


RT
233

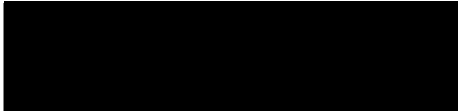
Lebanese Standards For Software Development Process

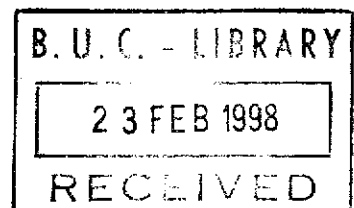
By
Ihab Z. Shallah
B.Sc., American University of Beirut

PROJECT

Submitted in partial fulfillment of the requirements for the degree of Master of
Science in Computer Science at the Lebanese American University
February 1998


Dr. Nash'at Mansour (Advisor)
Associate Professor of Computer Science
Lebanese American University

Gift

Dr. Ramzi Haraty
Assistant Professor of Computer Science
Lebanese American University



**To Maher Saadeh, the person who taught me how to fight till the end
Hope you are in heaven**

Table of Contents

List of Tables	viii
Acknowledgments	ix
Abstract	x
1 Introduction	1
1.1. Introduction	1
1.2. Analyzing the current status	2
1.3. Proposed solution	2
1.4. Thesis organization	2
2 Phases Overview	5
2.1. Introduction	5
2.2. The phases of a software life cycle	6
2.2.1. Phase One: Preliminary Specification and Planning	6
2.2.2. Phase Two: Requirement Specification Phase	7
2.2.3. Phase Three: Feasibility Prototype (Optional)	9
2.2.4. Phase Four: Planning Phase	10
2.2.5. Phase Five: Software Design Phase	11
2.2.6. Phase Six: Implementation Phase	13
2.2.7. Phase Seven: System Testing Phase	15
2.2.8. Phase Eight: Product Release Plan	16
2.2.9. Phase Nine: Product Release Phase	17
2.2.10. Phase Ten: Maintenance Phase	18
3 Purchaser/Supplier Contract	19
3.1. Introduction	19
3.2. The goal of the contract	19
3.2.1. Covered issues and responsibilities	20
3.2.2. Risk minimization	20
3.2.3. Terminology of the contract	21
3.2.4. Acceptance criteria	21
3.2.5. Requirement change procedure	22

3.2.6. Bug reporting procedure	22
3.3. Conclusion	22
4 Requirement Specification Document	24
4.1. Introduction	24
4.2. Issues addressed by the document	24
4.3. Characteristics of the document	25
4.3.1. Correctness	25
4.3.2. Unambiguity	25
4.3.3. Completeness	25
4.3.4. Consistency	26
4.3.5. Importance and/or stability	26
4.3.6. Verifiability	26
4.3.7. Modifiability	27
4.3.8. Traceability	27
4.4. Difference between the Design Document and RSD	27
4.4.1. Responsibilities of the customer and supplier	28
4.4.2. Incomplete requirements indication	28
4.4.3. Content of the document	28
4.5. Conclusion	29
5 Planning the Development Process	30
5.1. Introduction	30
5.2. Definition of Planning	30
5.3. Steps of the Planning Phase	32
5.3.1. Choice of a template	32
5.3.2. Choice of a process	32
5.3.3. Schedule/Budget	33
5.3.4. Update of the plan	33
5.4. Phase description	33
5.5. Verification process	34
5.6. Management responsibilities	34
5.6.1. Schedule of subsequent phases	34
5.6.2. Process control	35

5.6.3. Responsibilities	35
5.6.4. Inter-organization relations	35
5.7. Methods and tools definitions	36
5.8. Purpose of the plan	36
5.9. Conclusion	37
 6 Software Design Document	 38
6.1. Introduction	38
6.2. Cost/Time-friendly design	39
6.3. Evolutionary design approach	39
6.4. Coupling and cohesion	39
6.5. The document	40
6.5.1. Definition	40
6.5.2. Design object	41
6.5.3. Design Object Attributes	41
6.5.3.1. Identification	42
6.5.3.2. Type	42
6.5.3.3. Job	42
6.5.3.4. Interface	42
6.5.3.5. Resources	43
6.5.3.6. Data	43
6.6. Design Plan	43
6.6.1. Definition	43
6.6.2. System decomposition into functional modules	44
6.6.3. Individual decomposition of functional modules	44
6.6.4. Design of the database	45
6.7. Conclusion	45
 7 The Implementation Phase	 47
7.1. Introduction	47
7.2. Principles of Implementation Phase	48
7.2.1. Choice of the programming language	48
7.2.2. Programming style	48
7.2.3. Programming practices	49

7.2.4. Team organization	49
7.2.5. Portability	50
7.2.6. Reusability	50
7.3. Code Reviews	51
7.4. Conclusion	52
 8 The Testing Phase	 53
8.1. Introduction	53
8.2. What is testing?	54
8.3. What should the plan address	54
8.4. Test plan document	55
8.4.1. Test plan identifier	56
8.4.2. Introduction	56
8.4.3. Test items	56
8.4.4. Features to be tested	57
8.4.5. Features not to be tested	57
8.4.6. Testing phases and approaches	57
8.4.7. Item pass/fail	58
8.4.8. Suspension criteria and resumption requirements	58
8.4.9. Environmental needs	58
8.4.10. Responsibilities	59
8.4.11. Staffing and training needs	59
8.4.12. Schedule	59
8.4.13. Risks and contingencies	59
8.4.14. Approvals	60
8.5. More documents	60
8.5.1. Test Design Specification	60
8.5.2. Test-Case Specification	60
8.5.3. Test Procedure Specification	60
8.5.4. Test-Item Transmittal Report	61
8.5.5. Test Log	61
8.5.6. Test-Incident Report	61
8.5.7. Test Summary Report	61

8.6. Unit Testing	61
8.6.1. Plan the general approach, resources and schedule	62
8.6.2. Determine features to be tested	63
8.6.3. Refine the general plan	63
8.6.4. Design the set of tests	64
8.6.5. Implement the refined plan and design	64
8.6.6. Execute the test procedures	64
8.6.7. Check for termination	64
8.6.8. Evaluate the effort and unit	65
8.7. Conclusion	65
 9 The Maintenance Phase	 66
9.1. Introduction	66
9.2. Types of maintenance	66
9.3. Maintenance phases	67
9.4. Conclusion	68
 10 Conclusion	 70
 References	 72
 Bibliography	 74
 Appendix A IEEE Process model for software maintenance	 75
 Appendix B Improvement and ISO9001 Certification in BVR	 76

List of Tables

Table 5.1. IEEE Software Project Management Plan	31
Table 8.1. Test Plan Outline	55
Table 8.2. Unit Test Activities	62

Acknowledgments

I would like to thank the Arab Bank administration, especially Dr. Hisham Bsar and Mr. Marwan Baalbaki for all their support and encouragement.

I also thank my advisor, Dr. Nashat Mansour, for encouraging me the past four years and letting me believe in myself, and my other committee member Dr. Ramzi Haraty for his great interest and help.

I will not forget to thank also my friends who offered their aid when I needed it especially Fatima Mahfouz and Nabil Hoteit.

Finally, I thank my Mom and Dad, my family and all my friends especially Nabil Sharbagi for bearing with me.

Thank you God.

Abstract

It is well-known that "software engineers" in Lebanon do not use standard methods. The common belief is that using standards might add to the cost of the software. This work goes against the common belief and emphasizes software quality and documentation. In this project, we propose standards for software development in Lebanon and present guidelines for using these standards. The proposed standards are based on IEEE's and ISO's standards and on personal experience. They are simplified and tailored to suit the small- to minimum-size activities of lebanese software organizations. The proposed guidelines are presented in a checklist easy-to-use fashion.

Chapter 1

Introduction

1.1. Introduction

It was my second day at work when I received a call from an employee reporting an error in one output of the software I was responsible for its maintenance. I tried to find out what was the error and asked my supervisor. He convinced me that there was nothing wrong. I tried to find some document to prove his claim, but all was in his head. Day after day, I discovered that most of our practices were unorganized, unmonitored and unplanned. I tried to find out what was wrong with applying some rule or standard but I discovered that the sudden change will need a lot of work and a lot of acceptance. Moreover, I didn't have the power.

I turned around and asked friends working in other companies and I discovered that not only where I work, but in most of Lebanon's companies, the practices of software "engineering" were far from being standardized. But I didn't quit and set for myself a set of rules and tried not to break them. I also tried to monitor myself because I always searched for quality in my work but that was not enough.

One year ago, as I was searching for a subject for my project, I thought why not build a set of software engineering standards that could fit into our Lebanese practices. I started first to find out the problem with not applying international standards in local enterprises.

1.2. Analyzing the current status

The major problems that companies were faced with:

- ✓ These procedures took a lot of "time": the country was just coming out of a war and high production was the major concern
- ✓ The management was satisfied and didn't accept any change in the current practices
- ✓ There was no professional personnel to apply such standards
- ✓ There was no professional personnel to monitor this application
- ✓ There was a lack of training
- ✓ The international standards were too difficult to apply in one step

1.3. Proposed solution

The solution for such problems was to design a set of standards that was easy to implement, organized in a checklist or plan fashion, and formed of multiple short phases. Easy to implement, because there is an interesting demand on high quality products to compete with foreign companies. Organized in a checklist fashion, because management needs to understand and monitor the flow of the process. Formed of multiple short phases, because there is tendency that "engineers" get frustrated when they follow long and compact phases.

1.4. Thesis organization

This thesis or better-called manual can be read in two ways. To begin with, the second chapter alone could be taken as a baseline of standards and fit into the framework of any software company as convenient. The other way is to adopt the same chapter as an overall plan to the whole process of developing software, then follow the guidelines and recommendations of the following eight chapter that describe in detail the main phases of the process.

The proposed set of standards for Lebanese software systems development is formed of ten phases, each with a set of input, output and procedures. Each phase will have a goal. The exit criteria from a phase will be the satisfaction of its goal. The phases are short and form a checklist that is easy to follow and monitor. They are designed in a universal fashion in order to fit any paradigm or development environment. Moreover, the defined standards have the tendency to grow as the market's maturity grows. The ten "commandments" phases of software system development are then:

Phase One :Preliminary Specification and Planning

- i)Preliminary Requirement Document Creation
- ii)Preliminary Budgets and Schedules

Phase Two :Requirement Specification Phase

- i)Informal Specification Requirement
- ii)Semiformal/Formal Specification Requirement

Phase Three :Feasibility Prototype (Optional)

Phase Four :Planning (Detailed Budgets and Schedules)

- i)Software Development Plan
- ii)Documentation Plan

Phase Five :Software Design Phase

- i)Software Design Document
- ii)Detailed Test Plan

Phase Six :Implementation

- i)Code and Unit Test
- ii)Documentation
- iii)Test Design

Phase Seven :Testing

- i) Testing Result Report
- ii)Updated Software Design Document
- iii)Updated Software System

Phase Eight :Release Plan Phase

Phase Nine :Release Phase

Phase Ten :Maintenance Phase

In the second chapter, I will explain each step's description, purpose, output or product, and end conditions. I will also list the steps of each phase, their input, output, the teams involved in their accomplishment and the tasks done during the step. In chapters three to nine details of the main phases listed in chapter two will be given. The standards or as I better call them directives and guidelines can be used along the above cycle plan to accomplish higher transparency of the entire process.

Chapter 2

Phases Overview

2.1. Introduction

In this chapter we will concentrate on each phase of the software development process and its steps. All phases will have the same components: a goal, an output, procedures and an end condition. Moreover, each step will have: a goal, input, the responsible personnel, the necessary tasks to fulfil this step and an output. An important issue, that is not to be forgotten, is the assigned personnel for the accomplishment of each step: everybody has to know what s/he is responsible for. Moreover, high communication, collaboration and corporation between teams should be the main objective.

Therefore, we must describe, in addition to the development phases, the management configuration that will be responsible for the production of software systems where no conflict between parties will occur. This configuration should be formed mainly of six teams: the customers, the marketing department, the project management, the developers, the testers and the quality assurance team. The sales management and technical publication groups can also be considered to assist the development process. By assigning to each person specific tasks, the job of every employee can be easily monitored and the relation between employees will be well defined. Lesser conflicts between employees will be recorded.

2.2. The phases of a software life cycle

The following sections describe the proposed phases in detail.

2.2.1. Phase One: Preliminary Specification and Planning

Goal: Define the problem according to user's needs, developing a solution strategy and establish preliminary cost estimates for system development.

Output: The following documents should be delivered at the end of the phase: Preliminary Requirements Document (PRD) and Preliminary Budgets and Schedules (PBS).

End Conditions: This Phase is terminated after all documents have been reviewed and approved by the marketing department.

Description: This is a repetitive phase where the marketing personnel and engineering managers are involved. The needs of the users are carefully studied, the product functionality is drawn and the technical constraints of the product are created. Moreover, engineers start estimating the budget and schedule and decide on the resources needed. This phase is finalized when a compromise is reached between the requirements and resources.

Step 1. Preliminary Requirements Analysis

Goal: To identify the product's users, functionality, features, and technical constraints

Input: Market needs, surveys, customer interviews

Teams: Marketing and Purchaser and Engineering Managers and Sales

Tasks:

1. Interview the purchaser
2. Create a preliminary requirement document
3. Publicize the document for review
4. Collect feedback
5. Review the document with the purchaser

6. Resolve issues
7. Update the document
8. Repeat steps 5 through 7 until agreements on requirements, deliverables, cost and schedule are reached
9. Obtain final signature approval

Output: Preliminary Requirement Document (PRD)

Step 2. Preliminary Budgets and Schedules

Goal: To estimate the costs and time required to accomplish the entire project. These estimates will help in the accomplishment of a detailed and clear PRD

Input: PRD, development costs, and measurement from similar products

Teams: Engineering managers and Marketing department

Tasks:

1. Review the PRD
2. Estimate the time to accomplish all the functions described in the PRD
3. Review the measurements with the marketing department
4. Amend PRD according to cost and time
5. Repeat step 1 through 4 until ready for publicize final estimates
6. Finalize Preliminary Budgets and Schedules

Output: Preliminary Budgets and Schedules Estimates

2.2.2. Phase Two: Requirement Specification Phase

Goal: This phase's aim is to produce a clear and intelligible document to be used as the source of all information for the design team. The document should be clear because the customer wants to understand what s/he's paying for and intelligible because s/he's not a computer expert. It should incorporate constraints that the product has to satisfy and the acceptance criteria.

Output: The following document should be delivered at the end of the phase: Software Requirement Document (SRD).

End Conditions: This phase is complete when the document is produced, review with the marketing department and approved by both the marketing department and the project managers.

Description: This phase is one of the most important phases since its output will be the input of the design phase. This document will be used as the contract between the customer and the production house. It should include all the constraints that the product must satisfy: deadline to deliver the product, the parallel run phase, the portability, the reliability and the acceptance criteria. These conditions can be tested by using an optional phase (phase three), where a prototype is developed to reflect an image of the final product. The specifications will be written in both informal and semiformal/formal fashion because of the involvement of the customer and developer for later phases.

Step 1. Software Requirement Document

Goal: To identify in detail all input, output, and different resources (hardware and software) required to produce a software that will satisfy the previously defined functionalities and will meet the technical constraints

Input: Preliminary Requirement Document

Teams: Engineering department, Marketing and Software Quality Assurance

Tasks:

1. Create the SRD
2. Distribute the SRD for review
3. Collect reviews and identified issues
4. Find solutions for issues
5. Update the SRD
6. Repeat steps 1 through 5 until no amendments are made
7. Deliver document to the public

Output: Software Requirement Document

2.2.3. Phase Three: Feasibility Prototype (Optional)

Goal: To draw a picture of the product's offerings and to gain familiarity with the new interfaces, tools, platforms and algorithms.

Output: The following documents should be delivered at the end of the phase:
Software System Prototype (SSP).

End Conditions: This Phase is terminated when enough data has been produced in order to be reviewed by the engineering management. The prototype results will be used in further phases when estimating the time spent on coding a certain requirement.

Description: A subset of the requirements is chosen and coded. This approach help in resolving issued raised between the customer and the developers.

Step 1. Feasibility Prototype

Goal: To give a hint of what the product will look like after its completion and help in reviewing any last minute error in the SRD

Input: Software Requirement Document

Teams: Engineering department and Marketing and Software Quality Assurance

Tasks:

1. Chose a subset of all requirements
2. From a design of each requirement
3. Code the design
4. Demonstrate prototype
5. Review customer/marketing/management remarks
6. Repeat steps 1 through 5 until approval
7. Collect time spent on the design and coding and measure lines of code created (important data for the following phase)

Output: Prototype results, different metrics

2.2.4. Phase Four: Planning Phase

Goal: To plan in detail all the coming phases of the software development process from the view point of the resource use and the time needed to accomplish the job. Moreover, it is to identify the technical documents that are required for delivery along the product.

Output: The following documents should be delivered at the end of the phase: Software Development Cost, Software Schedule Plan and Software Documentation Plan.

End Conditions: The software engineers, the software manager and the marketing department approve the deliverables.

Description: This phase, like testing, must continue throughout the entire software and maintenance process. It is one necessary phase that organizes the core phases of the process (design, coding and testing). It is made to estimate the duration and cost of producing the software product. It also aims at the description of the work to be done, the resources that will be used and the money to pay it all. It could also describe the training requirements needed for both the developers and the users. It finally describes the documentation standards that will be used to produce the user manuals and the data needed for the maintenance phase.

Step 1. Software development cost and schedule plan

Goal: During this phase, different techniques are used to predict the resources and time needed to accomplish the requirements described in phase two

Input: Requirement Specification Document

Teams: Project Management and Marketing Department

Tasks:

1. Estimate time needed for different phases
2. Determine the start time of each phase
3. Build the organizational structure
4. Determine the staff and resources needed

5. Identify monitoring and control mechanisms
 6. Prepare a draft of the development plan
 7. Review plan with marketers
 8. Update plan if necessary
 9. Repeat steps 1 through 8 until satisfactory
- Output:** Software Development Cost and Schedule Plan

Step 2. Software Documentation Plan

Goal: In addition to the usual planning needed to lead the way for later phases, documentation is a major activity that should be planned ahead. Because documentation can help the user to understand the software product and because maintenance needs resources, different documents are needed

Input: Preliminary Requirement Document and Software Requirement Document

Teams: Technical Publication, Development engineer, Test engineers, and Marketing

Tasks:

1. Identify what document are to be created
2. Identify what sources are to be used
3. Create outline for such document
4. Create plan for the development of such document
5. Place rules to control the version of different document

Output: Software Documentation Plan

2.2.5. Phase Five: Software Design Phase

Goal: To translate the software requirement document in a model that will be used for implementing the functionalities of the desired product. It should explain the architecture of the software system, how errors will be handled and most importantly the database if applicable. Moreover, this phase should generate a set of test cases and a test strategy that will be used in the coming testing phase.

Output: The following documents should be delivered at the end of the phase:
Software Design Document and Software Test Description.

End Conditions: This phase is terminated when the project engineering leader has approved the Software Design Document and when the System Test Description has been approved by the marketing department, the project manager, and engineering leaders.

Description: At the end of this phase we will be able to view an architectural detailed design of the software system. Moreover, the user interfaces, the database design and the error handling design will be available for later phases. Detailed test case description will also be described.

Step 1. Software Design Document

Goal: Design the software product architecture, implementation, error handling, user interface, and database design

Input: Software Requirement Document

Teams: Software Engineers and Quality Assurance

Tasks:

1. Decompose the system into subsystems
2. Describe the input, output, and functions of each subsystem
3. Decompose each subsystem or module into programs and procedures
4. Describe the input, output and design of each program/procedure
5. Design how errors will be handled
6. Design the database
7. Use the prototype developed earlier (or) create prototype for windows/screens to interpret the current design
8. Demonstrate to marketing department, project manager, quality assurance and customer
9. Create document
10. Review document, identify and resolve issues
11. Correct document
12. Repeat steps 1 through 11 until satisfactory
13. Release document

Output: Software Design Document

Step 2. Software test description

Goal: Guided by all documents, the purpose of this step is to plan in detail how the software will be tested to satisfy functional requirements and technical constraints

Input: System Requirement Document, Software Design Document and Software Documentation Outline

Teams: Quality assurance, Development engineer, product manager, and documentation manager

Tasks:

1. Identify the hardware and software configuration for the test environment
2. Describe the installation procedure and prepare alternatives
3. Describe test cases
4. For each test case describe: input/output, test procedures, expected results, assumptions and constraints
5. Review the output document for issues identification
6. Identify issues
7. Resolve issues
8. Publicize test plan

Output: Software Test Description

2.2.6. Phase Six: Implementation Phase

Goal: To translate the Software Design Document into code, provide different and enough documentation and create test case.

Output: The following documents should be delivered at the end of the phase: Code, Documentation and Test Design.

End Conditions: This phase is terminated after the development team has reviewed the code and the quality assurance approved the documentation and the test design.

Description: During this phase, all design issues are transformed into code and different modules of the system are test individually. Later, the system as a whole will

be tested to insure that no errors occurred during its integration. The system test cases described in the previous phase will be created and documentation issued for review.

Step 1. Code and Unit Tests

Goal: Implement the software design and prove that the modules created fulfil the functionality of their design

Input: Software Design Document

Teams: Developers, Quality Assurance, development leaders

Tasks:

1. Implement software design
2. Perform code walk-through while preparing test cases
3. Run tests
4. Correct errors
5. Prepare Documentation
6. Repeat steps 1 through 4 until satisfactory

Output: System modules and Documentation

Step 2. Integration Test and Test Cases Creation

Goal: To integrate individual modules into one unit and test it preliminary. To create test cases for later phases

Input: System modules, Software Design Document, Software Test Description and Software Documentation Outline

Teams: Development engineer and Quality Assurance

Tasks:

1. Link individual modules into a single unit
2. Test the system as a whole
3. Review system test plan and Software Requirement Specification
4. For each test case create data
5. Review data, identify and resolve issues

Output: Test Plan Design

2.2.7. Phase Seven: System Testing Phase

Goal: To issue a report on the degree to which the software meets its functional and performance requirements.

Output: The following documents should be delivered at the end of the phase:
Updated Software Design Document and Updated Software product.

End Conditions: This phase is terminated when all test cases have been tried and all documents have been reviewed especially the Software Design Document.

Description: During this phase, the product is tested against the customer requirements. Functionality, performance and reliability are measured carefully. Bugs found are fixed and documentation is corrected to meet the final software product.

Step 1. Execute System Test Cases

Goal: To measure how much the software product adheres to its functional and performance requirements

Input: System Test Plan and Design, in addition to Previous documentation and Software System

Teams: Software Quality Assurance, Development Engineers and Technical Publication

Tasks:

1. Install the system on the test hardware
2. Test each test case
3. Issue report
4. Review report with the engineers
5. Update test cases, documentation and/or product as necessary
6. Repeat steps 1 through 5 until satisfactory
7. Update the Software Design Document

Output: Software, software documentation, test report

2.2.8. Phase Eight: Product Release Plan

Goal: To plan each step of the release phase. Installation strategy, training, and parallel run should be carefully described.

Output: The following documents should be delivered at the end of the phase:
Product Release Plan.

End Conditions: This phase is terminated when the marketing department, the engineering manager, the quality assurance team and the customer have approved the plan.

Description: During this phase a plan of how the software will be installed and running will be designed. Moreover, considerations should be taken for the training of the users before completely relying on the software system.

Step 1. Prepare Release Phase Plan

Goal: To prepare a plan that will guide the appropriate personnel for the training of the users, the installation of the software and the preliminary run phase.

Input: User Manual and Software System

Teams: Software Engineers, Marketing Department, Customer and Software Quality Assurance

Tasks:

1. Plan how the users are to be trained
2. Set schedule for installation
3. Set conditions for end of parallel run
4. Review plan and correct if necessary

Output: Software Release Plan

2.2.9. Phase Nine: Product Release Phase

Goal: To install a running software product that will fully replace the current applications.

Output: The following documents should be delivered at the end of the phase:
Evaluation Reports.

End Conditions: This phase is terminated when the user conditions for ending the parallel run period are satisfied.

Description: During this phase, users are trained to work on the new system. The system is installed as planned earlier and a parallel run for a predetermined period is run. Finally only system output is adopted.

Step 1. Release Phase

Goal: To ensure the installation of a running system that meets the customer's requirements

Input: Software Release Plan, User manual and Software

Teams: Training Department, Development Engineers and Quality Assurance

Tasks:

1. Train users on new software
2. Install software on real site
3. Start replacing old practices with new ones
4. Rely fully on the new Software

Output: Evaluation reports can be generated by all parties

2.2.10. Phase Ten: Maintenance Phase

Goal: To monitor the evolution of the software over time after the product has been handed over to the client.

Output: The following should be delivered at the end of each occurrence of this phase: Maintained Software and Maintained Documentation.

Description: Different types of maintenance can be done to correct any residential faults, improve the effectiveness of the product or to respond to changes in the working environment.

Step 1. Maintenance Request Process

Goal: To respond to any maintenance request report and process update to the software or the documentation previously delivered

Input: Maintenance Request Report

Teams: Marketing Department, Development Engineers and Quality Assurance

Tasks:

1. Accept any maintenance request report
2. Identify the type of the correction required (corrective, perfective or adaptive)
3. Issue priority for authorized changes
4. Process changes
5. Review product and ensure maintainability
6. Issue new product after planning and testing has occurred as described before

Output: Maintained product (Documentation and/or software)

Chapter 3

Purchaser/Supplier Contract

3.1. Introduction

A clear and well-defined process description can help in determining the relation between the customer and the supplier and most importantly the relationships between the personnel responsible for the system development. The approach described in the second chapter is somehow procedural and dictates steps to be taken to produce a software system according to a transparent and audible process. In this chapter we will provide recommendations for building the first output of the first phase of the development process: the purchaser and supplier contract.

3.2. The goal of the Purchaser/Supplier Contract

The following issues must influence the approach of the supplier and purchaser management:

- ✓ To understand the issues that the contract will cover
- ✓ Determine each organization's responsibilities
- ✓ Always measure and calculate to minimize risks
- ✓ Agree on the terminology of the contract
- ✓ Agree on the acceptance criteria
- ✓ Determine the procedure that will be followed when a requirement is to be changed
- ✓ Determine the procedure that will be followed when bugs are found, reported and fixed

3.2.1. Covered issues and responsibilities

Both the supplier and the purchaser must agree on procedures to review the contract and its content. This is done to ensure that each organization checks if it is capable of fulfilling the conditions listed in the document. By not realizing the level of commitment required to meet their respective obligations, many software projects are not delivered or exceed their original cost and schedule. Therefore, every issue identified during meetings should be recorded and reviewed by both organizations. Moreover, the supplier's management should follow an internal auditing procedure to ensure that every participant in the production process understand what s/he's supposed to do.

The relation between the supplier and the customer should be clearly documented in addition to the reviews of the purchaser and supplier and different development progress reports.

3.2.2. Risk minimization

Important attention should be paid for the identification of risks. When product development and maintenance risks are properly presented to the senior management in quantifiable terms related to budget and schedule, senior management can then make the proper business decisions needed to minimize these risks. This is done when a request to the project and product managers is made by senior management, who is often unaware of the technical, schedule, or budget risks involved with software projects. In their turn, the project and product managers will be responsible of identifying the assumptions made concerning the proposed work, the risks involved, and the plan to minimize risks.

3.2.3. Terminology of the contract

Most importantly, both parties must agree on the terminology that will be used when communication is done. Most of the times, it is the language of the purchaser. The way to ensure this, is to keep a dictionary of terms that will be used during the whole project. New comers on board the project will also use this dictionary during the development stage.

3.2.4. Acceptance criteria

In addition to the previously stated guidelines, acceptance criteria, handling of changes in purchaser's requirements during the development and handling of problems detected after acceptance must be stated in the contract.

Although acceptance testing is nearly the final stage of the product testing, the purchaser must prepare acceptance test cases to prove whether or not the product meets the functional and performance requirement.

Because the customer is not ready to accept the product or has built invalid test cases, problems could occur when decision is to be made. Moreover, the supplier will gain from the purchaser's definition of acceptance criteria and that is for two reasons. First, the purchaser would have understood the type of commitment needed toward the product's acceptance and ownership and will be supporting the testing phase. Second, there will be fewer arguments over whether the acceptance test is valid or if there is a problem with the product, because both parties decided together on the exit criteria.

For these reasons, it is recommended that the purchaser submits an acceptance plan that identifies test cases, test case description, test environment, and schedule. The supplier's job will be to review the acceptance test cases for accuracy.

3.2.5. Requirement change procedure

On the other hand, many changes can occur to the requirement during the product life cycle. A serious problem can occur when budgets and schedules are not updated to reflect changes in the requirements resulting in the projects being late or over budget. Therefore, conditions and authorized representatives from both parties should be identified as a part of the contract.

3.2.6. Bug reporting procedure

Moreover, every delivered product contains errors or bugs and should be corrected as they come visible to the purchaser. That is why a support plan should be designed to support this need. The supplier should be ready to provide resources to handle problems and set a time frame for solving problems.

Procedures should be made ready by the supplier to satisfy bug reporting (help desk, supplier's customer service), bug tracking (automated tools, manual paper work flow), bug prioritizing and processing (configuration control board), regression testing (test made after any change done to the code) and version control.

3.3. Conclusion

Finally, an important guideline could be added to the previous set of guidelines and that is the acceptance of the purchaser-taking role in requirement

specification, installation and acceptance. The purchaser must participate in the writing of the requirement specification, the installation procedure and the acceptance criteria of the product. If this condition is not met, it is most probable that the product will fail in meeting its technical, budget, and schedule goals. Therefore, it is recommended that the contract identifies these issues and identify key people from both the purchaser and supplier organizations that are to work together to ensure that the product is specified, installed and acceptance test correctly. Schedules for meetings and deliverables should be agreed on and listed in the contract. Facilities, tools and software items to be provided by the purchaser should be listed in the contract [Kehoe and Jarvis 1996].

Chapter 4

Requirement Specification Document

4.1. Introduction

This chapter is intended to describe the recommendations that should be followed when the second phase of the software development life cycle is to take place. The first translation of the purchaser's demands will produce a document that will identify the software requirement specifications. This document is not intended to specify the methods, approaches, or tools for developing the software system. It will draw in detail the specific functional requirements, external data flows, algorithms, error handling, and technical constraints.

4.2. Issues addressed by the document

This document can be implemented and tested and will be used as an input to the core phase of the entire process: the design phase. Each implementation requirement is studied: what it does, how does it start, its inputs and outputs, and the error handling. Moreover, user operations are described using all needed information and the error handling strategies are planned. In other words, issues that should be identified are:

Functionality: what the software is to do.

External interfaces: the inputs and outputs of the system and the platform on which the system will be installed.

Performance: for each software function, its speed, availability, response time and recovery time will determine its performance.

Attributes: portability, correctness, maintainability, and security.

Design constraints imposed on an implementation: any condition that could affect the implementation like the standards used, the language, the resource limits, the operation environment should be identified.

4.3. Characteristics of the document

All the issues addressed by the writers of the software requirements specifications document should be written in a correct, unambiguous, complete, consistent, ranked for importance and for stability, verifiable, modifiable and traceable fashion [IEEE std 830-1993].

4.3.1. Correctness

As defined by the Institute of Electrical and Electronics Engineering, software requirement specification is correct if, and only if, every requirement stated there in the RSD is one that the software shall meet. The "only" problem is that correctness is very difficult to prove, since there is no tool or procedure available to do the job. Traceability maybe the easiest way to assure some kind of correctness.

4.3.2. Unambiguity

Every requirement should have one meaning or interpretation. Therefore, it is recommended that every characteristic of the final product is described using a single unique term. Most requirement specifications are written in natural language. In nature these languages are ambiguous and could be reviewed by a third party to identify ambiguity. Other languages could be the solution but the choice of semi-formal or formal language should be done carefully. Such languages are difficult to learn and especially these formal methods which are not yet stable and not available in object oriented and non-procedural paradigm. As a compromise and a way of

communication between the supplier and purchaser, it is recommended that a representation tool be used to support any requirement method and language. When using any of these approaches it is best to retain the natural language descriptions. That way, customers unfamiliar with the notations can still understand the software requirement specifications.

4.3.3. Completeness

A Specification Requirement Document is complete if, and only if, every elements of any requirement is determined prior to the next phase: functionality, performance, design constraints, attributes and external interfaces. A document that contains the phrase "to be determined" is not complete. Everything must be labeled and referred to and all definitions and terms must be complete.

4.3.4. Consistency

The requirement specification should not contradict any other document. It should be submitted to an internal consistency, where "no subset of individual requirements described in it, conflicts".

4.3.5. Importance and/or stability

Any requirement should be labeled to identify its degree of stability and the degree of necessity. It will be easy to understand any change for an unstable requirement rather than a stable one. Moreover, the degree of necessity will distinguish between an essential, conditional or a potential requirement.

4.3.6. Verifiability

Any statement in the document should be verifiable. Therefore, they should use concrete terms and measurable quantities. Ambiguous requirements are not verifiable; therefore, any method that cannot be derived should be revised or removed.

4.3.7. Modifiability

The software requirements specifications can be modifiable if any change to the requirement is easy to do and will leave the document complete and consistent. In order to make a document modifiable, it is recommended that the document is organized in a coherent and easy to use fashion. Redundancy should be avoided because if one same requirement is mentioned more than once and is changed in one place then we lose the flexibility of modifying the requirement.

4.3.8. Traceability

A document is traceable if it is easy to find the origin of each requirement in the document (backward traceability) and if it is easy to refer the requirement in the rest of the development cycle (forward traceability). Forward traceability is very essential when the software product moves into implementation and maintenance. As code and design documents are modified, it is important to trace back all the requirements that may be affected by those modifications.

4.4. Difference between the Design Document and RSD

Moreover, these are some considerations others than the characteristics of a RSD that should be pointed out, and they are the commitment of both parties to

accomplish the job, the document's subjectivity to evolution and the distinction between design document and the RSD.

4.4.1. Responsibilities of the customer and supplier

To begin with, the RSD is a document that should be jointly prepared. This is true because neither the supplier nor the purchaser can write this document alone. The purchaser does not understand the process of design and development well enough to produce a usable RSD. Moreover, the supplier often does not understand the customer's problem and the environment it will be used in well enough to satisfy alone the customers' requirements. One important practice to be agreed upon is the style, language use and techniques of writing the document.

4.4.2. Incomplete requirements indication

In addition, the document should be written in a way that will make it easy to evolve with the revolution of the development process. An incomplete requirement should be labeled as such in order to be revised in later stages. Such requirements should be audited and trail of such changes should be recorded.

4.4.3. Content of the document

Finally, there should be an understanding of what the RSD should contain. Most importantly it should not embed any design feature. When the writers are identifying a required design constraint they are not projecting a specific design like the partitioning the software into modules, allocating functions to the modules, describing the flow of information or control between modules, and choosing data structures.

4.5. Conclusion

In summary, the supplier should have a complete, unambiguous set of functional requirements that include all aspects necessary to satisfy the purchaser's needs, that are testable, that are developed in close co-operation with the purchaser (if not provided by the purchaser), subject to document control or configuration management, and that specify all product interfaces. The requirements specification should be completed and approved before the development activities start [Schmauch 1994].

Chapter 5

Planning the Development Process

5.1. Introduction

Following the requirement specification document, a prototype for the resolution of "fuzzy requirements" can be prepared to correct any errors in the produced documentation. In this report, we will skip this third optional phase and move to the planning phase and the recommendations that should be followed in order to produce the most appropriate budget and schedule estimates.

Before coding can be started, it is necessary to plan in detail the entire software development effort. Planning, like testing, must continue throughout the software development and maintenance process. Its aim will be to create plans for the software development phases and the documentation techniques that will be used. For each phase, its inputs, outputs, schedule and resources will be identified. Moreover, auditing procedures and methods to test the status of progress of the development will be described. Finally, the tools and methods to be used during the rest of the process will be chosen.

5.2. Definition of Planning

Different definitions could be given to the planning phase, but all share the same components: the work to be done, the resources that will be used to do it, and the money to pay it all. The major resources required are the people who will develop the software, the hardware on which the software will be running and the support

software such as operating systems and text editors. We must note that the rate of resources usage will start small, climb rapidly to a peak in the middle of the life cycle, then decrease slowly till the final phases. A typical plan should have a similar construct to the framework of the software project management plan of the IEEE [Schach 1993] presented in table 5.1. As the document is clearly defined, the rest of the development process will be a direct application of the plan.

Table 5.1. IEEE Software Project Management Plan

IEEE Software Project Management Plan

- 1.1 Introduction:
 - 1.1.1 Project overview
 - 1.1.2 Project Deliverables
 - 1.1.3 Evolution of the Software Project Management plan
 - 1.1.4 Reference Materials
 - 1.1.5 Definitions and Acronyms
 - 1.2 Projection Organization:
 - 1.2.1 Process Model
 - 1.2.2 Organizational Structure
 - 1.2.3 Organizational Boundaries an Interfaces
 - 1.2.4 Project Responsibilities
 - 1.3 Managerial Process
 - 1.3.1 Management Objectives and Priorities
 - 1.3.2 Assumptions, Dependencies and Constraints
 - 1.3.3 Risk Management
 - 1.3.4 Monitoring and Controlling Mechanisms
 - 1.3.5 Staff Plan
 - 1.4 Technical Process:
 - 1.4.1 Methods, Tools, and Techniques
 - 1.4.2 Software Documentation
 - 1.4.3 Project Support Functions
 - 1.5 Work Packages, Schedule, and Budget
 - 1.5.1 Work Packages
 - 1.5.2 Dependencies
 - 1.5.3 Resource Requirements
 - 1.5.4 Budget and Resource Allocation
 - 1.5.5 Schedule
-

5.3. Steps of the Planning Phase

5.3.1. Choice of a template

The development plan should include a plan of each activity of the life cycle of the product. It should start first by defining the aim of the work and that is the satisfaction of the customer's requirements then the engineering process to be used to do the work. That is why it is recommended to use a template, similar to the IEEE framework presented earlier, in order not to forget any point of the plan. Any chosen template should include in its first section a definition of the project that should be understood by the various organizations and levels of management engaged in the development process. After this step, everybody will understand its responsibilities and work on identifying the resources they will use. Before defining roles, responsibilities, control strategies and communication methods no schedule, budget, or cost estimates can be done. Moreover, if any subcontractor is to offer support to the supplier, this latter one should ensure that the sub-contractor's efforts are included in the final product's effort.

5.3.2. Choice of a process

On the other hand, the supplier's aim at quality should lead to the adoption of a defined and documented engineering process. Whatever the project is, development goes through four main phases: analysis, design, code and test. Although this is true, the plan should include a choice of a precise cycle that will direct the development process. This choice will help in the identification of time and money needed for each phase.

5.3.3. Schedule/Budget

In addition to the definition of the project, the organization of the project resources and the development phases, the development plan should cover the project schedule identifying the jobs to be performed, the resources and time required for each job and any interrelationships between jobs. A phase is formed of a set of tasks to be performed. Each task has a set of inputs and outputs. By identifying the inputs and outputs of a task and its steps, a schedule and budget can be determined for each task. The project schedule and budget will then be the summation of all the tasks schedules and budgets.

5.3.5. Update of the plan

One important thing should be kept in mind and that is the plan is not a static document. It should always be updated especially at the end of each coming phase and before any new activity starts. Changes and problems often occur during software development. The problem is that the plan is not updated to reflect the new status of the project. Therefore, a mean to ensure this is to review regularly the plan and use a metric that is easy to keep track of and to compare with the predefined plan.

A good development plan then should include an identification of the development phases to be carried out and their attributes, define how the project will be managed and identify the methods and tools that will be used during development.

5.4. Phase description

The plan should define a disciplined process or methodology for transforming the purchaser's requirement specifications into a software product. This may involve

the division of work into phases, the identification of development phases to be carried out, the listing of required inputs for each phase and the analysis of the potential problems associated with the development phases and with the achievement of the specified requirements [Kehoe and Jarvis 1996].

5.5. Verification process

Main concerns should be held for the verification process and risk management. Verification should assure that the development process is following the procedures and task of the predefined plan. A set of risks should always be reviewed. The aim of the risk management plan is to study the problems that could occur and register new ones that could arise. As old risks may disappear new problems could occur and should be minimized.

5.6. Management responsibilities

In addition to the identification of the engineering process to be followed, the development plan should state how management of the project should be done. The management of the process should include a schedule for each subsequent phase, a mean to control progress, a definition of each job and the responsible for its accomplishment, and most importantly the definition of inter-organizations methods of communication.

5.6.1. Schedule of subsequent phases

To begin with, every task of the development process needs a set of inputs to produce a set of deliverables that will be used by other tasks. That is why it is very important to identify the schedules for these deliverables. As these schedules are

identified, the resources needed to create the deliverables within the predefined time period can be identified. The identification of such external dependencies will help in determining the ones that are beyond the control of the project management and the ones that could affect the quality of the overall project schedule.

5.6.2. Process control

Moreover, in spite the adoption of an SDP and a life cycle, something can go wrong. Therefore, there should be a way to review the status of various jobs to check if they conform to the SDP. The supplier management should regularly check progress against the original document and react accordingly.

5.6.3. Responsibilities

In addition, every organization should know its responsibilities, the resources it will use and the work it will do. That is why it is better to have the SDP identify these issues in order to be signed off by the managers of the various organizations responsible of implementing the plans.

5.6.4. Inter-organization relations

Finally and most importantly, various organizations need to communicate with each other. If there are no standards for the communication between different groups misunderstanding and low commitment may result. Therefore, techniques for communication and schedules for regular meetings should be drawn in the software development plan.

5.7. Methods and tools definitions

Parallel to management, development methods and tools must be identified to ensure that all activities are carried out correctly. Methods, rules, practices and conventions for development are not to limit personnel creativity. The management should create them as a base line of practices to be used uniformly and to enhance productivity. This choice will include the definition of the process steps, the programming standards, design standards and documentation. Quality, budget, and schedule will influence these choices.

On the other hand, tools and techniques for development should be listed in the plan. CASE tools, methodologies and techniques should be identified. Possible need for training should be budgeted and scheduled. It will be wrong to schedule training for the user only. Not only users need training but also members of the development teams need to be trained. Training could be needed to software as well as hardware.

5.8. Purpose of the plan

When writing the plan, one should keep in mind that this document will be used for the following purposes:

- ✓ Simulate how the project will be carried out
- ✓ Coordinate and communicate
- ✓ Increase participants' motivation
- ✓ Control the project
- ✓ Satisfy requirements
- ✓ Avoid problems
- ✓ Record the choice between options [Gilb 1988]

5.9. Conclusion

As a conclusion, we should always keep in mind that the plan should be the document that will make the development phase a transparent process. The project will be defined for all participants in the development life cycle. All phases will be identified along all inputs and outputs needed for each phase. Moreover, schedule and resources for each phase will be estimated using metrics that are easy to handle and modify in case of later updates. The plan should identify also procedures to control and determine the status process. Tools and methods should be known in advance and appropriate training should be given to the development and support teams. Finally, review, audits and testing procedures for each phase should be described.

Chapter 6

Software Design Document

6.1. Introduction

We have learned that the requirement phase's aim is to produce the software requirement document that explains what the product is to do. The aim of the design phase is to produce a document that explains how the product is to do it.

Similarly to the International Standardization Organization (ISO), we can consider this phase as the technical kernel of the software product and the phase that directly dictates the quality of the produced system. The output of this phase will specify how the product is to meet the requirements and it will influence all the steps of subsequent process (implementation, testing, maintenance).

The software design document is a translation of the software requirement specification. The output of this translation is a model that will be used when implementing, testing and maintaining the system and should always be updated to reflect any change in the product. It should reflect a precise image of the architecture of the software system, guide the implementation phase, explain how errors are to be handled, describe how the user will interact with the product, and most importantly explain how the database, if applicable, will be constructed.

6.2. Cost/Time-friendly design

The design should be cost-friendly and time-friendly. A cost-friendly design is the design that is made to cost. That is it does not include costs of effort, materials and tools, but cost of production or service environment in which they are to be made, costs of testing, lifetime costs to use, and so on. Time-friendly in the sense that the more useful time spent in this phase the better it will be for the next phases. Since, time goes by and cannot be reclaimed, we must be sure that we have taken into account everything. Till this phase staffing costs are low; therefore, we should take into consideration that it is better to stay as long as it is necessary in this phase and prepare everything for the upcoming processes.

6.3. Evolutionary design approach

The best solution is to use an evolutionary design approach: break everything down. Design and produce increments. By breaking the work down into chunks, potential problems can appear soon. When break down is used, work in a number of parallel and overlapping streams to reduce product cycle times can be done. In other words, reduction of risk can be reached by reducing complexity [Holdsworth 1994].

6.4. Coupling and cohesion

Design is not easily submitted to mechanization because we unfortunately do not have an exact set of rules, so the problem becomes one of how best to support software designers in their decision making. Apart from proposing the use of data flow diagrams (DFDs) and module hierarchy charts, two criteria can be used to evaluate design and is module coupling and module cohesion. The designer should seek to maximize module cohesion and minimize module coupling.

Module cohesion is the strength of the relationship between the elements within a module. The strongest type of cohesion is when all the module elements combine to accomplish a "single specific objective". The weakest type of cohesion is when no meaningful relationship between the internal module components is present. Changes to such modules will be difficult. The module will be more difficult to understand and, since the likelihood of anyone requiring the exact same set of functions is low, the probability of re-use is correspondingly low.

Module coupling is the strength of association between a module and its calling environment.

We can minimize coupling by:

- ✓ Letting the module call refer to the whole module and not any internal part
- ✓ Making communication between modules through parameters
- ✓ Lowering the number of distinct data objects communicated
- ✓ Lowering the control flags communicated [Shepperd 1995]

6.5. The document

6.5.1. Definition

The proposed document production follows the directions of the IEEE and ISO. The first standard stresses on the division of the problem into objects and the second on the definition of modules and databases. The advantages of both systems were gathered to form a realistic and applicable standard.

A model of the expected software system is described in the design document. All the information needed for the coding of the system should be available in the model representation. The best solution is to divide the entire system into design

objects (DO). Each object has a set of predetermined attributes and the relation between any two objects must be defined clearly and must follow a set of rules.

6.5.2. Design object

A DO as its name indicates is an entity (element) of a design that is unique by its structure and functionality. As the process is under way the system requirements are decomposed to produce more and more DOs. The aim of this decomposition is to divide the system into separate objects that can be implemented, tested and maintained without effecting other objects (high cohesion and low coupling).

Many factors can influence the design process like the system's complexity, the used design technique or the programming environment. These factors can produce object designs that describe systems, subsystems, data stores, modules, programs, and processes. Naturally, objects can be different but every object has a name, a purpose and a function. Moreover, objects can communicate through interfaces or shared data. The characteristics of such objects are described by DO attributes.

6.5.3. Design Object Attributes

Each DO we define has certainly some characteristics or properties called design object attributes. Such attributes form the identity of the DO. The attributes are to answer any question upon the nature of the object and should help in the building a profile for each object. Because of their necessity to all software projects, their harmful effect on the development phase if they were incorrectly described, and the intrinsic type of information they offer, a minimum set of such attributes must be adopted. This set of attributes should be defined for all the DOs that could be derived.

Sometimes, a value of null could be assigned to some attributes. Moreover, new attributes can be defined to include new design descriptions. These attributes can be summarized in the following subsections.

6.5.3.1. Identification

Logically, to reference any object a name is needed. Therefore, every name will refer to a unique object and no two objects should have the same name. This attribute will also help in the tracking of all DOs.

6.5.3.2. Type

Type is a description of the kind of object. This attribute will identify the nature of the object. Whether it is a subprogram, a module, a print file or a data store, this attribute will help in grouping objects under a same type-group and will help in the classification of different objects.

6.5.3.3. Job

This attribute is a description of the purpose and the function of the object entity. The job attribute will explain why the object has been created and the conditions under which the object will be used. In addition, this attribute will provide information on the transformation that will be submitted to the input and the production of a specific output and the rules that will govern this transformation.

6.5.3.4. Interface

A description of the interaction conditions between entities is done by the interface attribute. During the breaking down of the system DOs more and more of such entities are created. Moreover, the communication between DOs can grow. This

interchange of data must be governed by some methods (shared data, parameters, messages or direct access to internal data) and rules (communication protocols, data formats, acceptable values, and meaning to each value).

6.5.3.5. Resources

The definition of the resources attribute is the definition of external entities needed by the DOs. This very important attribute is to describe the external resources needed to accomplish the job of the DO. This information will help in determining most of the hardware resources needed to fulfil the requirements (e.g. printers, disk, memory) and will help in the prediction of any case of race and deadlock conditions and the finding of strategies to facilitate the management of such resources.

6.5.3.6. Data

The final attribute is the definition of data elements internal to the DO. This attribute will describe how data will be represented, used, initialized and the range of its acceptable values. Some data dictionary that explains the content, structure and use of all data elements could represent this data description. The format, number of elements and initial values will be included in this data description (DD) attribute. Moreover, the data structure like file structures, arrays, stacks, queues and memory partitions should be included.

6.6. Design Plan

6.6.1. Definition

We defined the design phase as the process of transforming the software requirements document into another document that will be used in subsequent

processes. Such a document must reflect an organized structure of the software system and such decomposition must be done following a planned approach. One thing for sure is that not all DOs will be ready from the first or second level of decomposition. Therefore, a suggested plan for the design process is presented next. We note that this is not the only secret recipe for system decomposition but a simple guideline for the organization of the activities of this critical phase.

6.6.2. System Decomposition into Functional Modules

To begin with the system must be partitioned into high level functional modules. Each module will have inputs, outputs, and processing attributes. This decomposition can be represented by a hierarchical decomposition diagram or by a plain natural language. The aim of this activity is to capture a high level view of the system. This representation when ready can be tested, walked-through and checked with the requirements of the software system. Corrections are necessary to be made as soon as this view is built in order not to project an error further to later processes.

6.6.3. Individual Decomposition of Functional Modules

At this stage deeper decomposition of functional modules can be done to produce more and more DOs. This decomposition should reflect the view of the dependencies and the interfaces between DOs. It also outputs a final list of the different components that form the system. Using structure charts, data flow diagrams, transaction diagrams and parameter tables the design team can build this view where low coupling and high cohesion should be the main objective.

In achieving these goals, the maintenance activity will be a much easier job and the possibility of change will not be a difficult task. Because software is a model

of the real world and because this real world is constantly changing the system must be designed in a flexible way in order to reflect these changes. Because the customer is always right and he must always be satisfied, the system should be ready for any additions, changes or deletion in the requirements of the system.

6.6.4. Design of the Database

One important issue in the design of software systems, especially when they are database systems is the choice of the relation schemas or better known as the database construction. Most of lebanese software systems are database systems. Therefore an important period of time must be spend on the design and grouping of fields into files. A more professional approach must be taken in order to build a "good" database.

The database should be easy to manipulate. It should be decomposed in a way to reflect a logical view, which refers to how the customer interprets the relation schemas and the meaning of the attributes. It is always recommended not to combine database attributes from multiple entity types and relationship types into a single relation. The mixture of such entities and relations leads to unclearness of the representation. We can also add that a database is well designed if no anomalies are produced when inserting, deleting or modifying records in the database.

6.7. Conclusion

Finally, we reach the lowest level of decomposition or description. At this level all the internal details of DOs are written. All attributes are determined and they will be ready for the next phase, that is the implementation phase. This data will help

also in producing certain test plans. At the end of this activity the SDD is ready to be approved and handed over to the implementation team.

Chapter 7

The Implementation Phase

7.1. Introduction

Few phases remain till the end of the process but most of the work has been terminated. We could define this phase as the translation of design objects into code. That is why "the presence of a good design document" will reduce the time of implementation and produce a robust and reliable software system. Actually, it is widely known that this phase should use half the resources used in the previous phases. On the other hand, this phase is the less standardised phase, since its role is to link the design and integration phases together. The most complete and standardised design phase will result in shorter and less standardised implementation phase [Bennatan 1992].

Little could be found in IEEE and ISO standards about this phase. Therefore, and for the sake of the personnel responsible for the maintenance phase, guidelines for coding will be given here below. In addition, proposal for the ways in which the product code should be implemented will be presented.

The most important issues that should be taken into consideration are:

- ✓ Definition of rules for programming, programming languages, consistent naming conventions, coding and adequate commentary
- ✓ Documentation of all the activities especially where problems were faced
- ✓ Preparation for the integration and test plans

7.2. Principles of implementation Phase

Many issues are to be considered when defining principles:

- ✓ The choice of the programming language
- ✓ The programming style (structured vs. non structured)
- ✓ The programming practices
- ✓ The coding standards
- ✓ The team organization
- ✓ Portability
- ✓ Reusability

7.2.1. Choice of the programming language

The choice of the programming language is not that difficult. If we know what our program is expected to do, its demands and the technical considerations, the choice will be easy. Most of the time our choice of language is limited by the knowledge of languages. Learning a new language can be time and money consuming. This cost should be included in a cost-benefit analysis that will help in determining the language. The management should compute the cost of implementation in every possible language and the benefits, present and future of the same languages. The language with the highest gain (the difference between estimated benefits and estimated cost) could be the most appropriate implementation language. The selection of a programming language can be also a result of a risk analysis. For each language, potential risks and ways of solving them are listed. The language with the smallest overall risk is chosen [Schach 1993]

7.2.2. Programming style

The second principle to be taken into consideration is the style of programming. When we talk of style we refer to the structured programming.

Structured programming as defined by Bohm and Jacopini in 1966 [Bohm and Jacopini 1966] is the possibility of writing programs without goto-statements:

A product is structured if the code blocks are connected by concatenation, selection, and iteration, only and every block has exactly one entry and one exit.

Moreover, Dijkstra in 1968 [Dijkstra 1968] considered that the goto statement was "harmful" and one more time inspired the need for a structured language. These classical definitions can be the basis to a more up-to-date one:

Structured programming is the writing of code with goto statements kept to a minimum, preferably only when an error is detected, and always in the forward direction.

The aim of these directions is to increase the readability, and hence the maintainability of the code. The validity of all procedures should be easily proven.

7.2.3. Programming practices

A third principle can be applied when writing code and that is the use of standards like the use of consistent and meaningful variable names, the issue of self-documenting code, the use of parameters and the implementation of a code layout that increases readability.

7.2.4. Team organization

Another important issue is the working environment and the team organization. The team should be run democratically. Everybody should have a word in the decisions of the team. The manager of the team should take part of the coding. In this way, s/he will gain the respect of his fellow members. Communication should always occur between team members and between teams. Therefore, meetings should

be held regularly and a way of running successful meetings includes: having a good agenda, starting on time, and staying on track [Perry, Ermel and Shields 1994].

7.2.5. Portability

With the rising cost of software, there should be a way that ensures the product can run on different hardware/operating system combinations. The software can then be resold to different parties or customers or fully or partly reused. Such a product can be called portable; that is easily adapted to run on new hardware without being rewritten again.

Different incompatibilities should be taken into consideration:

- ✓ Hardware incompatibilities
- ✓ Operating system incompatibilities
- ✓ Numerical software incompatibilities
- ✓ Compiler incompatibilities

Nevertheless, it is important to try to make all products as portable as possible. Ways of facilitating portability include using popular high-level languages, isolating the non-portable pieces of a product, and adhering to language standards [Schach 1993].

7.2.6. Reusability

Another aspect that relates to portability is reusability. Reuse refers to taking components of one product in order to facilitate the development of a different product with different functionality. The main reason for promoting reuse is that it can reduce the period of coding. Moreover, reuse can reduce the time spent on maintenance and testing. The same applies to the overall cost of the system. Although

some programmers tend to write new components rather than reuse old ones, either because they think their routine is better than others routine or because they are afraid of being put out of work, reusability has proved that it helps investing the lost time on other issues like design, lowering the cost of the whole product and enhancing the maintenance and testing phases.

7.3. Code Reviews

Methods can be used to review the written code. It can be one of three choices: inspections and walk-throughs, static analysis and metrics.

The easiest way is the inspection and walk-through of code. One group could study the code and try to find faults or the author describes the justification for the code to a team who tries to find errors in the logic.

The principle of static analysis involves operating on the source code to reveal facts and features in order for it to be compared with its specification for correctness. A program is validated algebraically and not through dynamic testing that uses specific values [Smith 1987].

The quest for a unique approach to measure software is far from being accomplished. The main cause is that all current practices have only been validated through empirical data. Moreover, the intrinsic nature of software from paradigm to paradigm makes this adventure full of pit falls. Therefore, the choice of the measuring tool should be done carefully and depending on the project we are managing.

7.4. Conclusion

As the functions of the system are completed, the integration of the product starts. Different units that make up a function are linked, function-level test cases can be created and run and test of the functions as a system can be done. Using the system test plan, the software requirements, and the product documentation, data is created for each test case. The hardware/software configuration and preset conditions, the inputs, the expected outputs, the test procedures and the assumptions and constraints for each test case are listed and issued [Kehoe and Jarvis 1996].

When these preparations are done, the system is ready to be tested before its final release.

Chapter 8

The Testing Phase

8.1. Introduction

The test phase is one of the most critical phases because this is where the "last" errors will be discovered and where the system will prove its robustness or just collapse. In the Lebanese market, little attention is paid to the test period: most tests are done by the customer himself and sometimes these tests are done on site. Testing is done neither by reviews nor by walkthroughs and if it is done it carried out after the code has been written. Therefore, it is recommended that testing should start during coding and not after. It should be done with a purpose of finding errors and not to prove that the product is error free.

This phase is one of the most documented phases and one of the most resource and effort consuming ones. Many standards could be found related to testing and could be a source of our proposed standards. We can refer to the IEEE standard for software test documentation, the IEEE standard for software unit testing, the ISO9000-3 testing and validation procedures and Marianne Hajjar's Suggested Lebanese Testing Standards. We will start by defining the testing phase based on the ISO's point of view, then we will describe the most important documentation associated with this phase and finally we will present the steps to be followed based on such documents.

8.2. What is testing?

Testing is "a complex engineering effort and must be well-planned [Kehoe and Jarvis 1996]." This is the supplier's responsibility and should not be done at the purchaser's facilities. Testing should be carried out on different levels of integration. Therefore, for each level of testing what is to be tested must be defined, the methods that will be used, the resources that will be required to perform the test, the time and money needed to develop the test cases and run them. The result of the test must be well-documented and referable in later stages in order to prove, if needed, that the software was tested.

8.3. What should the plan address

Because testing is a difficult and time-consuming process, that is why it needs to be carefully planned and performed by trained personnel. It should also be planned early in the software life cycle, maybe as early as the requirement and design phases and executed in the coding and testing phases.

The test plan should address:

- ✓ Types of testing
- ✓ Test cases
- ✓ Test environment
- ✓ Resources and schedule required to create the test
- ✓ Resources and schedule required to execute the test
- ✓ Entry and exit criteria for each phase of testing
- ✓ External dependencies
- ✓ Test tools
- ✓ Technical, budget, and schedule risks

In addition to the definition of all aspects of the test plan the International Organization for Standardization suggests that the test results should be recorded and used to identify problems with the software and possible units that should be re-tested. The test results should be recorded and compared to the expected output and documented in order to prove to the purchaser that tests were run. Moreover, any found bug and its possible impact on other parts of the product must be reported in order to correct or prevent any new errors. When errors are found in a certain component and corrections are made tests should be re-run [Kehoe and Jarvis 1996].

8.4. Test plan document

It is very obvious that documentation is a very important issue and should be completed carefully. The most needed document is the test plan used to guide the process. Its outline is presented in table 8.1 [Hajjar 1996].

Table 8.1. Test Plan Outline

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

8.4.1. 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 particular subset of tests.

8.4.2. 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.

8.4.3. 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 documentation should be supplied if they exist:

- ✓ Requirements Specification
- ✓ Design Specification
- ✓ User Guide
- ✓ Operations Guide
- ✓ Installation Guide

8.4.4. Features to be tested

It identifies all software features and combinations of software features to be tested and defined 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 purchaser.

8.4.5. 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.

8.4.6. Testing phases and approaches

It describes the overall approach to testing and specifies the major activities, techniques, and tools that are used to test the designated groups of features. It also identifies the techniques that are used to test the designated groups of features. It also identifies the techniques that 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 thoroughly. 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 that are emphasized in the corresponding construction document. For example, when we are doing a module test, the objects used are the algorithms and variables and the aim is to prove correctness of the module. On the other hand, when we are performing an

acceptance test, the objects used are the external functions, the external data and the documentation and the aim is to prove functionality of the whole product, its reliability and its performance.

8.4.7. Item pass/fail

L.A.U. LEBANESE AMERICAN UNIVERSITY P.O. Box: 13-5053 BEIRUT - LEBANON		LIBRARY BOOK REQUEST	
Fund RT		Author: Shallah, Ihab Z.	
Dealer		Title Lebanese Standards for Software Development Process	
Date Ordered Feb. 98		LC:	
Price gift		ISBN:	
Copies		Series	
<input type="checkbox"/> Public cat <input type="checkbox"/> oof. <input type="checkbox"/> ip <input type="checkbox"/> BIP <input type="checkbox"/> BBIP		Year 1998 Volume 15557 Received 23 FEB 1998	

determine whether each test item has passed or
 er defining several categories of "failures",
 ailure, serious failure, failure and
 ated in the Test Plan would then state the
 ire, which are permitted, while still considering

tion requirements

ispend all or a portion of the testing activity

n. The tester must consider two situations:

nd of a day and abnormal suspension, as

re malfunction. If there is any possibility that

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 database and

operating context may require restoration. This needs to be considered when selecting

the suspension points.

8.4.9. Environmental needs

It specifies both the necessary and desired properties of the test environment.

Physical characteristics include hardware, communications, the mode of usage and

any other software or supplies needed to support the test. Levels of security should also be specified for the test facilities.

8.4.10. 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 by the features to be tested.

8.4.11. 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 calculated by the test manager.

8.4.12. 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 planning phase.

8.4.13. 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.

8.4.14. Approvals

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

8.5. More documents

Although it may be early to adopt additional documentation, we can shed some light on possible documents that could trace the testing cycle and keep track of the process [IEEE 829- 1983].

8.5.1. Test Design Specification

This document will be used to explain in detail the approach described in the test plan. It will include specific test techniques to be used and the method of analyzing test results.

8.5.2. Test-Case Specification

This document will define a test case identified by a test-design specification. It will include specifications for the input and output and the environmental-hardware and/or software-needs.

8.5.3. Test Procedure Specification

It includes the steps for executing a set of test cases. It will also identify any special requirements, prerequisite procedures, special skill requirements or special environmental requirements. The procedure steps will include subsections on the way the test will be logged, how it will be set up, how it will start, proceed, be measured, shut down, restarted, stopped and wrapped up.

8.5.4. Test-Item Transmittal Report

It identifies the test items that will be used or transmitted for testing, including their version/revision level and the people responsible for the transmitted items, their location and status.

8.5.5. Test Log

This record will record the events that will occur during testing.

8.5.6. Test-Incident Report

This report will be used to document any event that occurs during the testing process that requires investigation.

8.5.7. Test Summary Report

To summarize the results of the designated testing activities and to provide evaluation based on these results.

The purpose of this listing is to prove that we are far from achieving the ultimate perfection and that we need to get more involved in the development process in order to reach international standards.

8.6. Unit Testing

Before concluding this chapter, we still need to define one more related standard and that is the standard of the unit testing. The process includes the preparation of a test plan, the execution of the plan and the comparison of the test

results of a test unit against its requirements. The unit testing activities are described in table 8.2. [IEEE 1219- 1992].

Table 8.2. Unit Test Activities

- (1) Perform test planning phase
 - (a) Plan the general approach, resources, and schedule
 - (b) Determine features to be tested
 - (c) Refine the general plan
- (2) Acquire test set phase
 - (a) Design the set of tests
 - (b) Implement the refined plan and design
- (3) Measure test unit phase
 - (a) Execute the test procedures
 - (b) Check for termination
 - (c) Evaluate the test effort and unit

These steps can be summarized as a sequential process of: Plan > Determine > Refine > Design > Implement > Execute > Check > Evaluate. Except for execute and check all the other steps will be performed once.

8.6.1. Plan the general approach, resources and schedule

- (1) *Specify a general approach to unit testing:* identify risk areas to be addressed by the process, existing resources of input and output, general techniques for data validation and general techniques to be used for the recording, collecting, reducing and validating the output.
- (2) *Specify completeness requirements:* identify the features, procedures, states, functions, data characteristics, instructions... that will be covered by the process and the degree they will be covered.

- (3) *Specify terminal requirements:* identify the conditions or requirements to end the test. Termination could be caused abnormally; therefore they should be taken into consideration.
- (4) *Determine resource requirements:* identify the resources needed to run the test and possible re-runs. Resources include hardware, access time, communications or system software, test tools, test files, forms and other supplies. Moreover, identify the personnel responsible for the test, their number, skills and the duration they will be involved.
- (5) *Specify a general schedule:* specify a timetable for all unit testing activity.

8.6.2. Determine features to be tested

- (1) *Study the functional requirements:* guided by the requirement document, identify feature to be tested.
- (2) *Identify additional requirements and associated procedures:* for clarification purposes, request any additional requirements and "related to" procedures whenever needed.
- (3) *Identify input and output data characteristics:* list all characteristics of data that will be input and output.

8.6.3. Refine the general plan

- (1) *Refine the approach:* review all previous issues and issue refined approach.
- (2) *Specify special resources requirements:* identify any special resources needed to test the unit such as any software that directly interfaces with the unit.
- (3) *Specify a detailed schedule:* prepare schedule listing all detailed steps.

8.6.4. Design the set of tests

- (1) *Design the architecture of the test set:* using the features to be tested and the conditions specified, design a hierarchically decomposed set of test objectives. In this way, lowest level objective can be tested by a few test cases.
- (2) *Obtain explicit test procedures as required:* define correlation between test cases and procedures.
- (3) *Record the test case specifications and complete the test design specification:* complete all the specification of test cases.

8.6.5. Implement the refined plan and design

- (1) *Obtain and verify test data:* get old test cases and add new ones. Verify all data against the software data structure specifications.
- (2) *Obtain special resources:* get any external resources needed to run the test.
- (3) *Obtain test items:* get all items needed for the test including manuals, operating system procedures, control data (i.e. tables and charts) and programs.

8.6.6. Execute the test procedures

- (1) *Run the test and record all incidents:* execute the test and document any incident regardless of its importance.
- (2) *Collect results and analyze any failure:* documentation, the test environment, the test procedure, the implementation, or the design could cause failures.

8.6.7. Check for termination

- (1) *Check for normal termination:* record the result and decide that no additional tests are needed.
- (2) *Check for abnormal termination:* re-execute the test procedure if needed.

8.6.8. Evaluate the effort and unit

- (1) *Describe the testing status:* specify the test variances between the expected and actual output. For abnormal termination, identify areas insufficiently covered by the test and note the possible reasons for the failures. Identify unresolved test incidents and the reasons for these problems.
- (2) *Describe unit's status:* evaluate the status of the unit.
- (3) *Issue test summary report:* organize all test products and document them in order to reference any of them in the future and when needed.

8.7. Conclusion

The testing phase should be taken seriously because higher-quality systems are in need. A high quality software product satisfies user needs, conforms to its requirements and design specifications, and exhibits an absence of errors. Different techniques can be used for assessing and improving software quality. Each technique has its strengths and weaknesses, and no technique is sufficient by itself.

The goal of testing is to assess and improve quality. High quality is achieved by careful attention to the details of systematic planning, analysis, design and implementation.

Chapter 9

The Maintenance Phase

9.1. Introduction

Software maintenance describes the software engineering activities that occur after delivery of a software product to the customer. Based on most surveys, it is commonly known that the typical lifespan of a software product is between 1 and 3 years in development, whereas maintenance can last between 5 and 15 years. Moreover, a study done by Boehm concludes that 48 to 60% of the effort in the software life cycle is spent on maintenance [Boehm 1976].

It is thus apparent that this final phase is one of the important phases, and one that will ensure the survival of the product. Before formulating the standard to be followed during this phase, definition of different types of maintenance must be stated in order not to label the maintenance phase only as the "error fix" phase.

9.2. Types of maintenance

There are three main reasons for making changes to a product. The first reason is to correct any remaining specification faults, design faults, coding faults, documentation faults, or any other fault. This type of maintenance is known as corrective maintenance. The second reason is to improve the functions of the product or even its maintainability, and thus this process seeks perfective maintenance. Adaptive maintenance is the third main type of maintenance, and it is done when external factors oblige the supplier/purchaser to make such changes. The most recent

and global example is the one of the year 2000 problem. Although most systems are still running, as the first days of the 21st century approaches, there is an urge to make the current code year 2K compliant.

9.3. Maintenance phases

The most realistic steps that will constitute the maintenance process will be a variant of all the previously defined phases. After a problem or a modification has been requested, it should be classified and given a certain priority. If the modification was found to be valid, the second step will be to analyze its feasibility and start changing the requirements. Design, implementation and finally regression test then follow analysis. A typical process model for software maintenance is given by the IEEE Std 1219-1992 and could be the plan to our proposed Lebanese Standards.

Any modification should be initiated by a written request. The modification request (MR) is then studied, classified as adaptive, corrective, perfective or maybe emergency, and given a priority. The output of this step will be a validated MR document that will be analyzed in more detail.

During this second step, all product documents are reviewed for any possible updates especially the requirement. A preliminary modification list is compiled along an implementation list and test strategy.

The third step consists of the design process: test cases are created and requirement and implementation plans are revised and issued. The overall design document and test plans are prepared for update.

Before testing starts, the code is altered to reflect the new changes. In addition to software, design, test plans and most importantly user documents are updated accordingly and released.

There are two aspects to the testing of changes to a product in operations mode. The first is checking that the required changes have been correctly implemented. The second aspect is ensuring that, in the course of making required changes to the product, no other damaging changes were made. Thus, once the programmer has determined that the desired changes have been implemented, the product must be tested against previous test cases to make certain that the functionality of the rest of the product has not been compromised. This procedure is called regression testing. To assist in performing regression testing, it is necessary that all previous test cases be retained, together with the results of running those test cases. It can be argued that this process is a waste of time because it requires the complete product to be retested against a set of test cases that appear to have nothing with the changes that were made. The problem is that some unaccepted side effects might appear after maintenance and caution must be taken. Finally, the system is presented to the customer for acceptance, personnel is trained to the new maintained product and software is delivered.

9.4. Conclusion

Because software maintenance is similar to software development, the tools, techniques and activities of software maintenance span the entire product life cycle. Most difficulties in maintenance are largely due to lack of systematic planning of the phase during the development process.

Planning for maintenance, developing the software product while increasing its maintainability can result in improvements in software quality, programmer productivity, and maintainer morale.

Chapter 10

Conclusion

In this thesis, we have proposed a framework for Lebanese software engineers in order to improve the quality of the software produced in this country. We have presented a list of basic steps that should be used when developing software. The list was formed of multiple short steps, easy to understand and implement in any environment. One can use these steps as a checklist for the whole process as standard steps that must be followed during the life cycle of software. We have also described each phase in detail, which can be used as a reference to make the process more transparent to all teams.

The entire project was not built from scratch but borrowed mostly from IEEE and ISO. My experience during the past seven years added some emphasis on many aspects of the standards, especially the need for documentation.

This approach was followed due to several reasons:

- ✓ A fast solution was needed in order for Lebanon to catch up with the world leaders and especially all his neighboring countries
- ✓ There was no belief in locally developed standards
- ✓ Europeans, Americans and Australians have serious experiences in writing standards and their work could be used as assistance for Lebanese software engineers

We feel it is about time that standards be used in Lebanese software houses.

The aim of this discipline will be the production of quality software that satisfies the

users needs, and is delivered on time and within budget. The only problem in applying these standards remains the lack of will and commitment to implement such standards. If only there are true intentions to improve the quality of software development in Lebanon, the suggestions given in the previous pages can be a useful tool to standardized local software development practices.

References

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.

IEEE Std 830-1993, *Recommended Practice for Software Requirements Specifications*, Institute of Electrical and Electronics Engineers, Inc., New York, Approved by American National Standards Institute, Dec. 1993.

IEEE Std 1016-1987, *Recommended Practice for Software Design Descriptions*, Institute of Electrical and Electronics Engineers, Inc., New York, Approved by American National Standards Institute, March 1987, Revised in December 1993.

Boehm, B. 1976, Software Engineering Education: Some Industry Needs, *Software Engineering Education: Needs and Objectives*, Springer-Verlag, Berlin.

Bohm C. and Jacopini G. 1966, Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules, *Communications of the ACM* 9.

Bennatan, E.M. 1992, *Software Project Management: A Practitioner's Approach*, McGraw Hill, London.

Dijkstra E.W. 1968, Go To Statement Considered Harmful, *Communications of the ACM* 11.

Gilb, Tom 1988, *Software Engineering Management*, Addison Wesley, Wokingham, England.

Hajjar, Marianne 1996, *Software Engineering Practices In Lebanon and Suggestions for Lebanese Requirements Analysis and Software Testing Standards*, Beirut, M.S. Project Report, LAU.

Holdsworth, J. 1994, *Software Process Design: Out of the Tar Pit*, McGraw-Hill, London.

Kehoe, R. and Jarvis, A. 1996, *ISO 9000-3: A Tool for Software Product and Process Improvement*, Springer-Verlag, New York.

Perry, P., Ermel, K. and Shields, J. 1994, *Software Development*, QUE Corporation, Indianapolis, Indiana.

Schach, Stephen R. 1993, *Software Engineering*, Aksen Associates, Boston, USA.

Schmauch, Charles H. 1994, *ISO 9000 or Software Developers*, ASQC Quality Press, Milwaukee, Wisconsin.

Shepperd, M. 1995, *Foundations of Software Development*, Prentice Hall, London.

Sovinski, Iris 1996, *Improvement and ISO9001 Certification in BVR*, Proceedings of the 7th Israeli Conference on Computer Systems and Software Engineering, IEEE Computer Society Press, Los Alamitos, California.

Smith, David J. 1987, *Achieving Quality Software*, Chapman and Hall, London.

Bibliography

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

Johnson, Perry L. 1993, *ISO 9000: Meeting the New International Standards*, McGraw-Hill, USA

Lewin, Marsha D. and Rosenau, Milton D. Jr. 1988, *Software Project Management: Step by Step*, Marsha D. Lewin Associates, Los Angeles, California

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

Appendix A

Process model for software maintenance [IEEE 1219- 1992]

	Problem identification	Analysis	Design	Implementation	System test	Acceptance test	Delivery
Input	MR	Project/system document Repository information Validated MR	Project/system document Source code Databases Analysis phase output	Source code Product/system document Results of design phase	Updated software documentation Test readiness review report Updated system	Test readiness review report Fully integrated system Acceptance test •Plans •Cases •Procedures	Tested/accepted system
Process	Assign change number Classify Accept or reject change Preliminary magnitude estimate Prioritize	Feasibility analysis Detailed analysis Redocument, if needed	Create test cases Revise •Requirements •Implementation plan	Code Unit test Test readiness review	Functional test Interface testing Regression testing Test readiness review	Acceptance test Interoperability test	PCA Install Training
Control	Uniquely identify MR Enter MR into repository	Conduct technical review Verify •Test strategy •Documentation is updated Identify safety & security issues	Software inspection/review Verify design	Software inspection/review Verify •CM control of software •Traceability of design	CM control of •Code •Listings •MR •Test documentation	Acceptance test Functional audit Establish baseline	PCA VDD
Output	Validated MR Process determinations	Feasibility report Detailed analysis report Updated requirements Preliminary modification list Implementation plan Test strategy	Revised •Modification list •Detail analysis •Implementation plan Updated •Design baseline •Test plans	Updated •Software •Design documents •Test documents •User documents •Training material Test readiness review report	Tested system Test reports	New system baseline Acceptance test report PCA report	PCA report VDD

Appendix B

Improvement and ISO9001 Certification in BVR

Iris Sovinski
Process Improvement Manager
BVR, Givataim Israel

Abstract

This paper describes the improvement process that was implemented in BVR in order to be certified as an ISO9001 software and hardware development company, and the associated effort, methodology and benefits. The predefined methodology, of involving the management, proved that an improvement process can take place without affecting work plans and brought the company to achieve all the goals that were set for the process.

1. Introduction

BVR is a company that develops and manufactures simulators of military systems and air combat debriefing systems.

A substantial part of the development of the company's products, in software, hardware and mechanical design is performed by the company's employees, while the other part is subcontracted. The software development process is based on DOD-STD-2167A [1], the US military standard for software development.

Following pressures by customers, BVR initiated the ISO9000 certification process and obtained this certification within a year and a half.

1.1. Process Goals

Obtaining ISO9000 certification for ISO9001 level including software development interpretation (ISO9000-3), while really improving the development and management processes of the company.

Changing the company's mentality from a "Start Up" organization to that of a mature organization which its processes are controlled, while still retaining the dynamic behavior of a small company.

1.2. Company's Initiatives

BVR is operating in the military area and is required to introduce the ISO9001 certification in each proposal and contract.

The company has increased its volume substantially in a short period and had to modify its procedures to its new volume, while improving and shortening the process. In addition, it had to retain its flexibility, in order to be able to respond to customer's requirements.

Due to the growth of the company, new procedures, such as testing, and documentation, had to be established.

2. The Process

2.1. The Process Methodology

The basic methodology for developing the company work procedures, was based on the definition of SEI concerning the work of the Software Engineering Process Group (SEPG) [2].

BVR appointed a Process Improvement Manager who served as the company's guide for the achieving of the process goals.

The Company's CEO is also the company's quality assurance manager and is personally involved in the process.

For this purpose, it was determined that:

- ✓ The procedures describing the global development process of a project in BVR, or inter-groups procedures, will be written by the Process Improvement Manager. The local procedures and work methods will be written by the employees involved in the process, guided by the Process Improvement Manager.
- ✓ Complicated procedures will be reviewed in meeting attended by the representatives of the departments that are involved in the execution of the procedures, or are directly affected by them.
- ✓ All the procedures will be authorized by a relevant professional manager, the Process Improvement Manager and the company's CEO.
- ✓ The process shall be as close to reality as possible and shall be tested in the field. Verifying the compliance of the procedures to the company's strategy and employees is emphasized.
- ✓ Each project manager is entitled to modify the company's procedures so as to comply to the specific requirements of his project. Such a modification should be consistent with ISO9000 requirements or according to customer need, and it requires the authorization of the CEO and is usually performed at the beginning of the project, as part of the project start up.
- ✓ Activity and quality criteria were set. The criteria goal was to define the efficiency and quality of the product and the development process, and identify the problems in the process to be improved. The criteria were set by the people who collect them, according to the efficiency of the data extracted from them and their availability in the company. The criteria were set in away they will serve both the data collectors and the management.

- ✓ The criteria goals were set by the measured employees and were authorized by the management. For example, a project manager sets his goals at the beginning of the project and the CEO authorizes them.

2.2. The Process Description

The process was divided into three phases: the facilities setup phase, the work methods definition phase, and the assimilation and audits phase.

The intensive investment in the first phase resulted in a stable and important basis for the ease of performance of the next phases, which are much more complicated, involve more central employees in the organization and cost more.

The order of defining the methods in the second phase was set by the procedures tree, but apart from that, the methods were defined separately and in parallel.

The third phase was finished when the certification by the Israeli Institute of Standards was obtained.

2.2.1. The Facilities Setup Phase

This phase included the following activities:

- ✓ Contacting the Israeli Institute of Standards, and analyze the path to obtain certification
- ✓ Comparing the ISO9001 requirements to the company's behavior and building a compliance matrix between requirements and the company's structure
- ✓ A list of the procedures that need to be written, according to the standard requirements and the company needs was issued. It was decided that the first phase shall mainly document the existing methods and shall not concentrate on developing new methods
- ✓ A list of the topics that require methodology improvements, to meet the requirements of the standard and the needs of the company was built
- ✓ The documentation facility for the procedures was built: procedures library, procedures format and table of contents for the procedures manual
- ✓ A methodology for the writing of procedures, which contained: a draft document of the procedure, meetings of the group, the final document, authorization processes and the assimilation and testing processes, was defined
- ✓ The employees that will write the procedures, the employees that will participate in the work groups, and the employees that will authorized the procedures were selected

- ✓ The training method, that will push the process throughout the company was defined and set into motion, starting at familiarizing the employees with the company goals and the requirements of the standard
- ✓ An internal monthly paper was published and put in visible locations in the company. It usually dealt with a subject that was at the focus at the publishing time, enlarged the knowledge in the company on the subject and described what was done in the company on the subject. The paper provided large circulation for the improvement processes even if they were confined to a relatively limited environment at the time
- ✓ A tool for managing the improvement activities was defined and built.

The facilities setup phase took a third of the length of the certification project.

2.2.2. Work Methods Definition Phase

This phase included the following activities:

- ✓ Employees training on the list/subjects of the procedures to be written and the decided methodology
- ✓ The full project development process, from the requirement phase to the end of the warranty period (Project Life Cycle - PLC) was defined
- ✓ Missing work methods, such as testing concepts were defined, a draft was written, the work groups were set into motion, the procedure was updated according to the work group comments and the requirements of the standard and at the end the procedure was authorized by the employees responsible to authorize it
- ✓ Criteria and goals were set for each department. For instance: meeting costs, schedules, the amount of problems in the product, the number of turnovers between the development department and the testing department
- ✓ The 'Internal Audit' and 'Corrective Action' processes were developed during this phase

BVR 'Quality System' was build from scratch during this phase, according to the ISO9001 standards requirements.

2.2.3. The Assimilation and Audits Phase

This phase included the following activities:

- ✓ Execution of the procedures in the field at one location and tracking the results
- ✓ Updating the procedures according to the results of the analysis

- ✓ Circulation of the methods in the company (for example to all the projects)
- ✓ Training on the central methods, even if the modification were scare and concerned only modified format, in: PLC, testing procedures, projects management etc.
- ✓ A simulation of the Israeli Institute of Standards inspection was executed to learn about our place in the process. This simulation helped to focus the efforts in the right direction and improve the process
- ✓ The methods were updated according to the simulation results analysis
- ✓ An internal assessment of the new methods implementation was executed in order to detect problems in execution of procedures after the update. The 'internal audits' methods that was developed during this process, was tested at this phase as a way of internal assessment
- ✓ Israeli Institute of Standards audit was performed.

All the above mentioned activities were not executed in series, but in parallel wherever required, according to the right timing of each activity and subject in the company. For example: the circulation of a method all over the company was performed when a specific subject was ready and tested and there were functions in the company that needed the method.

3. Analyzing the Process

3.1. The Points that strengthened the Process

3.1.1. "Real" Management Support

The company's management set the goals of the process and agreed in advance on the method of work. The CEO was the one who passed the message of the process to the employees, as a goal of the company both for certification and for improving the organization. Whenever it was required to push the employees to perform an activity and to set priorities for the process, it was done directly by the CEO.

Company management was aware of the importance of the definition of goals and setting of criteria and debriefing in order to reveal the severity of the problems encountered in the functioning of the company. It also demanded to receive the results of the measurements and used it on the spot. The immediate use of the criteria introduced an immediate influence of the quality system in the company and emphasized its clear place among the systems in the company.

3.1.2. Managing the Process as a Project

The process was defined as a project, including: schedule, budget, a manager that was measured by the success of the project, risks analysis, and

staffing. At any moment it was possible to inspect the rate of progress of the project versus the project plans and to warn about problems.

3.2. Problems with the Process and the Way to Deal with Them

3.2.1. Human Factors

A natural resistance has been roused, within the company's employees, to the idea of working according to procedures generally and to ISO9000 in particular.

This resistance was dealt with on various planes:

- ✓ The people who wrote the procedures were the employees who performed the work in the field, thus turning them interested in the success of the procedures. The employees who wrote the procedures were also responsible for their assimilation and were measured on their actual operation.
- ✓ The employees in the field were audited personally by the Israeli Institute of Standards. They were responsible for the results of the inspection in their area and, of course, were the ones who updated the procedures and improved the assimilation, according to the problems that were encountered.
- ✓ The ability of the employees to decide upon modifications in the procedures in accordance to the mentioned principles, was emphasized and implemented and even though it required the authorization of the CEO, the managers felt that the procedures are: going with" them rather than against them and that their work procedures are actually set by themselves.
- ✓ The Company management has dedicated effort and importance to work by the procedures.
- ✓ With time, the employees discovered that working by the procedures improves their work and began looking at them as resources of information and experience of the company.

3.2.2. Using Central Functions in the Organization

The majority of the employees who wrote the procedures were managers in the areas of which they wrote, and therefore the more busy employees in the company. It was only natural that they found it hard to allocate time for the procedures development. Management involvement in setting priorities, and the fact it was easy to monitor the progress of the projects very accurately, provided the tools to push the process forward.

3.2.3. Saving What Already Exists in the Organization

It is very easy to be carried along with such a process and to improve more than required, while losing the stability of the company and the advantage of natural process in the company, for instance short time response.

The process was initially defined as a first phase of the improvement which was aimed in producing a basis to a well defined improvement process, thus enabling us to keep most of the qualities of the company, intending to consider the quality of the process after they are documented.

4. The Company's Benefits

Due to the short time that passed from the termination of the project, only part of the profit from the process is visible. The immediate benefits visible today are mainly in the following aspects:

Company Visibility - distinct work procedures in the organization resulted in a state in which it is easier to view the activity executed in the company. It is easy to define the future needs of the organization, more accurate schedules and budgets, resulting in easier manageability of the organization.

Customers' Satisfaction - the customers are more satisfied by the results, it is easier to explain to them what happens and to obtain their cooperation and trust.

Atmosphere in the Organization - since the processes are clearer and so is the general direction of the company, there are less professional misunderstandings and personal collisions, and the general atmosphere in the organization is therefore better.

Documentation Organization - the organized documentation and well-defined processes create a clear path for the documentation in the organization. The documentation arrives better to its place, and it seems that we have less documentation, although there is more documentation demand.

Information Base - in a company where employees tend to keep high rate of relocation, working by the procedures saves the gathered data and experience for a long time, with less dependency on the stability of the employees in the field.

5. Project Costs

The project met the budget that was set at its beginning.

Most of the involved employees did not invest more than 5% of their time in the project. The employees that invested more were those who took upon themselves the definition of a new process and even that was done in less than 10% of their time, during their current work.

The profit gained from work by procedures was in most of the cases, higher than the investment in the project, so that no schedules was affected by the project.

6. Conclusion

During the work to obtain the certification of the ISO9001 standard, with the cooperation of the company's management and the employees and aiming at

assimilating the results of the process, BVR became a well organized company rather than a "Start Up" company, improved its management capability, the atmosphere in the company and still maintained most of its capabilities as a dynamic and highly development capable firm.

In addition, the company was qualified as an ISO9001 organization.

At the end of the process the company reached a stage in which it has well defined and documented work methods, collected and analyzed criteria and an adequate basis for a continuous and well controlled improvement process.

Bibliography

1. DOD-STD-2167A, Feb. 1988 Military standard, Defense system, Software development
2. AD-A235 784, SEI - Software Engineering Process Group Guide