, configuration management, or quality assurance. Testing is defined as the process of analyzing a software item to detect the differences between existing and required conditions, called bugs, and to evaluate the features of the software item.

C.5.1 Test Plan

It is mainly used to schedule testing activities. A test plan shall have the following structure:

1) *Test Plan Identifier:* Specifies the unique identifier assigned to this test plan.
2) *Introduction:* Summarizes the software items and software features to be tested.
3) *Test Items:* Identifies test items including their version / revision level
4) *Features to be Tested:* Identifies the test design specification associated with each feature and each combination of features
5) *Feature not to be Tested:* Identifies all features and significant combination of features which will not be tested and the reasons.
6) *Approach:* Describes the overall approach to testing, and specifies the major

activities, techniques and tools which are used to test the designated group of features.

7) *Item Pass/Fail Criteria:* Specifies the criteria to be used to determine whether each test item has passed or failed testing.
8) *Suspension Criteria & Resumption Requirements:* Specifies criteria used to suspend all or a portion of the testing activities and which are to be repeated when testing is resumed.
9) *Test Deliverables:* Identifies the deliverable documents such as test plan, test design specifications, test case specifications , test procedure specifications.
10) *Testing Tasks:* Identifies the set of tasks necessary to prepare for and perform testing.

11) *Environmental Needs:* Specifies both the necessary and desired properties of the test environment
12) *Responsibilities:* Identifies the groups responsible for managing, designing, preparing, executing, witnessing, checking and resolving.
13) *Staffing & Training Needs:* Specifies test staffing needs by skill level.
14) *Schedule:* Includes test milestones identified in the software project schedule as well as all item transmittal events.
15) *Risks & Contingencies:* Identifies the high-risk assumptions of the test plan.
16) *Approvals:* Specifies the names and titles of all persons who must approve this plan.

C.5.2 Test-Design Specification

The purpose is to specify refinements of the test approach and to identify the features to be tested by the design and its associated tests. A test-design specification shall have the following structure:

1) *Test-Design-Specification Identifier:* Unique identifier assigned to the test design specification

2) *Features to be Tested*

3) *Approach Refinements:* Specify refinements to the approach description in the test plan.

4) *Test Identification:* List the identifier and a brief description of each test case associated with this design.

5) *Feature Pass/Fail*

C.5.3 Test-Case Specifications

The purpose is to define a test case identified by a test design specification. The structure is the following:

1- Test-case Specification
2- Test Items
3- Input Specifications
4- Output Specifications
5- Environmental Needs
6- Special Procedural Requirements
7- Interface Dependencies

C.5.4 Test Procedure Specifications

The purpose is to specify the steps for executing a set of test cases. It has the following outline:

1- Test-Procedure specification Identifier
2- Purpose
3- Special Requirements
4- Procedure Steps

C.5.5 Test-Item Transmittal Report

The purpose is to identify the test items being transmitted for testing. Any variations from the current item requirements and designs are noted in this report. It has the following outline:

1- Transmittal Report Identifier
2- Transmittal Items
3- Location
4- Status
5- Approvals

C.5.6 Test Log

The purpose is to provide a chronological record of relevant details about the execution of tests. It has the following structure:

1- Test-Log Identifier

2- Description

3- Activity & Event Entries

C.5.7 Test-Incident Report

The purpose is to document any event that occurs during the testing process which requires investigation. It has the following structure:

1- Test-Incident-Report Identifier

2- Summary

3- Incident Description

4- Impact

C.5.8 Test-Summary Report

The purpose is to summarize the results of the designated testing activities and to provide evaluations based on these results. The outline consists of :

1- Test-Summary-Report Identifier

2- Summary

3- Variances

4- Comprehensive Assessment

5- Summary of Results

6- Evaluation

7- Summary Of Activities

8- Approvals

C.6 Software Reviews

The purpose of this standard is to provide definitions and uniform requirements for review processes. By definition, a review is an evaluation of software elements or project status to ascertain discrepancies from planned results and to recommend improvement.

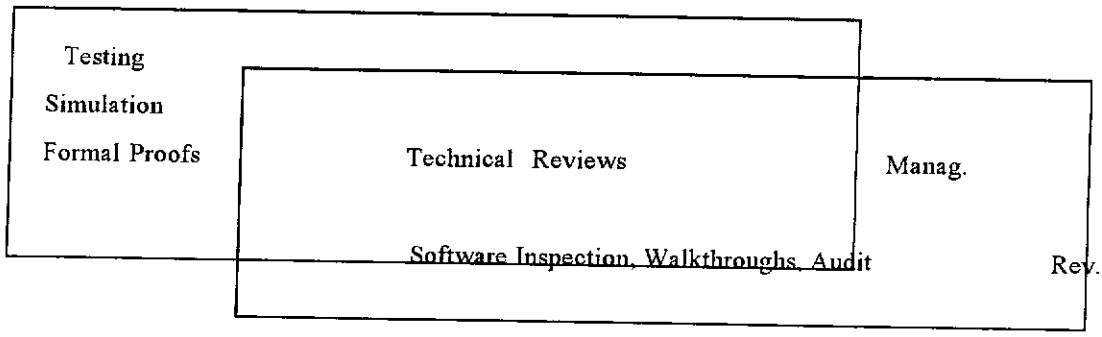
Reviews can be used in support of objectives associated with software quality

assurance, project management, and configuration management or similar control functions as shown in Table C.2:

OBJECTIVES	PRINCIPAL PROCESSES INCLUDE
Evaluation	Management Review Technical Review
Verification	Inspection, Walkthroughs
Validation	Test
Compliance, Confirmation	Audit

Table C.2: Some Principal Processes For Achieving Quality Objectives

Figure C.1 provides a view of how various quality assurance processes can be mapped to examine product and project issues.



FigureC5.1: View Of Quality Assurance Processes in Product/Project Issues

C.6.1 Management Review Process

A management review is a formal evaluation of a project level plan or project status relative to that plan by a designated review team. The need for conducting certain management reviews is initially established in the appropriate project planning documents such as software quality assurance plan.

C.6.2 Technical Review Process

The objective of a technical review is to evaluate a specific software element and provide management with evidence that it conforms to its specifications and changes are properly implemented. It is a formal team evaluation of a software element. The need for conduction technical reviews of specific software elements is defined by project planning documents.

C.6.3 Software Inspection Process

The objective of a software inspection is to detect and identify software element defects. The process is led by a moderator impartial to the software element under examination. The moderator is not the author.

C.6.4 The Walkthrough Process

The objective of a Walkthrough is to evaluate a software element. This includes finding defects emissions, and contradictions in order to improve the software elements and to consider alternative implementations. It also includes exchange of techniques and style variations and education of the participants. During the walkthrough meeting, the author makes an overview presentation of the software elements under review. This is followed by a general discussion from the participants after which the presenter "walks through" the software elements in detail.

Table C.3 provides further accent process distinction and guide in the application of review processes.

C.7 Software Project Management Plan (SPMP)

A software project management plan is the controlling document for managing a software project. It defines the technical and managerial processes necessary to satisfy the project requirements.

This standard identifies the minimal set of elements that shall appear in all software project management plans. The essential elements of a SPMP are presented is a detailed plan which starts with the Title Page, Revision Chart, Preface, Table of Contents, List of Figures, List of Tables then follows the following outline:

C.7.1 Introduction

1) ***Project Overview:*** Concise summary of the project objective.
2) ***Project Deliverables:*** List of all the items to be delivered to the customer.
3) ***Evolution of the SPMP:*** Plans for producing both scheduled and unscheduled updates of the SPMP.
4) ***Reference Materials:*** List of all documents and other sources of information referenced in the SPMP.
5) ***Definitions & Acronyms***

C.7.2 Project Organization

1) ***Process Model:*** Defines the relationships among major project functions and activities.
2) ***Organizational Structure:*** Internal management structure of the project.
3) ***Organizational Boundaries & Interfaces:*** Describes administrative and managerial boundaries between the project and various entities
4) ***Project Responsibilities:*** States the nature of each major project function and activity.

C.7.3 Managerial Process

1) ***Management Objectives & Priorities:*** Describes the philosophy, goals, and priorities for management activities during the project.
2) ***Assumptions, Dependencies, & Constraints.***
3) ***Risk Management:*** Assesses risk factors associated with the project.
4) ***Monitoring & Controlling Mechanisms:*** Defines the reporting mechanisms, report formats and other tools used in monitoring and controlling adherence to the SPMP.
5) ***Staffing Plan:*** Specifies the number and types of personnel requirements to conduct the project.

C.7.4 Technical Process

1) ***Methods, Tools, & Techniques***
2) ***Software Documentation***
3) ***Project Support Functions:*** Contains, either directly or by reference, plans for the supporting functions for the software project.

C.7.5 Work Packages, Schedule, & Budget

1) ***Work Packages***
2) ***Dependencies:*** Specifies the ordering relations among work packages to account for interdependencies among them and dependencies on external events.
3) ***Resource Requirements***
4) ***Budget & Resource Allocation***
5) ***Schedule:*** Schedules may be expressed in absolute calendar time or in increments relative to a key project milestone.

The plan can also include any additional components added as sections or subsections to the SPMP. An index of key terms and acronyms used follows. The plan ends with

available appendices.

C.8 Remarks

The above is a model of the Australian Standards. It serves to show how international standards are applied in Australia. The Software Test Documentation, Software Design Description, Software Reviews, Software Project Management Plan are identical to IEEE Standards. The User Documentation and Management Of Software Documentation are identical to ISO Standards. As for The Requirements, It is closely related to ISO9001 Standard but not identical. It places greater emphasis on design activities whereas ISO9001 emphasizes production and quality of conformance aspects.

QUESTIONNAIRE

Q: What kind of Software do you develop?
* Accounting * Educational * Other, Please Specify * Stock * Government * Banking * System Software
Q: What are the Software Development phases?
Q: How do you estimate the deadline to deliver the Software?
Q: On What basis do you set the price?
Q: How do you plan and schedule the various phases of development?
Q: Are there any case tools at this stage?
Q: When analyzing what the client wants, what are the fact finding techniques used?
* Questionnaire * Direct observation of the flow of operations * Interview * Analyze a client request report * Other, please specify
Q: Is there a perliminary document presented to the client specifying what the software can and will do?
Q: How do you represent the requirements specifications?
* Plain English * Decision tables/trees * Formal presentation * Data Flow diagrams * Structured English * Other, please specify

Q: At this stage, do you show the client an operational examples of the product?

Q: Do you then throw it away? or build the final product from it?

Q: How do you check the Requirements?

* Walkthroughs
* Reviews
* Inspections
* Other, please specify

Q: What is the Main Principle used in your designs?

* Process Oriented
* Modular
* Top down design
* Object Oriented
* Transaction Oriented
* Bottom up design
* Other, please specify

Q: Do you use the following?

* Data flow Graphs
* Flowcharts
* Algorithms
* Structured Charts
* Entity Relationship Diagrams
* Other, please specify

Q: In Quality Assurance Quest, do you use the following?

* Checklists
* Design reviews
* Other, please specify

Q: How do you keep a record of design documents?

Q: What languages do you use and in what percentage?

* Foxpro ----%
* Access ----%
* Paradox ----%
* Pascal ----%
* C ----%
* Cobol ----%
* Basic ----%
* Other, pls specify

Q: Do you use the following?

* Coding sheets
* Screen Forms
* Print Formats
* Internal Documentation (comments)
* Checklists to aid programming

Q: Any case tools used in programming?
Q: Which Methods of Unit/Module testing do you use?
* Statment testing * Branch testing * Path testing * Functional testing * Mutation testing * Other, please * Domain testing * Inspections specify * Reviews
Q: Do you use any testing tools?
Q: Do you think the best way to test Software is to let the customer try for himself in operational environment?
Q: Do you use Integration Testing?
Q: How do you perform product/Acceptance testing?
Q: How do you document your test cases?
Q: What are the main maintenance activities used, and in what percentage?
* Corrective ----% * Other, please specify * Adaptive ----% * Perfective ----%
Q: How do you train the client to use the Software?
* User's Manual * Invite him over to try * Show him on field * Other, please specify

Q:How do you record a client's report for an error or an improvement?
Q:How do you locate errors?
* Manually * Other, please specify * CASE Tools
Q: How do you estimate how maintainable your software is?
Q: Do you use past test cases to ensure correctness after a change is done?
Q: How do you keep a record of changes?
Q: What maintenance tools do you use?
Q: What metrics do you use to assess Software's Complexity?
* Lines of Code (LOC) * McCabe's * Halstead's * COCOMO * Function points * Other, please specify
Q:How do you assess Your software's Quality?

Q: What causes delays in delivering the software most of the times?
Q: How do you budget every software?
Q: Are the following documents used and updated regularly in your company?
* Design Plan * Management Plan * Test Plan * Other, please specify
Q: What is ,approximately, the percentage of errors discovered in the various phases of the software production?
* Requirements Analysis ----% * Testing ----% * Design ----% * After Delivery ----% * Coding ----%
Q: Do you consider testing a method to find errors or prove that there aren't any?
Q: Are the above methods used regularly in all your applications?

References

ANSI/IEEE Std 829-1983, *A Standard for Software Test Documentation*, Institute of Electrical and Electronics Engineers, Inc., New York, Approved by American National Standards Institute, Aug. 1983, Revised in March 1987

ANSI/IEEE Std 983-1986, *Guide of Software Quality Assurance Planning*, Institute of Electrical and Electrnonics Engineers, INC., New York, Approved by American National Standards Institute, Aug. 1986

Australian Standards 1991-1993, Published by Standards Association, Australia, North Sydney, NSW

Beizer, B. 1990., *Software Testing Techniques*, New York, Van Nostrand Reinhold

Brackett, John W. 1990, *Software Requirements*, SEI Curriculum Module, SEI-CM-19-1.2, January, Camegie Mellon University

British Standard 1986, *British Standard Guide to Specifying User Requirements For a Computer Based System*, British Standards Institute, London

Davis A. M. 1990., *Software Requirements, Analysis & Specification*, Prentice Hall, Englewood Cliffs, New Jersey

DOD-STD2167-A 1988., *Software Test Plan*, Naval Publications and Forms Center US Department Of Defense, Philadelphia

Fouser, Thomas J. 1988, Software Requirements Analysis Phase, Version 3.0, December, reprinted from *Software Requirements Analysis Phase Standard JPL D-4005*, by Jet Propulsion Laboratory

Goldberg, R. 1986., Software Engineering: An Emerging Discipline, *IBM Systems Journal*, Vol.25 No.3/4

Gray, Edwin M. and Rao, G. 1993, Software Requirements Analysis and Specification in Europe: An Overview, *Standards, Guidelines, and Examples on System and Software Requirements Engineering*, IEEE Computer Society Press, Los Alamitos, California

IEEE Std 830 1984, *Guide For Software Requirements Specification*, Institute of Electrical and Electronics Engineers, Inc., New York, Revised in 1988

ISO 9000-3 1991, *Quality Management and Quality Assurance Standards-Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software*, International Organization for Standardization (ISO), Geneva

Kaner, C., Falk, J. and Hung, Q. H. 1993., *Testing Computer Software*, Second Edition, Van Nostrand Reinhold, N.Y

Knight, J. C. and Meyers, E. A. 1993., An Improved Inspection Technique, *Communications Of The ACM*, November , Vol.36, No.11, pp.51-61

Mullerburg, M. 1990, Software Testing: A Stepwise Process, *Software Engineering, A European Perspective*, IEEE Computer Society Press, Los Alamitos, California

Schmauch, Charles. H. 1994, *ISO9000 For Software Developers*, ASQC Quality Press, Milwaukee, Wisconsin

STARTS 1988, *The STARTS Guide*, Second Edition, Chapter 5, Volume 1, pp 177-223, National Computing Centre, U.K.

Thayer, R. H. and Mc. Gettrick, A. D. 1993., Editors, *Software Engineering, A European Perspective*, IEEE Computer Society Press, Los Alamitos, California

Bibliography

Bandhary I. 1993, A Case Study Of Software Process Improvement During Development, *IEEE Transactions On Software Engineering*, December Vol.19, No.12

Blum Bruce I. 1995, Resolving The Software Maintenance Paradox, *Software Maintenance: Research And Practice*, Vol.7, 3-26

Boehm, B. 1994., Software Requirements As Negociated Win Conditions, Prasanta Bose, *The Proceedings Of The First International Conference On Requirements Engineering*, April 18-22, Colorado Springs, IEEE Computer Society Press

Boehm, B. and Belz, F. C. 1988., Applying Process Programming to The Spiral Model, *Proc. Fourth Software Process Workshop*, IEEE Computer Society Press, May

Bowen J. P., and Hinchey. G. M. 1995, Seven More Myths Of Formal Methods, *IEEE Software*, N.Y, July, Vol.12, No.14, pp34-40

Brooks, F. P. Jr. 1987., No Silver Bullet: Essence & Accidents Of Software Engineering, *IEEE Computer*, April

Bubenko, J., Rolland, C. and DeAntonellis, V. 1994., Facilitating Fuzzy To Formal Requirements Modeling, *The Proceedings Of The First International Conference On Requirements Engineering*, April 18-22, Colorado Springs, IEEE Computer Society Press

Dobson, J. and Sterns, R. 1994., Organizational Requirements Definition for Information Technology System, *The Proceedings Of The First International Conference On Requirements Engineering*, April 18-22, Colorado Springs, IEEE Computer Society Press

Edwards J. M. and Henderson-Sellers B. 1990., The Object Oriented Systems Life-Cycle, *Communications Of The ACM*, September

Fagan, M. E. 1976., Design & Code Inspections To Reduce Errors in Program Development, *IBM Systems Journal* 15, No.3, pp.182-211

Fairly, R. E. 1994., The Concept Of Operations: The Bridge From Operational Requirements To Technical Specifications, *The Proceedings Of The First International Conference On Requirements Engineering*, April 18-22, Colorado Springs, IEEE Computer Society Press

Gelpain, D. and Hetzel, B. 1988., The Growth Of Software Testing, *Communications of The ACM* 31, June pp.687-695

Ghezzi, C. 1991., *Fundamentals Of Software Engineering*, New .York

Gordon G., S. and Mc. Manus I. J. 1987., Editors, *Handbook Of Software Quality Assurance*, Van Nostrand Reinhold, N.Y

Hall, A. 1990., Seven More Myths Of Formal Methods, *IEEE Software*, September, pp.11/19

IEEE Standards Collection 1994, *Software Engineering*, New. York

Jalet, P. 1991., *An Inegrated Approach To Software Engineering*, New. York

Lea, R. J. 1991., On Generating Test Data From Prototypes, *Conference On Software Maintenance*, IEEE Computer Society Tech.,pp345-350

Macro, A. 1990., *Software Engineering, Concepts & Management*, New York

Meyers, G. J. 1979., *The Art Of Software Testing*, John Wiley & Sons, New York

Miller, D. and LittleWood, B. 1991., Editors, Software Reliability & Safety, reprinted from *Reliability Engineering & System Safety*, Vol.32, No.1/2, pp.135-154

Pohl, K. 1993., The Three Dimensions Of Requirements Engineering, IEEE *International Symposium On Requirements Engineering*, San Diego, California

Royce, W. W. 1970., *Managing The Development Of Large Software Systems:Concepts & Techniques*, Proc. Wescon, August

Royer, T. C. 1993., *Software Testing Management: Life On The Critical Path*, Prentice Hall, Englewood Cliffs, New Jersey

Saiedian, H. and Kuzara, R. 1995., SEI Capability Maturity Model's Impact on Contractors, *IEEE Computer*, January, pp.16-26

Salek, A. and Soreson, P. G. 1994., The Review System: From Formal Specifications To Natural Language, *The Proceedings Of The First International Conference On Requirements Engineering*, April 18-22, Colorado Springs, IEEE Computer Society Press

Thayer, R. H. 1992., *Software Engineering Project Management*, IEEE Computer Society Press, New Jersey

Thayer, R. H. and Dorfman, M. 1993, Editors, *Standards, Guidelines, and Examples on System and Software Requirements Engineering*, IEEE Computer Society Press, Los Alamitos, California

White S. M. 1994., Comparative Analysis Of Embedded Computer System Requirements Methods, *The Proceedings Of The First International Conference On Requirements Engineering*, April 18-22, Colorado Springs, IEEE Computer Society Press

Wolflang A. H. and Kramer, B. J. 1994., Safety Assurance In Process Control, *IEEE Software*, January