

PM.A DOCUMENT TEMPLATES

- PM.A1 Software Project Management Plan;
- PM.A2 User Requirements Document;
- PM.A3 Software Requirements Document;
- PM.A4 Architectural Design Document;
- PM.A5 Detailed Design Document;
- PM.A6 Software User Manual;
- PM.A7 Software Transfer Document;
- PM.A8 Project History Document
- PM.A9 Progress Report

PM.A1 Software Project Management Plan (SPMP)

PM.A1.1 Introduction

- PM.A1.1.1 Project Overview
Summarize the project plan.
- PM.A1.1.2 Project Deliverables
List what will be delivered, when and where.
- PM.A1.1.3 Evolution of the SPMP
Summarize the updates to this version of SPMP.
- PM.A1.1.4 Reference Materials
List applicable and reference documents.
- PM.A1.1.5 Definitions and acronyms
List all abbreviations and acronyms.

PM.A1.2 Project Organization

- PM.A1.2.1 Process Model
Define the life cycle approach to the project.
- PM.A1.2.2 Organizational structure
Describe project roles and reporting lines.
- PM.A1.2.3 Organizational boundaries and interfaces
Describe the interfaces with customers and suppliers.
- PM.A1.2.4 Project responsibilities
Describe the responsibilities of the roles described in 2.2.

PM.A1.3 Managerial process

- PM.A1.3.1 Management objectives and priorities
Describe the management goals and priorities.
- PM.A1.3.2 Assumptions, dependencies and constraints
Describe assumptions, external dependencies and constraints that influence the project plan.
- PM.A1.3.3 Risk management
Describe the risks and how they will be managed.
- PM.A1.3.4 Monitoring and controlling mechanisms
Describe how the project will be controlled and monitored.
- PM.A1.3.5 Staffing plan
Describe the number of people required in each skill category.

PM.A1.4 Technical Process

- PM.A1.4.1 Methods, tools and techniques
Summarize the software development methods and tools.
- PM.A1.4.2 Software documentation
Define or reference the documentation plan.

PM.A1.5 Activities, Schedule and Budget

- PM.A1.5.1 **Activities**
For each activity, describe the input, tasks, outputs and verification process.
- PM.A1.5.2 **Dependencies**
*Describe the interdependencies between the activities.
Describe the external dependencies of the activities.*
- PM.A1.5.3 **Resource requirements**
Describe who and what he or she is required to do on the project.
- PM.A1.5.4 **Budget and resource allocation**
Allocate the resources to work on the activities.
- PM.A1.5.5 **Schedule**
Describe when the activities will be done.

PM.A2 User Requirements Document (URD)

PM.A2.1 Introduction

This section should provide an overview of the entire document and a description of the scope of the software.

PM.A2.1.1 Purpose of the document

This section should:

- a) define the purpose of the particular URD;
- b) specify the intended readership of the URD.

PM.A2.1.2 Scope of the software

This section should:

- a) explain what the proposed software will do (and will not do, if necessary);
- b) name;
- c) identify the software product(s) to be produced by describe relevant benefits, objectives, and goals as precisely as possible;
- d) be consistent with similar statements in higher-level specifications, if they exist.

PM.A2.1.3 Evolution of the URD

Summarize the updates to this version of the URD.

PM.A2.1.4 Definitions, acronyms and abbreviations

This section should provide the definitions of all terms, acronyms, and abbreviations, or refer to other documents where the definitions can be found.

PM.A2.1.5 References

This section should provide a complete list of all the applicable and reference documents, identified by title, author and date. Each document should be marked as applicable or reference. If appropriate, report number, journal name and publishing organization should be included.

PM.A2.1.6 Overview of the document

This section should:

- a) describe what the rest of the URD contains;
- b) explain how the URD is organized.

PM.A2.2 General Description

This chapter should describe the general factors that affect the product and its requirements. This chapter does not state specific requirements but makes those requirements easier to understand.

PM.A2.2.1 Product perspective

Describe related external systems and subsystems.

This section puts the product into perspective with other related systems. If the product is 'standalone', it should be stated here.

PM.A2.2.2 General capabilities

Describe the main capabilities required and why they are needed.

This section should describe the main capabilities and why they are needed. This section should describe the process to be supported by the software, indicating those parts of the process where it is used.

PM.A2.2.3 General constraints

Describe the main constraints that apply and why they exist.

This section should describe any items that will limit the developer's options for building the software.

This section should not be used to impose specific requirements or specific design constraints, but should state the reasons why certain requirements or constraints exist.

PM.A2.2.4 User characteristics

Describe who will use the software and when.

This section should describe those general characteristics of the users affecting the specific requirements.

Many people may interact with the software during the operations and maintenance phase. Some of these people are users, operators and maintenance personnel. Certain characteristics of these people, such as educational level, language, experience and technical expertise impose important constraints on the software.

Software may be frequently used, but individuals may use it only occasionally. Frequent users will become experts whereas infrequent users may remain relative novices. It is

important to classify the users and estimate the likely numbers in each category. If absolute numbers cannot be stated, relative numbers can still be useful.

PM.A2.2.5 Operational environment

Describe what external systems do and their interfaces with the product.

This section should describe the real world the software is to operate in. This narrative description may be supported by context diagrams, to summarise external interfaces, and system block diagrams, to show how the activity fits within the larger system. The nature of the exchanges with external systems should be specified.

If a URD defines a product that is a component of a parent system or project then this section should:

- outline the activities that will be supported by external systems;
- reference the Interface Control Documents that define the external interfaces with the other systems;
- describe the computer infrastructure to be used.

PM.A2.2.6 Assumptions and dependencies

Describe the assumptions upon which the requirements depend.

This section should list the assumptions that the specific requirements are based on. Risk analysis should be used to identify assumptions that may not prove to be valid.

A constraint requirement, for example, might specify an interface with a system that does not exist. If the production of the system does not occur when expected, the URD may have to change.

PM.A2.3 Specific Requirements

List the specific requirements with attributes.

Specific requirements should be described in this section, which is the core of the URD. The acceptability of the software will be assessed with respect to the specific requirements.

Each requirement must be uniquely identified. Forward traceability to subsequent phases in the life cycle depends upon each requirement having a unique identifier.

Essential requirements have to be met for the software to be acceptable. If a requirement is essential, it must be clearly flagged. Non-essential

requirements should be marked with a measure of desirability (e.g. scale of 1, 2, 3).

Some user requirements may be 'suspended' pending resources becoming available. Such non-applicable user requirements must be clearly flagged.

The priority of a requirement measures the order, or the timing, of the related software becoming available. If the transfer is to be phased, so that some parts of the software come into operation before others, then each requirement must be marked with a measure of priority.

Unstable requirements should be flagged. These requirements may be dependent on feedback from other phases. The usual method for flagging unstable requirements is to attach the marker 'TBC'.

The source of each user requirement must be stated. The source may be defined using the identifier of a system requirement, a document cross-reference or even the name of a person or group. Backwards traceability depends upon each requirement explicitly referencing its source.

Each user requirement must be verifiable. Clarity increases verifiability. Each statement of user requirement should contain one and only one requirement. A user requirement is verifiable if some method can be devised for objectively demonstrating that the software implements it. For example, statements such as:

- 'the software will work well';
 - 'the product shall be user friendly';
 - 'the output of the program shall usually be given within 10 seconds';
- are not verifiable because the terms 'well', 'user friendly' and 'usually' have no objective interpretation.

A statement such as: 'the output of the program shall be given within 20 s of event X, 60% of the time; and shall be given within 30 s of event X, 99% of the time', is verifiable because it uses concrete terms and measurable quantities. If a method cannot be devised to verify a requirement, the requirement is invalid.

The user must describe the consequences of losses of availability and breaches of security, so that the developers can fully appreciate the criticality of each function.

PM.A2.3.1 Capability requirements

The organization of the capability requirements should reflect the problem, and no single structure will be suitable for all cases.

The capability requirements can be structured around a processing sequence, for example:

- a) RECEPTION OF IMAGE

b) PROCESSING OF IMAGE

c) DISPLAY OF IMAGE

Perhaps followed by deviations from the baseline operation:

d) HANDLING LOW QUALITY IMAGES

Each capability requirement should be checked to see whether the inclusion of capacity, speed and accuracy attributes is appropriate.

PM.A2.3.2 Constraint requirements

Constraint requirements may cover any topic that does not directly relate to the specific capabilities the users require.

Constraint requirements that relate to interfaces should be grouped around the headings:

- communications interfaces;
- hardware interfaces;
- software interfaces;
- human-computer interactions (user interfaces).

If the software is part of a larger system then any documents that describe the interfaces should be identified.

Requirements that ensure the software will be fit for its purpose should be stated, for example:

- adaptability;
- availability;
- portability;
- security;
- safety;
- standards.

PM.A3 Software Requirements Document (SRD)

PM.A3.1 Introduction

- PM.A3.1.1 Purpose of the document
- PM.A3.1.2 Scope of the software
- PM.A3.1.3 Definitions, acronyms and abbreviations
- PM.A3.1.4 References
- PM.A3.1.5 Overview of the document

PM.A3.2 General Description

- PM.A3.2.1 Relation to current projects
Describe the relationship to other current projects.
- PM.A3.2.2 Relation to predecessor and successor projects
Describe the relationship to previous and future projects.
- PM.A3.2.3 Function and purpose
Describe the main functions the product must perform.
- PM.A3.2.4 Environmental considerations
Describe where the product will be used, who will use it, who will operate it, the hardware it will run on, and the operating system.
- PM.A3.2.5 Relation to other systems
Describe related external systems and subsystems.
- PM.A3.2.6 General Constraints
Describe the main constraints that apply and why they exist.
- PM.A3.2.7 Model description
Describe the logical model using a recognized analysis method.

PM.A3.3 Specific Requirements

- List the specific requirements with attributes.*
- Subsections may be regrouped around high-level functions.*

- PM.A3.3.1 Functional requirements
- PM.A3.3.2 Performance requirements
- PM.A3.3.3 Interface requirements
- PM.A3.3.4 Operational requirements
- PM.A3.3.5 Resource requirements
- PM.A3.3.6 Verification requirements
- PM.A3.3.7 Acceptance testing requirements
- PM.A3.3.8 Documentation requirements
- PM.A3.3.9 Security requirements
- PM.A3.3.10 Portability requirements
- PM.A3.3.11 Quality requirements
- PM.A3.3.12 Reliability requirements
- PM.A3.3.13 Maintainability requirements
- PM.A3.3.14 Safety requirements

PM.A3.4 Use Vs Software Requirements Traceability Matrix
Give a table cross-referencing software requirements to user requirements.

PM.A4 Architectural Design Document (ADD)

PM.A4.1 Introduction

PM.A4.1.1 Purpose of the document

PM.A4.1.2 Scope of the software

PM.A4.1.3 Definitions, acronyms and abbreviations

PM.A4.1.4 References

PM.A4.1.5 Overview of the document

PM.A4.2 System Overview

Summarise the system context and system design

PM.A4.3 System Context

Describe the system context, with diagrams.

Define the external interfaces.

PM.A4.4 System Design

PM.A4.4.1 Design method

Describe or reference the design method.

PM.A4.4.2 Decomposition description

Describe the system design, with diagrams.

Show the components and the control and data flow between them.

PM.A4.5 Component Description

Describe each component.

Structure this section according to the design.

5.n [Component identifier]

5.n.1 Type

Say whether the component is a module, a file, a program etec

5.n.2 Purpose

Trace the component to the software requirements.

5.n.3 Function

Say what the component does.

5.n.4 Subordinates

List the immediate children.

5.n.5 Dependencies

Describe the pre-conditions for using this component.

PM.A5 Detailed Design Document (DDD)

Part 1 – General Description

PM.A5.1. Introduction

- PM.A5.1.1 Purpose of the document
- PM.A5.1.2 Scope of the software
- PM.A5.1.3 Definitions, acronyms and abbreviations
- PM.A5.1.4 References
- PM.A5.1.5 Overview of the document

PM.A5.2. Project Standards, Conventions and Procedures

- PM.A5.2.1 Design standards
Describe or reference the design method used
- PM.A5.2.2 Documentation standards
Describe the formats, style and tools for documentation.
- PM.A5.2.3 Naming conventions
Describe the conventions for naming files, modules etc.
- PM.A5.2.4 Programming standards
Define and reference the coding standards
- PM.A5.2.5 Software development tools
Define and reference the design and production tools.

Part 2 – Component Design Specifications

Describe each component.

Structure this section according to the design.

- n [Component identifier]
Give the name of the components
 - n.1 Type
Say whether the component is a module, a file, a program etc.
 - n.2 Purpose
Trace the components to the software requirements.
 - n.3 Function
Say what the component does.
 - n.4 Subordinates
List the immediate children.
 - n.5 Dependencies
Describe the preconditions for using this component.
 - n.6 Interfaces
Define the control and data flow to and from the components.
 - n.7 Resources
List the resources required, such as displays and printers.
 - n.8 References
Reference any document needed to understand the components.
 - n.9 Processing
Describe the control and data flow within the component using flow/bubble diagram, pseudo-code or a PDL.
 - n.10 Data
Define in detail the data internal to components.

PM.A6 Software User Manual (SUM)

PM.A6.1. Introduction

PM.A6.1.1 Intended readership

Describe who should read the SUM.

PM.A6.1.2 Applicability statement

*State which software release the SUM applies to.
State what it does not cover.*

PM.A6.1.3 Purpose

*Describe the purpose of the document.
Describe the purpose of the software.*

PM.A6.1.4 How to use this document

Say how the document is intended to be read.

PM.A6.1.5 Related documents

Describe the place of the SUM in the project documentation.

PM.A6.1.6 Conventions

Describe any stylistic and command syntax conventions used.

PM.A6.1.7 Problem reporting instructions

Summarize the SPR system for reporting problems.

PM.A6.2. [Overview section]

Describe the process to be supported by the software, and what the software does to support the process, and what the user and/or operator needs to supply to the software.

PM.A6.3. [Instruction section]

From the trainee's viewpoint, for each task, provide:

PM.A6.3.1 Functional description

What the task will achieve.

PM.A6.3.2 Cautions and warnings

Do's and don'ts.

PM.A6.3.3 Procedures

Include:

Set-up and initialisation

Input operations

What results to expect

PM.A6.3.4 Probable errors and possible causes

What to do when things go wrong

PM.A6.4 [Reference section]

From the expert's viewpoint, for each basic operation, provide:

PM.A6.4.1 Functional description

What the operation does.

PM.A6.4.2 Cautions and warnings

Do's and Don'ts.

PM.A6.4.3 Formal description

Required parameters

Optional parameters

Default options

Parameter order and syntax

PM.A6.4.4 Examples

Give worked examples of the operation.

PM.A6.4.5 Possible error messages and causes

List the possible errors and likely causes.

PM.A6.4.6 Cross references to other operations

Refer to any complementary, predecessor or successor operations.

PM.A7 Software Transfer Document (STD)

PM.A7.1 Introduction

- PM.A7.1.1 Purpose of the document
- PM.A7.1.2 Scope of the software
- PM.A7.1.3 Definitions, acronyms and abbreviations
- PM.A7.1.4 References
- PM.A7.1.5 Overview of the document

PM.A7.2 Installation Procedures

Describe how to get the software up and running on the target machine. Include estimated time required for installation.

PM.A7.3 Build Procedures

Describe how to build the software from source code.

PM.A7.4 Configuration Item List

List all the deliverable configuration items.

PM.A7.5 Acceptance Test Report Summary

*For each acceptance test, give the:
user requirement identifier and summary
test report identifier in the SVVP/AT/Test Reports
test result summary*

PM.A7.6 Software Problem Reports

List the SPRs raised during the TR phase and their status at STD issue.

PM.A7.7 Software Change Requests

List the SCRs raised during the TR phase and their status at STD issue.

PM.A7.8 Software Modification Reports

List the SMR's completed during the TR phase.

PM.A8 Project History Document (PHD)

- PM.A8.1 Description of the project
Describe the product developed by the project, the context of the project, the constraints on the project, and the development environment.
- PM.A8.2 Management of the project
- PM.A8.2.1 Contractual approach
Describe the contractual arrangement (e.g. in-house, external contractor).
- PM.A8.2.2 Project organisation
Describe the project organisation in each phase of the life cycle.
- PM.A8.2.3 Methods used
List the methods and tools used.
- PM.A8.2.4 Planning
Describe the original plan and what actually happened.
- PM.A8.3 Software Production
- PM.A8.3.1 Estimated vs. actual amount of code produced
*Give the estimated volume of code at the end of the AD phase.
Give the actual volume of code at the end of the TR phase.*
- PM.A8.3.2 Documentation
List the project documents, with the number of words and pages.
- PM.A8.3.3 Estimated vs. actual effort
*For each work package:
Give the predicted man-months of effort;
Give the actual man-months of effort.*
- PM.A8.3.4 Computer resources
*List the computer resources predicted at the end of AD phase.
List the computer resources used at the end of the TR phase.*
- PM.A8.3.5 Analysis of productivity factors
Give productivity estimates (e.g. LOC/day).
- PM.A8.3.6 Software Reliability
Give the software reliability measures that have been applied and results collected.

PM.A8.4 Quality Assurance Review

Summarise the SQA activities including

- *What SQA attributes had been planned;*
- *What SQA attributes were actually performed;*
- *What were the results of SQA activities performed;*
- *What were the areas of improvement;*
- *Statement of where SQA objective was achieved.*

PM.A8.5 Financial Review

Give the total cost for each phase, and for the whole development.

For each work package:

Give the costs predicted at the end of the UR, SR and AR phases;

Give the actual costs.

PM.A8.6 Conclusions

Say whether the project was a success or failure.

Evaluate against original project objectives.

Describe the lessons learned.

PM.A8.7 Performance of the system in OM phase

Summarise the users' views on the system after final acceptance.

Record data related to quality, reliability, maintainability and safety.

PM.A9 Progress Report

The following table of contents is recommended for the progress report:

- PM.A9.1. Introduction
 - PM.A9.1.1 Purpose
 - PM.A9.1.2 Summary
 - PM.A9.1.3 References
 - PM.A9.1.4 Definitions and acronyms

- PM.A9.2. Technical Status
 - PM.A9.2.1 Work package technical status
 - PM.A9.2.2 Configuration status
 - PM.A9.2.3 Forecast for next reporting period

- PM.A9.3. Resource Status
 - PM.A9.3.1 Staff utilisation
 - PM.A9.3.2 Work package resource status
 - PM.A9.3.3 Resource summary

- PM.A9.4. Schedule Status
 - PM.A9.4.1 Milestone trends
 - PM.A9.4.2 Schedule summary

- PM.A9.5. Problems (new, outstanding and resolved)

- PM.A9.6. Financial Status Report
 - PM.A9.6.1 Costs for the reporting period
 - PM.A9.6.2 Cost to completion
 - PM.A9.6.3 Limit of liability
 - PM.A9.6.4 Payments

PM.B FORM TEMPLATES

PM.B1 Document Change Record;

PM.B2 Document Status Sheet;

PM.B3 Review Item Discrepancy;

PM.B4 Software Change Request;

PM.B5 Software Modification Report;

PM.B6 Software Problem Report;

PM.B7 Software Release Note;

PM.B8 User Change Request;

PM.B9 Time Sheet;

PM.B1 Document Change Record

Originator:		DCR No.:
Approved by:		Date No.:
1. Document Title:		
2. Document Reference No.:		
3. Document Issue/Revision No.:		
4. Page	5. Paragraph	6. Reason for Change

PM.B2 Document Status Sheet

1. Document Title:			
2. Document Reference No.:			
3. Issue	4. Revision	5. Date	6. Reason for Change

PM.B3 Review Item Discrepancy

Originator:	RID No.:
	Date:
1. Document Title:	
2. Document Reference No.:	
3. Document Issue/Revision No.:	
4. Problem Location:	
5. Problem Description:	
6. Recommended Solution:	
7. Author's Response:	
8. Review Decision: Close/Update/Action/Reject (<u>underline choice</u>)	

PM.B4 Software Change Request

Originator:	SRC No.:
Related SPRs:	Date:
6. Software Item Title:	
7. Software Item Version/Release Number:	
8. Priority: Critical/Urgent/Routine (underline choice)	
9. Changes Required:	
10. Responsible Staff:	
6. Estimated Start Date, End Date and Manpower Report:	
7. Attachments:	

PM.B5 Software Modification Report

Originator:		SMR No.:
Related SCRs:		Date:
1. Software Item Title:		
2. Software Item Version/Release Number:		
3. Changes Implemented:		
4. Actual Start Date, End Date and Manpower Effort:		
5. Attachments	Source Code	<input type="checkbox"/>
	Test Procedures	<input type="checkbox"/>
	Test Data	<input type="checkbox"/>
	Test Results	<input type="checkbox"/>
	Documentation Updates	<input type="checkbox"/>

PM.B6 Software Problem Report

Originator:	SPR No.:
	Date:
1. Software Item Title:	
2. Software Item Version/Release No.:	
3. Priority: Critical/Urgent/Routine (underline choice)	
4. Problem Description:	
5. Description of Environment:	
6. Recommended Solution:	
7. Review Decision: Close/Update/Action/Reject (underline choice)	
8. Review Decision: Close/Update/Action/Reject (underline choice)	

PM.B7 Software Release Note

Originator:	SRN No.:
Approved by:	Date:
1. Software Item Title:	
2. Software Item Version/Release No.:	
3. Changes in this Release:	
4. Configuration Items included in this Release:	
5. Installation Instructions:	

PM.B8 User Change Request

PM.B9 Time Sheet

PM.B10 Work Description and Completion Report

Project: Phase:	Sheet of Issue Ref.: Issue Date:
Work title: Primary responsibility:	
Planned start: end:	Actual start: end:
Inputs: Activities: Output:	
Remarks: Team Leader/ Manager:	

PM.C PROJECT LIFE CYCLE PHASE ACTIVITIES

PM.C.1 Requirements Phase

The UR phase is the ‘problem definition phase’ of a software project. The scope of the system must be defined. The user requirements must be captured. This may be done by interview or survey, or by building prototypes. Specific user requirements must be identified and documented in the User Requirements Document (URD).

The involvement of the developers in this phase varies according to the familiarity of the users with software. Some users can produce a high quality URD, while others may need help from the developers.

The URD must always be produced. The review of the URD is done by the users, the software and hardware engineers and the managers concerned. The approved URD is the input to the Analysis phase.

Before the completion of the User Requirements Review (UR/R), a Software Project Management Plan outlining the whole project must be produced by the developer. This plan must contain a cost estimate for the project. Detailed plans for the SR phase must also be produced.

PM.C.2 Analysis Phase

A vital part of the analysis activity is the construction of a ‘model’ describing ‘what’ the software has to do, and not ‘how’ to do it. Building prototypes to clarify the software requirements may be necessary.

The principal deliverable of this phase is the Software Requirements Document (SRD). The SRD must always be produced for every software project. Implementation terminology should be omitted from the SRD. The SRD must be reviewed formally by the users, by the computer hardware and software engineers, and by the managers concerned.

As an important part of this phase, the section of the Software Project Management Plan outlining the rest of the project must be updated. Accordingly, the plan must contain an estimate of the total project cost. Detailed plans for the Design phase must also be produced.

PM.C.3 Design Phase

The purpose of the Design phase is to define the structure of the software followed by detailing the design of the software. The model constructed in the Analysis phase is the starting point. This model is transformed into the overall design by allocating functions to software components and defining the control and data flow between them.

This phase may involve several iterations of the design. Technically difficult or critical parts of the design have to be identified. Prototyping of these parts of the software may be necessary to confirm the basic design assumptions. Alternative designs may be proposed, one of which must be selected.

The deliverable item which constitutes the formal output of this phase is the Design Specifications (DS). The DS must always be produced for every software project. The DS must be reviewed formally by the computer hardware and software engineers, by the users, and by the management concerned.

During the Design phase, the Design specifications initially contain the sections corresponding to the top levels of the system. As the design progresses to lower levels, related subsections are added leading to the completion of program specifications as a deliverable of this phase.

As an important output of this phase, the section of the Software Project Management Plan outlining the rest of the project must be updated. Accordingly, the plan must contain an estimate of the project cost (10% accuracy is a good target). Detailed plans for the coding phase must also be produced.

PM.C.4 Coding phase

The purpose of the Coding phase is to code and test specified units of the software.

During this phase, units of the software code are produced in accordance to program specifications and unit testing activities are performed. As well as these tests, there should be checks on software quality.

PM.C.5 Testing phase

The purpose of this phase is to perform integration and systems tests of the software.

Tested units of the software are combined into larger components of the system which are then tested individually in accordance to the DS until the overall system is built.

When the overall system is built, system testing is performed mainly to test overall system functionality, performance and operations capability.

Apart from integration and system-tested results, the Software User and Operations Manuals are being developed to form the major deliverables of this phase.

PM.C.6. Acceptance

The purpose of this phase is to establish that the software fulfils the requirements laid down in the URD. This is done by installing the software in a simulated production environment and conducting acceptance tests.

When the software has been demonstrated to provide the required capabilities, the software can be *provisionally accepted* and implementation procedures started.

The Software Transfer Document (STD) must be produced during the Acceptance phase to document the transfer of the software to the operations team.

PM.C.7 Implementation

The operations team must follow the instructions on the STD in implementing the software into the production environment.

When the software has been implemented, it should be carefully monitored to confirm that it meets all the requirements defined in the URD. Some of the requirements, for example those for availability, may take a period of time to validate. When the software has passed all the acceptance tests, it can be *finally accepted*.

The Project History Document (PHD) summarises the significant managerial information accumulated in the course of the project. This document must be issued after final acceptance. It should be reissued at the end of the life cycle, with information gathered in the OM phase.

After final acceptance, the software may be modified to correct errors undetected during earlier phases, or because new requirements arise. This is called 'maintenance'.

For the whole period of operation, particular attention should be paid to keeping the documentation up to date. Information on faults and failures should be recorded to provide the raw data for the establishment of software quality metrics for subsequent projects. Tools should be used to facilitate the collection and analysis of quality data.

PM.D TYPES OF PROJECT LIFE CYCLE PHASE APPROACHES

PM.D.1 Life cycle approaches

This software life cycle (SLC) model, summarizes the phases and activities which must occur in any software project. A life cycle approach, based upon this model, should be defined, for each project, in the Software Project Management Plan.

This section defines three common approaches. In the diagrams, the SLC phases have been reduced to boxes. Arrows connecting the boxes represent permitted transitions.

PM.D.2 The waterfall approach

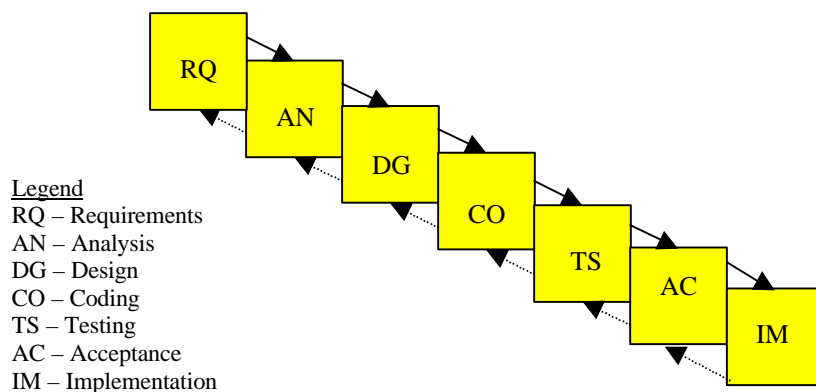


Figure 2 The water fall approach

The ‘waterfall’ approach, shown in Figure 2 is the simplest interpretation of the SLC model. The phases are executed sequentially, as shown by the heavy arrows. Each phase is executed once, although iteration of part of a phase is allowed for error correction, as shown by the dashed arrows. Delivery of the complete system occurs at a single milestone at the end of the AC phase. The approach allows the contractual relationship to be simple.

PM.D.3 The Incremental Delivery Approach

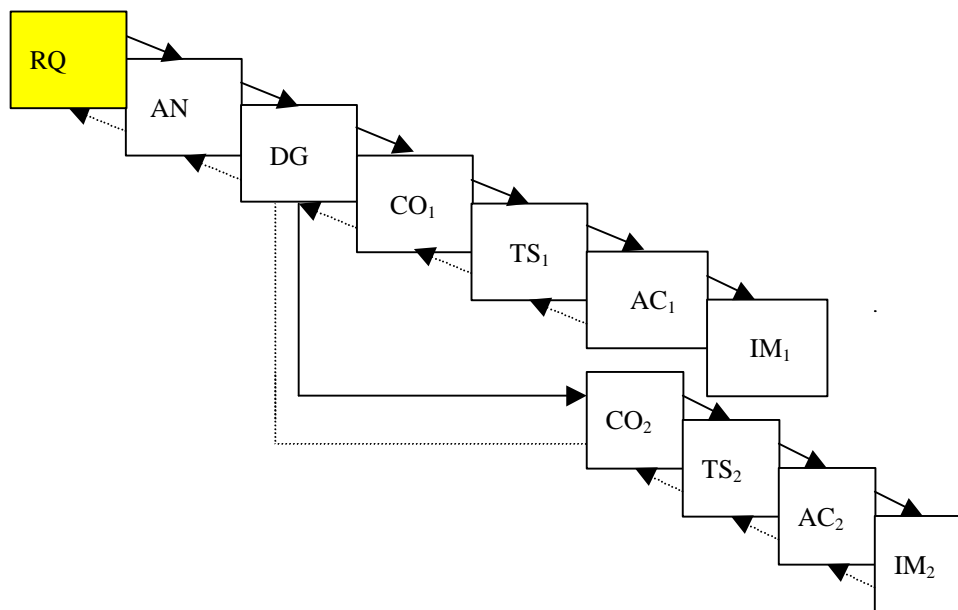


Figure 2.1 *The incremental delivery approach*

The ‘incremental delivery’ approach, shown in 2.1, is characterised by splitting the CO, TS, AC and IM phases into a number of more manageable units, once the complete architectural design has been defined. The software is delivered in multiple releases, each with increased functionality and capability. This approach is beneficial for large projects, where a single delivery would not be practical. This may occur for a number of reasons such as:

- certain functions may need to be in place before others can be effective;
- the size of the development team may necessitate subdivision of the project into a number of deliveries;
- budgeting considerations may only allow partial funding over a number of years.

In all cases, each deliverable should be usable, and provide a subset of the required capabilities.

A disadvantage of the incremental delivery approach is that regression testing is required to confirm that existing capabilities of the software are not impaired by any new release. The increased amount of testing required increases the cost of the software.

PM.D.4 The evolutionary development approach

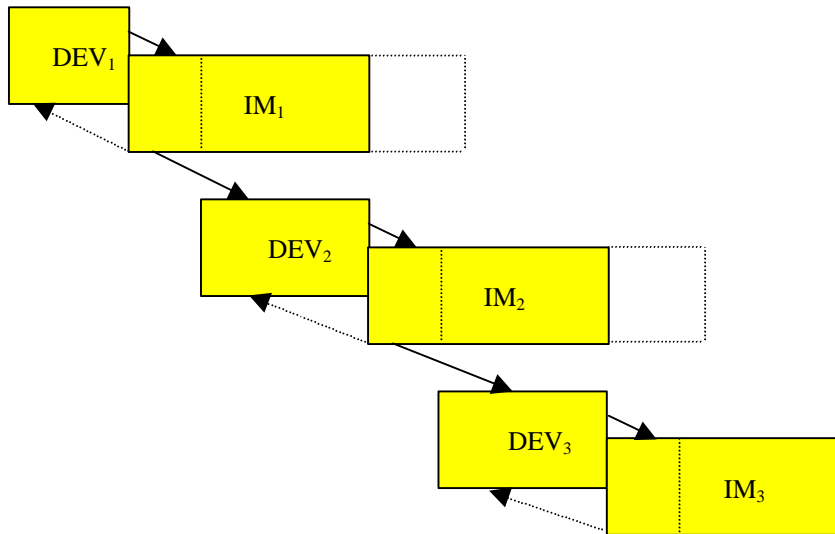


Figure 3 *The evolutionary development approach*

The 'DEV' box is equivalent to the RQ, AN, DG, CO, TS and AC phases shown in Figure

The 'evolutionary' approach, shown in Figure 3 is characterised by the planned development of multiple releases. All phases of the life cycle are executed to produce a release. Each release incorporates the experience of earlier releases. The evolutionary approach may be used because, for example:

- some user experience is required to refine and complete the requirements (shown by the dashed line within the IM boxes);
- some parts of the implementation may depend on the availability of future technology;
- some new user requirements are anticipated but not yet known;
- some requirements may be significantly more difficult to meet than others, and it is decided not to allow them to delay a usable delivery.

The dashed extensions to the boxes in Figure 3 show that some overlap of IM phases will occur until each new delivery is finally accepted.

In an evolutionary development, the developer should recognise the user's priorities and produce the parts of the software that are both important to the user and, possible to develop with minimal technical problems or delays.

The disadvantage of the evolutionary approach is that if the requirements are very incomplete to start with, the initial software structure may not bear the weight of later evolution. Expensive rewrites may be necessary. Even worse, temporary solutions may become embedded in the system and distort its evolution. Further, users may become impatient with the teething troubles of each new release. In each development cycle, it is important to aim for a complete statement of requirements (to reduce risk) and an adaptable design (to ensure later modifiability). In an evolutionary development, all requirements do not need to be fully implemented in each

development cycle. However, the architectural design should take account of all known requirements.

PM.D.5 Prototyping

The use of prototypes to test customer reaction and design ideas is common to many engineering disciplines. A software prototype implements selected aspects of proposed software so that tests, the most direct kind of verification, can be carried out.

Prototyping is the process of building prototypes. Prototyping within a single phase is a useful means of reducing the risk in a project through practical experience. The output of a prototyping exercise is the knowledge that is gained from implementing or using the prototype software.

The objective of the prototyping activity should be clearly stated before the process starts. Prototyping to define requirements is called 'exploratory' prototyping, while that for investigating the feasibility of proposed solutions is called 'experimental' prototyping.

Prototypes usually implement high risk functional, performance or user interface requirements and usually ignore quality, reliability, maintainability and safety requirements. Prototype software is therefore 'pre-operational' and should never be delivered as part of an operational system.

PM.E CONSIDERATIONS FOR BUILDING ACTIVITY NETWORK

The primary objective of building an activity network is to design a feasible pattern of activities that takes account of all dependencies.

An activity network represents the work in the project as a set of nodes with arrows linking them. A sequence of arrows defines a path, or part of a path, through the project.

Two important by-products that can be derived from the activity network are:

- Critical path;
- Work float.

The critical path is the longest path through the network in terms of the total duration of activities. The time to complete the critical path is the time to complete the project.

The float time of a piece of work is the difference between the earliest and latest work start or end dates, meaning the amount of time each activity can be moved without affecting the total time to complete the project. By definition, activities on the critical path have zero float time.

In general, activity networks should only include work that depend on other work for input or produce output for other work to use, meaning activities that are just connected to the start and end nodes should be excluded.

In addition, the activity network for large projects should be broken down into sub-networks e.g. one per phase. Such a modular approach makes the project easier to manage, understand and evaluate.

PM.F RISK MANAGEMENT GUIDELINES

PM.F.1 Introduction

Project managers should manage risks by:

- Continuously looking out for likely threats to the success of the project;
- Adjusting the plan to reduce the likelihood of threats being realized;
- Defining contingency plans;
- Implementing contingency plans if necessary.

Risk management must never start from the premise that ‘all will go well’ but rather ‘what is most likely to go wrong?’ This is realism, not pessimism.

Project plans should identify the risks to a project and show how the plan takes account of them.

The following sections discuss the common types of risk factor groups that may impact software projects, with possible actions to counteract them:

- Experience;
- Planning;
- Technology;
- External.

PM.F.2 Experience Factors

Experience factors that can be a source of risk include:

PM.F.2.1 Experience and qualifications of the project manager

The project manager is the member of staff whose performance is critical to the success of a project. Inexperienced project managers are a significant risk and organizations should match the difficulty of the project to the experience of the project manager.

Part-time project managers are also a risk as this reduces the capability of the project to respond to problems quickly.

Contract project managers are also a risk as here lies a potential conflict of loyalty between the organization owning the project and the organization employing the project manager.

Untrained project managers poses a risk because they may be unaware of what is involved in management. Staff moving into management should receive training for their new role.

Finally, project management should be the responsibility of a single person and not be divided. This ensures unity of command and direction.

PM.F.2.2 Experience and qualifications of staff

Staff will be a risk to the project if they have insufficient experience, skills and qualifications for the tasks they are assigned.

Project managers should avoid such risks by:

- Assessing staff before they join the project;
- Allocating tasks that match staff experience, skills and qualifications;
- Retaining staff within the project who have the appropriate skills, experience and qualifications to cope with future tasks.

PM.F.2.3 Maturity of suppliers

The experiences of suppliers are key factors in assessing the risks to a project. Indicators of maturity are the:

- Successful development of similar systems;
- Use of software engineering standards;
- The possession of software Quality Management certificates like ISO 9001.
- The existence of a software process improvement programme.

Experience of developing similar systems is an essential qualification for a project team. Lack of experience often results in poor estimates, avoidable errors and higher costs.

Software engineering, coding and administrative standards are all important for a project and project managers should ensure that the standards are understood, accepted and applied.

PM.F.3 Planning Factors

Planning factors that can be a source of risk are:

PM.F.3.1 Accuracy of estimates

Accurate estimates of resource requirements are needed to complete a project successfully. Underestimates are especially dangerous if there is no scope of being given additional resources later. Over-estimation can result in waste and can prevent resources from being deployed elsewhere.

Some of the activities which are often poorly estimated are:

- Testing;
- Integration, especially of external systems;
- Implementation phase;
- Reviews including rework.

PM.F.3.2 Short time-scales

Short time-scales increase the amount of parallel working required, resulting in a larger team. Progressive reduction in the time-scale increases this proportion to a limit where the project becomes unmanageable. Project managers should not accept unrealistic time-scales.

Project managers should avoid artificially contracting time-scales by attempting to deliver software before it is required. They should use the time available to optimize the allocation of resources such as staff.

When faced with accumulating delays and rapidly approaching deadlines, project managers should remember Brooks' Law: 'Adding manpower to a late project makes it later'.

PM.F.3.3 Long time-scales

Projects with long time-scales run the risk of:

- Changes in requirements;
- Staff turnover;
- Being overtaken by technological changes.

Long time-scales can result from starting too early. Project managers should determine the optimal time of starting the project through careful planning.

Long projects should consider using an incremental delivery or evolutionary development life-cycle approach. Both approaches aim to deliver some useful functionality within a time-scale in which the requirements should be stable. If this cannot be done, then the viability of the project should be questioned, as the product may be obsolete or unwanted by the time it appears.

Ambitious objectives coupled with constraints imposed by annual budgets, often cause projects to have long time-scales. As management costs are incurred at a fixed rate, management consumes a large proportion of the project cost as the time-scale lengthens. Furthermore, technical and economic change can make the objectives obsolete resulting in the abandonment of the project before anything is achieved.

PM.F.3.4 Single-point failures

A single-point failure occurs in a project when a resource vital to an activity fails and there is no backup. Project managers should look for single-point failures by examining each activity and considering:

- The reliability of the resources;
- Whether backup is available.

Project managers should devise contingency plans to reallocate resources when

failures occur.

PM.F.3.5 Location of staff

Dispersing staff to different locations can result in poor communication. Project managers should co-locate staff in contiguous accommodation wherever possible. This improves communication and allows for more flexible staff allocation.

PM.F.3.6 Definition of responsibilities

Poor definition of responsibilities is a major threat to the success of a project. On one hand, vital jobs may not be done simply because no one was assigned to do them. On the other hand, tasks may be repeated unnecessarily because of ignorance about responsibilities.

Projects should be well organized by defining project roles and allocating tasks to the roles. Similarly, responsibilities should be clearly defined to perform the tasks.

PM.F.3.7 Staffing profile evolution

Rapid increases in the number of staff can be a problem because new staff need time to familiarize themselves with a project. After the start of the project much of the project knowledge comes from the existing staff. These teaching resources limit the ability for a project to grow quickly.

Rapid decreases in the number of staff can be a problem because of the loss of expertise before the project is finished can drastically slow the rate at which problems are solved. When experienced staff leave a project, time should be set aside for them to transfer their knowledge to other staff.

The number of staff on a software project should grow smoothly from a small team in the software definition phase to a peak during coding and unit testing, and fall again to a smaller team for the implementation phase. Project managers should avoid sudden changes in the manpower profile and ensure that the project can absorb the inflow of staff without disruption.

PM.F.4 Technology factors

Technology factors that can be a source of risk are:

- Technical novelty;
- Maturity and suitability of methods;
- Maturity and efficiency of tools;
- Quality of commercial software.

PM.F.4.1 Technical novelty

Technical novelty is an obvious risk. Project managers should assess the technical novelty of every part of the design. Any radically new component can cause problems because the:

- Feasibility has not been demonstrated;
- Amount of effort to build it has not been demonstrated.

Project managers should estimate the costs and benefits of technically novel components. The estimates should include the costs of prototyping the component.

Prototyping should be used to reduce the risk of technical novelty. The cost of prototyping should be significantly less than the cost of developing the rest of the system, so that the feasibility is demonstrated before major expenditure is incurred. More accurate cost estimates are an added benefit of using the prototyping approach.

PM.F.4.2 Maturity and suitability of methods

New methods are often immature as practical experience with a method is necessary to refine it. Furthermore new methods normally lack tool support.

Choosing an unsuitable method results in unnecessary work and possibly unsuitable software. Analysis and design methods should match the application being built. Verification and validation methods should match the criticality of the software.

Project managers should evaluate the strengths e.g. high suitability, and the weaknesses e.g. lack of maturity, of a method before making a choice. Project managers should always check whether the method has been used for similar applications.

PM.F.4.3 Maturity and efficiency of tools

Tools can prove to be a hindrance instead of a benefit if:

- Staff have not been trained in their use;
- They are unsuitable for the methods selected for the project;
- They are changed during the project;
- They have more overheads than manual techniques.

The aim of using tools is to improve productivity and quality. Project managers should carefully assess the costs and benefits of tools before deciding to use them on a project.

PM.F.4.4 Quality of commercial software

Including new commercial software or using commercial software for the first time in a new application area can be a risk because:

- It may not work as advertised or expected;

- It may not have been designed to meet the quality, reliability, maintainability and safety requirements of the project.

New commercial software should be comprehensively evaluated as early as possible in the project so that there are no surprises later. Apart from technical requirements, other aspects to evaluate when considering the acquisition of commercial software are:

- Supplier size, capability and experience;
- Availability of support;
- Future development plans.

PM.F.5 External Factors

External factors that can be a source of risk are the:

- Quality and stability of user requirements;
- Definition and stability of external interfaces;
- Quality and availability of external systems.

PM.F.5.1 Quality and stability of user requirements

A project is unlikely to succeed without a high quality User Requirements Document (URD). A URD is a well-defined baseline against which to measure success. Project managers should ensure that a coherent set of requirements is available. This may require resources very early in the project to help users define their requirements e.g. prototyping.

PM.F.5.2 Definition and stability of external interfaces

External interfaces can be a risk when they are poorly defined and/or are non-standard. Interfaces with external systems should be defined in the Interface Control Document (ICD refer Appendix A). These are needed as early as possible because they can constrain the design. Once agreed, changes should be minimized, as a change in an interface could imply rework by multiple teams.

Standard interfaces have the advantage of being well defined and stable. For these reasons a standard interface is always preferred, even if the design has to be modified slightly.

PM.F.5.3 Quality and availability of external systems

External systems can be a problem if their quality is poor or if they are not available when required owing to late delivery or to unreliability.

Project managers should initiate discussions with suppliers of external systems when the project starts. This ensures that suppliers are aware of project requirements as early as possible. In addition, project managers should allocate adequate resources to the integration of external systems.

The effects of late delivery of the external systems can be mitigated by:

- Adding the capability of temporarily ‘bypassing’ the external system;
- Development of software to simulate the external system.

Note that the cost of the simulator must be traded-off against the costs incurred as a result of late delivery.

PM.G FACTORS USED IN DEFINING SCHEDULE, RESOURCE REQUIREMENTS AND TOTAL COSTS

The start and end dates for work affected by time and resource constraints should be set first as these reduce the number of possibilities to consider. If the total time to complete the project violates time constraints, project managers are required to return to the define activities stage, redefine work, re-estimate resources and duration, and then modify the activity network.

The minimum total cost is the sum of all the work required to be carried out. However labour costs should be calculated from the total amount of time spent by each person on the project. This ensures that the overhead created by time spent waiting for work to start and end is included in the total cost estimate.

Activities with high risk factors should be scheduled to start at their earliest possible start times so that the activity float can be used as a contingency.

Project Managers should adjust the start times of activities using the same resource so that it is used continuously rather than at discrete intervals. Minimizing fragmentation of a resource allocation is important for the following reasons:

- Interruptions mean that project staff have to re-familiarize or relearn procedures that may have been forgotten;
- Equipment may be charged to the project even when it is not in use.

PM.H MEASURING PROJECT PROCESSES AND PRODUCTS

Project managers should as far as possible adopt a quantitative approach to measuring software processes and products. This is essential for effective project control, planning future projects and improving the software development process in the organization.

The following measurement steps should be followed in a project:

- Assess the project environment and define the primary goals (e.g. financial targets, quality, reliability etc.);
- Analyze the primary goals into sub-goals that can be quantitatively measured and define metrics e.g. man-months of effort, for each sub-goal;
- Collect the raw metric data and measure the performance relative to the project goals;
- Improve the performance of the project by updating project plans to correct deviations from project goals;
- Improve the performance of the organization by passing the metric data and measurements to the software process improvement group or to the software managers who are responsible for the standards and procedures the project uses.

Steps one to four are discussed in more detail below.

PM.H.1 Assess the project environment and define the primary goals

The purpose of assessing the project environment is to understand and define what primary goals are needed and what are practical in delivering the software product on time, within budget and with the required quality.

Common methods used for assessing the project environment involve audits to measure conformance to Software Engineering Standards, SEI CMM Model, ESI BOOTSTRAP Model or ISO 9000 Quality Management Standards.

PM.H.2 Analyze goals and define metrics

Common sub goals related to the primary goal are:

- Do not exceed the planned effort on each activity;
- Do not exceed the planned time on each activity;
- Ensure that the product is complete;
- Ensure that the product is reliable;

Project managers should define one or more metrics related to the sub goals defined for the project. Metric definitions should as far as possible follow organizational or industry standards so that comparisons with other projects are possible.

Examples of some types of metrics are provided below:

Process Metrics for measuring project *development effort*:

- Amount of resources used;
- Amount of resources left.

Process Metrics for measuring project *development time*:

- Actual duration of activities in days, weeks or months;
- Slippage of activities (actual start-planned start).

Process Metrics for measuring project *progress*:

- Number of tasks or activities completed;
- Number of software problems solved.

Product Metrics for measuring *amount of product produced*:

- Number of lines of software code produced, excluding comments;
- Number of modules coded and unit tested;
- Number of function points implemented;
- Number of pages of documentation written.

Actual value should be compared with target values to measure product completeness.

Product Metrics for measuring *product reliability*:

- Test coverage;
- Integration complexity of programs;
- Number of critical Software Problem Reports;
- Number of non-critical Software Problem Reports;
- Number of changes to products after first issue or release.

PM.H.3 Collect metric data and measure performance

Project managers should ensure that metric data collection is a natural activity with the project. Preferably, the actual effort expended for metric data collection should always be assessed before a piece of work is signed off as being completed e.g. Counts of Software Problem Reports should be accumulated in the Software Testing process.

PM.H.4 Improving performance

Metric data should be made available for project management reporting, planning future projects, and software process improvement studies.

The project manager should use the metric data to improve processes and products.

For example:

- Comparisons of predicted and actual effort early in the project can quickly identify deficiencies in the initial estimates, prompting major re-planning of the project to improve progress;
- Analysis of the trends in software problem occurrence during integration and systems testing can show whether further improvement in quality and reliability is necessary before the software is ready to be implemented