

**Guidelines for the
Life Cycle Objectives (LCO)
and the
Life Cycle Architecture (LCA)
deliverables for
Model-Based (System) Architecting and
Software Engineering (MBASE)**

Inception and Elaboration

- ❑ **Operational Concept Description (OCD)**
- ❑ **System and Software Requirements Definition (SSRD)**
- ❑ **System and Software Architecture Description (SSAD)**
- ❑ **Life Cycle Plan (LCP)**
- ❑ **Feasibility Rationale Description (FRD)**

General permission to make fair use in teaching or research of all or part of these guidelines is granted to individual readers, provided that the copyright notice of the Center for Software Engineering at the University of Southern California is given, and that reference is made to this

publication. To otherwise use substantial excerpts or the entire work requires specific permission, as does reprint or republication of this material.

© Center for Software Engineering, University of Southern California. All Rights Reserved. 1997-1999.

General Guidelines

Please read the following general guidelines carefully, before proceeding to the guidelines for the individual deliverables.

MBASE 577 Process

Model-based System Architecting and Software Engineering (MBASE) is an approach that integrates the process, product, property and success models for developing a software system. The essence of the approach is to develop the following six system definition elements concurrently and iteratively (or by refinement) using the WinWin Spiral approach defined in [Boehm, 1996].

- Operational Concept Description (OCD)
- System and Software Requirements Definition (SSRD)
- System and Software Architecture Description (SSAD)
- Life Cycle Plan (LCP)
- Feasibility Rationale Description (FRD)
- Risk-driven prototypes

The two critical project milestones are the Life Cycle Objectives (LCO) and Life Cycle Architecture (LCA). The system definition elements have to satisfy specific completion criteria at each anchor point.

- The system definition elements are strongly integrated and a strong traceability thread ties the various sections: e.g., the System Definition (documented in the SSRD) is a refinement of the Statement of Purpose (documented in the OCD). Therefore, to enforce conceptual integrity, it is essential that team members work collaboratively, particularly on strongly related sections.
- Due to the strong interdependencies, it may be a good idea to follow some order when producing the deliverables, at least initially: e.g., write core sections of the OCD before the SSRD. During successive iterations, the documents generally should not be traversed in a linear fashion. Forward consistency should always be enforced (if an Entity is added to the Entity Model, then it should be examined as to how it affects the Component Model). Backward consistency can be less strongly enforced, but is useful to do where feasible.
- Strongly dependent sections are indicated by [Consistent with DDD x.x.x] where DDD is the LCO/LCA deliverable, and x.x.x the section number. When reviewing the deliverables and checking the overall conceptual integrity, it is very helpful to review strongly connected sections in sequence (e.g., OCD: Statement of Purpose, SSRD: System Definition), as opposed to reviewing the deliverables in a linear fashion.
- Conceptual integrity and consistency between the various deliverables, at a given milestone (LCO/LCA), is critical. In particular, a system definition element should not be "incomplete" with respect to the remaining ones. For instance, if the SSRD specifies more requirements, than the architecture described in the SSAD supports, but the FRD claims that the architecture will satisfy all the requirements, the SSAD would be considered incomplete. It is important to reconcile the deliverables, and make sure that one deliverable is not "one iteration ahead" of the other deliverables.
- The general differences between the LCO and the LCA are as follows:
 - Life Cycle Objectives (LCO):**
 - less structured, with information moving around
 - focus on the strategy or "vision" (e.g., for the Life Cycle Plan), as opposed to the details
 - could have some mismatches (indicating unresolved issues or items)
 - no need for complete forward and backward traceability
 - may still include "possible" or "potential" elements (e.g., Entities, Components, ...)
 - some sections could be left as TBD
 - Life Cycle Architecture (LCA):**
 - more formal, with everything tracing upward and downward
 - no major unresolved issues or items, and closure mechanisms identified for any unresolved issues or items (e.g., "detailed data entry capabilities will be specified once the Library chooses a Forms Management package on February 15")
 - no more TBDs
 - there should no longer be any "possible" or "potential" elements (e.g., Entities, Components, ...)

- no more superfluous, unreferenced items: each element (e.g., Entities, Components, ...) either should reference, or be referenced by another element. Items that are not referenced should be eliminated, or documented as irrelevant

For further information: Refer to the completion criteria for each deliverable, for each phase.

- The Completion Criteria for each LCO/LCA deliverable, within the LCO/LCA phase respectively, can be used as "Exit criteria". There is no mandated number of pages per se, for a deliverable. Each package should meet all the phase completion criteria, and thus should thus contain the pertinent information. It is generally desirable to minimize the amount of detail, through conciseness: "less is more", as long as it conveys the appropriate amount of information, and meets all the exit criteria.
- The level of detail of each section should be risk-driven. For example, interface specification between the projects should be rigorously specified, as it is very risky to leave them ambiguous. However, one should avoid premature rigorous specification of user screen layouts, as it is risky to lock these in before users have had a chance to interact with them, and GUI-builder tools make it a low risk to iterate the screens with the users.
- Use visual models (whenever possible):
 - OCD/SSRD: block diagrams, context diagrams
 - OCD/SSRD/SSAD: UML diagrams
 - LCP: tables, Gantt charts, PERT charts
- Repetition of information within the various deliverables should be discouraged, and referencing the information should be encouraged. It is not enough to make things consistent by SIMPLY repeating sections. For example, there is no need to repeat the System Requirements in the Feasibility Rationale. The feasibility rationale should establish the feasibility and consistency of the operational concept, requirements, architecture, prototypes and plans, with respect to particular (referenced) System Requirements. While redundancy, among other deficiencies, leads to lengthy and repetitious documentation and creates extra update-consistency problems, referencing items enforces traceability.
- When referencing, avoid having:
 - "broken" or invalid references (e.g., references to something, such as Project Goal, Entity, Component, etc., that does not exist)
 - "blind" or vague references (e.g., "See FRD 2.2.3"—What exactly in FRD 2.2.3 is relevant?).
- If assumptions are made in the LCO/LCA package, it is important to reality-check the assumptions as much as possible. If you say somewhere "This assumes that COTS package will do X", determine the likelihood that the assumption is true. If the likelihood is low, identify this as a risk, and determine a risk management strategy for it. Avoid introducing non-customer and non-domain-expert assumptions.
- Do not just include text from the guidelines or outside sources in your deliverables, without relating the material to your project's specifics: no need to repeat in great detail software engineering principles and explanations taken from elsewhere.

General Tool Guidelines

- Rational Rose is the standard tool to develop the diagrams. Rose may not directly support all the diagrams (e.g., block diagrams, layered views). Some add-ins provide additional capabilities (e.g., ErWin for E-R diagrams). However, avoid having your architecture models developed with a large variety of tools, ranging from picture editing programs to other specialized software (e.g., flowcharting software).

General Formatting Guidelines

- There should be an explanation after each heading for the following subheadings: i.e., no two headings should be immediately next to each other.
- All documents should have the following information on the cover page
 - Document Title
 - Project Title
 - Team
 - Team Members and Roles
 - Date

- Document Version Control Information
- In general, use an outline form, e.g., for Organization Activities, instead of wordy paragraphs. In an outline form, items are easier to read, and important points stand out.
- Use numbered lists as opposed to bulleted lists to be able to reference items by their number, e.g., 'Organization Goal #2', which helps traceability.
- Include captions for each figure, table, etc., to encourage referencing and enforce traceability.

Final Remark

We can only suggest outlines for the LCO/LCA deliverables: in particular, there is no one-size-fits-all Requirements Description, or Life Cycle Plan structure. Authors should consider all of the items in the outline. If some of them are not applicable, it should be noted as "Not applicable" or "N/A" for future reference with some justification as to why this is so. Do not feel compelled to artificially create information simply to fill out a section that is not applicable to your project. Similarly, the document outline can be expanded if there is a need. However, it is not recommended to radically change the ordering of the various sections and to freely delete critical sections. The overriding goal is clear, concise communication. Standardized guidelines help with this: if you make substantial alterations, make sure they are clear, and well justified. Haphazard documentation is a major point of project failure.

Conventions Used

The following conventions are used in the guidelines.

Common Pitfalls: to warn against common mistakes

Course Guidelines: In general, the document templates are applicable to general classes of large software projects. However, guidelines should be tailored to the types and sizes of projects in this class.

The Five MBASE Invariants

The primary MBASE invariant is (1) Defining and sustaining a stakeholder win-win relationship throughout the system's life-cycle. If this is not accomplished, any win-lose relationships among the key stakeholders will generally lead to lose-lose outcomes in the long run.

Achieving stakeholder win-win involves getting the stakeholders to clarify, understand, and reconcile their success models. This activity drives the choice of the project's process, product, and property models.

Some additional candidate invariants to discuss are the terms used above, e.g.:

- a. A project organized to achieve the stakeholder win-win objectives is the focus of MBASE activity.
- b. A system characterized by a system boundary scopes the project's authority and responsibility.
- c. The system's entire life cycle is the appropriate time scope of the project's authority and responsibility.

Factors complicating the nature of these as invariants are product lines, strategic partnerships for families of products, and enterprise integration frameworks.

Two further MBASE invariants are (2) Using the MBASE Model Integration Framework and (3) Using the MBASE Process Integration Framework. For (2), I think chart 1 in Figure 1 of the IT Pro article is the best representation. For (3), I think the expanded version of chart 2 in Figure 1 is better: the chart called "MBASE Conceptual framework" in most of the MBASE presentations. These provide static (2) and dynamic (3) relationships and constraints across the various MBASE models being integrated. If these relationships and constraints are not satisfied, model clashes will occur. The Process Integration Framework (3) also introduces the WinWin Spiral Model and LCA Anchor Point, which are discussed below.

The next MBASE invariant is (4) Using the LCO, LCA, and IOC Anchor Point milestones. These milestones represent fundamental stakeholder life cycle commitment points analogous to the real-life commitment points of getting engaged, getting married, and having your first child. If the project fails to address or satisfy such commitment points, it will fall into such tar pits as analysis paralysis, unrealistic expectations, requirements creep, architectural drift, COTS shortfalls and incompatibilities, unsustainable architectures, traumatic cutovers, or useless systems.

Implicit in the identification of the Anchor Point milestones as invariants are their constituent elements and associated reviews and pass/fail conditions. Thus, for LCO and LCA, the essential content of an Operational Concept Definition, Requirements Definition, Architecture Definition, Life Cycle Plan, Key Prototypes, and Feasibility Rationale are included, as are the LCO and LCA Architecture Review Board reviews and their Feasibility Rationale-based pass-fail criteria. For IOC, the essential content of the IOC deliverables for software, personnel, and facility preparation are included, as are the Transition Readiness Review, Release Readiness Review, and their associated pass-fail criteria.

This does not imply that these all need to be separate events or definition documents. For a 4GL application, the project has committed to the 4GL infrastructure's architecture, and the project can merge its LCO and LCA packages and reviews. For simple systems or upgrades, the project may choose to integrate its OCD, SSRD, and SSAD.

These and similar considerations lead to the Final MBASE invariant: (5) Ensuring that the content of MBASE artifacts and activities is risk-driven. If this is not the case, the project will either fail to achieve critical success conditions or will waste effort in performing unnecessary or dysfunctional activities. The risk criterion is the best way for a project to determine "how much is enough" specifying, prototyping, reusing, testing, documenting, reviewing, etc. Thus for example, a simple application to automate some prespecified tax tables may have no risk of not prototyping, and no need to prototype at all.

MBASE Variants

These are initially described in less detail in order to move the process of reviewing and iterating these definitions along.

- (1) Use of particular success, process, product, or property models. The Process Model Decision Table is a good example of such variation, and, as Dan and Nikunj have discussed, a good objective for deriving product, property, etc. model variants, although I think this a more long-term objective.
- (2) Choice of process or product representation. Thus, UML may be appropriate for many applications, but not for a small upgrade to a well-documented legacy system using structured analysis and design, for example. The choice may vary by legacy constraints, nature of application (pure dataflow vs. pure event-based), staff familiarity, or tool support.
- (3) Degree of detail of process, product, property, or success modeling. Given Invariant 5, the degree of detail will vary based on risk considerations.
- (4) Number of spiral cycles or builds between anchor points. This can vary based on risk, project size, staff availability, or other system constraints (e.g., hardware, budget, schedule).
- (5) Mapping of activities onto Inception-Elaboration-Construction-Transition phases. With respect to Nikunj's MBASE Activity Diagram, most of the activity elements will be spread across multiple phases. Their relative activity levels by phase may vary a lot. For example, stakeholders may require a lot of negotiation of top-level requirements in Inception and little negotiation of detailed requirements in Elaboration, or vice versa.
- (6) Mapping of staff levels onto activities. The example just discussed is a good example of this. Thus, any MBASE effort and schedule distributions by phase and activity that we put into COCOMO II will be necessarily imprecise.

Summary Table. MBASE Invariants and Variants

Invariants	Variants
1. Defining and sustaining a stakeholder win-win relationship through the system's life-cycle.	1. Use of particular success, process, product, or property models.
2. Using the MBASE Model Integration Framework.	2. Choice of process or product representation.
3. Using the MBASE Process Integration Framework	3. Degree of detail of process, product, property, or success modeling.
4. Using the LCO, LCA, and IOC Anchor Point milestones.	4. Number of spiral cycles or builds between anchor points.
5. Ensuring that the content of MBASE artifacts and activities is risk-driven.	5. Mapping of activities onto Inception-Elaboration-Construction-Transition phases.
	6. Mapping of staff levels onto activities.

Operational Concept Description (OCD)

Purpose

- Describe the overall context of the system to be developed, why it's being built, what exists now, and where the project is starting from
- Describe to the stakeholders of the system to be developed (“developed” is meant to include such terms as “enhanced”, “updated”, “re-engineered”, “automated”), how the system will work in practice once it is deployed
- Enable the operational stakeholders to evolve knowledgeably from their current operational concept to the new operational concept, and to collaboratively adapt the operational concept as developments arise, to make clear the value of developing the new system

Completion Criteria

Below are the completion criteria for the Operational Concept Description for the two phases:

- Life Cycle Objectives (Inception Phase)
- Life Cycle Architecture (Elaboration Phase)

Life Cycle Objectives (LCO)

- Top-level system objectives and scope
 - Organization Context and Goals
 - Current system overview and shortfalls
 - System Boundary: project focus
 - System Environment
 - Evolution Considerations
- Operational concept
 - Operational stakeholders identified
 - Organizational responsibilities determined and coordinated with clients
 - Main operational scenarios coordinated with clients
 - System Concept
- Shared vision and context for stakeholders
 - Common vision and goals for system and its evolution
 - Common language and understanding of system constraints
 - Operational concept satisfiable by at least one system/software architecture
 - Capabilities rationalized by business case analysis in Feasibility Rationale

Life Cycle Architecture (LCA)

- Elaboration of system objectives and scope by system increment
- Elaboration of operational concept by system increment
- All stakeholder-critical nominal and off-nominal scenarios coordinated with clients
- Operational concept satisfiable by the architecture in the SSAD
- Tracing between Project Goals, and Organization Goals and Activities
- Tracing between Capabilities and Project Goals and Organization Activities

Intended Audience

- Customer for Domain Description and shared vision
- Domain Expert for initial System Analysis
- Use language, and define CDL, appropriate to intended audience

Participants

- Same stakeholders as WinWin negotiation
- Establish a concept of operation that all stakeholders agree upon

High-Level Dependencies

- WinWin negotiations give:
 - Capabilities (Priority and Rationale for proposed changes)
 - Terms for the Domain Description
 - Project Goals and Constraints
 - Levels of Service
- OCD yields:
 - Project, Capability and Level of Service Requirements for SSRD
 - Domain Description and Initial Analysis for SSAD
 - Stakeholder and Organizational Responsibilities for LCP
 - Business Case analysis parameters for FR

Outline

1. Introduction
 - 1.1 Purpose of the Operational Concept Description
 - 1.2 References
2. Domain Description
 - 2.1 Organization Background
 - 2.2 Organization Goals
 - 2.3 Current System
 - 2.4 Entity Model
 - 2.5 Interaction Model
 - 2.6 Organization Activity Model
 - 2.7 Current System Shortfalls
3. Proposed System
 - 3.1 Project Goals and Constraints
 - 3.2 Capabilities
 - 3.3 Levels of Service
 - 3.4 Proposed System Description
 - 3.4.1 Statement of Purpose
 - 3.4.2 Proposed Entities
 - 3.4.3 Proposed Interactions
 - 3.4.4 Proposed Activities
 - 3.5 Redressal of Current System Shortfalls
 - 3.6 Effects of Operation
 - 3.6.1 Operational Stakeholders
 - 3.6.2 Organizational Relationships
 - 3.6.3 Operational Policies and Constraints
 - 3.6.4 Operational Impacts
 - 3.6.5 Organizational Impacts
4. Common Definition Language
5. Appendix

1. Introduction

1.1 Purpose of the Operational Concept Description

- This paragraph shall summarize the purpose and contents of this document and identify the project stakeholders
 - Current life cycle phase
 - The specific system whose operational concept is described here is: [name-of-system]
 - Its operational stakeholders are: [Describe the stakeholder roles and organizations]
 - Use specific names, titles and roles
- Show how your particular Operational Concept Description meets the completion criteria for the given phase

Common Pitfalls:

- Simply repeating the purpose of the document from the guidelines

1.2 References

- Provide complete citations to all documents, meetings and external tools referenced or used in the preparation of this document and their outputs.
- This should be done in such a manner that the process and information used can be traced and used to reconstruct the document if necessary

2. Domain Description

The Domain Description (which focuses on the current system and organization) establishes the context for the system to be developed and determines what is and is not relevant to the project. It consists of several views, which describe the domain of the project (i.e., the context in which the project will be developed and deployed, including the organization, stakeholders, etc.) at various levels of generality from the customer's and domain expert's perspective. It provides the distilled rationale for:

- **Why** the system is being built
- **What** overall organization goals and activities the proposed system will support and be responsible for when deployed
- **Where** the project is starting from (i.e. "what" is there at present to build upon, what is missing, and what is needed, etc.)

The goal is to describe the sponsoring and user organization as relevant to the project, and provide a working context for the System Analysis ("What" the proposed system is precisely). The working context serves to avoid building a system that is too general by restricting its scope to what adds value for the critical stakeholders; this provides a tangible means to measure what is or is not relevant to the project. All sections of the Domain Description should be written in a language understood by all the stakeholders in the project, in particular customers and domain experts. This generally means describing concepts at a high, non-technical level.

CS 577 Guidelines:

Don't go too high in the organization for your project's organization background and goals. USC's overall goals may include improving USC's rank in lists of the top U.S. universities, but it is too hard to relate the project goals for a multimedia archive to such organization goals. We recommend using USC's Digital Library Initiatives as an appropriate organizational context. Here is a working good statement for these initiatives:

"To make USC's reference materials more rapidly, reliably, easily and effectively accessible to the USC community, subject to appropriate information protection, fairness, and economic constraints."

At the level of organization goals shown above, the mapping to project goals is more meaningful and straightforward. For your library information system, it is appropriate to elaborate these overall organizational goals to relate to your project goals (e.g., defining an aspect of "easily accessible" as bringing the reference materials to the user rather than vice versa), or to particular goals of your client's organization (e.g., Seaver Science Library, Marshall School of Business Library).

2.1 Organization Background

- Provide a brief overview (a few sentences) of the organization (within the project's context) sponsoring the development of this system
- Provide brief overview (a few sentences) of the organization that would be the end user and maintainer of the system (these may or may not be the same as the sponsoring organization)
- Include the above organizations' mission statements and/or their objectives and goals (summarize relevant portions)

2.2 Organization Goals

- Identify the broad and high-level objectives and/or aspirations of the sponsoring organization and the organization using the system. The goals should be relevant to, but independent from, the proposed system. System-specific goals would be documented as Project Goals (OCD 3.1)
- Include only the goals that indicate what the organization wishes to achieve by having the proposed system, e.g., improve efficiency of Inter-Library Loan borrowing
- The Organization Goals should be Measurable and Relevant to the current project (M.R.).
- Use a brief enumerated list. E.g.:
 1. Improve cost ... via ...
 2. Improve speed ... via ...
 3. Improve quality... via ...
 4. Improve customer satisfaction ... via ...

Test Questions for M.R.: By LCA, each organization goal should be able to clearly answer:

M: "What is a measure of this goal?"

R: "Why is it relevant to the organization?"

Common Pitfalls:

- Specifying Project Goals as Organization Goals
- Not clearly indicating the Measure and/or the Relevance of the goals, to the Organization and the Proposed System
- Developers introducing Organization Goals. Organization Goals should only come from interviewing customers and domain experts: have them describe the M. and R.
- Having superfluous organization Goals that are never referenced by Organization Activities, Project Goals, Capabilities or System Requirements (they should be eliminated by the LCA).

2.3 Description of Current System

Provide a brief, high-level overview of the current operational system as it exists in the organization prior to building the new system. Keep in mind that the current system may be manual or ad hoc.

- Explain the current system's (if available) scope within the organization
 - What the current system **does**
 - What it **does not do** (avoid overlap with system shortfalls)
- Include a high-level Block Diagram of current system which depicts the boundaries of the current system.
- Orient the content of this section strongly towards the proposed system, which will be described in the System Analysis. Leave out clearly irrelevant items. E.g., the Inter-Library Loan department at USC may be interested in automating the current manual process of Inter-Library Loan Borrowing (by which USC borrows reference material from other libraries). In that case, there is no need to describe in great detail the Inter-Library Loan Lending process (by which other libraries borrow items from the USC Libraries), if the latter is already automated.

2.4 Entity Model

- The domain entities are "things" exist in the domain in which the current system exists, are relevant to the current system and have significance to the organization goals. Many of these "things" are relevant to the proposed system: they will all be represented in the proposed system. Therefore, it is vital to identify and clarify these "things" as early as possible to encourage faithfulness of the proposed system to the domain. The entity model shows these "things" and their inter-relations.
- Only top level entities should be identified, an example of the desired level of abstraction of an entity would be the digital assets and a catalog of assets within a digital library system; videos, manuscripts and pamphlets are more low-level and not appropriate for being included in the OCD. More details can be provided in the SSAD Enterprise Model.
- The Entity Model should not include any software components or any proposed entities that do not currently exist in the domain. E.g. library information systems (e.g. *SIRSI*), software components (e.g., *Database*) or users (e.g., *System Administrator*) introduced by the proposed system.
- Identify the information represented by each of the entities. An entity is something whose information needs to be stored in the system.

Common Pitfalls:

- Including the current or proposed system as an Entity
- Including entities for which no information needs to be stored.
- Not listing a large number of possible entities before selecting which ones to include
- Using system components for the proposed system as domain entities. These do not exist until the system is built
- Including an Entity that has no direct relevance or relation to a component in the Component Model [SSAD 2.1]
- Having superfluous entities that are never referenced by components (they should be eliminated by the LCA)

- Including design related details, they belong to the Enterprise Model [SSAD 2.3]

2.5 Interaction Model

- The Interaction Model shows how the users of the system perform activities related to the domain using the current system.
- The interactions should identify external systems and users with which the current system has to interact. Even when the current system does not provide automation, the interactions among users and the system can be determined based on the current system boundary as described in OCD 2.3.
- [Consistent with Organization Activity Model (OCD 2.6)]

2.6 Organization Activity Model

- The Activity Model provides a simple overview of the sponsoring and user organization's activities within the domain and describes their relevant workflows. The Activity Model should describe only those activities that are relevant to the proposed system (e.g., activities that the proposed system will automate or enhance or the activities that the proposed system will interact with). The Activity Model may include activities of the current system (if one exists).
- The activity model may show which domain entities are exchanged by the system users (including external systems) during interactions.
- Organization activities support or carry out organization goals: note which goal the activity supports.
- The Organization Activity Model provides the contextual basis and scope for the proposed system's behaviors
- Avoid overly technical or implementation related activities unless they are already present in the current system
- An example of an appropriate level of aggregation of an activity for a digital library would be “Add an asset to the library’s collection”
- [Consistent with Interaction Model 2.5]

Common Pitfalls:

- Including system behaviors (of only the proposed system) as activities
- Having superfluous activities that are not referenced by anything later. These should be eliminated by the LCA
- Including organization activities that do not relate to any organization goals. These should be eliminated by the LCA

2.7 Current System Shortfalls

- Describe limitations of the current system, in particular, how the current system does not fulfill the Organization Goals (OCD2.2), or does not support some of the Organization Activities (described in detail in OCD 2.5).

3. Proposed System

This section describes the concept and effects of the proposed system. Specifically it addresses the following questions:

- *What* the proposed system is
- *How Well* it should perform
- NOT *How* it is implemented in software

3.1 Project Goals and Constraints

- Project Goals are factors, project-level constraints and assumptions that influence or contribute to the eventual outcome of the project: such as legacy code or systems, computer system compatibility constraints, COTS compatibility constraints, budget limits and time deadlines. Project Goals may carry out or support Organization Goals and Activities.

- Project-level constraints correspond to the Constraints in the Spiral Model cycles; Capabilities and Levels of Service correspond with spiral model Objectives.
- Project Goals are separate from Capabilities: Project Goals usually affect many parts of the system, whereas Capabilities address more local and specific areas
- Project Goals should be M.R.S. (Measurable, Relevant, Specific). Note that the project goals may also be relative to the infrastructure on which the system is based.
- Defer Levels of Service till OCD 3.3

Test Questions for the MRS criteria:

M: "How is the goal measured with respect to the proposed system?"

R: "Is this related to any Organization Goal or any external constraint?"

S: "What specific part of the system is this relevant to? What are the specific acceptable levels or thresholds with respect to the measures used? What specific parts of the system are to be measured?"

Common Pitfalls:

- Including Organization Goals as Project Goals
- Including Levels of Service as Project Goals (defer those till OCD 3.3)
- Including Capabilities as Project Goals, these should be described in OCD 3.2
- Including Project Goals that do not reference Organization Goals or Activities
- Including Project Goals that are not referenced by Project Requirements [SSRD 2]

3.2 Capabilities

- This section describes overall what products and services the domain expert ideally expects from the proposed system with respect to their organization, including desired modifications to the current system.
- Capabilities provide a high level overview of **broad categories** of system behaviors, as opposed to an operational breakdown provided by System Requirements. Capabilities should realize high-level activities in the Organization Activity Model (Reference as appropriate).
- Capabilities correspond with spiral model Objectives.
- Capabilities should be testable (so one can determine if the responsibility has been handled)
- An example of the desired level of granularity of a Capability would be "Provide a virtual experience of touring the Doheny Library" or "Report all leave records of the employees for a given period of time"
- Each capability may require several iterations. Use the "just do it" approach to eliminate the pressure to get it all right on the first pass (like writing a rough draft for a term paper). "Go with what you know" and plan to iterate it and make adjustments.
- [Consistent with Organization Activity Model (OCD 2.6)]

Common Pitfalls:

- Including System Requirements as Capabilities. Those belong in SSRD 3.2
- Including Levels of Service as Capabilities. Those belong in OCD 3.3
- Including System Behaviors as Capabilities. Those belong in SSAD 2.2
- Including too many Capabilities for a relatively small system (some of them may be either System Requirements or System Behaviors)

3.3 Levels of Service

- Define the kinds of levels of Service required in the System (i.e., "how well" the system should perform a given capability).
- Levels of Service should be M.R.S. (Measurable, Relevant, Specific). Measures should specify the unit of measurement and the conditions in which the measurement should be taken (e.g., normal operations vs. peak-load response time). Where appropriate, include both desired and acceptable levels. Again, don't get too hung up on measurability details.

- Indicate how the Levels of Service are relevant to the Organization Goals, Capabilities and Project Goals
- Levels of Service correspond with Spiral Model Objectives.
- It is important at this point, not to overburden the system's analysis with Levels of Service that are not expressly requested by the customer.
- Level of Service Requirements (SSRD 5) is supposed to be more specific than the Levels of Service (OCD 3.3). However, it is often recommended to specify both acceptable and desired quality levels, and leave the goals flexible to produce the best balance among Level of Service Requirements (since some Level of Service Requirements conflict with each other, e.g., performance and fault-tolerance).
- If the Level of Service is well-defined, it is possible to simply refer to it in the OCD, without repeating it, in the SSRD
- [Consistent with Organization Goals (OCD 2.2)]
- [Consistent with Level of Service Requirements (SSRD 5)]

Common Pitfalls:

- Overburdening the system with Levels of Service that are not expressly requested by the customer
- Including Levels of Service that do not reference Project Goals or Organization Goals
- Levels not satisfying the M.R.S. criteria
- Including Project Goals as Levels of Service, these are described in OCD 3.1
- Including Capabilities as Levels of Service, these are described in OCD 3.2

3.4 Proposed System Description

- The section provides a brief description of the proposed system, and explains how the new system will address the current system's shortfalls.

3.4.1 Statement of Purpose

- Provide a brief synopsis of the overall system. It should be more general than a problem statement (it does not need to specify a problem). The statement of purpose identifies the overall solution of the problem through the proposed system.
- Include a top-level block diagram of the proposed system. The block diagram should not identify any software components such as a web server or a database. It should only focus on the major subsystems of the proposed system.
- Identify the systems with which the proposed system is expected to interact.
- [Consistent with Organization Background (OCD 2.1)]
- [Consistent with Organization Goals (OCD 2.2)]
- [Consistent with Operational Stakeholders (OCD 3.6.1)]

Common Pitfalls:

- Simply listing Capabilities and Behaviors as a Statement of Purpose
- Not including on the block diagram all the key operational stakeholders as listed in OCD 3.6.1
- Confusing the Statement of Purpose with Organization Goals
- Including architectural decisions or implications (e.g., "The purpose is to design a client-server ...")
- Including too many architectural details
- Not including relevance to the Organization Background (OCD 2.1)

3.4.2 Proposed Entities

- At times, the system will introduce new entities that had no analogical parts in the existing domain. Such entities should be described in this section. The components in the system will often represent entities or groups of entities relevant to the proposed system.
- The proposed entities should not include new software components (e.g., *Database*) or roles for which information is not required to be tracked (e.g., *System Administrator*) introduced by the proposed system.

- An example of a proposed entity for a new Digital Manuscript System would be a unique catalog of items being digitized.
- Relations of the proposed entity to the existing domain entities should be depicted.

Common Pitfalls:

- Including the proposed system as an Entity
- Using system components and external systems in the proposed system as proposed entities.
- Including an Entity that has no direct relevance or relation to a component in the Component Model
- Including “possible” proposed entities in LCA (they are acceptable at the LCO)

3.4.3 Proposed Interactions

- This section should describe a set of scenarios that illustrate, from the user's perspective, what will be experienced when utilizing the system under various situations.
- Scenarios should illustrate the role of the new or modified system, its interaction with users, its interface to other systems, and modes identified for the system.
- Identify the operational usage characteristics for each of the proposed interactions to understand the scale needs of the proposed system.
- Scenarios are defined as follows (IEEE Software, March 1994),:

In the broad sense, a scenario is simply a proposed specific use of the system. More specifically, a scenario is a description of one or more end-to-end transactions involving the required system and its environment. Scenarios can be documented in different ways, depending up on the level of detail needed. The simplest form is a use case, which consists merely of a short description ; more detailed forms are called scripts. These are usually represented as tables or diagrams and involve identifying an action and the agent (doer) of the action.
- Scenarios are illustrated through the interfaces that focus on the appearance and style aspects of user interaction. You may have to develop several prototypes to specify the look and feel of the intended system. This section may reference prototype screens included in the Appendix. Other diagrams, such as storyboards (low-fidelity prototypes) may be also used as necessary
- Although scenarios are useful in acquiring and validating requirements, they are usually not themselves requirements, because they describe the system's behavior only in specific situations; a requirement, on the other hand, usually describes what the system should do in general.

Common Pitfalls:

- Simply including screen shots without any scenario description
- Too many screenshots. Including all screens even though they may not represent important interactions in the proposed system.

3.4.4 Proposed Activities

- Describe the workflows in the proposed concept of operation, which describes how the various operational stakeholders interact with the system and each other and exchange information through proposed entities. The workflow can also identify the artifacts and information flowing between these stakeholders with or without the proposed system
- Proposed activities should demonstrate how the organization activities are being supported through the proposed system.

3.5 Redressal of Current System Shortfalls

- Describe how the successful development and installation of the proposed system would address the shortfalls in the current system and allow the Organization to meet its Goals. Note that the proposed system can either, extend, enhance or replace the current system.
- [Consistent with Current System Shortfalls (OCD 2.7)]
- [Consistent with Organization Goals (OCD 2.2)]

Common Pitfalls:

- Confusing with Organization Goals
- Not including relevance to the Organization Background (OCD 2.1)

3.6. Effects of Operation

This section presents the effects of the proposed concept of operation and describes how the system's operational stakeholders (users, operators, maintainers, inter-operators, managers, etc.) will interact with the system, and how they will interact with each other in the context of the system.

3.6.1 Operational Stakeholders

- Describe the operational stakeholders (e.g., users, system administrator, etc...) who will interact with the new or modified system, including, as applicable, organizational structures, training/skills, responsibilities, and interactions with one another.
- Do not include development-related stakeholders and organizations such as developers, software maintainers and customers.
- Provide organization charts showing the responsibility relations between the various organizations involved in the software life cycle process, and identify the key responsible personnel within each organization.
- For each stakeholder, list:
 - Major activities performed by that stakeholder
 - Assumptions about User Characteristics
 - Frequency of usage
 - Expected expertise (with software systems and the application domain)
- [Consistent with Activity Model (OCD2.6)]
- [Consistent with Stakeholder Responsibilities (LCP 3.1)]

Common Pitfalls:

- Including development-related agents and stakeholders

3.6.2 Organizational Relationships

- Include a specialized (i.e., derived from the main organizational chart) organization chart indicating the relations among the system's operational stakeholders' management hierarchies.
- This serves to verify the following:
 - Project scope fits within client's authority scope or cross organizational boundaries
 - Solution does not introduce organizational friction
 - Solution does not shift power, confuses lines of authority nor puts outside parties on critical path for regular operational procedures
- The operational stakeholders' development-related responsibilities, as well as development-related stakeholders, during the various phases of the project life cycle, will be defined in LCP 3.1
 - Organizational Responsibilities
 - Global Organization Charts
 - Organizational Commitment Responsibilities
 - Stakeholder Responsibilities

Common Pitfalls:

- Mixing class hierarchies and reporting hierarchies in an Organization Chart
- Mixing people and organization units in the same Organization Chart
- Including development-related agents and stakeholders

3.6.3 Operational Policies and Constraints

- Include additional proposed policies and constraints for usage of the new capabilities (e.g., policies on information access, borrowing of materials, copyright protection, etc.)

- You may also reference any existing organization policies (include in the Appendix)

3.6.4 Operational Impacts

List impacts of the new operational concept on operational personnel, procedures, performance and management functions due to parallel operation of new and existing system, during transition, and likely evolution of roles and responsibilities, thereafter.

3.6.5 Organizational Impacts

Describe anticipated organizational impacts on the user, customer, once the system is in operation. These impacts may include modification of responsibilities; addition or elimination of responsibilities or positions; need for training or retraining; and changes in number, skill levels, position identifiers, or location of personnel in various modes of operation.

4. Common Definition Language

- Include an alphabetical listing of all uncommon or organization-specific terms, acronyms, abbreviations, and their meanings and definitions, to understand the Domain Description
- Avoid implementation technology terms at this point
- CDL items are often answers to questions that you ask to the client: “What does this mean?”

5. Appendix

- As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided.
- Include supporting documentation or pointers to electronic files containing:
 - Policies (e.g., applicable Copyright Laws)
 - Descriptions of capabilities of similar systems
 - Additional background information
 - Prototyping results

System and Software Requirements Definition (SSRD)

Purpose

- Describe capability requirements (both nominal and off-nominal): i.e., the fundamental subject matter of the system, measured by concrete means like data values, decision-making logic and algorithms.
- Describe Level of Service Requirements (sometimes referred to as Non-functional requirements): i.e., the behavioral properties that the specified functions must have, such as performance, usability, etc. Level of Service Requirements should be assigned a unit of measurement
- Describe Global constraints: requirements and constraints that apply to the system as a whole. For example, the customer for the system is a global constraint, as is the Purpose of the System. Those constraints include:
 - Interface Requirements
 - Budget and Schedule Requirements
 - Implementation Requirements
- Mandates and instructions on how the system must be implemented ("must", "shall", "will"), with respect to the general technology

Completion Criteria

Below are the completion criteria for the System and Software Requirements Definition for the two phases:

- Life Cycle Objectives (Inception Phase)
- Life Cycle Architecture (Elaboration Phase)

Life Cycle Objectives (LCO)

- Top-level capabilities, interfaces, quality attribute levels, including:
 - Growth vectors (evolution requirements)
 - Priorities
- Stakeholders' concurrence on essentials
- Requirements satisfiable by at least one system/software architecture

Life Cycle Architecture (LCA)

- Elaboration of capabilities, interfaces, quality attributes by iteration
 - Resolution of TBD's (to-be-determined items)
 - Elaboration of evolution requirements
- Stakeholders' concurrence on their priority concerns (prioritization)
- Traces to SSAD (and indirectly to FRD, LCP)
- Requirements satisfiable by the architecture in the SSAD

Intended audience

- Domain expert and Customer (decision makers)
- Implementers and Architects

Participants

Same stakeholders as WinWin negotiation

High-Level Dependencies

- SSRD depends on WinWin taxonomy
 - Outline of SSRD evolves from taxonomy
 - There is no one-size-fits-all taxonomy or requirements description
 - Importance of adapting taxonomy to domain
- SSRD depends on OCD for:
 - Statement of Purpose

- Project Goals and Constraints
- Capabilities
- SSRD depends on prototype for:
 - User interface requirements
- SSRD depends on FRD for:
 - Changes Considered but Not Included
- Additional documents depend on SSRD:
 - SSAD to obtain (and consistency trace):
 - System Requirements
 - Project Requirements
 - LCP to relate requirement priorities to system increments or to requirements to be dropped in a design-to-cost/schedule development plan
 - FRD to check for satisfaction of:
 - Capability Requirements
 - Interface Requirements
 - Level of Service Requirements
 - Evolution Requirements

Outline

1. Introduction
 - 1.1 Purpose of the System and Software Requirements Definition Document
 - 1.2 References
2. Project Requirements
 - 2.1 Budget and Schedule
 - 2.2 Development Requirements
 - 2.3 Packaging Requirements
 - 2.4 Implementation Requirements
 - 2.5 Support Environment Requirements
3. Capability Requirements
 - 3.1 System Definition
 - 3.2 System Requirements
 - 3.2.1 Nominal Requirements
 - 3.2.2 Off-Nominal Requirements
4. System Interface Requirements
 - 4.1 User Interface Requirements
 - 4.1.1 Graphical User Interface Standards
 - 4.1.2 Command-Line Interface Requirements
 - 4.1.4 Diagnostics Requirements
 - 4.2 Hardware Interface Requirements
 - 4.3 Communications Interface Requirements
 - 4.4 Other Software Interface Requirements
5. Level of Service Requirements
6. Evolution Requirements
 - 6.1 Capability Evolution Requirements
 - 6.2 Interface Evolution Requirements
 - 6.3 Technology Evolution Requirements
 - 6.4 Environment and Workload Evolution Requirements
7. Common Definition Language for Requirements
8. Appendices
 - A. Standards Specifications
 - B. Interface Specifications

1. Introduction

1.1 Purpose of the System and Software Requirements Definition Document

- Summarize the purpose and contents of this document with respect to the particular project and people involved
- Avoid generic introductions as much as possible: for instance, you can show how your particular System and Software Requirements Definition meets the completion criteria for the given phase

Common Pitfalls:

- Simply repeating the purpose of the document from the guidelines

1.2 References

- Provide complete citations to prior and current related work and artifacts, documents, meetings and external tools referenced or used in the preparation of this document
- Useful for consistency checking and traceability

2. Project Requirements

- Project Requirements are general constraints and mandates placed upon the design team, as well as non-negotiable global constraints: e.g., solution constraints on the way that the problem must be solved, such as a mandated technology. Project Requirements could summarize process-related considerations from the Life Cycle Plan such as preliminary Schedule and Budget considerations.
- Project Requirements are such that, if they were left unmet, then the proposed system would not be acceptable or would not satisfy Win conditions for the success-critical stakeholders.
- Project Requirements should be a refinement of Project Goals (OCD 3.1): Include reference to the corresponding Project Goal
- Project Requirements should be M.A.R.S. (Measurable, Achievable, Relevant, Specific)
- Defer Project Requirements about "how well" the system should perform to the Level of Service Requirements section (SSRD 5)
- Example: "The system shall use the Microsoft Active Server Pages technology"
- Example: "The system must have the core capabilities [specify which ones] by IOC within twelve weeks"

Common Pitfalls:

- Including Level of Service Requirements as Project Requirements. Those belong in SSRD 5.
- Introducing Project Requirements that do not parallel or trace back from Project Goals (OCD 3.1). One Project Goal (OCD 3.1) may lead to several Project Requirements (SSRD 2)
- Introducing Project Requirements not mandated by the customer
- Introducing superfluous Project Requirements that are not referenced by System Requirements
- Not relating each Project Requirement to the corresponding Project Goal

Additional Guidelines:

Project Requirements should be able to answer the following Test Questions:

M: "How is the requirement measurable and testable with respect to the proposed system?"

A: "How must this requirement be realized in the system (what are the general technology considerations)?"

R: "Is this requirement relevant to the proposed system?"

R: "Does this requirement achieve any Project Goal?"

S: "What are the specific details, values, or conditions that must be measured to test the satisfaction of this requirement?"

2.1 Budget and Schedule

- Identify the available time for developing and delivering the system
- Provide the budget limits for the software and system development. Often the clients would require that existing systems be used instead of buying new ones and COTS software be used based on existing licenses. This fact should be noted in the budget requirements

2.2 Development Requirements

Describe any requirements that constrain the design and implementation of the system. These requirements may be specified by reference to appropriate standards and specifications.

- **Tools Requirements**

Describe any requirements that constrain the use of tools for the design and construction of the system (e.g., program generators, integrated development environments, COTS tools, etc.). Include version requirements (if applicable).

- **Programming Languages Requirements**

Describe constraints on the use of a particular programming language for the design and the construction of the system.

- **Computer Resource Requirements**

Describe any constraints on the hardware or software to be used for the development, testing or deployment of the system.

- **Computer Hardware Requirements**

Describe any requirements regarding computer hardware that must be used by the system. The requirements shall include, as applicable, number of each type of equipment, type, size, capacity, and other required characteristics of processors, memory, input/output devices, auxiliary storage, communications/network equipment, and other required equipment.

- **Computer Hardware Resource Utilization Requirements**

Describe any requirements on the system's computer hardware resource utilization, such as maximum allowable use of processor capacity, memory capacity, input/output device capacity, auxiliary storage device capacity, and communications/network equipment capacity. The requirements (stated, for example, as percentages of the capacity of each computer hardware resource) shall include the conditions, if any, under which the resource utilization is to be measured.

- **Computer Software Requirements**

Describe any requirements regarding computer software that must be used by, or incorporated into, the system. Examples include operating systems, database management systems, communications/network software, utility software, input and equipment simulators, test software, and manufacturing software. The correct nomenclature, version, and documentation references of each such software item shall be provided.

- **Computer Communication Requirements**

Describe any requirements concerning the computer communications that must be used by the system. Examples include geographic locations to be linked; configuration and network topology; transmission techniques; data transfer rates; gateways; required system use times; type and volume of data to be transmitted/received; time boundaries for transmission/reception/response; peak volumes of data; and diagnostic features.

- **Standards Compliance Requirements**

Describe any particular design or construction standards that the system must comply with, and provide a reference to the standard.

Example: "The system's object broker capabilities shall comply with the OMG CORBA standard".

2.3 Packaging Requirements

Describe any requirements for packaging, labeling, and handling the system for delivery

- Installation
 - Assumptions
 - Deployment hardware and software
 - Installer experience/skills
- Post-installation requirements
 - Re-packaging
 - Uninstall
- Transport and delivery

2.4 Implementation Requirements

- Personnel
- Training

These should be consistent with personnel and training identified in the LCP 3.2.3

- Development environment
- Development software
- Development hardware

2.5 Support Environment Requirements

- Describe any required Software Support Environments to be used for the support of the delivered system
- Describe the skill levels of required support personnel and the frequency of support interactions required in terms of bug fixes, future software releases and the reporting and tracking of problems.

3. Capability Requirements

This section describes the capability requirements of the proposed system.

3.1 System Definition

- Provide a brief overview of what the software system is. This could consist of enumerating at a high-level the various components or modules of the system.
- The System Definition should be a refinement of the Statement of Purpose (OCD 3.4.1). The System Definition needs to focus on what the system does with respect to the technology that will do it, and therefore, may introduce very high-level design indications
- [Consistent with Statement of Purpose (OCD 3.4.1)]

Common Pitfalls:

- Not tracing back the System Definition to the Statement of Purpose (OCD 3.4.1)
- Simply repeating the Capabilities or the System Requirements as a System Definition
- Too much detail in the System Definition

3.2 System Requirements

- System Requirements should be a refinement of Capabilities (OCD 3.2). They need to trace from and parallel Capabilities. Each Capability must translate into at least one System Requirement (be sure to reference which one)
- System Requirements need high-level design specifics (i.e., *what* and *how* it must be implemented generally, and *how* the system will work)

Common Pitfalls:

- Confusing between Operational Modes and sub-systems
- Confusing between Operational Modes and Off-Nominal Requirements
- Confusing between modes and states
- Including Level of Service Requirements ("how well the system does something") as functional System Requirements ("what the system is to do")
- Including System Requirements that do not parallel or trace back from Capabilities (OCD 3.2). One Capability may lead to several System Requirements

3.2.1 Nominal Requirements

- Include Nominal Functional Requirements or Capabilities
- During LCO, include only major requirements
- During LCA, add less important requirements
- For every Capability (OCD 3.2), describe the corresponding System Requirement(s)
- Prioritize the System Requirements, to validate that the overall life cycle strategy matches the system priorities (FRD 3.2).
- Check that every requirement has its most critical scenarios specified in the Proposed Activities (OCD 3.4.3)

Common Pitfalls

- It is important that the requirements are testable and specific: if one can interpret different behavioral sequences (not operational) from the statement of the requirement, the requirement is not well specified.
- Including System Requirements that do not reference Capabilities, Project Goals, Levels of Service, Project Requirements or Level of Service Requirements
- If a System Requirement traces back to multiple Capabilities, it probably indicates that you have included System Behaviors as Capabilities

3.2.2 Off-Nominal Requirements

- Include Off-Nominal Functional Requirements (i.e., Requirements on how to deal with special circumstances or undesired events, errors, exceptions and abnormal conditions.
- Example: "If the request cannot be completed, the server should add an entry to the error log file indicating the time the error occurred and the returned error code."
- During LCO: define high-risk off-nominal requirements; list others
- During LCA: define moderate to high-risk off-nominal requirements; list others

Well-specified off-nominal requirements make a difference between a Byzantine system (e.g., System just fails or stops responding, or gives wrong answers, without any warning), and a fault-tolerant system (e.g., a system that gives some warning signs before failing, does an orderly shutdown, or degrades gracefully). Off-Nominal requirements may lead into additional Level of Service Requirements (Availability, Reliability...)

Examples of Off-Nominal Requirements for a Business Q&A system, which allows patrons to pose queries in English, search a local database, and also runs the same query against some common search engines.

- "If the system sends a query to a remote search engine, and the remote search engine does not respond within 10 seconds, the system should timeout and try a different search engine, up to 6 different search engines."
- "If the search results exceed 1000 hits, then the system should prompt the user to refine their query instead of attempting to return all search results, which make take a very long time to process, or may overload the client machine"

Special Emphasis: Modes and User Classes

Modes

- Some systems behave quite differently depending on the operational mode. If that's the case, identify the various modes, and organize the System Requirements (Nominal and Off-Nominal) around the operational modes, to avoid forgetting some critical system requirement.
- For example, a voice-operated computer system may have two operational modes:
 - Operational Mode: where the system is actually being used to perform productive work
 - Training Mode: where the operators are training the Voice Recognition module in the system to properly interpret their voice commands, to be saved for later use.

The following template shows a way of organizing Section 3.3.x (this depends on whether the off-nominal requirements are also dependent on mode) around operational modes:

3.2 System Requirements

3.2.1 Mode 1

3.2.1.1 Nominal Requirements

3.2.1.1.1 Functional Requirement 1.1

⋮

3.2.1.1.n Functional Requirement 1.n

```

    3.2.1.2 Off-Nominal Requirements
        :
3.2.2 Mode 2
    :
3.2.m Mode m
    3.2.m.1 Nominal Requirements
        3.2.m.1.1 Functional Requirement m.1
        :
        3.2.m.1.n Functional Requirement m.n
    3.2.m.2 Off-Nominal Requirements
        :

```

User Classes

- Some systems provide different sets of functions to different classes of users. If that's the case, it may be helpful to organize the requirements by User Class (see below for a possible organization) to avoid forgetting some critical system requirement
- For example, a multimedia archive system can have two user classes
 - User: can access, browse and search the archive
 - Administrator: can add, modify, remove items from the archive

The following template shows a way of organizing Section 3.2.x (this depends on whether the off-nominal requirements are also dependent on user class) around user classes.

```

3.2 System Requirements
3.2.1 User Class 1
    3.2.1.1 Nominal Requirements
        3.2.1.1.1 Functional Requirement 1.1
        :
        3.2.1.1.n Functional Requirement 1.n
    3.2.1.2 Off-Nominal Requirements
        :
3.2.2 User Class 2
    :
3.2.m User Class m
    3.2.m.1 Nominal Requirements
        3.2.m.1.1 Functional Requirement m.1
        :
        3.2.m.1.n Functional Requirement m.n
    3.2.m.2 Off-Nominal Requirements
        :

```

4. System Interface Requirements

- In the following sections, describe any applicable requirements on how the software should interface with other software systems or users for input or output. Examples of such interfaces include library routines, token streams, shared memory, data streams, and so forth.
- Use high-level block diagrams (as applicable)

Common Pitfalls:

- Focusing only on user interface requirements and neglecting interfaces with inter-operating systems, such as Homer, SIRSI, IBM Digital Library, HTTP servers, external databases, etc...

- Providing low-level interface requirements, for systems or sub-systems which are outside of the boundary/scope of the proposed system, or which have implicit and standard interfaces (such as TCP/IP for a Web-based application)

4.1 User Interface Requirements

- Describe any requirements on the various User Interfaces that the system presents to the users (who may belong to various user classes, such as end-user, programmer, etc.), which can be any of the following:
 - Graphical User Interface(s) Requirements
 - Command-Line Interface(s) Requirements
 - Diagnostics Requirements

4.1.1 Graphical User Interface Standards

- Describe any Graphical User Interface (GUI) standards to which the proposed system should adhere.
- Include a few screen dumps or mockups, either directly or by reference to Operational Scenarios (OCD 3.4.3), to illustrate graphical user interface *features*.
- If the system is menu-driven, a description of all menus and their components should be provided.
- If the system provides tool bars, describe the tools provided
- Describe button types, shortcut commands, help features and window types in the GUI
- Identify the need for special accessibility needs of users.

Common Pitfall

- Including detailed screen shots of the prototype in the standards, these are likely to evolve and should not be included in the SSRD. They are best included in the OCD appendix.
- Not identifying the key features required of every GUI element such as a menu, tool bars, editors etc.

4.1.2 Command-Line Interface Requirements

- Describe any Command-Line Interface (CLI) requirements
- For each command, provide:
 - Description of all arguments
 - Example values and invocations

4.1.3 Diagnostics Requirements

- Describe any requirements for obtaining debugging information or other diagnostic data

4.2 Hardware Interface Requirements

- Describe any requirements on the interfaces to hardware devices (if they are part of the system)
- Such devices include scanners, bar code readers and printers

4.3 Communications Interface Requirements

- Describe any requirements on the interfaces with any communications devices (e.g., Network interfaces) if they are part of the system

4.4 Other Software Interface Requirements

- Application Programming Interface(s) Requirements
- APIs used and provided to external systems (such as SIRSI, OCLC, BRS)
- Describe any requirements on the remaining software interfaces not included above
- Device drivers for special hardware devices

5. Level of Service Requirements

- Describe the desired levels of service of the System (i.e., "how well" the system should perform a given requirement)
- Level of service requirements in the SSRD should be more specific than the Levels of Service in the OCD, and indicate how they will be achieved
- Level of Service Requirements should be M.A.R.S. (Measurable, Achievable, Relevant, and Specific).
- Measures should specify the unit of measurement and the conditions in which the measurement should be taken. Where appropriate, include both desired and acceptable levels and indications on how the quality will be achieved. Note that the measure of a Quality Attribute need not be absolute but could be a function of another measure. E.g., if a component of the proposed system is an add-in to an existing system, an acceptable measure would be to say that the "The spell-checker add-in should not degrade the reliability of the current editor by more than 10%".
- Trace the Level of Service Requirements back to the Levels of Service and to the Organization Goals.
- Note that some "non-functional" requirements could either be considered "functional requirements", as Nominal Requirements (e.g., Security) or Off-Nominal Requirements (e.g., Reliability)
- The Feasibility Rationale will validate (FRD 2.2.5) that the Quality Requirements are achievable with the given architecture. Do not overburden the system's design with Quality Requirements that are clearly unachievable.
- The following subsections provide possible quality attribute requirements: adapt to the project at hand, and do not feel obliged to create a requirement for each one of them.
- [Should be consistent with OCD 3.3 (Levels of Service)]

Common Pitfalls:

- Simply repeating Levels of Service from OCD 3.3
- Including functional System Requirements ("what the system is to do") as Level of Service Requirements ("how well the system does something"). Note that in some areas (e.g., reliability, security, etc.), the distinction may not be very clear
- Including superfluous Level of Service Requirements not strictly mandated by the customer (avoid having the developers introduce additional requirements). Table 1 shows typical stakeholder concerns for Quality Attributes.
- Including superfluous Level of Service Requirements that do not trace back to Levels of Service or to Organization Goals
- Including Level of Service Requirements not satisfying the M.A.R.S. criteria:
 - Example of non-measurable:** "The system should be as fast as possible"
 - Example of non-relevant:** "The system should be available 24/7", for an organization that operates only 8 hours a day and does not want to perform activities beyond that. Many systems have been overloaded with requirements that are not necessary or relevant: e.g., instant response time on information used on a day-to-day basis or pinpoint accuracy when users only needed two-digit accuracy.
 - Example of non-specific:** "The system shall be implemented as per the standards laid out by USC."
 - Example of non-achievable:** "The system shall be available 100% of the time" for a network-based system, knowing that the network itself may not be available 100% of the time.

Table 1: Stakeholder Roles / Quality Attribute Concerns Relationship

Stakeholder	Roles and Primary Responsibilities	Quality Attribute Concerns	
		Primary	Secondary
General Public	Avoid adverse system side effects: safety, security / privacy.	Dependability	Evolvability & Portability

Interoperator	Avoid current and future interface problems between system and interoperating system	Interoperability, Evolvability & Portability	Dependability, Performance
User	Execute cost-effective operational missions	Dependability, Interoperability, Usability, Performance, Evolvability & Portability	Development Schedule
Maintainer	Avoid low utility due to obsolescence; Cost-effective product support after development	Evolvability & Portability	Dependability
Developer	Avoid non-verifiable, inflexible, non-reusable product; Avoid the delay of product delivery and cost overrun.	Evolvability & Portability, Development Cost & Schedule, Reusability	Dependability, Interoperability, Usability, Performance
Customer	Avoid overrun budget and schedule; Avoid low utilization of the system	Development Cost & Schedule, Performance, Evolvability & Portability, Reusability	Dependability, Interoperability, Usability

Use the following taxonomy of quality attribute requirements as a checklist. Appendix C has some standard definitions for these terms.

1. Dependability
 - 1.1 Reliability/Accuracy
 - 1.2 Correctness
 - 1.3 Survivability/Availability
 - 1.4 Integrity
 - 1.5 Verifiability
2. Interoperability
3. Usability
4. Performance (Efficiency)
5. Adaptability
 - 5.1 Verifiability
 - 5.2 Flexibility
 - 5.3 Expandability
 - 5.4 Maintainability/Debuggability
6. Reusability

Additional Guidelines:

The M.A.R.S. criteria of a quality attribute requirement are critical for ensuring that the requirement has been met.

Measurable:	<ul style="list-style-type: none"> - Time required to perform common tasks - Number of keystrokes required to perform common activities - No need to maintain a hard copy log of operator interactions with the system
Relevant	Having a system with a high usability will definitely increase the chances of the users acceptance of the system, since they will perceive it as reducing the number of repetitive tasks they have to perform
Specific	Currently, the manual system relies on having an Order Number on the various forms, that the users have to remember. The system shall allow users to search by name, and tie closely related forms. Users will not have to remember the unique identifier assigned to new Orders to lookup an Order

Achievable	<ul style="list-style-type: none"> - Minimize the number of operations performed by the operator for a typical activity - Minimize the need to retype the same information multiple times - Provide the operator the ability to interrupt processing - Provide simple and consistent of operator messages that are self-explanatory without requiring the use of the user's manual in the majority of the cases - Allow the user to review and modify all input data prior to execution - Allow the user to undo/redo their last changes - Providing the user with the ability to customize their interface (e.g., add or remove buttons from their toolbar)
------------	---

6. Evolution Requirements

- Describe any requirements on the flexibility and expandability that must be provided to support anticipated areas of growth or changes in the proposed system or the domain itself
- Describe foreseeable directions of the system growth and change
- Describe how the software and data assets will be maintained
 - Facilities
 - Equipment
 - Service-provider relations
 - Maintenance levels
 - Maintenance cycles
 - Emergency software fixes
 - Planned software upgrade releases

6.1 Capability Evolution Requirements

- Major post-IOC capability requirements that are within the current horizon but not important to be implemented in the initial capability.
- These requirements should be described in detail so that if the need arises, some of these can be built into the initial capability.
- These requirements are referred to by software and system support to plan their activities.
- [Consistent with Changes Considered but Not Included (FRD 5.1.4)]

6.2 Interface Evolution Requirements

- Describe any proposed systems with which this system must interoperate and evolve such as standards and protocols of interaction.
- How must the system adapt to interface changes?
 - Organizational changes in use on system
 - Personal changes (more, less, different style)
 - New or expanded product lines
 - Policy changes
 - Organization restructure
 - New/additional/dissolved relationships
- External systems
 - New/additional/replace system
 - Changes in external interfaces
- [Consistent with System Interface Requirements (SSRD 4)]

6.3 Technology Evolution Requirements

Describe how the desired system should address evolution of the underlying technology, how future versions of the COTS products, such as browser upgrades, new versions of server and databases, would be integrated with the proposed system and what new delivery mechanisms would be explored for the proposed system as a part of providing evolutionary support.

6.4 Environment and Workload Evolution Requirements

- Workload Characterization
 - Identify the projected growth of system usage and the scalability needs for the system.
- Data Storage Characteristics
 - As more information is stored in the system in order to support larger number of users or provide richer interactions, the storage needs of the system will grow. Identify such growth vectors.

7. Common Definition Language for Requirements

- Provides definitions of unfamiliar definitions, terms, and acronyms encountered or introduced during the Requirements elicitation process: the definitions express the understanding of the participants and the audience.
- No need to repeat the Domain Description Common Definition Language

8. Appendix

- As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided.
- Include any supporting documentation
 - Detailed software and hardware specifications
 - Standards (used for compliance)

System and Software Architecture Description (SSAD)

Purpose

- Document the Architectural Analysis and the System Design
- Serves as a bridge between the Engineering (Inception and Elaboration) and Construction Phase: during the Construction Phase, the SSAD is refined into a detailed design specification.

Completion Criteria

Below are the completion criteria for the System and Software Architecture Description for the two phases:

- Life Cycle Objectives (Inception Phase)
- Life Cycle Architecture (Elaboration Phase)

Life Cycle Objectives (LCO)

- Top-level definition of **at least one feasible** architecture:
 - ***Feasibility Criterion: a system built to the architecture would support the operational concept, satisfy the requirements, be faithful to the prototypes, and be buildable within the budgets and schedules in the Life Cycle Plan***
 - Physical and logical elements and relationships
 - Essential features of likely components, behaviors, objects, operations
 - Choices of COTS and reusable software elements
 - Detailed analysis, high-level Design
- Identification of infeasible architecture options

Life Cycle Architecture (LCA)

- Choice of architecture and elaboration by iteration
 - Physical and logical components, connectors, configurations, constraints
 - Precise descriptions of likely components, behaviors, objects, operations
 - COTS and technology reuse choices
 - Architectural style choices, deployment considerations
 - Critical algorithms, analysis issues resolved in design
- Architecture evolution parameters
- Complete design for each component of the system
- Tracing to and from OCD/SSRD
- Assurance of satisfaction of ***Feasibility Criterion***

Intended audience

- Domain expert for System Analysis
- Implementers for System Design

Participants

- System Architect
- Domain Experts (to validate analysis models)
- Implementers (to validate design models)
- Project Manager (for feasibility)

High-Level Dependencies

- SSAD depends on OCD for:
 - Statement of Purpose
 - Project Goals and Constraints
 - Levels of Service
 - Capabilities

- SSAD depends on SSRD for:
 - System Definition
 - System Requirements
 - Level of Service Requirements
 - System Interface Requirements
 - Project Requirements
- FRD depends on SSAD to ensure that:
 - Project Requirements
 - Capability Requirements
 - Interface Requirements
 - Quality Requirements
 - Evolution Requirements
 ... are achievable

Overall Tool Support

- Use of Rational Rose as the visual modeling tool is strongly encouraged.
- Avoid large, complex UML diagrams with a lot of overlapping connections. If your diagram is hard to read or overwhelmingly complex, try to reduce amount of information shown on a diagram, by breaking it into several logically connected diagrams. For example, for a class model:
 - Instead of showing all attributes/operations on a large class model, show only inheritance/aggregation
 - Turn off attribute/operation visibility for the overall class model
 - Add for each class or set of classes, a separate diagram showing class, attributes, operations...

Remark

- The SSAD should not repeat information from other documents, and should reference the other information wherever applicable. Reviewing and referencing items in external documents is vital to project integration, coherence and cohesion. Conciseness is paramount.

Outline

1. Introduction
 - 1.1 Purpose of the System and Software Architecture Description Document
 - 1.2 Standards and Conventions
 - 1.3 References
2. Architectural Analysis
 - 2.1 Component Model
 - 2.2 Behavior Model
 - 2.3 Enterprise Model
3. System Design
 - 3.1 Architectural Views
 - 3.1.1 System Topology
 - 3.1.2 Component Specifications
 - 3.1.3 Framework and Protocol Specifications
 - 3.1.4 System Deployment View
 - 3.1.5 Logical Component View
 - 3.2 Class Model
 - 3.3 Operations Model
 - 3.3.1 Critical Algorithms
 - 3.3.2 Operation Specifications
 - 3.4 Configuration Model
4. System Design Common Definition Language
5. Appendices
 - 5.1 Reference
 - 5.2 Vendor documents

1. Introduction

1.1 Purpose of the System and Software Architecture Description Document

- Summarize the purpose and contents of this document with respect to the particular project and people involved
- Avoid generic introductions as much as possible: for instance, you can show how your particular System and Software Architecture Description meets the completion criteria for the given phase

Common Pitfalls:

- Simply repeating the purpose of the document from the guidelines

1.2 Standards and Conventions

- Standards used (DOD, IEEE)
- Notation used (UML)
 - Any exceptions to the standard notation employed
- Naming Conventions
 - Consistent use of names for elements
 - e.g., anObject, the_attribute, MyClass, theOperation()
 - e.g., nouns for Components, Objects; verbs for Behaviors, Operations
 - e.g., label for relationships and outlets

1.3 References

- Provide complete citations to prior and current related work and artifacts, documents, meetings and external tools referenced or used in the preparation of this document
- Useful for consistency checking and traceability

2. Architectural Analysis

The Architectural Analysis is the high level analysis of the problem and a general solution structure independent of the implementation technology: "what" is wanted is more pertinent than the "how" it can be done. The deliverables are component, behavior, and enterprise models, which are detailed representations of the proposed system from different perspectives.

Each analysis view has a counterpart in the Domain Description [OCD 2], which provides the initial starting point and context. Analysis models draw basic information and elaborate in greater detail the aspects of the system to be built as specified by the System Requirements.

Your main architectural task is to discover the fundamental components and behaviors of the proposed system that arise within the Domain Description and document these in a concise way. This provides the critical high-level architecture that will be used as a blueprint by the designers to map out a sound and faithful design for implementation of the proposed system.

2.1 Component Model

- Components are used to describe the system to the domain experts at a higher level of abstraction, independent of software. Objects are used to represent the system in software.
- The component model provides the architectural breakdown of the entities in terms of basic tangible objects of the proposed system that arise from the capability requirements [SSRD 3]. How the components can or will be implemented, is a design issue.
- Analysis components should provide a decomposition and refinement of the entities in the Proposed Entity Model (OCD 3.4.2). All components should be understandable by the **Domain Experts**.
- A Component is an abstraction that has memory and/or computation:
 - Memory: a component's static qualities such as attributes and relationships.
 - Computation: a set of behaviors that embody operations
- An example of a digital archive entity from the Proposed entity would be a digital archive component that contains the objects: multimedia item, item catalog and item metadata. The breakdown allows further detailed design to be performed later.
- [Consistent with Entity Model (OCD 2.4)]

Common Pitfalls:

- Repeating the Domain Entity Model as a Component Model
- Including Components that do not reference Entities
- Including Components that do not have counterparts as Design-Components (SSAD 3.1.4) or Objects (SSAD 3.2.1)
- Including "possible" Components in LCA (they are acceptable at the LCO)

2.2 Behavior Model

- The behavior model describes the organization of the system behavior in terms of its interactions with users. This is achieved by analyzing the interdependence and structure of requirements in order to identify a design that will be able to realize the required behaviors in software.
- The actual sequences of events and actions are identified along with a description of the information flowing in and out of the system.
- [Consistent with the Proposed Interactions (OCD 3.4.3)]
- [Consistent with the System Requirements (SSRD 3.2)]
- [Consistent with Capabilities (OCD 3.2)]

Common Pitfalls:

- Including Behaviors that do not reference Capabilities, Project Goals nor Levels of Service

2.3 Enterprise Model

- The Enterprise Model is the complete model of the system domain that provides a concise overview of the overall application domain structure through description of the various objects and their taxonomies.
- The component model shows the internal architecture of each component, the enterprise model goes one step further to show interrelationships among the constituents of different components.
- Behaviors are mapped to the objects that will carry out the operations.
- The goal is to get an overall organization of the system so that choices on the "elegance" and faithfulness of the entities can be made.
- [Consistent with Component Model (SSAD 2.1)]
- [Consistent with Proposed Entities (OCD 3.4.2)]

Common Pitfalls:

- Including design classes in the Enterprise model, these are not a natural part of the application domain but belong to the class model in SSAD 3.2
- Including non-business attribute types such as string and integer in the Enterprise Model, appropriate types are currency, date, etc

3. System Design

- Describe how the system will be implemented in software using specific technology solutions that meet System Requirements, Project Requirements, Level of Service Requirements, etc.
- You propose direct implementation considerations, such as the use of databases, web-servers, hardware, critical algorithms, operation sequence, significant events, GUI's, etc.

3.1 Architectural Views

Architectural Views provide the high-level design information about the proposed system. The different architectural views project different dimensions of the proposed system and identify Logical Components, System topology and structure and Deployment and Physical Arrangement of the proposed system

- What are they? Describe how system components are mapped into low-level architecture
- Why? Help identify what objects are needed by grouping components into technology representation "clusters" discovers straightforward implementations
- Identifies "gaps" (often due to communication between components) for which particular system objects must be created to fill (i.e., no direct relevance to domain, only makes components "work" in software)
- It is critical that the System Design Views be consistent with the System Block Diagram (SSRD 3.1)

3.1.1 System Topology

- The system topology concerns itself with the logical software/hardware/network module organization in the production environment, taking into account derived requirements related to ease of development, software management, reuse, and constraints imposed by programming languages and development tools.
- The System topology shows the components and their interconnections as well as the configuration in which they are connected. The System Topology is useful for:
 - Design: it allows the designer to identify the various subsystems, or inter-operating systems, especially for COTS frameworks
 - Transfer considerations, i.e., requirements and considerations for software assembling and packaging and the approach to be used for transferring the software to the library
 - Identifying representational layers, which may be handled with the existing frameworks
- Should be a refinement of the system block diagrams in OCD 3.4.1
- Assigns components to system block diagram or other logical system group
 - conceptual understanding and completeness

- ensures consistency, completeness, accessibility, and necessity of system parts and relationships to outside (system boundaries)
- may introduce specific technology choices (some may exist from block diagram or requirements)

Common Pitfalls:

- Inconsistency with the System Block Diagram (SSRD 3.1)
- Omitting System Components introduced in SSAD 2.1

3.1.2 Component Specifications

- Describe the components identified in the system topology and specify their interfaces.
- Add to the components from the Component Model (SSAD 2.1), new components such as mechanisms, to make the system realizable in software (i.e., pure software components which do not directly map to entities in the domain)
- Can be specific technology choices (e.g., database tables, HTML templates, HTTP servers, COTS products, API's, class libraries, design patterns, etc.)

3.1.3 Framework and Protocol Specifications

- The architecture describes the interactions of the system components and how the component interactions support the system activities. The interactions take place with the help of various mechanisms and frameworks. These frameworks are often picked off-the-shelf and are industry standards. This section should describe the specific frameworks to be used and the nature of interactions among the logical components (SSAD 2.1) and design components (SSAD 3.1.4) in order to support all the behaviors of the system as described in SSAD 2.2
- Protocols and services are typically independent of the domain and their choice depends upon the various requirements of the system, most notably Level of Service Requirements (SSRD 5).
- Examples of frameworks include CORBA Services and Facilities, Java JDK, TCP/IP and various network protocols as well as security and audit mechanisms.

3.1.4 System Deployment View

- The System Deployment View concerns itself with the physical software/hardware/network module organization
- Identify the specific kinds of hardware required, such as PCs, mainframes and RAID disks, and identify where the custom built and COTS products would be installed.
- Assign components to deployed hardware and software
 - if known, includes OS, mechanisms, and frameworks
- Splits system into physical groups
 - very common that components that communicate across physical groups will require “glue” objects

Common Pitfalls:

- Inconsistency with the System Block Diagram (OCD 3.1.1)
- Omitting System Components introduced in SSAD 2.1
- Not indicating the placement of components identified in SSAD 3.1.1 on specific hardware devices.

3.1.5 Logical Component View

- The logical component view identifies the top-level development units for the system.
- This view describes the dependencies among development units and identifies the self developed software and the software being reused.

Common Pitfalls:

- Inconsistency with the System Topology (SSAD 3.1.1)

- Not clearly demarcating the components being developed in the system against those being reused off the shelf.

3.2 Class Model

- The Class Model is a realization of the classes identified in the Enterprise Model (SSAD 2.3)
- The class model identifies the classes as implemented in programming languages along with their attributes and operations with signatures and type information.
- Each category of classes should be represented in a separate class diagram in this section for the purpose of easier comprehension..
- It also identifies the important state transitions for describing system behaviors.
- Identify the relations among classes in software.
- [Consistent with Operations Model (SSAD 3.3)]

Common Pitfalls:

- Including one model to contain all types of classes, it is possible that there are programs and help files, they belong in separate class models.

3.3 Operations Model

- The Operations Model should be a refinement of the Behavior Model (SSAD 2.2)
- Create operations by refining behaviors from the Behavior Model (SSAD 2.2)

3.3.1 Critical Algorithms

- Detail and explain critical custom algorithms
- Detailed policies and associated algorithms for carrying out policies as identified in OCD 3.6.3

Common Pitfalls:

- Identifying algorithms without providing a description of the algorithm (this is allowed at LCO, but at LCA all critical algorithms should either be assigned to identified COTS or described in detail)

3.3.2 Operation Specifications

- Describe the operations performed to demonstrate all the system behaviors described in Behavior Model 2.2.
- Identify the flow of control through the system during system execution and provide operation signatures and entry/exit conditions.

Common Pitfalls:

- Including analysis level operations specifications which do not identify the operation signatures
- Including operations in Operation Model that are not assigned to Objects in the Class Model
- Haphazard introduction of objects to handle operations: only create new classes as absolutely needed
- Not tying operations to System Behaviors and Capabilities
- Including operations that do not references Behaviors, System Requirements or Level of Service Requirements
- Including operations in the Operation Model that are not assigned to classes in the Object Model
- Not covering all the behaviors from the Behavior Model (SSAD 2.2)

3.4 Configuration Model

- The configuration model describes how classes in the Class Model (SSAD 3.4) are implemented in programs and files. The model describes the dependency structure and directory structure of the software development.
- Include all required files, scripts, programs, images and libraries in the directory structure.

4. System Design Common Definition Language

- Define new terms and acronyms encountered or introduced during System Design
- May include technology implementation terms

Appendix

As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided.

A Reference

- Provide supplementary data such as algorithm descriptions, alternative procedures, tabular data, or other document excerpts from technical publications, etc...

B Vendor documents

- Provide information, technical specification sheets on the COTS products used
- Describe/refine domain or application independent components
- Frameworks
- Components
- Class Libraries

Life Cycle Plan (LCP)

Purpose

- To serve as a basis for monitoring and controlling the project's progress
- To serve as the basis for controlling the project's progress in achieving the software product objectives.
- To help make the best use of people and resources throughout the life cycle.
- To provide evidence that the developers have thought through the major life cycle issues in advance.
- Organized to answer the most common questions about a project or activity: why, what, when, who, where, how, how much, and whereas.

Completion Criteria

Below are the completion criteria for the Life Cycle Plan for the two phases:

- Life Cycle Objectives (Inception Phase)
- Life Cycle Architecture (Elaboration Phase)

Life Cycle Objectives (LCO)

- Identification of life-cycle stakeholders
 - Users, customers, developers, maintainers, interfacers, general public, others
 - Identification of life-cycle process model
- Top-level stages, increments
 - Top-level WWWWWHH (Why, What, When, Who, Where, How, How Much) by stage
- Major risks identified
- Deliverables, budgets and schedules achievable by at least one system/software architecture

Life Cycle Architecture (LCA)

- Elaboration of WWWWWHH for Initial Operational Capability (IOC)
- All major risks resolved or covered by risk management plan
- Deliverables, budgets and schedules achievable by the architecture in the SSAD

Intended Audience

Primarily the performer teams in each stage, but also important for customers, and useful for other stakeholders.

Participants

The Project Manager for each stage leads the baselining of the plan for that stage. Plans for future stages are normally done by a designated team member during Engineering Stage. Stakeholders affected by plan elements should participate in their definition.

High-Level Dependencies

Products specified by Requirements and Architecture must be buildable and supportable within the budgets and schedules in the Life Cycle Plan. Plans for transition and support must be consistent with the Operational Concept Description.

Overall Tool Support

Use of the planning and control tool Microsoft Project '98 is encouraged for CS 577a but required for planning and tracking CS 577b. COCOMO II should be used to help develop budget and schedule estimates. If COCOMO II estimates conflict with good engineering judgment, use and document the engineering judgment. Rational Dashboard would also be used to collect progress metrics and analyze project effort.

Potential Pitfalls/Best Practices: It should be noted that through the high degree of dependencies between tasks and people, delays in critical areas might cause schedule problems in many (if not all) activities later on. Even the simplest reasons such as a vacation of a key person not considered may be responsible for such schedule upsets. Utmost care should therefore be devoted into planning and maintaining this document, and to ensuring its feasibility via the Feasibility Rationale.

Quality Criteria: The plan adds a time component to the other documents. Thus, there is a high degree of dependency especially between the SSRD (and SSAD) and the life cycle plan. If the tasks identified in this document do not reflect the requirements and the components of the product to be developed, then the plan will be useless. Thus, maintaining the conceptual integrity between the tasks and the things they create/solve is a prime quality criterion.

Further, the timing and scheduling of the tasks is highly dependent on not only the SSAD and SSRD as explained above, but also on the people who are working on them. Fortunately, that part is fairly independent from other documents. However, the ultimate core of this document is the creation of a responsibility trace (or matrix) which maps the people to components via tasks. The plan is a record of the personal and organizational commitments of each of the stakeholders to their part of the overall project. If these commitments are vaguely defined or poorly matched to people's capabilities, there is a high risk of project misunderstandings, frustrations and failures. The bottom-line quality criteria are the assurance of resource level feasibility and business-case return on investment in the Feasibility Rationale.

Outline

1. Introduction
 - 1.1 Purpose of the Life Cycle Plan Document
 - 1.2 References
 - 1.3 Assumptions
2. Milestones and Products
 - 2.1 Overall Life Cycle Strategy
 - 2.2 Phase Milestones and Schedules
 - 2.2.1 Engineering Stage
 - 2.2.2 Production Stage
 - 2.2.3 Support Stage
3. Responsibilities
 - 3.1 Stakeholder Responsibilities
 - 3.1.1 Stakeholder Representatives
 - 3.1.2 Engineering Stage
 - 3.1.3 Production Stage
 - 3.1.4 Support Stage
 - 3.2 Development Responsibilities
 - 3.2.1 Development Organization Charts
 - 3.2.2 Staffing
 - 3.2.3 Training
4. Approach
 - 4.1 Risk Management and Monitoring Procedures
 - 4.2 Quality Management
 - 4.3 Reviews
 - 4.4 Configuration Management
 - 4.5 Project Communications
 - 4.6 Status Monitoring and Control
 - 4.7 Facilities and Related Concerns
 - 4.8 Support Environment, Methods, and Tools
5. Resources
 - 5.1 Work Breakdown Structure
 - 5.2 Budgets
- Appendices
 - A. COCOMO Results
 - B. Gantt Charts

1. Introduction

1.1 Purpose of the Life Cycle Plan Document

- Summarize the purpose and contents of this document with respect to the particular project and people involved
- Avoid generic introductions as much as possible: for instance, you can show how your particular Life Cycle Plan meets the completion criteria for the given phase

Common Pitfalls:

- Simply repeating the purpose of the document from the guidelines

1.2. Assumptions

This Section identifies the conditions that must be maintained in order to implement the plans above within the resources specified. If one or more of these assumptions is no longer satisfied, then the requirements (System Requirements or Project Requirements) may need to re-negotiated; or the Life Cycle Plan may need to be re-evaluated

Develop a list of assumptions on which the project planning decisions are based so that everyone on the project understands them. It is important to uncover unconscious assumptions and state all assumptions up front. Assess and state the likelihood that the assumption is correct, and where relevant, a list of alternatives if something that is assumed does not happen.

These assumptions might cover such items as:

- Stability of software product requirements, including external interfaces;
- Stability of required development schedules;
- Continuity of funding;
- On-schedule provision of customer-furnished facilities, equipment, and services;
- On-schedule, definitive customer response to review issues and proposed changes.
- What the developers expect to be ready in time for them to use. For example, other parts of your products, the completion of other projects, software tools, software components, etc.
- Dependencies on computer systems or people external to this project
- Any tasks which are on the critical path of the project
- [Consistent with Process Match to System Priorities (FRD 3.2)]
- [Consistent with Project Requirements (SSRD 2)]

Integration and Dependencies with other components:

Assumptions are made because the necessary detail is not yet known. Thus, this section will change as the system evolves. The assumptions can be based on any aspect of the development and thus they can be dependent on any document (e.g. SSRD and SSAD). The risk identification and analysis part in section 4.1 should reflect those assumptions.

1.3 References

- Provide complete citations to prior and current related work and artifacts, documents, meetings and external tools referenced or used in the preparation of this document
- Useful for consistency checking and traceability

2. Milestones and Products

This section tells *what* project functions will be performed during life cycle, and what products will be developed. It contains schedules and milestones that indicate when each function and product will be completed.

Integration and Dependencies with other components:

- The milestones defined for stages and phases in Section 2 must be consistent with the stage and phase responsibilities, reviews, budgets, etc. in Sections 3, 4, and 5.
- The content of the LCO, LCA, and LCA Rebaseline milestones is exactly the content of the OCD, SSRD, SSAD, Prototype, LCP, and FR for those milestones.

Additional Guidelines:

[Boehm, 1996] provides additional information about the LCO, LCA, and IOC milestones and [Royce, 1998] about their respective Inception, Elaboration, Construction, and Transition phases. COCOMO II may be used as a validity check for the schedule and the milestones in Items 2.1 and 2.2.

2.1 Overall Life Cycle Strategy

Describe the overall approach to be used in defining, developing, and supporting the product:

- (a) The choice of process model(s) used (waterfall, evolutionary, spiral, etc.) and the major life cycle stages, phases and milestones. Departures from this approach can be identified in section (d);
- (b) If prototyping is employed, the nature and phasing of the planned prototypes;
- (c) If incremental development is employed, describe the content and phasing of the successive increments to be developed. The phasing of the increments should correspond to the system priorities.

- (d) Specifics of any other departures from the approach in item (a) (e.g., design-to-cost, design-to-schedule, multiple parallel design or development teams, product-line as well as product development);
- (e) Top-level milestone charts (Gantt charts) and activity networks (PERT charts) showing the sequence of major products and activities.

Focus on the external products and milestones that the customer will see. Later sections should give the internal project details. Emphasize the critical process drivers and process strategies analyzed in FRD 3.2.

Course Guidelines:

The recommended process model is the WinWin Spiral Model; and the major life-cycle stages, phases, and milestones are (See Figure 1):

- Engineering Stage
 - Inception Phase (Life Cycle Objectives): one WinWin Spiral cycle, completed by an LCO ARB review
 - Elaboration Phase (Life Cycle Architecture): one WinWin Spiral cycle, completed by an LCA ARB review
- Production Stage
 - Construction Phase: a short WinWin Spiral cycle, to incorporate changes that may have occurred since the LCA milestone, and to incorporate Product Improvement suggestions from the Individual Critiques, completed by an Architecture Rebaseline Review; an initial core-capability development increment; followed by several risk-driven incremental iterations or design-to-schedule adjustments, completed by a Transition Readiness Review
 - Transition Phase, completed by a Release Readiness Review
- Support Stage (USC-ISD responsibility)
 - A series of releases, each with its appropriate choice of the stages and phases above, completed by a Release Readiness Review

The process drivers will likely include the production stage, including the need to fully transition the product to the client. Additional process drivers could include client infrastructure constraints, dependencies on other projects, and challenging performance or dependability requirements.

Common Pitfalls

- Only identifying the current stage in the life cycle strategy
- Confusing the stages and phases of the life cycle
- Not identifying the need for transition and support

2.2 Phase Milestones and Schedules

Provide more detailed milestone charts and activity networks (PERT charts) identifying the activities (in each increment, if applicable), showing initiation and completion of each activity and depicting sequential relationships and dependencies among activities. Identify those activities that impose the greatest time restrictions on the project (i.e., which are on the critical path). The activities described here should be tracked with progress metrics. Artifacts to be delivered at the end of each phase should be identified and the completion criteria for each artifact should be specified. For each artifact (plans, specifications, manuals, reports, code, data, equipment, facilities, training material, person-hours, etc.) to be delivered to the customer, provide the following information on the nature of the deliverable:

- (a) The name or title of the artifact
- (b) The date on which the artifact is due
- (c) The required format of the artifact (disk format, tape format, document format, etc.)
- (d) The completion or "exit" criteria for each artifact (evidence required of being produced, delivered, received, tested, approved, etc.);
- (e) Pointers to relevant contract requirements

Completion criteria are defined in [Royce, 1998] Chapter 5 and Section 9.1. The criteria will vary by phase but often involve stakeholder concurrence, evidence of completeness and stability, actual vs. planned

estimates, prototype acceptability, and more. The completion criteria for each of the components of the LCO and LCA package components are indicated in this document.

Common Pitfalls

- Not providing a graphical representation of the entire life cycle
- Only describing the current stage in the life cycle
- Not identifying the deliverables at each milestone (both minor and major)

Course Guidelines:

Deliverables include the LCO/LCA versions of the:

- Operational Concept Description
- System and Software Requirements Description
- System and Software Architecture Description
- Life Cycle Plan
- Feasibility Rationale
- Prototype and optional Prototyping Results (See Appendix A)
- COCOMO II run for the product defined in the LCO/LCA package
- Weekly Effort Forms

Internal deliverables include the following documents and the added transition package to be delivered to the client:

- LCA package (kept up-to-date with as-built architecture and design)
- Software Test Plan
- Inspection Plan
- Quality Assurance Plan
- Configuration Management Plan
- Test Plan
- Iteration Plans
- Iteration Assessment Reports
- Inspection Reports
- Test Description and Results
- Software Development Files
- Weekly Effort Forms
- Weekly Status Reports
- Release Notes (for each internal release)

The transition package contents includes the software library:

- Source Code
- Load Modules
- Installation Scripts

Client-side deliverables include:

- User Manual
- Training Plan
- Transition Plan
- Support Plan
- Training materials (including tutorials and sample data)
- Regression Test Package
- Tools and Procedures
- Version Description Document (or Final Release Notes)
- Facilities, Equipment and Data (these may not be the responsibility of the team)

All deliverables are properly stored in the Class Archive, in accordance with the course guidelines.

Elaborate on the top-level information given in section 2.1. The following example illustrates the typical level of detail to be provided for software integration and test activities:

- Section 2.1. For each increment, indicate completion of integration, of product test, and of acceptance test; and indicate major dependencies on life cycle activities, on other increments, on facilities, etc.;
- Section 2.2. Indicate milestones showing the overall order in which components are integrated, and the intermediate stages of increment and acceptance testing. Show how these are synchronized with milestones for preparation of test drivers, facilities, equipment, data, post- processors, etc. for the various increments.
- Detailed Integration and Test Plan (not part of LCP). Indicate the integration order-of-build for all software components. Identify each test to be performed and indicate which itemized requirement(s) it will test, and where it fits in the overall sequence of tests.
- [Consistent with Process Match to System Priorities (FRD 3.2)]

Course Guidelines:

- Use actual milestone dates rather than week numbers.
- Develop phase capabilities concurrently rather than sequentially and provide meaningful increments to assess progress and assure customers.
- Transition planning and preparation should be done to allow for an initial delivery to the customer two weeks before the end of the semester, followed by 2 weeks of installation, hands-on training, usage testing, refinement, and completion of IOC deliverables.
- Provide graphical activity charts: in particular, show externally provided resources and components, and highlight the ones (e.g., equipment purchase) which are on the critical path for the transition

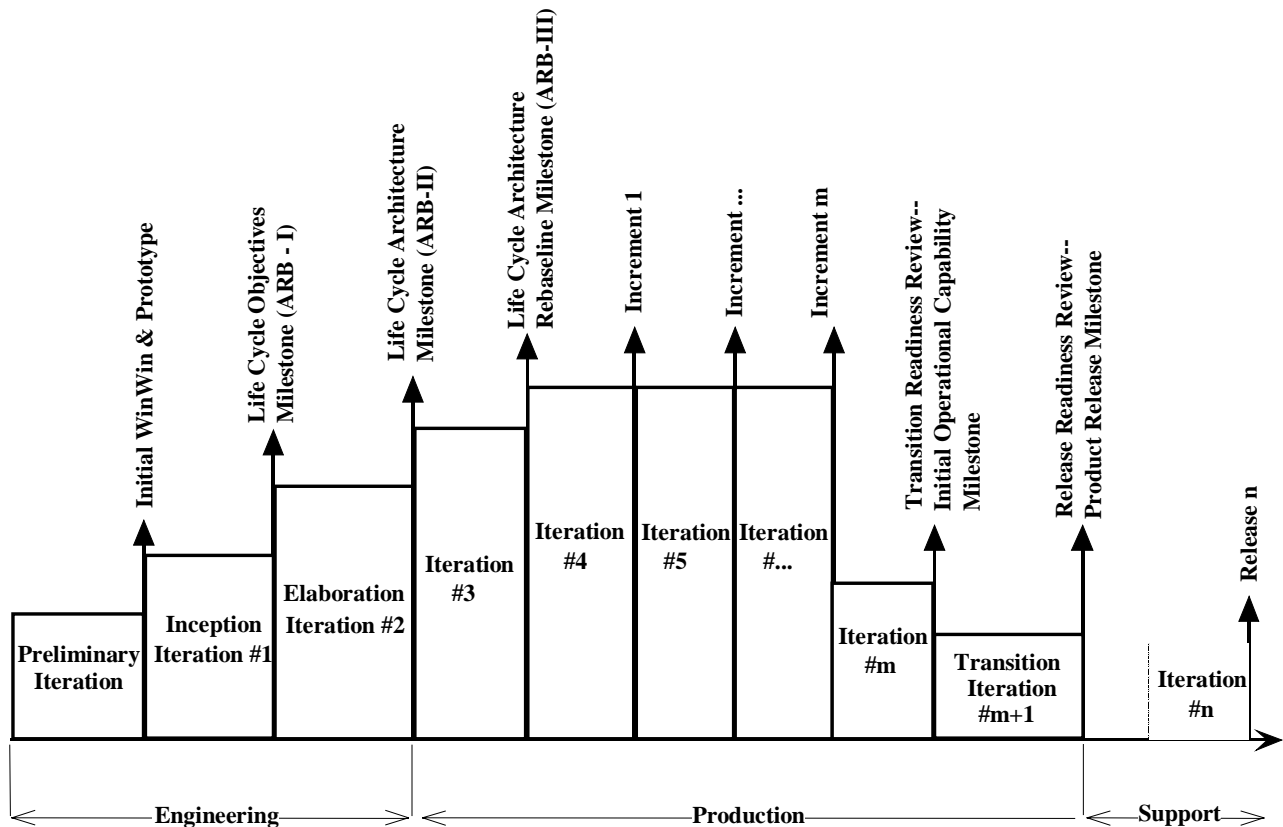


Figure 1 Process Model

2.2.1 Engineering Stage

2.2.2 Production Stage

Identify the priorities for development in terms of the system requirements being fulfilled in a timely manner for the initial operational capability.

2.2.3 Support Stage

Indicate the degree of client concurrence on any client commitments expressed or implied in this section.

3. Responsibilities

This Section tells *who* will be responsible for performing the various software life cycle functions, and *where* organizationally they will be performed. It identifies the major life cycle- related agents (developer, customer, maintainer, users and interfacers) and establishes their roles and responsibilities. It defines the major organizational components of the project, and indicates their responsibilities across the phases of the life cycle.

It presents organization charts showing individual and organizational responsibilities, and includes plans for project staffing and training.

- [Consistent with OCD 3.6]

Integration and Dependencies with other components:

This section of the document is relatively independent from other documents. However, within the SDP this section describes human resources, which are needed in other sections of this document. The availability of people is the basis for scheduling and creating milestones in Section 2 because the people identified in this section must be connected to the tasks which were identified in Section 2. Together, the responsibilities can be defined and possible inconsistencies can be eliminated. Inconsistencies are for example:

- no one assigned to a task
- too many assigned to a task (or it is not clear who is primarily responsible for it)
- input to a task may not be available, etc.

It is particularly important to ensure that the non-developer stakeholders (users, operators, customers, etc.) concur with any responsibilities assigned to them in the LCP.

Additional Guidelines:

No special tools or techniques required for the staffing part. For creating responsibilities, we recommend using MS Project. To ensure completeness, try to look at that problem from different perspectives. For instance, look at each person and check his or her tasks and whether they are feasible. Alternatively, look at each task and verify whether people are assigned to it for its entire life. Further, as people shortfalls are a top candidate risk item, make sure your people approaches in Section 3 are coordinated with the people-oriented risks in LCP 4.1

3.1 Stakeholder Responsibilities

Identify which organizations will assume responsibility for carrying out the functions of the major life cycle- related agents: developer, customer, maintainer, users, and interfacers. Adapt **Table 2** as appropriate to indicate the life-cycle responsibilities of each agent. Indicate the roles and responsibilities of subcontractors, vendors, team members, independent verification and validation agents, and other external organizations. Identify any special life cycle-related responsibilities assumed by the customer, owner, or users, e.g.

- Performance of design or development functions
- Providing facilities, equipment, or software
- Supplying data
- Performing conversion and installation functions
- Supplying support services (computer operations, data entry, transportation, security, etc.)

- Describe the following items for each stage in Sections 3.2.1, 3.2.2, and 3.2.3:
- **Organization:** Identify the major organizational components of the project and indicate their responsibilities during the various stages of the life cycle. Adapt Table 2 and elaborate OCD 4.1.2 as appropriate. Provide organization charts showing the major organizational components of the project during this stage and the key responsible personnel within each organizational component. Record any special agreements on organizational boundaries (e.g., the boundaries between "integration" and "system test"; the boundaries between quality assurance functions, test functions, and management functions).
- **Staffing:** Identify the types of personnel required by the project (analyst, programmer, operator, clerical, etc.) and the number of personnel of each type required to perform each major function during each life cycle phase. Identify critical skills required of the personnel (e.g., experience with Java, image compression, Sybase). Provide charts or tables showing the staffing requirements as a function of calendar time. Provide any special plans for hiring, relocation, security clearances, organizational transfers, use of temporary personnel, special compensation, etc.
- **Training:** Identify the organizations responsible for internal (project personnel) and external (customer, owner, user personnel) training. For both internal and external training, indicate the number of personnel to be trained, the length of training classes, the schedules for training preparation and performance, and the required facilities, equipment, software, instructors, training materials, etc.]

Table 2 - Stakeholder responsibilities during the software life cycle

	Inception	Elaboration	Construction	Transition
Users	Support definition of requirements specification, operational concept and plan. Review prototype and exercise if available.	Review designs and prototypes during ARB. Help provide test data and test scenarios.	Review and test product (or its increment) in development environment. Provide test support.	Review and test product (or its increment) in operational environment. Provide usage feedback to Maintainer
Customer	Support definition and review of requirements specification, operational concept and plan – accept or reject options	Monitor progress at milestones. Review designs, prototypes, plans and feasibility during ARB. Help provide test data and test scenarios.	Monitor progress at milestones. Review and test product. Provide administrative support. Review system performance	Monitor progress. Provide administrative support in transitioning the product. Review system performance
Developer/Maintainer	Prepare requirements specification, operational concept, architectural sketches and plan. Build user interface prototype.	Refine architecture and design and present them during ARB. Refine or rebuild further prototype to investigate risks. Prepare test plan.	Refine design, implement, and integrate product. Perform and support test.	Provide development support in transitioning the product. Adapt product if development environment differs from operational one.
Interfacer	Support definition of requirements specification and interface specification.	Refine interface specification and review design. Build prototype to investigate risks.	Review interface design and implementation. Validate interface in development environment	Validate interface in operational environment

Common Pitfalls:

- Simply copying Table 2. You should adapt it to the particular project

3.1.2 Stakeholder Representatives

Identify by organizational position the personnel responsible for committing their stakeholder organization to any changes in project scope, budget, and schedule.

Course Guidelines:

This section should name individuals responsible for making project commitments for their organizations, not the organizations.

3.1.1 Engineering Stage

The normal project team responsibilities for the Engineering Stage will include four performers responsible for the OCD and LCP, SSRD and LCP, SSAD, and Prototype, reporting to a project manager also responsible for the FRD. Identify any differences from this approach, plus any key client roles and responsibilities during the Engineering Stage.

User and Customer responsibilities may include meetings/discussions regarding the new system, participating in prototyping, requirements gathering, architecture review boards, etc...

3.1.2 Production Stage

User and Customer responsibilities may include development or modification of databases; training; parallel operation of the new and existing systems; impacts during testing of the new system; preparation for deployment in the organization; and other activities needed to aid or monitor development.

3.1.3 Support Stage

Operational Roles and Responsibilities can largely reference OCD 3.6..1. Ensure that maintenance roles and responsibilities are comparably defined.

3.2 Development Responsibilities

Describe a specific organization and set of team member roles and responsibilities. Use COCOMO II to calibrate the necessary team size. It should be detailed enough so that it could be used as a detailed construction plan, assigning tasks and responsibilities to team members. Having a clear distribution of tasks would make the team members work more efficiently and help them finish their tasks quicker. It is also important to minimize the overlap between tasks assigned to different team members.

Below is an ordered list of neighboring roles and responsibilities, which can be clustered in various ways to best fit the project's situation:

- Project management, planning and control
- Client facilitation: operational procedures, transition planning and support, conversion and data preparation support, training, user manuals
- Quality Assurance and system test
- Configuration Management and document integration
- Detailed design, Programming, Unit Testing, Integration Testing , Maintenance Manuals

Course Guidelines:

The Work breakdown structure is used to address team roles and responsibilities. Teams should develop the WBS (LCP 5.1) and the Effort Estimates (LCP 5.2), before proceeding to Section 3.

3.2.1 Development Organization Charts

Describe a Development Organization Chart.

3.2.2 Staffing

Describe (as applicable):

1. The estimated staff-loading for the project (number of personnel over time). This plan will be tracked using effort metrics.

2. The breakdown of the staff-loading numbers by responsibility (for example, management, software engineering, software testing, software configuration management, software product evaluation, software quality assurance) and by software development unit.
 3. A breakdown of the skill levels of personnel performing each responsibility
- The staffing plan should reference (as applicable) the work breakdown structure.

Common Pitfalls:

- Not identifying the staffing roles per WBS and instead only including total levels.

3.2.3 Training

Describe as applicable plans for getting the development team up to speed on critical skills.

4. Approach

This Section tells *how* the project will implement the software life cycle strategy described in Section 2.1. It identifies the activities, tools, and techniques that will be employed in performing the project functions. It presents the project's approach for risk management, and for performing software requirements analysis, design, development, test, and implementation functions. It describes how the project will perform associated project functions such as technical reviews, project communications, configuration management, quality management, facilities management, security, and subcontractor management.

Integration and Dependencies with other components:

The Risk Management and Monitoring Procedures section is strongly coupled with the Project Risk Assessment section of the Feasibility Rationale (FRD 4.0). In addition, since MBASE life cycle processes are risk-driven, the overall life cycle strategy in Section 2.1 (incremental, design-to-schedule, etc.), will reflect the major risks. Quality Assurance is closely coupled with the quality requirements in the SSRD. Facilities plans are coupled with the OCD. The section on Reviews focuses on the major milestones and deliverables, and is thus coupled with Section 2. The milestone content also creates dependencies with the capabilities and requirements described in the SSRD. Once the capabilities have been refined in more detail, the milestone reviews will also address more detailed knowledge from the SSAD.

Additional Guidelines:

difficulties.

- For Configuration Management, see [CMU-SEI, 1995] Chapter 7.6 and [Royce, 1998] Section 12.2.2.
- For Quality Assurance, see [CMU-SEI, 1995] Chap. 7.5 and [Royce, 1998] Sections 3.5 and 11.2 (assessment)

4.1 Risk Management and Monitoring Procedures

Document procedures for monitoring the risk factors and for reducing the potential occurrence of each risk. Identify contingency procedures for each area of risk. Show how the project will address, monitor, and resolve these sources of risk, by providing plans for mitigating the identified risks

- Monitoring of risk management plan milestones
- Corrective Action: Decision Point to invoke Contingency Plans
- Top-10 Risk Item Tracking (See Table 3)
 - Identify top 10 Risk Items
 - Highlight these in regular project reviews (focuses review on manager-priority items)
 - Focus on new entries and slow-progress items
- Risk Reassessment

Table 3: Top-N Risk Item List (Assuming weekly risk reassessment)

Risk Items	Weekly Ranking			Risk Resolution Progress
	Current	Previous	# Weeks	

For each critical risk item, include a detailed risk management plan in the appendix, and provide a reference to it. The Risk Management Plan answers the following questions:

1. **Objectives (Why?):** Risk item importance, relation to project objectives
2. **Deliverables and Milestones (What? When?):** Risk resolution deliverables, milestones, activity nets
3. **Responsibilities and Organization (Who? Where?):**
4. **Approach (How?):** Prototypes, surveys, models, ...
5. **Resources (How Much?):** Budget, schedule, key personnel, ...

Course Guidelines:

A risk management technique commonly used in is a weekly top-10 Risk Items Lists

Common Pitfalls:

- Describing the risks in detail here, they belong to the FRD 4

4.2 Quality Management

The objective of software quality management is to ensure the delivery of a high-quality software product by determining the project's prioritized quality objectives and verifying that the project's agreed-upon plans, policies, standards, and procedures for achieving those objectives are all adhered to.

This section should elaborate on quality attributes, and roles & responsibilities of team members in achieving them. As emphasized in [Royce, 1998], Appendix 5, quality is everyone's responsibility, but it is still generally useful to include traditional quality assurance functions such as:

- a) Development of documentation and code standards;
- b) Verification of the project's compliance with its documentation and code standards;
- c) Auditing the project's compliance with its plans, policies, and procedures;
- d) Monitoring the performance of reviews and tests;
- e) Monitoring corrective actions taken to eliminate reported QA deficiencies

Course Guidelines:

It will be sufficient to identify a specific team member to perform the QA function. Following the Theory W principle ("Match people's tasks to their win conditions"), it would be best to have the QA function (and probably related functions such as configuration management) performed by the team member responsible for successful product transition to the client.

Common Pitfalls:

Do not promise more than you want to deliver. Identify just the key subset of standards the team should follow and compliance-manage (e.g., header block information for code modules; document formats; avoidance of error-prone code constructs, CGI, exotic OS features limiting portability, etc.).

4.3 Reviews

This Section identifies the major project reviews and their objectives. It provides the plans to prepare for, conduct, and follow up on the review meeting in order to achieve the objectives of each review.

The primary objective of each major review is to determine whether or not the project is ready to proceed to the next life cycle phase. If so, the phase products reviewed are baselined and put under configuration

management, to provide a stable foundation for the following phase. Note that the LCO package is not baselined, since the high priority is to evolve it into an LCA version.

Preparing the Reviews section often requires a good deal of tailoring. Each project review is actually a small project in itself, and should thus have its own small project plan, indicating its objectives, milestones, responsibilities, approach, resources, and assumptions.

In practice, though, particularly on small projects, there may be a good deal of overlap between these sections (e.g., between "milestones" and "approach"), and between the plans for the various reviews. In such cases, it is best to compress Section 4.3 into a single generic plan, accompanied by a table indicating the unique characteristics of each review.

Review plans should emphasize the following pre-meeting activities:

- getting user, owner, and interfacers organizations to participate;
- securing commitments from capable reviewers;
- preparing review assignments;
- distributing review material well in advance;
- getting itemized written comments from reviewers;
- providing the comments to the developer, and getting the developer to prepare his or her response;
- setting up the review meeting agenda;

and the following post-meeting activities:

- publishing review meeting minutes, documenting agreements reached at the meeting;
- assigning, tracking, and closing out action items from the review meeting;

The review materials should include not only the phase products, but also evidence of the developer's having verified and validated them (e.g., the Feasibility Rationale in the LCO and LCA packages).

Subsections of Section 4.3 are indicated below for the reviews corresponding to the nominal phase organization in Table 4. These subsections should be modified to reflect any significant departures of the project's development strategy from this nominal approach.

General principles for ARB reviews are available in [AT&T, 1993]. ARB reviews need to put review materials on the Web a week in advance, and to arrange a satisfactory review time for their clients. Reviews involve short highlight presentations by each team member, including a prototype demo. A review scribe should summarize the review results, with associated actions, agents, and due dates.

Course Guidelines:

- You may reference plans and reports which will be done in the Production Stage
 - Review Plan
 - Inspection Plan
 - Inspection Reports
- Review plan should indicate how client wants to accomplish the reviews (one vs. more meetings; demos; training sessions, etc.) and identify candidate dates for each

Table 4 Key products and the reviews (not including In-process Reviews and Inspections)

Phases:	Inception	Elaboration	Construction			Transition
	(LCO)	(LCA)	ARB-III	TRR	RRR	
Reviews:	ARB-I	ARB-II	ARB-III	TRR	RRR	...
Package Elements						
• Operational Concept Description (OCD)	▲	▲	▲	▲	▲	
• System and Software Requirements Definition (SSRD)	▲	▲	▲	▲	▲	
• System and Software Architecture Description (SSAD)	▲	▲	▲	▲	▲	

• Life Cycle Plan (LCP)	▲	▲	▲	▲	▲
• Feasibility Rationale (FRD)	▲	▲	▲	▲	▲
• Prototype	▲	▲	▲	▲	▲
...					
Deployment Set		▲			▲
• Transition Plan			▲	▲	▲

Note: this table is adapted from [Royce, 1998] Figure 6-10. The dark triangles are controlled (strong) baselines whereas the light triangles are informal (weaker, less defined) baselines.

- Architecture Review Board I (ARB-I)
 - The Review Criteria for the LCO Architecture Review are the completion criteria for LCO Feasibility Rationale.
- Architecture Review Board II (ARB-II)
 - The Review Criteria for the LCA Architecture Review are the completion criteria for LCA Feasibility Rationale.
- Architecture Review Board III (ARB-III)
 - The LCA Rebaseline Review is an incremental review of the LCA package with respect to its Feasibility Rationale Criteria
- Inspections and In-Process Reviews
 - Detailed Design Inspection
 - Code Inspection
 - User Manual Inspection
 - Unit Test Completion Reviews (UTCR)

Course Guidelines:

Refer to the inspection guidelines on the class web site, and their associated inspection reports. See also [CMU-SEI, 1995], Chapter 8.7.

- Transition Readiness Review (TRR)
 - The Transition Readiness Review (TRR) should verify that the following transition pre-conditions are satisfied:
 - Ready-to-install software, verified for compliance with the requirements in the SSRD;
 - Ready-to-use User's Manual, Maintenance Manual, training material, installation and operational procedures;
 - Draft Version Description Document;
 - Ready-to-use client facilities, equipment, software infrastructure, and applications data;
 - Committed client personnel for transition and training
 - A Transition Plan (a mini WWWWWHH (*Why, What, When, Who, Where, How, How Much*) plan) for the transition activities, including completion criteria for the Release Readiness Review (RRR).

The Transition Plan may identify transition preconditions whose satisfaction is deferred for the Release Readiness Review (e.g., final tailoring of user interfaces and operational procedures). The completion criteria for the Release Readiness Review (RRR) will include the completion criteria for the deliverables in Section 2.3, plus any situation-specific criteria (e.g., the degree of cutover from the existing operation to be accomplished by the Release Readiness Review).

- Release Readiness Review (RRR)
 - The Release Readiness Review (RRR) should verify the successful completion of the Transition Plan and the Readiness of the system and the clients to transition into client operations. The Release Readiness Review (RRR) should review all operations-critical items, such as:
 - system preparation;
 - training;
 - usage;
 - evolution support with clients.
 - It should reference section 2.3 for “acceptability” criteria for deliverables

- Review plan should indicate how client wants to accomplish the final review to assure satisfactory system transition and identify candidate dates for each

4.4 Configuration Management

The objective of Configuration Management (CM) is to provide a stable foundation for software life cycle by establishing baseline versions of key products, and maintaining them under a formal change control process. To achieve this objective, CM involves five major functions:

- (1) Configuration Identification: Identify the key products to be baselined and the project milestones at which they enter the CM process. These would correspond to the various phase deliverables in Section 2.2.
- (2) Change Control: Provide a flow chart indicating the sequence of tasks and decisions involved in submitting, analyzing, approving, and implementing proposed changes to software baseline items. Provide the associated change control forms and procedures, and a chart indicating what level of management is responsible for approving the various classes of proposed changes.
- (3) Configuration Status Accounting: Identify the purpose, content, and format of the various status accounting records and reports, and the procedures for operating the status accounting system.
- (4) Configuration Audits: Identify the responsibilities, schedules, and procedures involved in performing audits of the integrity of the CM baselines and records.
- (5) Project Library Management: Describe the operation of the project library, including:
 - (a) organizational responsibilities;
 - (b) library contents;
 - (c) services provided;
 - (d) operational procedures for general usage, storage and release of master copies, security, backup and recovery;
 - (e) library facilities and support services;
 - (f) staffing and resource requirements

Course Guidelines:

It will be sufficient to identify:

- Which items will be baselined when (e.g., LCA package rebaselined at ARB-III)
- How changes to the baseline will be coordinated with the client (e.g., meeting for major changes, email for moderate changes, none for trivial changes)
- How outstanding problem reports will be tracked
- Who will be the custodian of the master baselined versions, and how he/she will preserve the integrity and recoverability of the master versions

4.5 Project Communications

Provide a plan for how team members will communicate with each other and the client(s) covering:

- Meeting schedules
- Use of email, web, etc.
- Conference calls, etc.
- Use of word processing systems for document integration

4.7 Facilities and Related Concerns

Identify, as appropriate, the functions, milestones, responsibilities, physical configurations and operational procedures involved in preparing and operating project facilities, and in handling related concerns, including:

- Support services
- Support software
- Customer furnished facilities, equipment, and services
- Security
- Subcontractor operations
- Use of commercial software

Facilities may include:

- computer rooms, flooring, power supply, and air conditioning
- computers, peripherals, and supplies
- data communications capabilities
- office space, furniture, equipment, utilities, and supplies
- transportation, parking, and employee services

Course Guidelines:

Identify library-furnished equipment, software, services, documentation, data, and facilities. A schedule detailing when these items will be needed shall also be included. Also, include other required resources, including a plan for obtaining the resources, dates needed, and availability of each resource item.

4.8 Status Monitoring and Control

Describe the techniques, procedures, and reports to be used in tracking project progress vs. plans and expenditures vs. budgets. Include, as appropriate:

- Risk Monitoring Procedures
- Summary Task Planning Sheets
- Earned Value Status Reports
- Project Expenditure Summaries
- Cumulative Milestone Progress Reports
- PERT/ COST Systems
- Budget-Schedule-Milestone Charts
- Personnel Loading Charts
- Detailed Expenditure vs. Budget Reports

Course Guidelines:

Most of the items above would be an over-kill. It will generally be sufficient to identify how milestones will be tracked (e.g., via text schedules, Microsoft Project, etc.), and who is responsible for monitoring and controlling what (e.g., project manager for major milestone completion, QA person for inspections and product content).

Describe the software metrics used for tracking and controlling the project development, and the process used to collect and analyze the metrics. Each team must report weekly progress, effort and trouble report metrics as well as risk items, using Weekly Effort Forms submitted by the team members. Describe all the different sets of progress metrics that will be tracked. Examples include major development milestones, lines of code, etc. Progress metrics can also be broken down by function or sub-teams. Optional metrics include requirements volatility.

4.8 Support Environment, Methods, and Tools

Describe the key environment, methods, and tools choices: Use **Figure 2 Software Tools Coverage by Activity** as checklist. Refer to [Royce, 1998] Chapter 12 for more information

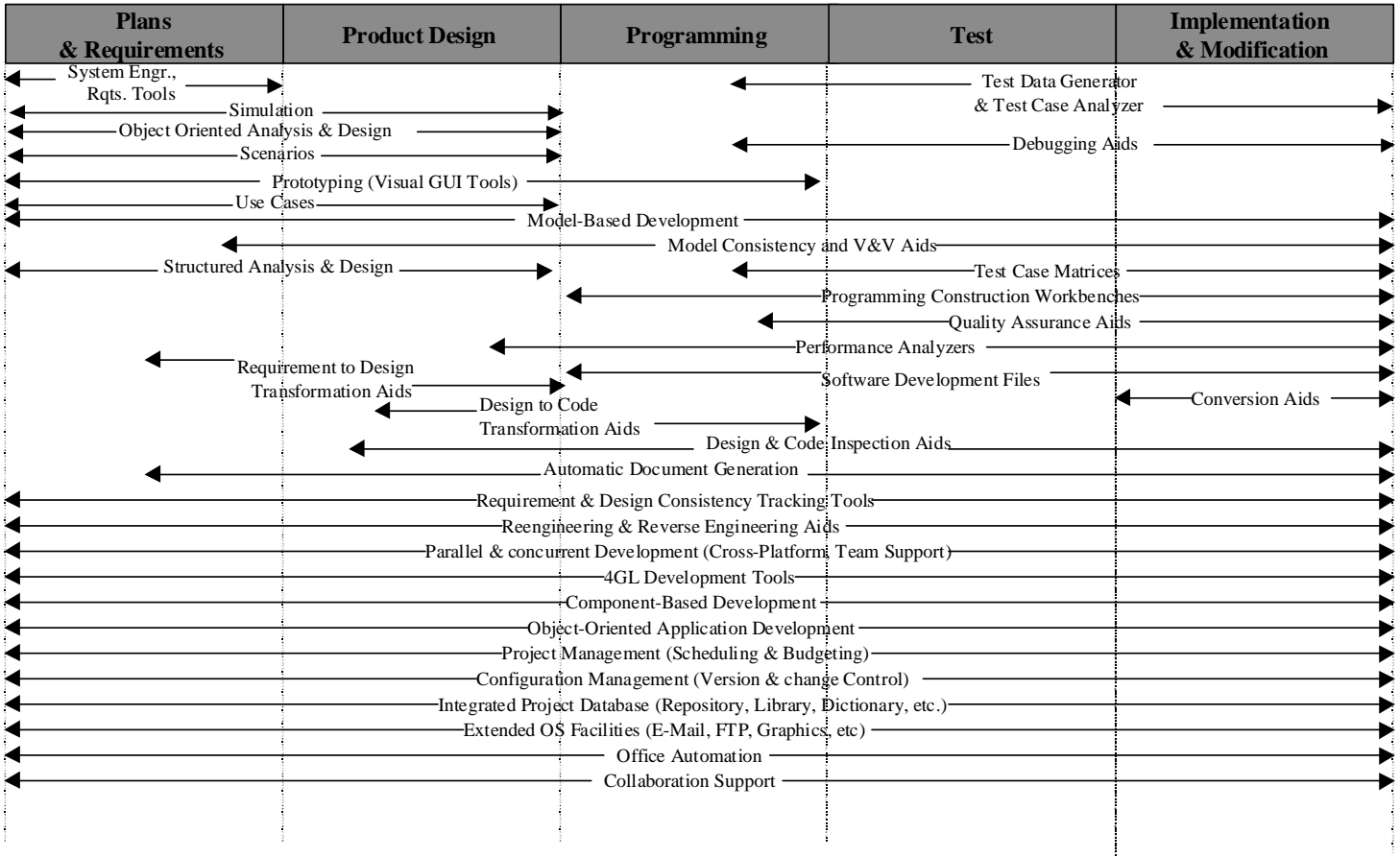


Figure 2 Software Tools Coverage by Activity

5. Resources

This Section tells *how much* resources the project will require to perform the functions indicated in the previous sections. It identifies the resources by type (personnel, calendar time, capital dollars, operations dollars, etc.), by expenditure category (labor, computer, travel, publications, etc.), by life cycle phase, and by project activity.

Integration and Dependencies with other components:

The tasks must fit into the required capabilities as described in the SSRD and SSAD, which are summarized in Item 5.1. Based on that summary, the budget can be estimated and refined. The resource requirements are the levels of investment needed for the business case in the Feasibility Rationale.

Additional Guidelines:

Use COCOMO II for the cost estimation. See [Royce, 1998] Section 10.1 for WBS guidelines.

5.1 Work Breakdown Structure

The WBS provides a hierarchical ordering of project tasks and activities which serves as a basis for project budgeting, cost collection, and control.

Tailor the following WBS chart indicating the project's WBS elements, their associated budgets, and the person responsible for the tasks and budgets. Form the WBS by an appropriate tailoring of the project-specific WBS product.

- A Management
 - AA Inception phase management
 - AAA Top-level Life Cycle Plan (LCO version of LCP)
 - AAB Inception phase project control and status assessments
 - AAC Inception phase stakeholder coordination
 - AAD Elaboration phase commitment package and review (LCO package preparation and ARB review)
 - AB Elaboration phase management
 - ABA Updated LCP with detailed construction plan (LCA version of LCP)
 - ABB Elaboration phase project control and status assessments
 - ABC Elaboration phase stakeholder coordination
 - ABD Construction phase commitment package and review (LCA package preparation and ARB review)
 - AC Construction phase management
 - ACA Updated LCP with detailed transition and evolution plans
 - ACB Construction phase project control and status assessments
 - ACC Construction phase stakeholder coordination
 - ACD Transition phase commitment package and review (IOC package preparation and PRB review)
 - AD Transition phase management
 - ADA Updated LCP with detailed next-generation planning
 - ADB Transition phase project control and status assessments
 - ADC Transition phase stakeholder coordination
 - ADD Evolution stage commitment package and review (PR package preparation and PRB review)
- B Environment and Configuration Management (CM)
 - BA Inception phase environment/CM scoping and initialization
 - BB Elaboration phase environment/CM
 - BBA Development environment installation and administration
 - BBB Elaboration phase CM
 - BBC Development environment integration and custom toolsmithing
 - BC Construction phase environment/CM evolution
 - BCA Construction phase environment evolution
 - BCB Transition phase CM
 - BD Transition phase environment/CM evolution\
 - BDA Construction phase environment evolution
 - BDB Transition phase CM
 - BDC Evolution stage environment packaging and transition
- C Requirements
 - CA Inception phase requirements development
 - CAA Operational Concept Description and business modeling (LCO version of OCD)
 - CAB Top-level System and Software Requirements Definition (LCO version of SSRD)
 - CAC Initial stakeholder requirements negotiation
 - CB Elaboration phase requirements baselining
 - CBA OCD elaboration and baselining (LCA version of OCD)
 - CBB SSRD elaboration and baselining (LCA version of SSRD)
 - CC Construction phase requirements evolution
 - CD Transition phase requirements evolution
- D Design
 - DA Inception phase architecting
 - DAA Top-level system and software architecture description (LCD version of SSAD)
 - DAB Evaluation of candidate COTS components
 - DB Elaboration phase architecture baselining
 - DBA SSAD elaboration and baselining
 - DBB COTS integration assurance and baselining

- DC Construction phase design
 - DCA SSAD evolution
 - DCB COTS integration evolution
 - DCC Component design
- DD Transition phase design evolution
- E Implementation
 - EA Inception phase prototyping
 - EB Elaboration phase component implementation
 - EBA Critical component implementation
 - EC Construction phase component implementation
 - ECA Alpha release component coding and stand-alone testing
 - ECB Beta release (IOC) component coding and stand-alone testing
 - ECC Component evolution
 - ED Transition phase component evolution
- F Assessment
 - FA Inception phase assessment
 - FAA Initial assessment plan (LCO version; part of SDP)
 - FAB Initial Feasibility Rationale Description (LCO version of FRD)
 - FAC Business case analysis (part of FRD)
 - FB Elaboration phase assessment
 - FBA Elaboration of assessment plan (LCA version; part of SDP)
 - FBB Elaboration feasibility rationale (LCA version of FRD)
 - FC Construction phase assessment
 - FCA Detailed test plans and procedures
 - FCB Evolution of feasibility rationale
 - FCC Peer reviews
 - FCD Alpha release assessment
 - FCE Beta release (IOC) assessment
 - FD Transition phase assessment
- G Deployment
 - GA Inception phase deployment planning (LCO version; part of SDP)
 - GB Elaboration phase deployment planning (LCA version; part of SDP)
 - GC Construction phase deployment planning and preparation
 - GCA Transition plan development
 - GCB Evolution plan development
 - GCC Transition preparation
 - GD Transition phase deployment

5.2 Budgets

Provide breakdowns of the software life cycle project budget and staffing level requirements. These should include, as appropriate, breakdowns by:

- WBS element;
- Phase and by calendar month;
- Labor grade (analyst, programmer, operator, clerical, etc.);
- Budget category (capital dollars, operations dollars, etc.);
- Expenditure category (labor, computer, travel, publications, miscellaneous).

A checklist of potential miscellaneous expenditures is given in Table 5.

Include Effort and Schedule estimates using at least two different, credible, and repeatable cost estimation techniques.

- Object points (You should have more or less a fair idea on the number of screens, reports, and 3GL components in your application, especially after doing your prototype, SSRD, and SSAD)
- SLOC
- Function Points (backfiring tables can convert function-points to SLOC)

It is critical to document any assumptions used to come up with the estimate: e.g., when using COCOMO, ratings for the Effort Multipliers, Scale factors, etc... The Effort (Person-months) and Schedule should be used to validate the staffing requirements and confirm that the project is feasible within the allotted budgets and schedule (FRD 3.3).

Table 5: Miscellaneous Software Project Expenditure Sources

<i>Clerical Costs</i>	
<i>Related Personnel Costs</i>	Overtime, benefits, hiring, termination, relocation, education; personnel costs for product acquisition: contracts, legal, receiving, inspection, etc.
<i>Related Computer Costs</i>	Installation, maintenance, insurance, special equipment: terminals, control units, data entry devices, etc.
<i>Office Equipment Costs</i>	Computers, telephones, copiers, file cabinets, desks, chairs, software, etc.
<i>Software Product Costs</i>	Purchase, rental, licensing, maintenance of software components, utilities, tools, etc.
<i>Supplies Costs</i>	Disks, forms, paper, print cartridges, office supplies, etc.
<i>Telecommunication Costs</i>	Line charges, connection charges, special equipment: modems, video conferencing, etc.
<i>Facility Costs</i>	Office Rental, electricity, air conditioning, heating, water, taxes, depreciation, cleaning, repairs, insurance, security, fire protection, etc.
<i>Other costs</i>	Travel, postage, printing, consulting fees, books, periodicals, conventions, equipment relocation, etc.

Course Guidelines:

The budget breakdowns in the guidelines are more for big projects than for class projects. Provide a manual COCOMO II Object-Points analysis, and a USC COCOMO II SLOC-based estimate, and provide breakdowns by item of the overall equipment, data preparation and other costs identified in the WBS in Section 5.1.

Common Pitfalls:

Inconsistent with SSRD budget requirements (SSRD 2.1) and FRD development costs (FRD 2.1.1)

Appendices

The Appendix may be used to provide additional information published separately. As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided. As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided.

- Properly formatted COCOMO Results including the CLEF, EAF for each module, schedule and scale factors and phase distributions.
- Detailed Gantt Charts for milestones and schedules

Feasibility Rationale Description (FRD)

Purpose

- Ensure feasibility and consistency of other package components (OCD, SSRD, SSAD, LCP, Prototype)
- Demonstrate viable business case for the system
- Identify shortfalls in ensuring feasibility, consistency, and business case as project risk items for LCP
- Demonstrate that a system built using the specified architecture (described in the SSAD) and life cycle process (described in the LCP) will:
 - satisfy the requirements described in the SSRD
 - support the operational concept described in the OCD
 - satisfy the success-critical stakeholders in the OCD and LCP
 - remain faithful to the key features determined by the prototype described in the OCD and SSRD
 - stay within the budgets and schedules in the LCP
- Rationalize life cycle decisions in a way the prime audience (the customer and users) and other stakeholders can understand
- Enable the customers to participate in the decision process and to express their satisfaction with the product

Completion Criteria

Below are the completion criteria for the Feasibility Rationale Description for the two phases:

- Life Cycle Objectives (Inception Phase)
- Life Cycle Architecture (Elaboration Phase)

Life Cycle Objectives (LCO)

- Assurance of consistency among elements above for at least one feasible architecture
 - Via analysis, measurement, prototyping, simulation, etc.
 - Business case analysis for requirements, feasible architectures

Life Cycle Architecture (LCA)

- Assurance of consistency among elements above for the architecture specified in the SSAD
- All major risks resolved or covered by risk management plan

Intended Audience

- The primary audiences are the LCO and LCA Architecture Review Board members
 - Key system stakeholders
 - Experienced peers
 - Technical Specialists in critical areas
- The parts dealing with client satisfaction must be understandable by the client representatives on the ARB.
- The technical parts must be sufficiently detailed and well organized to enable the peers and technical experts to efficiently assess the adequacy of the technical rationale.
- The FRD is of considerable value to developers and other stakeholders in providing a rationale for important decisions made by the project.

Participants

- Project manager responsible for content
- OCD author should prepare business case
- All stakeholders responsible for consistency and feasibility via Win-Win negotiations
- Agreements can be contingent on demonstration of feasibility

High Level Dependencies

The thoroughness of the Feasibility Rationale is dependent on the thoroughness of all the other LCO and LCA components. Issues incompletely covered in the Feasibility Rationale are sources of risk, whose management should be covered in the Life Cycle Plan's (LCP) Risk Management and Monitoring Procedures section (LCP 4.1)

Overall Tool Support

Well-calibrated estimation models for cost, schedule, performance, or reliability are good sources of feasibility rationale. Others are prototypes, simulations, benchmarks, architecture analysis tools, and traceability tools. The rationale capture capability in the WinWin tool is also useful.

Outline

1. Introduction
 - 1.1 Purpose of the Feasibility Rationale Document
 - 1.2 References
2. Product Rationale
 - 2.1 Business Case Analysis
 - 2.1.1 Development Cost Analysis
 - 2.1.2 Implementation Cost Estimate
 - 2.1.3 Operational Cost Estimate
 - 2.1.4 Maintenance Cost Estimate
 - 2.1.5 Estimate of Value Added and Return on Investment
 - 2.2 Requirements Satisfaction
 - 2.2.1 Operational Concept Satisfaction
 - 2.2.2 Project Requirements Satisfaction
 - 2.2.3 Capability Requirements Satisfaction
 - 2.2.4 Interface Requirements Satisfaction
 - 2.2.5 Level of Service Requirements Satisfaction
 - 2.2.6 Evolution Requirements Satisfaction
 - 2.3 Stakeholder Concurrence
3. Process Rationale
 - 3.1 System Priorities
 - 3.2 Process Match to System Priorities
 - 3.3 Consistency of Priorities, Process and Resources
4. Project Risk Assessment
5. Analysis Results
 - 5.1 Product Features
 - 5.1.1 Advantages
 - 5.1.2 Limitations
 - 5.1.3 Tradeoffs Considered
 - 5.1.4 Changes Not Included
 - 5.2 Commercial-Off-The-Shelf Solutions
6. Appendices
 - A. Cash Flow Statement

1. Introduction

1.1 Purpose of the Feasibility Rationale Description Document

- Summarize the purpose and contents of this document with respect to the particular project and people involved
- Avoid generic introductions as much as possible: for instance, you can show how your particular Feasibility Rationale Description meets the completion criteria for the given phase

Common Pitfalls:

- Simply repeating the purpose of the document from the guidelines

1.2 References

Provide complete citations to all documents, meetings and external tools referenced or used in the preparation of this document.

2. Product Rationale

This section furnishes the rationale for the product being able to satisfy the system specifications and stakeholders (e.g. customer, user). It should also provide the rationale as to why the proposed system is better than the current system.

Integration and Dependencies with other components:

This section is highly dependent on all other documents. The cost estimates in Item 2.1 are strongly dependent on development cost (from LCP) and operational cost (from OCD). Item 2.2 maps requirements to design, which create a high dependency between the System and Software Requirements Description (SSRD), the System and Software Architecture Description (SSAD), and often the prototype. Similarly, item 2.3s create a dependency between the OCD, the SSAD, and often the prototype. The stakeholder concurrence in Item 2.4 summarizes the findings so that green light can be given to proceed with the development.

Additional Guidelines:

Architecture attribute analysis methods can be used to assess feasibility of quality attribute requirement levels (See Table 6). The rationale capture capability in the WinWin tool is also useful.

Table 6 Top-Level Field Guide to Software Architecture Attribute Analysis Methods

Method	Examples	Strengths	Potential Concerns
Interface Checking	StP, RDD-100	<ul style="list-style-type: none"> • Static integrity (partial) • Traceability 	<ul style="list-style-type: none"> • Dynamic integrity • Performance, cost, schedule analysis • Subjective attributes
Formalized Models	Rapids, Wright, HDM, AAA	<ul style="list-style-type: none"> • Static, dynamic integrity • Security • Interoperability 	<ul style="list-style-type: none"> • Model granularity and scalability • Cost, schedule, reliability, full performance • Subjective attributes
Scenario Analysis	SAAM	<ul style="list-style-type: none"> • Subjective attributes <ul style="list-style-type: none"> - Usability, modifiability • Human-machine system attributes: <ul style="list-style-type: none"> - Safety, security, survivability 	<ul style="list-style-type: none"> • Largely manual, expertise-dependent • Scenario representativeness; method scalability • Verification/Validation/Accreditation • Integrity, performance, cost, schedule analysis
Simulation; Execution	Network VOA; UNAS	<ul style="list-style-type: none"> • Performance analysis • Dynamic integrity • Reliability, survivability, accuracy 	<ul style="list-style-type: none"> • Model granularity and scalability • Input scenario representativeness • Verification/Validation/Accreditation • Cost, schedule, subjective attributes
Parametric Modeling	A4, COCOMO, Queuing Models	<ul style="list-style-type: none"> • Cost, schedule analysis • Reliability, availability analysis • Performance analysis 	<ul style="list-style-type: none"> • Subjective attributes • Static, dynamic integrity • Verification/Validation/Accreditation • Input validation

2.1 Business Case Analysis

This section describes the impact of the product in mainly monetary terms.

- How much does it cost to develop and to operate?
- How much added value does it generate?
- How high is its return on investment?

However, non-monetary factors may be also decisive. For instance, “added value” can include the improved quality of the service provided by the product.

- For a commercial system, the business case analysis will generally demonstrate an acceptable financial return on investment.
- For a research and education support system, the rationale would be expressed in terms of improvements in research and educational effectiveness as expressed by the users, or in terms of cost savings to achieve the desired level of effectiveness

2.1.1 Development Cost Analysis

- Using estimates computed in the section Budgets (LCP 5.2), provide a summary of the full development cost, including hardware, software, people, and facilities costs.
- Provide a rationale for the EAF cost drivers included in the COCOMO estimates.

Common Pitfalls:

- Repeating the analysis from LCP 5.2. Provide only a summary, and reference the detailed analysis

2.1.2 Transition Cost Estimate

- Provide a rough estimate of costs to be incurred during the transition of the product into production
- These costs may include:
 - Training Time
 - Data preparation
 - COTS licenses
 - Operational readiness testing
 - Site preparation
 - Facilities preparation
 - Equipment purchase

2.1.3 Operational Cost Estimate

- Provide a summary of the operational costs, including costs for the operational and additional support software

2.1.4 Maintenance Cost Estimate

- Provide a summary of maintenance costs if applicable
- Use COCOMO Maintenance data (optional)

Common Pitfalls:

- Repeating the analysis from LCP 5.2. Provide only a summary, and reference the detailed analysis

2.1.5 Estimate of Value Added and Return on Investment

- Provide a summary of cost with and without the product, and how much value it adds
- The value added may also describe non-monetary improvements (e.g. quality, response time, etc.), which can be critical in customer support and satisfaction.
- Include a Return-On-Investment (ROI) analysis as appropriate.

Common Pitfalls:

- Not providing a dollar value to the estimate and return on investment.

2.2 Requirements Satisfaction

- This section summarizes how well a system developed to the product architecture will satisfy the system requirements.

Common Pitfalls:

- Simply restating the requirements, without showing how and why the proposed architecture guarantees that they will be met

2.2.1 Operational Concept Satisfaction

- Summarize product's ability to satisfy the **key** operational concept elements and **critical** scenarios, including critical off-nominal scenarios (Exception-Handling Scenarios)
- [Consistent with Operational Scenarios (OCD 3.4.3)]

2.2.2 Project Requirements Satisfaction

- Summarize how project requirements are being met through the approach adopted for the project and described in LCP 4.

2.2.3 Capability Requirements Satisfaction

- Show evidence that the system developed to the product architecture will satisfy the capability requirements, e.g., “capability described/demonstrated/exercised as part of included COTS component”, with a pointer to the results.
- No need to restate obvious mappings from the requirements to the architecture.
- For each **critical** requirement, indicate:
 - Criticality:** Describe how essential this requirement is to the overall system
 - Technical issues:** Describe any design or implementation issues involved in satisfying this requirement.
 - Cost and schedule:** Describe the relative or absolute costs associated with the technical issues associated with satisfying that particular requirement
 - Dependencies:** Dependencies on COTS package capabilities, externally furnished components, etc.
 - Side effects:** Interactions with other requirements
 - Risks:** Describes the circumstances under which this requirement might not able to be satisfied, and what actions can be taken to reduce the probability of this occurrence. Describe some Risk resolution options
- [Consistent with System Requirements (SSRD 3.2)]

2.2.4 Interface Requirements Satisfaction

- Show evidence that the system developed to the product architecture will satisfy the **critical** interface requirements.
- [Consistent with System Interface Requirements (SSRD 4)]

2.2.5 Level of Service Requirements Satisfaction

- Show evidence that the system developed to the product architecture will satisfy the **critical** quality requirements.
- Table 6 summarizes the most effective analysis methods available for each quality attribute
- Table 7 and Table 8 show some effective architecture, product and process strategies for ensuring Level of Service Requirements Satisfaction
- [Consistent with Level of Service Requirements (SSRD 5)]

Table 7 Quality Attribute Strategies and Relations: Architecture Strategies

Primary Attribute	Architecture Strategy	Other Attribute Reinforcement	Other Attribute Conflicts	Special Cases, Comments
Dependability	Input acceptability checking	Interoperability, Usability	Development Cost/schedule, Performance	
	Redundancy		Development Cost/schedule, Evolvability, Performance, Usability	
	Backup/recovery		Development Cost/schedule, Evolvability, Performance	
	Monitoring & Control		Development Cost/schedule, Performance	Performance reinforcement in long term via tuning

Interoperability	Input acceptability checking	Dependability, Usability	Development Cost/schedule, Performance	
	Layering	Evolvability/Portability, Reusability	Development Cost/schedule, Performance	
Usability	Error-reducing user input/output	Dependability	Development Cost/schedule, Performance	
	Input acceptability checking	Dependability, Interoperability	Development Cost/schedule, Performance	
Performance	Architecture balance	Cost/Schedule		
	Domain architecture-driven	Cost/Schedule		
Evolvability/Portability	Layering	Interoperability, Reusability	Development Cost/schedule, Performance	
Cost/Schedule	Architecture balance	Performance		
	Domain architecture-driven	Performance		
Reusability	Domain architecture-driven	Interoperability, Reusability	Development Cost/schedule, Performance	
	Layering	Interoperability, Evolvability/Portability	Development Cost/schedule, Performance	

Table 8: Quality Attribute Product and Process Strategies

	Product Strategies	Process Strategies
Dependability	Accuracy Optimization, Backup/Recovery, Diagnostics, Error-reducing User Input/output, Fault-tolerance Functions, Input Acceptability Checking, Integrity Functions, Intrusion Detection & Handling, Layering, Modularity, Monitoring & Control, Redundancy	Failure Modes & Effects Analysis, Fault Tree Analysis, Formal Specification & Verification, Inspections, Penetration, Regression Test, Requirements/Design V & V, Stress Testing, Test Plans & Tools
Interoperability	Generality, Integrity Functions, Interface Specification, Layering, Modularity, Self-containedness	Interface Change Control, Interface Definition Tools, Interoperator Involvement, Specification Verification
Usability	Error-reducing User Input/output, Help/explanation, Modularity, Navigation, Parametrization, UI Consistency, UI Flexibility, Undo, User-programmability, User-tailoring	Prototyping, Usage Monitoring & Analysis, User Engineering, User Interface Tools, User Involvement
Performance	Descoping, Domain Architecture-driven, Optimization (Code/ Algorithm), Platform-feature Exploitation	Benchmarking, Modeling, Performance Analysis, Prototyping, Simulation, Tuning, User Involvement
Adaptability (Evolvability / Portability)	Generality, Input Assertion/type Checking, Layering, Modularity, Parameterization, Self-containedness, Understandability, User-programmability, User-tailorability, Verifiability	Benchmarking, Maintainers & User Involvement, Portability Vector Specification, Prototyping, Requirement Growth Vector Specification & Verification

Development Cost / Schedule	Descoping, Domain Architecture-driven, Modularity, Reuse	Design To Cost/schedule, Early Error Elimination Tools And Techniques, Personnel/Management, Process Automation, Reuse-oriented Processes, User & Customer Involvement
Reusability	Domain Architecture-driven, Portability Functions	Domain Architecting, Reuser Involvement, Reuse Vector Specification & Verification
All of Above	Descoping, Domain Architecture-driven, Reuse (For Attributes Possessed By Reusable Assets)	Analysis, Continuous Process Improvement, Incentivization, Inspections, Personnel/Management Focus, Planning Focus, Requirement/design V&V, Review Emphases, Tool Focus, Total Quality Management

2.2.6 Evolution Requirements Satisfaction

- Show evidence that the system developed to the product architecture will satisfy the **critical** evolution requirements (e.g., show which parts of the architecture ensure an easy transition to support via the IBM Digital Library package).
- [Consistent with Evolution Requirements (SSRD 6)]

2.3 Stakeholder Concurrence

- Summarize stakeholder concurrence by reference to :
 - WinWin negotiation results
 - Memoranda of agreements
- Stakeholders may be anybody involved in the development process. For instance, a developer may claim that a certain response time cannot be achieved in a crisis mode unless nonessential message traffic is eliminated. Similarly, a customer may claim that the product does not satisfy his/her win conditions (e.g. cost).
- This section serves as a record of how such claims were resolved to the stakeholders' satisfaction.

3. Process Rationale

This section analyzes the ability of the development to satisfy the stakeholders' (e.g. customer) cost and schedule constraints.

Integration and Dependencies with other components: Like the previous section, this section is also highly dependent on other documents, foremost the Life Cycle Plan (LCP) and System and Software Requirements Description (SSRD). Item 3.1 maps primarily to the capabilities in SSRD and milestones in LCP 2.2. Item 3.2 is a summary of LCP 2.1 and 2.2, with emphasis on priorities above. Item 3.3 is reasoning that the LCP is consistent and doable (especially LCP 4).

3.1 System Priorities

- Summarize priorities of desired capabilities and constraints. Priorities may express time and date as well as quality and others (e.g. performance).
- These priorities should be derived from the Organization Goals (OCD 2.2) and Project Goals (OCD 3.1) as well as System Requirements (SSRD 3.1.1)

Common Pitfalls:

- Prioritizing on a capability by capability basis instead of requirement by requirement basis

3.2 Process Match to System Priorities

- Provide rationale for

- Ability to meet milestones
- Choice of process model: The decision table (Table 9) provides guidance on selecting an appropriate process model for various combinations of system objectives, constraints and alternatives.
- Spiral Cycles, Anchor points
- Increments; Design-to-Schedule options

3.3 Consistency of Priorities, Process and Resources

- Provide evidence that priorities, process and resources match
 - Budgeted cost and schedule are achievable
 - No single person is involved on two or more full-time tasks at any given time
 - Low priority features can be feasibly dropped to meet budget or schedule constraints
- Using the estimated Effort (Person-months) and Schedule from Budgets (LCP 5.2), show that the staffing levels are enough, and that the project is achievable within the schedule.
- It is important to use a credible and repeatable estimation technique for the Effort and the Schedule.

Table 9: Process Model Decision Table

Objectives, Constraints			Alternatives		Model	Example
Growth Envelope	Understanding of Requirements	Robustness	Available Technology	Architecture Understanding		
Limited			COTS		Buy COTS	Simple Inventory Control
Limited			4GL, Transform		Transform or Evolutionary Development	Small Business - DP Application
Limited	Low	Low		Low	Evolutionary Prototype	Advanced Pattern Recognition
Limited to Large	High	High		High	Waterfall	Rebuild of old system
	Low	High			Risk Reduction followed by Waterfall	Complex Situation Assessment
		High		Low		High-performance Avionics
Limited to Medium	Low	Low-Medium		High	Evolutionary Development	Data Exploitation
Limited to Large			Large Reusable Components	Medium to High	Capabilities-to-Requirements	Electronic Publishing
Very Large		High			Risk Reduction & Waterfall	Air Traffic Control
Medium to Large	Low	Medium	Partial COTS	Low to Medium	Spiral	Software Support Environment

Conditions for Additional Complementary Process Model Options

Design-to-cost or Design-to-schedule	Fixed Budget or Schedule Available
Incremental Development	Early Capability Needed
(only one condition is sufficient)	Limited Staff or Budget Available

Downstream Requirements Poorly Understood
 High-Risk System Nucleus
 Large to Very Large Application
 Required Phasing With System Increments

4. Project Risk Assessment

Any combinations of capabilities or objectives whose feasibility is difficult to assure, are major sources of risk. Risk Assessment consists of risk identification, risk analysis and risk prioritization. Frequent major sources of risk and techniques for resolving them are given in Table 10. The project's overall life cycle strategy described in Section 2.1 should be consistent with its approach to risk management. The initial set of risks defined here will be updated throughout the project.

- Identify the major sources of risk in the project.
- Provide a description of all identified risks for the project, including risk exposure quantities.
- For critical risks, indicate the following:
 - Description
 - Risk Exposure: Potential Magnitude and Probability of Loss
 - Risk Reduction Leverage: in reducing risk exposure
 - Actions to Mitigate Risk
 - Contingency Plan
- Identify low-priority requirements that can be left out in the case of schedule slippage

Table 10 Software Risk Management Techniques

Source of Risk	Risk Management Techniques
1. Personnel shortfalls	<ul style="list-style-type: none"> • Staffing with top talent; key personnel agreements; team-building; training ; tailoring process to skill mix; walkthroughs.
2. Schedules, budgets, process	<ul style="list-style-type: none"> • Detailed, multi-source cost and schedule estimation; design to cost; incremental development; software reuse; requirements descoping; adding more budget and schedule; outside reviews.
3. COTS, external components	<ul style="list-style-type: none"> • Benchmarking; inspections; reference checking; compatibility prototyping and analysis
4. Requirements mismatch	<ul style="list-style-type: none"> • Requirements scrubbing; prototyping; cost-benefit analysis; design to cost; user surveys
5. User interface mismatch	<ul style="list-style-type: none"> • Prototyping; scenarios; user characterization (functionality; style, workload); identifying the real users
6. Architecture, performance, quality	<ul style="list-style-type: none"> • Simulation; benchmarking; modeling; prototyping; instrumentation; tuning
7. Requirements changes	<ul style="list-style-type: none"> • High change threshold: information hiding; incremental development (defer changes to later increments)
8. Legacy software	<ul style="list-style-type: none"> • Reengineering; code analysis; interviewing; wrappers; incremental deconstruction
9. Externally-performed tasks	<ul style="list-style-type: none"> • Pre-award audits, award-fee contracts, competitive design or prototyping
10. Straining computer science	<ul style="list-style-type: none"> • Technical analysis; cost-benefit analysis; prototyping; reference checking

Additional Guidelines:

There are numerous risk identification and analysis tools that can be applied in this section (COCOMO II is again integral here). However, they can only give guidelines, not real answers. The best preparation for this section is to try to construct the Feasibility Rationale and see where you have difficulties.

Common Pitfalls:

- Repeating the same table in the Risk Management section of LCP (LCP 4.1)

5. Analysis Results

- Identify architectural alternatives and tradeoffs. Identify unfeasible architectures or rejected alternatives; document criteria for rejection to avoid having the rejected architectural alternative selected in ignorance at some other point
- Describe feasible architectural alternatives which were rejected due to solution constraints on the way that the problem must be solved, such as a mandated technology. Those architectural alternatives may be reconsidered should the solution constraints be relaxed.

5.1 Product Features

5.1.1 Advantages

This paragraph shall provide a qualitative and quantitative summary of the advantages to be obtained from the new or modified system with respect to the Organization Goals and Activities. This summary shall include new capabilities, enhanced capabilities, and improved performance, as applicable, and their relationship to deficiencies identified in the Current System Shortfalls, as well as the rationale for new capabilities. For a quantitative analysis, you may reference the Business Case Analysis from the FRD 2.1.

You may also describe the relationship of this system with any other systems if they exist. Specify if this system is intended to be stand-alone, used as a component in a larger product, or one of a family of products in a product line. If the latter, this section discusses the relationship of this system to the larger product or to the product line.

5.1.2 Limitations

This paragraph shall provide a qualitative and quantitative summary of potential disadvantages or limitations of the new or modified system. These disadvantages and limitations shall include, as applicable, degraded or missing capabilities, degraded or less-than-desired performance, greater-than-desired use of computer hardware resources, undesirable operational impacts, conflicts with user assumptions, and other constraints. These are used either for stakeholder expectations management or as a basis for further negotiation of system capabilities or tradeoffs.

5.1.3. Tradeoffs Considered

This paragraph shall identify and describe major alternatives for the concept of operation of the system, their characteristics, the tradeoffs among them, and rationale for the decisions reached. Also discuss alternative architectures and their pros and cons.

5.1.4 Changes Considered but Not Included

- In general, the results of the WinWin requirements negotiation activity will be to drop or defer some capabilities from the initially proposed system. It is valuable to capture these for future reference, along with the rationale for dropping or deferring them. Some of those changes considered but not included may become Evolution Requirements.
- Include Reference to WinWin artifact (if applicable)
- You may include a threshold for including some of the deferred capabilities (e.g., depending on the availability of a specific COTS package, etc.)
- [Consistent with Evolution Requirements (SSRD 6)]

5.2 Commercial-Off-The-Shelf solutions

- List of existing COTS products that should be investigated as potential solutions
- Reference any surveys or evaluations that have been done on these products
- Is it possible to buy something that already exists or is about to become available? It may not be possible at this stage to say with a lot of confidence, but any likely products should be listed here.
- Consider whether there are products that must not be used, and state the reason.

6. Appendix

- List or provide any references to supporting documentation
- Provide details of cash flow and project earnings statement.

Sources and References

The guidelines have drawn upon material from the following:

- [Boehm, 1996] Boehm, B., "Anchoring the Software Process", *IEEE Software*, July 1996, pp. 73-82.
- [Boehm, 1989] Boehm, Software Risk Management, IEEE Computer Society Press, 1989.
- [Royce, 1998] Royce, W. Software Project Management: A Unified Framework. Addison Wesley, 1998.

Boehm, B., Egyed, A., Kwan, J., and Madachy, R. (1997), "Developing Multimedia Applications with the WinWin Spiral Model," Proceedings, ESEC/ FSE 97, Springer Verlag.

Boehm, B., Egyed, A., Kwan, J., and Madachy, R. (1998), "Using the WinWin Spiral Model: A Case Study," *IEEE Computer*, July, pp. 33-44.

[MIL-STD-498] Defense Standards MIL-STD-498

[EIA/IEEE J-STD-016] Commercial Standard EIA/IEEE J-STD-016

[Cichocki et al., 1997] Cichocki, A., Abdelsalam, A., Woelk, D., Workflow and Process Automation : Concepts and Technology (Kluwer International Series in Engineering and Computer Science, Secs 432), Kluwer Academic Pub, 1997.

Potts, *IEEE Software*, 1994

Sommerville, I., *Software Engineering*, Addison Wesley, 1995.

IEEE, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990, February 1991.

Since the guidelines have been integrated from multiple sources, they will not necessarily be fully consistent with the guidelines in any individual source. See the following resources for useful perspectives and additional guidelines

Operational Concept Description

[AIAA, 1992] AIAA Recommended Technical Practice, Operational Concept Description Document (OCD), Preparation Guidelines, Software Systems Technical Committee, American Institute of Aeronautics and Astronautics (AIAA), March 1, 1992.

[Fairley et al, 1994] Fairley, R., Thayer R, and Bjorke P., *The Concept of Operations: The Bridge from Operational Requirements to Technical Specifications*, *IEEE*, 1994.

[Lano, 1988] Lano, R.J., A Structured Approach for Operational Concept Formulation (OCF), In *Tutorial: software Engineering Project Management*, edited by R. Thayer, Computer Society Press, 1988.

System and Software Requirements Definition

In, H. (1998). *Conflict Identification and Resolution for Software Attribute Requirements*. Ph.D. Thesis, University of Southern California, Los Angeles, CA.

Template. James & Suzanne Robertson. Atlantic Systems Guild
<http://www.atlsysguild.com/Site/Robs/Templsects.html>

Typical list of data items for system/software development

<http://www.airtime.co.uk/users/wysywig/didlist.htm>

System and Software Architecture Description

C2 Architectural style (see <http://www.ics.uci.edu/pub/arch/c2.html>)

Port, D., Integrated Systems Development Methodology, Telos Press (to appear).

Life Cycle Plan

[AT&T, 1993] Best Current Practices: Software Architecture Validation, Lucent/AT&T, 1993.

[CMU-SEI, 1995] Paulk, M., Weber, C., Curtis, B. The Capability Maturity Model : Guidelines for Improving the Software Process (SEI Series in Software Engineering), Addison-Wesley, 1995.

Appendices

- A. Prototyping Results
- B. Suggested WinWin Taxonomy for MBASE
- C. Level of Service Requirements
- D. Common Definition Language (CDL) For MBASE

Appendix A. Prototyping Results

1. Objectives

- Describe the critical issues and risks that the prototype is attempting to resolve and the uncertainties that the prototype is trying to address

Common Pitfall

One common pitfall when prototyping is to fail to describe the prototype from the perspective of the client. In particular, the prototype should be user-oriented, and should avoid abstracting elements. It helps to use realistic sample data in the various prototype screens.

E.g., use 'History', 'Geography', 'Astronomy', as opposed to 'Subject 1', 'Subject 2', 'Subject 3'.

2. Approach

2.1 Participants

- Describe any participation on the part of the clients in the prototyping effort: e.g., changes requested after initial evaluation
- Describe how effective was the prototype in overcoming initial IKIWISI (I'll Know It When I See It) client expectations
- Mention whether this is the first prototype, or a revised one, including changes suggested by client, etc...

2.2 Tools

- Describe briefly the tool used to develop the prototype and the reasons for choosing that tool.
- Describe how adequate the tool turned out to be to your needs, or whether you are contemplating using a different tool
- Example: "We started by creating a Web based prototype. But we decide to move to Microsoft Access since the system does not require public access and will be used only at the reference librarian desk".

3. Initial Results

For each aspect of the system that you prototyped, describe the:

a. Current way of performing activity

Example: "Currently, to reserve a room, patrons go and look at the Facility schedule and look for empty slots"

b. Proposed way of performing activity

- Include screen shot of relevant prototype screen
- Brief explanations on how system will be used as illustrated by prototype screen (You may annotate explanations directly on screen shots)
- You may propose multiple screens, and indicate which one your client preferred (or maybe hasn't decided yet which one to use).

Example:

"ScreenName1: After entering date, Patron clicks on 'Find Unused Time Slot' button to obtain a list of all the time slots when the room is not booked.

ScreenName2: Patron enters Date and Time, and clicks on 'Check for Availability' button, to obtain whether the

Room is available on that day and time, or not. The patron keeps trying until he finds a available slot".

4. Conclusions

- List by order of priority the items that you will be looking into next, during the next round of prototyping
- List the most critical risks that you hope to resolve by doing further prototyping
- Example: "Current prototype suffers from navigability problems: we will be looking into improving the usability and the navigability using frames, site maps, etc."

Appendix B. Suggested WinWin Taxonomy for MBASE

The suggested domain taxonomy to be used as a checklist and organizing structure for the WinWin requirements negotiation. Each WinWin stakeholder artifact should point to at least one taxonomy element (modify taxonomy as appropriate). Each taxonomy element should be considered as a source of potential stakeholder win conditions and agreements. The WinWin taxonomy roughly corresponds to the table of contents of the System and Software Requirements Definition (SSRD). Mapping the WinWin taxonomy to the SSRD outline is straightforward, but in some cases, some sections need to be combined. In particular, Operational Modes are described in the SSRD with System Requirements. The reason is that the same system functionality may lead to different results depending on the mode.

1. Project Constraints (====>SSRD 2.2 Project Requirements)
 - 1.1 Budget Constraints
 - 1.2. Schedule Constraints
 - 1.3 Staffing Constraints
2. Application Capabilities (====>SSRD 2.3 System Requirements)
 - 2.1 Operational Modes
 - 2.2 User Classes
 - 2.3 Mission Capabilities. These will vary depending on whether the mission involves a multimedia archive, selective dissemination of information, data analysis, etc.
 - 2.4 Support Capabilities
 - 2.4.1 Help
 - 2.4.2 Administration
 - 2.4.2.1 User Account Management
 - 2.4.2.2 Usage Monitoring and Analysis
 - 2.4.3 Maintenance and Diagnostics
3. Quality Attributes (====> SSRD 3. Level of Service Requirements)
 - 3.1 General Qualities
 - 3.1.1 Correctness
 - 3.1.2 Simplicity
 - 3.1.3 Consistency
 - 3.1.4 Completeness
 - 3.1.5 Coherence
 - 3.2 Dependability
 - 3.2.1 Reliability
 - 3.2.2 Accuracy
 - 3.2.3 Availability
 - 3.2.4 Survivability
 - 3.2.5 Serviceability
 - 3.2.6 Verifiability
 - 3.2.7 Resilience
 - 3.3 Security
 - 3.3.1 Integrity
 - 3.3.2 Privacy
 - 3.3.3 Audit
 - 3.3.4 Confidentiality
 - 3.4 Safety
 - 3.5 Interoperability
 - 3.5.1 Compability
 - 3.6 Usability
 - 3.6.1 Mission Orientation
 - 3.6.2 Comprehensiveness
 - 3.6.3 Controllability
 - 3.6.4 Ease of Learning
 - 3.6.5 Ease of Use

- 3.6.6 Help requirements
- 3.7 Performance
 - 3.7.1 Processing Efficiency
 - 3.7.2 Memory Efficiency
 - 3.7.2 Storage Efficiency
 - 3.7.3 Network Efficiency
- 3.8 Adaptability (Evolvability)
 - 3.8.1 Portability
 - 3.8.2 Flexibility
 - 3.8.3 Scalability/Expandability/Extendability/Extensibility
 - 3.8.4 Modifiability
 - 3.8.5 Maintainability
 - 3.8.6 Reconfigurability
- 3.8 Reusability
- 4. Interfaces (====>SSRD 4. System Interface Requirements)
 - 4.1 User Interfaces Requirements
 - 4.1.1 Graphical User Interfaces
 - 4.1.2 Command-Line Interfaces
 - 4.1.3 Application Programming Interfaces
 - 4.1.4 Diagnostics
 - 4.2 Hardware Interfaces
 - 4.3 Communications Interfaces
 - 4.4 Other Software Interfaces (SIRSI, BRS, etc.)
- 5. Environment and Data (====> SSRD 5. Environment and Data)
 - 5.1 Design and Construction Constraints
 - 5.1.1 Tools
 - 5.1.2 Programming Languages
 - 5.1.3 Computer Resources
 - 5.1.4 Standards Compliance
 - 5.2 Packaging
 - 5.3 Implementation
 - 5.4 Software Support Environment Requirements
- 6. Evolution (====>SSRD 6. Evolution Requirements)
 - 6.1 Capability Evolution
 - 6.2 Interface Evolution
 - 6.3 Technology Evolution
 - 6.4 Environment Evolution
 - 6.5 Workload Evolution

Appendix C. Level of Service Requirements

The following glossary is based on the IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, February 1991.

Accuracy

(1) A qualitative assessment of correctness, or freedom from error; (2) A quantitative measure of the magnitude of error

Adaptability

Adaptability is defined by the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed. Syn: *Flexibility*

Audit

Specification of the required audit checks or various audit trails the system should keep to build a system that complies with the appropriate audit rules. This section may have legal implications

Availability

The degree to which a system or component is operational and accessible when required for use. Often expressed as a probability. See also: Error Tolerance; Fault-tolerance; Robustness

Compatibility

(1) The ability of two or more systems or components to perform their required functions while sharing the same hardware or software environment (2) The ability of two or more systems or components to exchange information (See also: Interoperability)

Complexity

(1) The degree to which a system or component has a design or implementation that is difficult to understand and verify. Contrast with: simplicity

Consistency

The degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component.

Correctness

Correctness is defined by: (1) The degree to which a system or component is free from faults in its specification, design, and implementation; (2) The degree to which software, documentation, or other items meet specified requirements; (3) The degree to which software, documentation, or other items meet user needs and expectations, whether specified or not.

Dependability

Dependability is defined as “that property of a computer system such that reliance can justifiably be placed on the service it delivers” [Laprie, 1992]. Depending on the intended application of the system dependability is usually expressed as a number of inter-dependent properties such as reliability, maintainability and safety. It refers to a broad notion of what has historically been referred to as “fault tolerance”, “reliability”, or “robustness”

Efficiency

Efficiency is defined by the degree to which a system or component performs its designated functions with minimum consumption of resources.

Error Tolerance

The ability of a system or component to continue normal operation despite the presence of erroneous inputs. See: Fault tolerance, robustness

Expandability/Extendability/Extensibility

Expandability is defined by the easy with which a system or component can be modified to increase its storage or functional capability.

Fault-tolerance

(1) The ability of a system or component to continue during normal operation despite the presence of hardware or software faults. See also: error tolerance; fail safe; fail soft; fault secure; robustness

Flexibility

Flexibility is defined by the easy with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.

Integrity

Integrity is defined by the degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data.

Example: "Identical up-to-date booking information must be available to all users of the system."

Interoperability

Interoperability is defined by the ability of two or more systems or components to exchange information and to use the information that has been exchanged.

Describe other platforms or environments on which the system is expected to run without recompilation.

Example: "The program should be binary compatible with Windows 3.1, Windows 95 and Windows 98 "

Legality

Describe any legal requirements for this system, to comply with the law to avoid later delays, lawsuits and legal fees.

If the legal requirements are above average, then this section might need to be entirely revisited

Example: "Personal information must be implemented so as to comply with the data protection act."

Example: "The system shall not use any image formats that might infringe with existing copyrights or pending legislation (e.g., GIF)"

Maintainability

Maintainability is defined by: (1) The easy with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment; (2) The easy with which a hardware system or component can be retained in, or restored to, a state in which it can perform its required functions.

Memory Efficiency

List of memory usage requirements that have a genuine effect on the system's ability to fit into the intended environment to set the client and user expectations.

Example: "The system should be able to run on a multi-tasking system with 4MB of free memory"

Example: "Upon exit, the server shall return all the memory it allocates to the pool of free memory on the host computer without any memory leaks".

Modularity

The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

Network Efficiency

List of network usage requirements that have a genuine effect on the system's ability to fit into the intended environment to set the client and user expectations.

Example: "The system should not increase the traffic on the current network by more than 10% "

Performance

Performance is defined by the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage.

Political Correctness

Describe any special factors about the product are necessary for some political or socioeconomic reason: the reality is that the system has to comply with political requirements even if you can find a better/more efficient/more economical solution.

Example: "Our company policy says that we must buy our hardware from Unisys."

Portability

The ease with which a system or component can be transferred from one hardware or software environment to another.

- Describe other platforms or environments to which the system must be ported to quantify client and user expectations about the platforms and future environments in which the system is expected to run.
- Example: "The source code should compile correctly on Solaris and Linux"

Privacy/Confidentiality

Specification of who has authorized access to the system, and under what circumstances that access is granted.

Processing Efficiency

List of response time requirements that have a genuine effect on the system's ability to fit into the intended environment to set the client and user expectations for the response time of the system.

Reliability

Reliability is defined by the ability of a system or component to perform its required functions under stated conditions for a specified period of time.

Reusability

Reusability is the degree to which a software module or other work product can be used in more than one computer program or software system.

Software reusability means that ideas and code are developed once, and then used to solve many software problems, thus enhancing productivity, reliability and quality. Reuse applies not only to source-code fragments, but to all the intermediate work products generated during software development, including requirements' documentation, system specifications, design structures and any information the developer needs to create software

Robustness

The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions. See also: error tolerance; fault tolerance

Scalability

A quantification of how the system should be able to adapt to an increase in the workload without imposing additional overhead or administrative burden.

Storage Efficiency

The degree to which a system or component performs its designated functions with minimum consumption of available storage.

Example: "The system should be able to run with only 40MB of available disk space "

Usability

Usability is defined by the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.

Ease of Learning

A statement of the expected learning time, and any special training needed, for the expected users of the system to guide the designers of the system's interface and functionality and to determine whether or not the user can use the system after the number of hours training/familiarization/use (plus description of training program if applicable) for each type of user.

Example: "The average user should be able to produce a virtual tour within 1 hour of beginning to use the system, without resorting to the manual."

Make sure that you have considered the ease of learning requirements from the perspective of all the different types of users.

Ease of Use

A statement of how easy the system is to use to guide the system's designers.

Example: "The average user must have an error rate less than 2%."

Make sure that you have considered the usability requirements from the perspective of all the different types of users. It is necessary to make some measure of the system's intended usability. This may be that the user labs pronounce that the system is usable, or that it follows all the Apple/Windows interface guidelines, or simply that it must be popular with the majority of users.

Help requirements

A description of the help that the system will provide. The help requirements might become so complex that it is better to treat help as a separately specified system.

Example: "The system must provide context specific help. The user must be able to select an artifact and receive instruction about its use."

These might be requirements that relate to all events (globally accessible help facilities) or they might be requirements that relate to individual events or functional requirements.

Appendix D. COMMON DEFINITION LANGUAGE (CDL) for MBASE

List of Abbreviations

A : Analysis

DD: Domain Description

D: Design

Abstraction DD: A simple interface on complex information. An abstraction is a representation of something, either tangible or conceptual.

Algorithm D: That portion of a behavior that actually carries out the work of the behavior, independent of any decision-making (policy making) necessary to determine which algorithm to execute.

Analogy: The identification of groups of similar relationships between abstractions. An operation abstraction.

Analysis: The part of the software development process whose primary purpose is to formulate a model of the problem domain. Analysis focuses on what to do, design focuses on how to do it. See design.

Architect: The person or persons responsible for evolving and maintaining the system's architecture. Ultimately, the architect gives the system its conceptual integrity.

Architecture: A description of the organization and structure of a system. Many different levels of architectures are involved in developing software systems, from physical hardware architectures to the logical architecture of an application framework.. Describes the static organization of software into subsystems interconnected through interfaces and defines at a significant level how nodes executing those software subsystems interact with each other.

Audience: The person or persons who act as the consumers of an abstraction's interface.

Behavior Model: A representation of the behaviors in a system.

Behavior: Maps to objects.

Boundaries of Control: The point at which a behavior requires interaction with users or other elements outside the system.

Classification: Organizing a set of abstractions according to their common qualities. Classification allows you to reduce complexity in a object model. by making general statements about groups of objects.

Coherence: A measure of an abstraction's elegance. An abstraction is coherent if all of its quality resolutions are both correct and consistent.

Cohesiveness: A measure of an abstraction's elegance. The degree to which an abstraction's qualities fit with the defining quality and with each other.

Common Definitional Language (CDL): A glossary. A common and consistent set of problem space and solution space terminology developed during modeling. Used as a catalog and thesaurus during systematic reuse domain engineering to describe the key concepts.

Comparator: A similarity between two abstractions. If the similarity between the two abstractions is expressed in terms of each abstraction's defining quality, the Comparator is referred to as the Main Comparator. Contrast with Discriminator.

Completeness: A measure of elegance describing whether all of an abstraction's information can be accessed from the interface.

Component: A meta-type whose instances are 'part-of' another abstraction. Components are needed to describe the system to domain experts. Components are compositions of objects (sometimes only one). What an entity is in the domain description, a component is in the system analysis. Components are nouns.

Composition: Creating a new abstraction by combining smaller components into a larger abstraction. Contrast with Decomposition. An operation on an abstraction.

Conceptualization: The earliest phase of development, focused upon providing a proof of concept for the system and characterized by a largely unrestrained activity directed toward the delivery of a prototype whose goals and schedules are clearly defined.

Constraint: A restriction on the resolution of a component's or an object's quality. For instance, an attribute might have a minimum value, or a set of illegal values. An attribute quality

Context: The surrounding environment of a software project.

COTS: Commercial Off The Shelf

Coverage: A measure of elegance, with regard to an object's defining quality. A defining quality has good coverage if it includes everything that should be a part of the abstraction.

Customer: Customers request application systems, place requirements on them, and usually pay for the systems. Customers also interact when deciding on needed features, priorities, and roll-out plans when developing new versions of component systems and the layered system as a whole. Customers can be both internal, such as business process owner, or external, such as another company.

Decomposition: Breaking an abstraction into smaller components. An operation on an abstraction.

Dependency: A requirement that specifies how one element of a system affects another. There are three possible dependencies. Dependencies can be **Semantic** – how a quality of an abstraction(object, attribute, behavior, relationship) is resolved under a given set of conditions, e.g. vacation depends on start date;

Functional – describes how a component uses other components to assist in providing behavior, e.g. alarm – response; or **Conceptual** – effects the defining of qualities, e.g. what a car is may depend on what model it is. Dependency is an attribute quality.

Design: The part of the software development process whose primary purpose is to decide how the system will be implemented. During design, strategic and tactical decisions are made to meet the required functional and quality requirements of a system. See analysis.

Discriminator: A difference between two abstractions. If the difference between the two abstractions is expressed in terms of each abstraction’s defining quality, the discriminator is referred to as the Main Discriminator.

Domain: The part of the real world that a particular software project is intended to model. Compare with Context.

Domain Expert: A person very familiar with the subject matter of the domain and how it fits together.

Domain Model: The sea of classes in a system that serve to capture the vocabulary of the problem space; also known as a conceptual model. A domain model can often be expressed in a set of class diagrams whose purpose is to visualize all of the central classes responsible for the essential behavior of the system, together with a specification of the distribution of roles and responsibilities among such classes.

Elegance: An abstraction that conveys its underlying information using the simplest possible interface. A measure of an abstraction’s elegance is its Information-to-interface Ratio. Qualities that directly impact elegance are: Completeness, Cohesiveness, Sufficiency, and Coherence.

Encapsulation: A property of object-oriented programs in which an object’s implementation is hidden behind its interface.

Engineering: Creating cost-effective solutions to practical problems by applying scientific knowledge to building things of value.

Engineering Abstractions: Construction of elegant abstractions to increase the information/interface.

Enterprise Model: The complete model of a domain, including object structures and behaviors.

Entity (Organization): Any identifiable set of individuals, policies or systems.

Entities Model: Entities are the fundamental building blocks of the Domain Description that represent information stored in the system.

Factoring: Identifying common elements of two or more abstractions, typically decomposition followed by composition. An operation abstraction.

Fixed: A value that once set remains unchanging. A quality of an attribute. A measure of accessibility.

Framework: A collection of interdependent classes that provides a set of services for a particular domain; a framework thus exports a number of individual classes and mechanisms that clients can use or adapt.

Functional: See Functional Dependency under Dependency.

Generalization: Creating a more inclusive classification. Within an abstraction hierarchy, generalization results in “kind of” relationships.

Contrast with Specialization. An operation on an abstraction. There are three special cases of

generalization: **Leading Special Case:** easy to handle and very accessible in which it is seen that other cases follow;

Representative Special Case: is a specialization achieved by resolving some of the abstraction’s qualities in an arbitrary way; **Extreme Special Case:** sets boundaries for other cases.

Goal: motivation neither expressed nor implied by responsibilities. From notes, “factors that contribute to the choices and aspirations of the organization.”

Hierarchy: A directed tree of abstractions with transitive relationships between levels.

Identity: Designation of a component such as a name or phone number. An attribute quality.

Information: Processed data that conveys more than the data itself; relationships or descriptions of particular data.

Input: An operation quality. Any data that is required to carry out the operation.

Interaction: mutual or reciprocal action or influence between entities and/or systems.

Interface: Set of qualities of an object/entity that may be extracted or changed. Refers to that part of an object/entity which is accessible to others.

Law of Demeter: A hierarchy of operations with respect to where messages should be sent, e.g. first to itself.

Levels: Abstractions that can be classified together are considered to be at the same level. See Metalevel

Mapping: A constraint requiring the presence of a particular model element whenever another is present.

Mappings are most commonly used to express two-way relationships between objects.

Mechanism: An element of software that identifies or provides system-wide object behavior.

Metadata: An object which holds information that describes another object. For example, a recipe is metadata.

Meta Level: Abstractions that describe other abstractions.

Metric: Measures. E.g. elegance metrics such as cohesiveness, consistency etc.

Model: An organized collection of abstraction levels.

Object: An encapsulated packet of data and a behavior that acts as abstraction of a particular domain element or programming construct.

Operation: A task which is executed in response to the stimulus of an event. The functionality of a component. Involve data flow, control flow and state diagrams. Map to abstractions (see behavior).

Operations Engineering: Facilitates ways of organizing and managing complex operations, e.g. through assembling operations into hierarchies.

Operations Qualities: Trigger, Scenario, Preconditions, Postconditions, Inputs, Outputs, Actions, Exceptions.

Operations Classification: Organizing operations according to abstractions.

Output: An operation quality. Any data that is produced by the operation.

Policy: That portion of a given behavior that decided what the behavior should be doing. Contrast with algorithm.

Postcondition: An operation quality. The state of a system after an operation has been executed.

Precision: A measure of elegance, with regard to an object's defining quality. A defining quality is precise if it excludes everything that is not part of the abstraction.

Precondition: An operation quality. A prerequisite set of conditions that must be true in order for an operation to proceed,

Primitive Method: A method that directly accesses an instance variable.

Readability: Visibility of the value. All attributes should be readable. A quality of an attribute. A measure of accessibility

Reflexivity: Indicates whether a relationship can have the same object at both ends. Reflexive relationships can; irreflexive relationships cannot.

Relationship: A conceptual connection between two or more components or objects. A complex relationship is a composition of simple relationships and include bidirectional relationships (simple "one to many" relationships) and symmetric relationships (a relationship that has the same qualities when viewed from either direction (e.g. "next to")).

Relationship Constraints: Three of them (Reflexivity – relationships that have the same components as both the source and destination e.g. lawyers as own clients; Directed – limits the possible relationships between two components to one or more relationships e.g. selected from; Mappings – the existence of a relationship requires the existence of another – e.g. 'works for' implies 'employed by').

Relationship Types: 'Part of' relationship (the relationship objects within a component have to their container).

'**Contains**' relationship is the reverse of the 'part' relationship (deletion of the container deletes the parts).

'**Composite**' relationships, four of them: (Collections – e.g. address book; Aggregations – e.g. an automobile engine with its aggregation of parts; Groupings – like an aggregation but if you delete all the parts the group is deleted as well; Partnerships – where the deletion of a relationship between objects causes the container to be deleted)

Relevance: The degree to which a quality is important in the current context.

Responsibility: System responsibilities are a list of tasks the final system will be responsible for. Something to be done.

Reuse: Further use or repeated use of an artifact. Typically software programs that were designed for use outside their original context.

Scenario: An operation quality. A description of the steps taken to complete the operation.

Scope: The value of an attribute and whether or not another component of the same type can have the same value. A quality of an attribute.

Selector: A relational attribute which uniquely identifies an object in the context of the relationship to which it belongs.

Semantic: Semantic equivalence is one component simply representing the state of another component. E.g., bank account can be represented by open account.

Source Component: The component that originates the relationship. (See also Destination component).

Source: Specialization: Refining a classification by adding more qualities to it. Contrast with Generalization.

Specialization: Refining a classification by adding additional qualities to it (sub class). An operation on an abstraction.

Spiral: A model of a system development which repeatedly cycles from function to form, build, test, and back to function.

State: A combination of attributes and relationships that affects the behavior of an object and has well-defined transitions to or from other discreet states. E.g. solvency of a bank account. State may be represented by an attribute.

Subsystem: A model element which has the semantics of a package, such that it can contain other model elements, and a class, such that it has behavior. (The behavior of the subsystem is provided by classes or other subsystems it contains). A subsystem realizes one or more interfaces, which define the behavior it can perform.

Sufficiency: A measure of an abstraction's elegance. The degree to which all of an abstraction's information can be accessed in a reasonable amount of time, or with a reasonable amount of knowledge about the domain.

Super-type: Similar to generalization. Creating a less-specific, more inclusive class from a set of subclasses

System: As an instance, an executable configuration of a software application or software application family; the execution is done on a hardware platform. As a class, a particular software application or software application family that can be configured and installed on a hardware platform. In a general sense, an arbitrary system instance.

Test case: A set of test inputs, execution conditions, and expected results developed for particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. [NOTE: not the same definition as used in IEEE J-016.]

Test class: An optional (only used if designing and implementing test specific functionality) stereotype of Class in the design model.

Test model: A collection of test cases and test procedures and their relationship. A test case can be implemented by one or more test procedures, and a test procedure may implement (the whole or parts of) one or more test cases.

Test procedure: A set of detailed instructions for the set-up, execution, and evaluation of results for a given test case (or set of test cases).

Transitive (Transitivity): The relationship: If A then B, If B then C. Therefore if A then C.

Trigger: An operation quality. A set of conditions which, when true, cause an event to be sent to stimulate an operation..

Types: Types of components in the same class must share all qualities of other components in that class. E.g. names.

Value Class: A class that is used to represent an atomic value, such as a string or a number.

Waterfall: A development model based on a single sequence of steps