

Guidelines for Model-Based (System) Architecting and Software Engineering (MBASE)

Construction

- ❑ **Iteration Plan**
- ❑ **Iteration Assessment Report**
- ❑ **Release Description**
- ❑ **Quality Management Plan**
- ❑ **Test Plan**
- ❑ **Test Description and Results**
- ❑ **Inspection Plan**
- ❑ **Inspection Report**

Transition

- ❑ **Transition Plan**
- ❑ **User's Manual**

Support

- ❑ **Support Plan**

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. 1997-2000.

Version control

| Date | Author | Version | Changes made |
|-------------|---------------|----------------|---|
| 1/31/00 | Nikunj Mehta | 1.1.0 | Added some details to the LCP Approach section |
| 2/3/00 | Dan Port | 1.2.0 | <ul style="list-style-type: none"> • Elaborated and re-wrote parts of Invariants/Variants section • Added model classification per section guide • Added model classifications to OCD, SSRD, SSAD, LCP, FRD outlines • Elaborated on Architectural Views SSAD 3.1 • Added details to LCP |
| 2/14/00 | Barry Boehm | 1.3.0 | <ul style="list-style-type: none"> • Elaborated several LCP sections • Re-organized several LCP sections |
| 2/18/00 | Barry Boehm | 1.3.1 | <ul style="list-style-type: none"> • Further elaborations to LCP • BRA material added to LCP 4.3 • Need to re-sync LCP sections with outline TOC |
| 2/22/00 | Ebru Dincel | 1.4.0 | <ul style="list-style-type: none"> • Synchronization of Section numbers with the ones in TOC • Reference check • Took out course related material |
| 2/23/00 | Ebru Dincel | 1.4.1 | <ul style="list-style-type: none"> • Added some abbreviations |
| 2/24/00 | Ebru Dincel | 1.4.2 | <ul style="list-style-type: none"> • Corrected some typos • Appendix A of MBASE moved to OCD 5 |
| 2/26/00 | Nikunj Mehta | 1.4.3 | <ul style="list-style-type: none"> • Edits and corrections. • Changed link section under LCP Approach to bullets |
| 2/29/00 | Ebru Dincel | 1.4.5 | <ul style="list-style-type: none"> • Edits and corrections to LCP |
| 7/10/00 | Dan Port | 1.5 | <ul style="list-style-type: none"> • Updated OCD, added CTS |
| 8/1/00 | Dan Port | 1.5.1 | <ul style="list-style-type: none"> • Moved material after FRD to after CTS |
| 8/7/00 | Ebru Dincel | 1.5.2 | <ul style="list-style-type: none"> • LCP Rework, split participants into agents |
| 8/14/00 | Dan Port | 1.6 | <ul style="list-style-type: none"> • Updated OCD, reworked SSRD (removed User Classes, moved Typical Levels of Service table to OCD), reworked SSAD (sample specification templates, object model) |
| 8/20/00 | Ebru Dincel | 1.7 | <ul style="list-style-type: none"> • New OCD 2.0 |
| 8/28/00 | Dan Port | 1.7.1 | <ul style="list-style-type: none"> • Lots of OCD and SSRD revisions, added Degree of Detail and Tailoring sections |
| 8/31/00 | Dan Port | 1.8 | <ul style="list-style-type: none"> • More SSRD revisions, LCP and FRD revisions |
| 9/4/00 | Dan Port | 1.9 | <ul style="list-style-type: none"> • Added Course Guidelines to all sections |
| 9/5/00 | Ebru Dincel | 2.0 | <ul style="list-style-type: none"> • RUP related material |

General Guidelines

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

MBASE 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 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.
- A primary characteristic of the MBASE process is to be risk driven at all times (see MBASE invariant 5). Use this to help resolve tricky "how much should be specified" problems. Note that the assumption "more is better" and "It doesn't hurt to have extra" often may introduce added risks (such as confusion, presumptive specification, decreased coherence and cohesion, etc.). The risk principle may often be applied as follows:
 - If it's risky not to specify precisely, Do (e.g., a safety-critical hardware-software interface)
 - If it's risky to specify precisely, Don't (e.g., a GUI layout that can be easily evolved to match uncertain user needs with a GUI-builder)

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

577 Guidelines: indicate MBASE variants for CS577. In general, the MBASE active templates also contain recommended CS577 variants that may not be indicated in the guidelines (however they are typically applicable to general classes of large software projects),. however, guidelines should be tailored to the particular types and sizes of projects in the CS577 class on a case by case, risk driven basis.

RUP GL: indicate model variants compatible with the Rational Unified Process. Keep in mind that these may not always apply well to your particular project and the extent of you use RUP needs to be risk driven.

MBASE Variants and Invariants

MBASE systems engineering is intentionally very broad in order to encompass a broad spectrum of projects large and small. Within the MBASE superset, there are five elements, the model integration guidelines, that are universal for all MBASE projects. These elements are called *invariants*. Additionally, there are elements of MBASE, the process and method guidelines, which are categorized as *variants*, and can be adjusted according to particular project parameters (team size, application scope, etc.).

The Five MBASE Invariants

(1) Defining and sustaining a stakeholder win-win relationship throughout the system's life-cycle. 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.

- a. MBASE activity strives to create and organize a project that achieves stakeholder win-win objectives.
- b. System boundaries scope the project's authority and responsibility, and clarifies the win-win objectives.
- c. The life cycle provides appropriate scope to the project's duration; this scope is influenced by the project's authority and responsibility.

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

(2) Using the MBASE Model Integration Framework. As shown at the bottom of Figure 1, a cost and schedule property model would be used to evaluate and analyze the consistency of system models. In other cases, the success model might make a process product model the primary driver for model integration. An IKIWISI ("I'll know it when I see it") success model would initially establish a prototyping and evolutionary development process model, with most of the product features and property levels left TBD by the process. A success model focused on developing a product line of similar products would initially focus on product models (domain models, product line architectures), with process models and property models subsequently explored to perform a business-case analysis of the most appropriate breadth of the product line and the timing for introducing individual products.

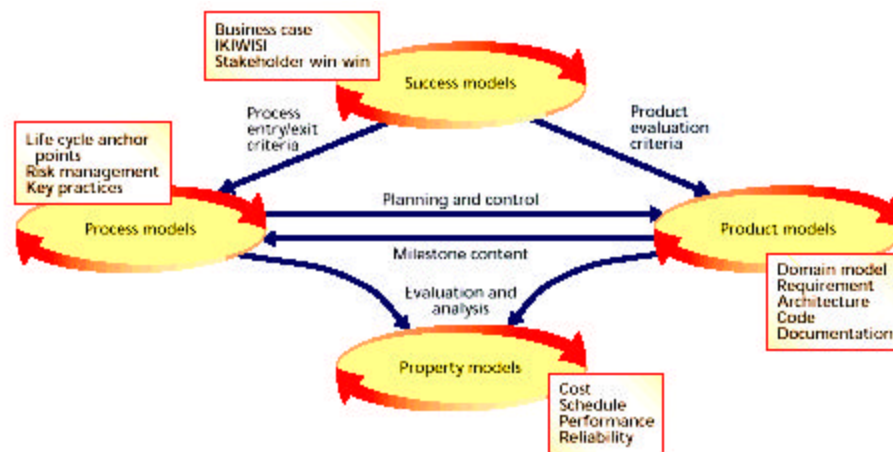


Figure 1.

(3) Using the MBASE Process Integration Framework.

Figure 2 shows the process framework within which stakeholders express their initial desired success models, and proceed to adjust these and their associated product, process and property models to achieve a consistent and feasible model set to guide the project and its stakeholders. The actual process generally takes several iterations, and requires intermediate checkpoints. Figure 3 shows the MBASE extension of the original spiral model to include stakeholder win-win model negotiation and common anchor point milestones: key life-cycle decision points at which a project verifies that it has feasible objectives (LCO); a feasible life-cycle architecture and plan (LCA); and a product ready for operational use (IOC).

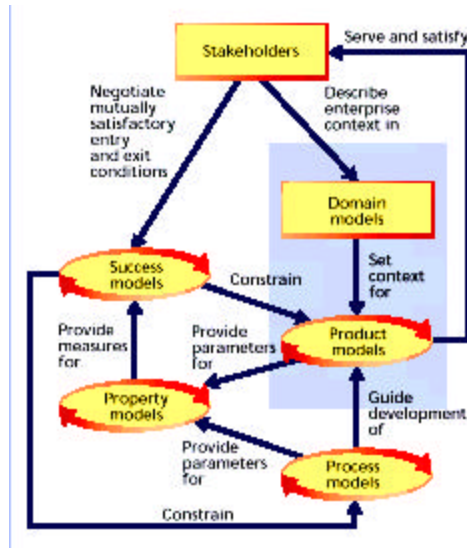


Figure 2.

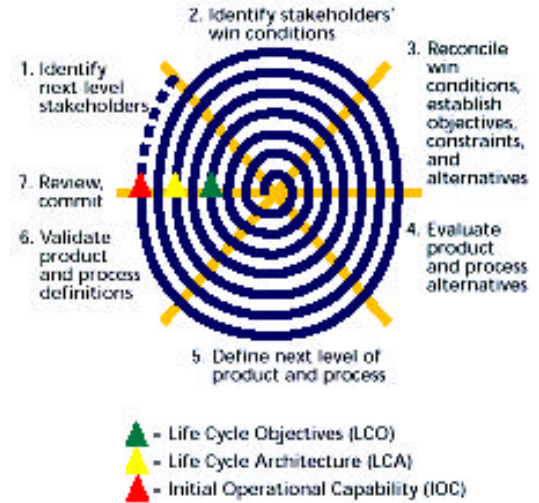


Figure 3.

These charts 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.

(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 user-less 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 Fourth-Generation 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.

(5) Ensuring that the content of MBASE artifacts and activities is risk-driven.

Use a risk-driven model in order to achieve critical success conditions and to avoid wasting effort. 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 pre-specified tax tables may not incur risk by not prototyping.

MBASE Variants

- (1) Use of particular success, process, product, or property models. The Process Model Decision Table is a good example of such variation, and a good objective for deriving product and property model variants.
It guides the project team toward selecting the specific model that works best for that project.
- (2) Choice of process or product representation. Thus, for example, UML may be appropriate for many applications, but not for a small upgrade to a well-documented legacy system using structured analysis and design. 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 the 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 discussed above is a good example of such mapping. Thus, any MBASE effort and schedule distributions by phase and activity, 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 and LCA Anchor Point milestones. | 4. Number of spiral cycles or builds between anchor points. |
| 5. Ensuring that the content of MBASE | 5. Mapping of activities onto Inception-Elaboration-Construction-Transition |

| | |
|--|--|
| artifacts and activities is risk-driven. | phases. 6. Mapping of staff levels onto activities. |
|--|--|

MBASE Model Classifications

| Success Models (SS) | Property Models (PY) | Process Models (PS) | Product Models (PD) |
|--|--|----------------------------------|--|
| OCD 3.2 Organization Goals | OCD 2.2 Key Stakeholders | LCP 2. Milestones and Products | OCD 2.3 System Boundary and Environment |
| OCD 3.7 Current System Shortfalls | OCD 2.4 Major Project Constraints | LCP 3. Responsibilities | OCD 3.1 Organization Background |
| OCD 4.2 Project Goals and Constraints | OCD 4.4 Levels of Service | LCP 4. Approach | OCD 3.3 Current Organization Activity Model |
| | OCD 4.7.3 Operational Policies and Constraints | LCP 5.1 Work Breakdown Structure | OCD 3.4 Description of Current System |
| OCD 4.6 Redressal of Current System Shortfalls | SSRD 2. Project Requirements | FRD 3. Process Rationale | OCD 3.5 Current Entity Model |
| OCD 4.7.3 Operational Impacts | SSRD 5. Level of Service Requirements | | OCD 3.6 Current Interaction Model |
| OCD 4.7.4 Organizational Impacts | LCP 5.2 Budgets | | OCD 3.3 Current Organization Activity Model |
| | FRD 2.1 Business Case Analysis | | OCD 4.2 Capabilities |
| OCD 2.1 System Capability Description | FRD 4. Project Risk Assessment | | OCD 4.5 Proposed System Description |
| FRD 2.3 Stakeholder Concurrence | FRD 5. Analysis Results | | OCD 4.7.1 Operational Stakeholders |
| | | | OCD 4.7.2 Organizational Relationships |
| | | | OCD 6. Common Definition Language for Domain Description |
| | | | SSRD 3.1 System Definition |
| | | | SSRD 3.2 System Requirements |
| | | | SSRD 4 System Interface Requirements |
| | | | SSRD 6. Evolution Requirements |
| | | | SSRD 7. Common Definition Language for Requirements |
| | | | SSAD 2. Architectural Analysis |
| | | | SSAD 3. System Design |
| | | | SSAD 4. Common Definition Language for System Design |
| | | | FRD 2.2 Requirements Satisfaction |

- Construction Transition and Support (CTS)

Construction

- ❑ Iteration Plan
- ❑ Iteration Assessment Report
- ❑ Release Description
- ❑ Quality Management Plan
- ❑ Test Plan
- ❑ Test Description and Results
- ❑ Inspection Plan
- ❑ Inspection Report

Transition

- ❑ Transition Plan
- ❑ User's Manual

Support

- ❑ Support Plan

General Construction Process Guidelines

- The process for the Construction and Transition Phase should be risk-driven: in particular, you should follow process strategies, to accommodate your particular requirements. For instance, if you have stringent performance requirements, you should plan accordingly for some of the following process strategies, such as Benchmarking, Modeling, Performance Analysis, Prototyping, Simulation, Code Profiling, Tuning and Optimization (Reference Table 8 in Hoh In's Dissertation)
- It is critical to keep all the artifacts properly baselined. In particular, at the end of each iteration, the Operational Concept Description (OCD), System and Software Requirements Definition (SSRD), System and Software Architecture Description (SSAD), Life Cycle Plan (LCP), Feasibility Rationale Description (FRD) must be consistent with the IOC plans and implementation (e.g. source code comments, component and object names, etc.) documentation. This is consistent with the concept of "continuous integration" aspect of iterative and incremental application development.
- As part of making winners of all the success critical stakeholders, it is recommended that your clients assess and evaluate each one of the intermediary or incremental releases, to avoid any changes introduced late in the process, which might introduce schedule slippage--something you want to avoid in a design-to-schedule situation
- Reference information where applicable, as opposed to repeating information. In particular, you should provide references to the information, when the reader will have to look at multiple documents to get hold of the information. In particular, it is recommended to use hyperlinks with traceability matrices that reference specific related areas in the documentation.
- Although the purpose of concurrent engineering is to having coding and testing proceeding in parallel, it is advisable to have a functional freeze at some point during the iteration. Ideally, the release at the end of the iteration should have thoroughly tested the features implemented in that increment. If the programmers don't stop adding features at some point before the end of the

iteration, the added features may not be thoroughly tested, and furthermore, may compromise the quality or correctness of the current feature set, leading to an unusable increment.

CS577b Guidelines:

- Rose Model Files should also be kept up-to-date. The code generation as well as the round-trip engineering capabilities of Rose (Java, C++, ...) should be used, where applicable.
- Ideally, during the rebaselining of the LCA packages, you should set your own dates for performing inspections: make sure that you turn in the required deliverables by the date indicated on the class schedule.
- Make sure that your project plan also identifies which increments are important to be evaluated by the customer and the users. It is important that the client periodically reviews the software as it is being developed, in particular, regarding user interface considerations. It is very important, due to the short schedule, to minimize rework, by avoiding making assumptions: when in doubt, refer to your customer. We recommend that most teams adopt the practice of delivering intermediate working increments to the clients, and keep incorporating the feedback.

During Construction, you will be performing the following activities:

- Requirements Management
- Detailed Design
- Coding
- Unit and Integration Testing
- Inspections
- Configuration Management
- Quality Assurance

You will be generating the following artifacts:

- Iteration Plan
- Iteration Assessment Report
- Release Descriptions and Notes
- Test Plans and Results
- Inspection Plans and Reports
- Quality Management Plans

Requirements Management

Changes in the requirements will be documented, as appropriate, in the Operational Concept Description, and the System and Software Requirements. Subsequently this may affect the architecture in the SSAD, impact the schedule and appear in the LCP, for which the FRD will then need to be updated. For instance, some of the changes might be moved to the Changes Considered but not Included (FRD 5.1.4) or Evolutionary Requirements (SSRD 6.). Accordingly, the Feasibility Rationale Description should be updated to reflect the impacts of the changes on the feasibility criterion of the project (e.g. is the value of the revised system greater than the cost?).

Design Creating or Modification

During Construction, a lot of effort will be spent on developing, detailing or changing the system design. The design activities are reflected in the System and Software Architecture Description, and the associated model files (e.g., the Rational Rose model). Low-level design details should also be included as comments in the source code and be consistent with the SSAD (especially naming). Another related activity would be to review the design and implementation (includes design meetings and consultations, as well as formal and informal reviews, walkthroughs, and inspections) and generating accordingly Inspection Reports.

Since there is no separate Detailed Design artifact, the Detailed Design information is documented in the following 3 artifacts:

1. SSAD
2. Rose MDL Files
3. Source Code (comments always related back to SSAD)

CS577b Guidelines:

You should try to strike a good balance as to what goes in the SSAD, v/s what goes in the MDL files, v/s what goes in the source code. Because having many objects/operations without tool support can lead to an unwieldy SSAD, you may want to leave very detailed information (e.g., method argument types, return values, ...) in the Rose MDL file. Once you have the information in the Rose model file, you can generate a report out of it (e.g., using SoDA), and include that information in the SSAD, as Operation Specification Templates, and so forth.

At any point in time, you should make sure that there are no glaring problems, such as having your architecture/design as represented in the MDL file conflict what is represented in the SSAD (e.g., block diagram), or with how the system was built.

Code Generation/Modification

During construction, most of the effort will be spent on actually coding the system. During coding, care should be taken to follow proper coding standards and programming style, emphasizing on code readability and maintainability, including the proper use of comments in the source code. An associated activity will consist of code reviews and inspections, where you will be inspecting code for defects, and generating Inspection Reports. WARNING: risk managed scheduling and resource allocation as well as careful and regular assessment and control are essential for a successful outcome.

Some related activities during the Construction stage will include creating or modifying prototypes, assessing various Commercial Off The Shelf (COTS) components for the application, tailoring COTS products, and integrating COTS products into application including glue code design, development and test.

Testing

Testing is an integral part of the Construction stage. This includes testing individual components of the system, writing test drivers, simulations, and gages, generating regression test packages, writing test descriptions, matching with requirements scenarios and reporting test results.

Project Management and Special Functions

Throughout the project, you will be performing planning and control activities, such as creating or modifying plans, reporting status, collecting and analyzing metrics, managing or coordinating work (configuration management, quality control). In particular, the Project Manager will be generating Iteration Plans and Iteration Assessment Reports.

Configuration Management and Quality Assurance: Hours spent performing configuration management and quality assurance functions, including developing Quality Management Plan, Inspection Plan, coordinating tools and the like must be planned and accounted for on the construction schedule.

In preparation for, and during the Transition Phase, you will be performing several activities, such as developing and executing the transition plan, coordinating deliverables with the client, meeting with key personnel for transition strategy and readiness discussions. You will also be training the users on the application, developing training material, as well as developing user documentation (e.g., user's manual and online help). Finally, you will need to spend some time coordinating, preparing and packaging customer deliverables for delivery (source code files, installation scripts, maintenance package, regression test package, support tools and environment, etc.). Most importantly documenting how the system must be supported and will support anticipated evolutionary changes.

Guidelines for the Deliverables

The artifacts of the process grouped in "logical" sets. These groupings do not imply physical document groupings. They indicate what areas are the main focus within a particular phase.

Requirements, Architecture, Design and Management Set

- Operational Concept Description (OCD)

- System and Software Requirements Definition (SSRD)
- System and Software Architecture Description (SSAD) and Rose Model Files (MDL)
- Feasibility Rationale Description (FRD)
- Life Cycle Plan (LCP). Must include effort/cost estimates such as:
 - COCOMO II run
 - COCOMO II Data Collection Form (as an Appendix) as a rationale capture for the COCOMO Estimate
- Risk-Driven Prototype(s)

Construction Planning Set

- Life Cycle Plan (LCP)
- Quality Management Plan (including guidelines for Configuration Management, Testing and Inspections)
- Inspection Plan
- Test Plan

Status Assessment Set

- Weekly Effort Forms
- Weekly Status Reports

Construction Working Set

One Construction Set is delivered at the end of each iteration.

- ❖ Documentation
 - As-built specs
 - As-built Operational Concept Description (OCD)
 - As-built System and Software Requirements Definition (SSRD)
 - As-built System and Software Architecture Description (SSAD)
 - As-built Rose Model Files (MDL)
 - As-built Feasibility Rationale Description (FRD)
 - Updated Risk Management Plans
 - *Summary of the revisions*
 - Iteration Plans (*one per iteration*)
 - Inspection Reports (*at least 1 inspection report per iteration*)
 - Test Reports (*at least 1 test report per iteration*)
 - Release Description (*one per iteration*)
 - Iteration Assessment Reports (*one per iteration*)
- ❖ Implementation
 - Source Code Baselines (including comments in the source files and “Read Me” files)

- Associated Compile-Time Files
- Component Executables
- Test drivers and simulations

Transition Set

- Transition Plan (including some Training planning)
- User Manual
- Transition readiness assessment

Support Set

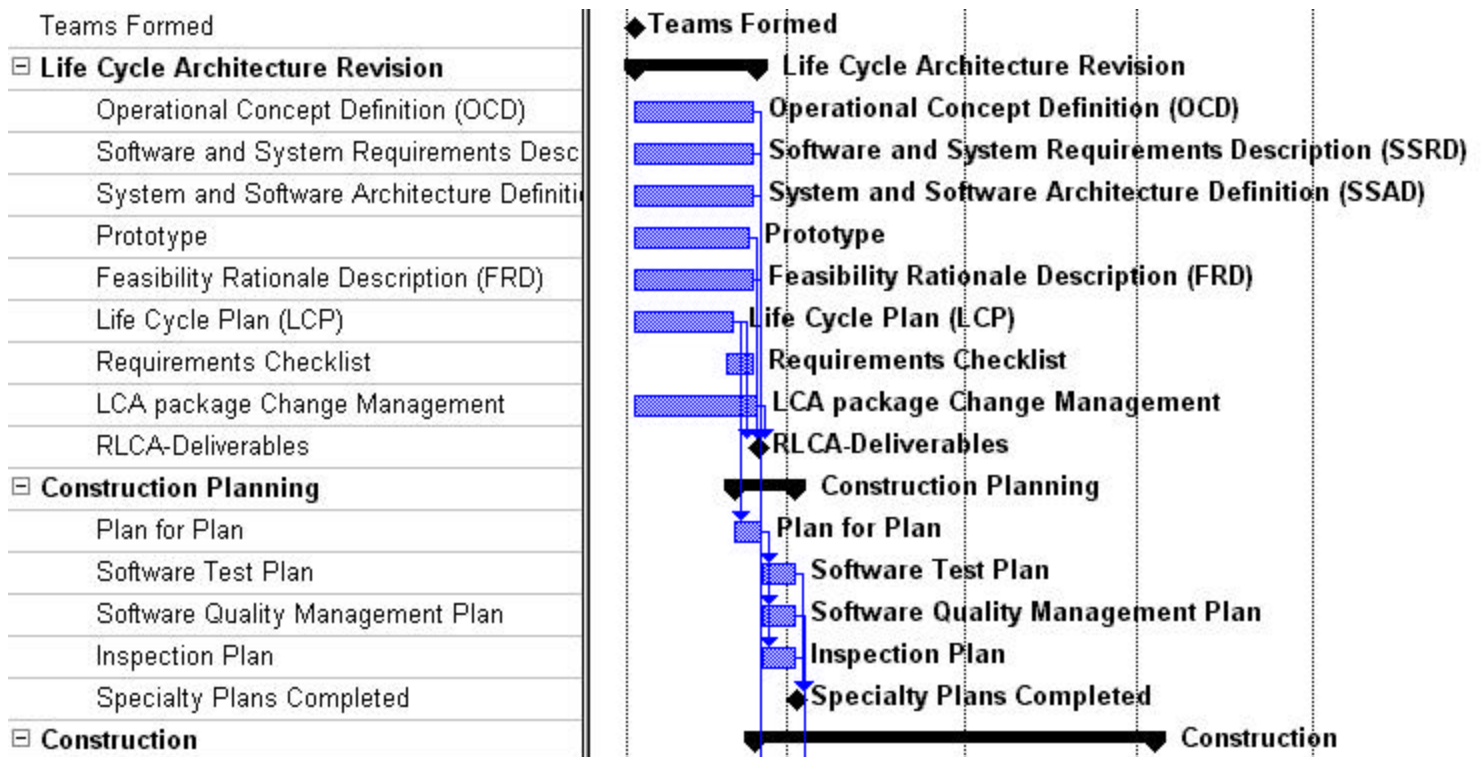
- Support Plan (including evolution support plan)
- Training materials (including tutorials and sample data)
- Regression Test Package
- Packaged Tools and Procedures

Data Collection Set

- Size Report (including Source Lines Of Code (SLOC) estimates) (*one size report per iteration*)
- Other data collection items such as:
 - COCOMO Data Collection Form (*including actuals*)
 - COCOTS Data Collection Form

CS577b Guidelines:

All deliverables should be properly stored in the CS 577 Archive, in accordance with the course guidelines. Below is a sample CS 577b start of construction schedule:



General Guidelines for Plans and Reports

The following guidelines for plans and reports are very general. As such some items and activities they describe may not be apply to the particular project at hand. In some cases items will need to be added. The choice as to what to include and at what level of detail should be risk driven in accordance with achieving high system assurance and effective development team communication given the constraints of the project. Below are some questions that may be helpful in determining the appropriate items to include and their respective level of detail:

- What should be documented to indicate that the correct system would be constructed?
- What should be documented to indicate that the system would be constructed correctly?
- Are the plans and processes helping to guide and control the construction or do they hinder it?
- Are all the development team members being utilized effectively?
- Do the plans address all the significant construction issues (in accordance with the FRD feasibility analysis)?

Try to keep plans and reports short, tightly focused, and as concise as possible. Keep in mind that the audience is generally developers who are likely already familiar with the system concept and as such extended explanations, justifications, and so forth are unnecessary. Be direct, brief, and clear as possible about what is asked for or being reported. Consider the use of tables, diagrams, bullet lists and so forth over large blocks of text.

The following table presented within the “High-Level Dependencies” will indicate the general level of integration a particular plan or report has with LCO/LCA MBASE deliverables:

| OCD | SSRD | SSAD | LCP | FRD | Prototype |
|-----|------|------|-----|-----|-----------|
| X | X | X | X | X | X |

Where X is one of:

- $\bar{0}$ = direct integration
- + = strong influence
- ~ = moderate integration
- - = indirect integration

Iteration Plan

Purpose

Overall the purpose of the iteration plans are to detail the incremental implementation and control of the SSRD requirements following the designs within the SSAD according to the schedule and approach specified in the LCP. The Iteration Plan for an upcoming iteration is planned in the current iteration. It is modified as needed during the iteration. The current iteration plan is an input to the next iteration plan. There are often two such plans: one for the current iteration, and one under construction for the next iteration. An iteration plan is realized and frozen after the scheduled iteration time is exhausted. The next iteration plan is then executed and an iteration assessment report is generated for the previous iteration. The Iteration Plan corresponds to the 'Establish next level objectives, constraints and alternatives' in the WinWin spiral model.

Timing

Intended Audience

The purpose of the Iteration Plan is to keep the construction on track and focused on realizing the SSAD designs. All the project stakeholders should be familiar with the Iteration Plan, and in particular, the development team.

Participants

Responsibility

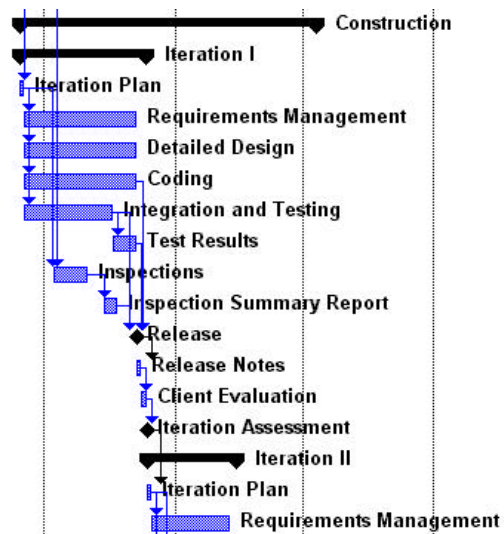
The Project Manager is responsible for authoring the Iteration Plan and keeping it up-to-date.

Completion Criteria Additional Information

CS577b Guidelines:

The following is a sample schedule for iteration activities within 577b

| |
|---------------------------|
| Construction |
| Iteration I |
| Iteration Plan |
| Requirements Management |
| Detailed Design |
| Coding |
| Integration and Testing |
| Test Results |
| Inspections |
| Inspection Summary Report |
| Release |
| Release Notes |
| Client Evaluation |
| Iteration Assessment |
| Iteration II |
| Iteration Plan |
| Requirements Management |



High-Level Dependencies

The Iteration Plan requires the following as inputs:

- Life Cycle Plan for the overall milestones to be achieved for each iteration (i.e. schedule estimates, dependencies, etc.)
- Life Cycle Plan and Feasibility Rationale for the identification and assessment of the risks and the risk management strategy to be implemented during each iteration
- System and Software Requirements Definition (SSRD) for the list of requirements that must be completed
- Current status of the project (as represented by the set of Weekly Status Reports) to-do's, unrealized tasks from previous iterations.
- Current Test Reports and Inspections Reports for a summary of the defects that must be removed prior to next release.

| OCD | SSRD | SSAD | LCP | FRD | Prototype |
|-----|------|------|-----|-----|-----------|
| - | + | + | 0 | + | + |

Outline

1. Iteration Overview

Provide a high-level overview of the content of the given iteration. Indicate which LCP milestones will be addressed.

1.1 Capabilities to be Implemented

- Identify the features, requirements or use cases that are being developed (implemented, tested, ...) for this iteration.
- Provide reference to particular System Capabilities (OCD 3.), System Requirement (SSRD 3.2), Level of Service Requirement (SSRD 5.)
- Provide also reference to the various artifacts (non-source code) that will be developed during the iteration. E.g. COTS configuration

Additional Guidelines

Each component should be accounted for in at least one iteration. All requirements should be implemented and tested (or re-negotiated) by the completion of all the iterations. Be mindful of implementation dependencies. Document complex dependencies and communicate them to the appropriate development staff.

1.2 Capabilities to be Tested

- Identify the software features and combinations of software features to be tested this iteration. This may also include non-functional requirements or extra-functional requirements, such as performance, portability, and so forth.

Every requirement listed in the SSRD LCA package, should be tested:

- Project requirements (SSRD 2.)
- System requirements (SSRD 3.)
- Interface requirements (SSRD 4.)
- Level of Service requirements (SSRD 5.)
- Evolutionary requirements (SSRD 6.)

Additionally you may need to test non-requirement component features such as COTS capabilities and quality, API functionality, etc.

1.3 Capabilities not to be tested

Identify notable features, and significant combinations of features, which will not be tested this iteration and why (e.g. a given feature uses a feature which will be implemented in following iteration).

1.4 Objectives

State measurable goals to be achieved during the iteration in terms of items such as:

- Implemented use cases
- Defect density
- Successfully executed test cases
- Risks Addressed
- Performance Levels
- Functionality
- Capacity

Describe specific, measurable, achievable, relevant and time-limited (“SMART” objectives that can be demonstrated (for instance within an in Iteration Assessment Report, or to a Review Board) with this iteration. It is acceptable to specify both desirable as well as acceptable levels. The time-limited aspect should be emphasized to ensure the objectives are realized, e.g. “at least ten test cases will be executed per day.”

2. Plan

This is the core section of the Iteration Plan. It is important to keep the Plan up-to-date during a given iteration. Thus, this section should be written so that it is very easily modified and updated. Be sure to keep careful version control.

Tool Support

We recommend the use of a project management tool, such as Microsoft Project, which provides ways of describing timelines, milestones and resources.

2.1 Schedule of Activities

Provide detailed diagrams showing timelines, intermediate milestones, when testing starts, beta version, demos etc. for the iteration. This should details major milestones indicated on the lifecycle schedule within LCP x.y

2.2 Resources

- Describe the Resources needed for this iteration – human, financial, etc.
- Highlight the resources that are on the critical path.
- Describe constraints or dependencies on resources during this iteration.

2.3 Team Responsibilities

- Provide detailed team responsibilities, covering the possible range of activities for this particular iteration.

3. Approach

Describe the general approach to be followed for this iteration. Note any special need, constraints, or opportunities to be leveraged during this iteration. Provide references to specific relevant items within various plans (do not repeat them here):

- **Risk Management plans (e.g., from LCP x.y and FRD x.y)**
- **Quality Management Plan from CTS identifying quality assurance strategies**
- **Test Plan identifying the test strategy for this particular iteration.**

4. Assumptions

Describe briefly the specific (significant) assumptions, under which this plan will hold: i.e., if those assumptions were no longer satisfied, the Iteration Plan would have to be revisited.

Iteration Assessment Report

Purpose

An iteration is concluded by an iteration assessment, where the actual results of construction actively are assessed in the light of the evaluation criteria that were established within the iteration plan. Iteration Assessments are not updated, but should be maintained for future reference. One aspect of the Iteration Assessment Report is to come up with "Lessons Learned", which corresponds to the 'Evaluate Product and Process alternatives' in the WinWin Spiral Model.

Timing

Intended Audience

Participants

Responsibility

The Project Manager is responsible for the Iteration Assessment. However, all the development team contributes to the content of the Iteration Assessment Report.

Completion Criteria

Additional Information

This assessment is a critical step in an iteration and should not be skipped. If iteration assessment is not done properly, many of the benefits of an iterative approach will be lost. Note that sometimes the right thing to do in this step is to revise the evaluation criteria rather than reworking the system. Sometimes the benefit of the iteration is in revealing that a particular requirement is not important, too expensive to implement, or creates an unmaintainable architecture. In these cases, a cost/benefit analysis must be done and a business decision must be made. Sound metrics should be used as the basis of this assessment.

CS577b Guidelines:

High-Level Dependencies

Outline

1. Overview

1.1 Capabilities Implemented

- List the features, use cases and scenarios (from OCD x.y), and their respective requirements (SSRD x.y), components and objects that were actually implemented.
- Indicate divergence from items planned to be implemented within 1.1 of the Iteration Plan.

1.2 Summary of Test Results

- Provide an overall assessment of the system as demonstrated by the test results
- Summarize the evaluation of the test items
- Report any variances of the test items from their design specifications
- Indicate any variances from the test plan, test designs, or test procedures. Specify the reason for each variance.
- Evaluate the comprehensiveness of the testing process against the comprehensiveness criteria specified in the test plan if the plan exists.

- Identify the features or combinations which were not sufficiently tested and highlight those as needing further testing in the following iteration
- Identify all resolved incidents and summarize their resolutions
- Identify all unresolved incidents.

1.3 Open Problems

- Identify any remaining (open) deficiencies, limitations, or constraints that were detected by the testing performed. Problem/change reports may be used to provide deficiency information.. For each remaining (open) deficiency, limitation, or constraint, describe the following:
Its impact on system performance, including identification of requirements not met
The impact on system design to correct it
A recommended solution/approach for correcting it

1.4 Objectives Reached

- Assess the results of the iteration relative to the evaluation criteria that were established for 1.4 Objectives within the Iteration Plan.

2. Adherence to Plan

Describe how well the iteration ran according to plan. Was it on budget and on time? Provide some insight to avoid mistakes for future iterations.

3. Approach Used and Suggested Changes

With respect to Iteration Plan 3.0:

- Summarize the major activities and events: resource consumption, total staffing level, ...
- Evaluate any improvements in the approach that should be incorporated into the following iteration.
- Provide suggestions for improvements to the environment: tools , resources, etc...

4. External Changes Occurred

- Describe any changes that have occurred with respect to the original assumptions in Iteration Plan 4.0: e.g., changes in requirements, new user needs, competitor's plan, discovery of a more efficient algorithm, ...
- Provide indications on the amount of rework required for the following iteration

5. Suggested Actions

- State any actions suggested due to unexpected results of the analysis.
- Provide any recommended improvements in the design, operation, or testing of the system tested. For each recommendation, describe the impact on the system.

Release Description

Purpose

The purpose of the Release Description is to describe items, particularly executables that will be made available after the completion of a development incremental plan. A Release Description is prepared right at the end of an iteration. In particular, the final Release Description before the Product Release is the most critical one, as it details the system outcome and aids in transition. Release descriptions are good candidates for “Read Me” files.

Timing

Intended Audience

The intended audience of a Release Description consists of the development team, as well as the customers. Since it is recommended to have the users and customer evaluate each one of the releases, the Release Description will serve to manage their expectations and smooth the evaluation by noting and anticipating possible problem areas or missed expectations (e.g. don’t use feature X as it presently non functional)

Participants

The developers, the testers and the quality management workers, contribute to the Release Description.

Responsibility

Completion Criteria

Additional Information

CS577b Guidelines:

High-Level Dependencies

| OCD | SSRD | SSAD | LCP | FRD | Prototype |
|-----|------|------|-----|-----|-----------|
| - | ~ | + | ~ | - | - |

Outline

1. About This Release

Provide version information, what the release consists of, documentation, licensing, etc. Include the following information, as applicable.

1.1 Physical Inventory of materials released

List all the physical media, documentation, and hardware that make up the software version being released by identifying numbers, titles, abbreviations, dates, version numbers, and release numbers as applicable. Include privacy considerations and restrictions regarding duplication and license provisions.

1.2 Inventory of software contents

- List all computer files that make up the software version being released by identifying numbers, titles, abbreviations, dates, version numbers, and release numbers as applicable.

- List the number, title, revision and date of all documents pertinent to this software. This should include applicable requirements, SSRD, design, SSAD, test (CDC Test Description and Results) and user documents.

2. Compatibility Notes

- Describe software (include version) that interact with the system and that are known to be compatible.
- Describe significant software that is known to be incompatible, and if there are any workarounds

3. Upgrading

- Describe installation, data conversion from information produced by earlier versions, etc.

New Features and Important Changes

Provide an accounting of the differences between this release and the previous (or what there is if this is the first release) for the following areas:

4.1 New Features

- List all the new features incorporated into the software since the previous version.

4.2 Changes since previous release

- List all changes incorporated into the software since the previous version.
- Identify as applicable the problem reports, inspection reports, test results and change notices for each change.

4.3 Upcoming Changes

- List all new features and changes that will be incorporated in future releases.

5. Known Bugs and Limitations

- Identify any possible problems or known errors with the software at the time of release, and instructions for recognizing, avoiding, correcting or handling each error (if known).

6. Defect and Change Request Reporting Procedures

- Provide information for reporting change requests, problems and defects with the current version
- Provide a point of contact to be consulted if there are problems or questions

7. Appendix

Use the appendix for information that does not fit directly in the body of the document.

Quality Management Plan

Purpose

The objective of construction is to follow a high quality process and deliver high quality products. Quality is elusive and poses particularly challenging issues when addressed only after the majority of a system has been implemented. As such it is difficult to achieve directly and a sound set of guidelines established prior to and followed during implementation can help achieve this indirectly and incrementally. It is also difficult to ensure that quality will be achieved as a matter of course, however the main concern is to avoid unspecified, haphazard or ad-hoc and often clashing approaches.

Intended Audience

Developers as well as the maintainers of the software system would use this plan. Organizational quality assurance personnel can also use the plan to assess the overall quality of the project.

Participants

Responsibility

Each project would identify suitable personnel to prepare the plan and implement it so that each team member can carry out their designated tasks for achieving required quality without significant additional efforts. The ideal person to perform quality management would be the project manager.

Completion Criteria Additional Information

CS 577b Guidelines:

For most projects, Quality Management Plan would be dictated by Organizational quality assurance mechanisms. For CS 577b, this document should help the project team members understand each person's contribution to quality. Each team has the flexibility to choose their own standards for quality management and should take the initiative in defining additional quality mechanisms as dictated by project requirements. It should be clear from the plan as to who is primarily responsible for the implementation of this plan.

High-Level Dependencies

- The has the following dependencies within other sections:

| OCD | SSRD | SSAD | LCP | FRD | Prototype |
|-----|------|------|-----|-----|-----------|
| - | 0 | + | ~ | ~ | - |

Outline

1. Purpose

1.1 Overview

This section should describe the purpose, scope and audience of this plan. Quality of software is the degree to which software components meet specified requirements (SSRD x.y) and user/customer operational concept (OCD x.y) expectations as well as provide clear value (FRD x.y) with respect to the cost and effort. The purpose of this plan is to ensure that there is adequate direction to achieve the goal of delivering quality software.

1.2 References

This document should refer to external quality management documents (such as process guidelines). References made to all documents along with their versions and numbers should be identified in this section.

2. Quality Guidelines

This section describes the guidelines for quality management. This section should be very brief and only cover those quality tasks that are significant and meaningful to the project.

2.1 Design Guidelines

Briefly describe design guidelines to improve or maintain modularity, reuse and maintenance, etc. In particular indicate how the designs in SSAD x.y will map to the implementation of those designs (e.g. how will the object models be translated into code)

2.2 Coding Guidelines

A team would probably choose to implement various components of its software system in various programming languages. It is necessary though to follow the same spirit of documentation and coding throughout the system for ease of understanding among all the team members and to ensure smooth transition to maintenance. The approach used should attempt to document the code in such a way that it could easily be communicated to an outside developer or new team member that understood the guidelines.

We are providing links to some industry standards for coding in some of the implementation languages:

C: http://www.gnu.org/prep/standards_toc.html

C++: <http://www.nfra.nl/~seg/cppStdDoc.html>

Java http://www.infospheres.caltech.edu/resources/code_standards/java_standard.html

Visual Basic <http://construxsoftware.com/carmac/DocumentationProject/vbcodestd.pdf>

A NASA web site gives comparative coding standards for C, C++ and Java.

http://v2ma09.gsfc.nasa.gov/coding_standards.html

It is not important which coding standard is chosen, but it is very important that the project identifies and complies with defined coding standards (for each language used). You can develop your own coding standards and provide them in the appendix of this document.

There should also be sufficient comments in the code to support maintenance of the software. Each module, class, interface and function should be described and related to the SSAD x.y designs.

The header of each source code file should contain the following information. This header should not replace in line code comments (at functions, methods, etc.) that explain non-trivial implementation points.

Version Control and History

Provide a chronological log of the changes introduced to this unit.

Implementation Considerations

Provide detailed design and implementation for **as-built** considerations. A description of how well this code implements an SSAD design (for example object defining qualities are a useful start for this). The SSAD design document may have to be updated to reflect any discrepancies.

Unit Verification

Provide links or references to any of the following, as applicable:

- Unit/integrated test descriptions
- Unit/integrated test results
- Code walkthrough/inspection results

Integration

How the component(s) this code implements fit within the application together with the tests used to verify each version

Additional Information

Include any other information that could be useful in understanding the software element.

The main purpose of the comments in the source code is to provide a trail of implementation decisions that aren't documented elsewhere. Ideally, most of the detailed design should be documented in the SSAD. However, this often isn't practical and hence in contrast to external documentation, internal documentation is found within the program listing itself. It's the most detailed kind of information, at the source-statement level. Because it's most closely associated with the code, internal documentation is also the kind of documentation most likely to remain current and correct as the code is modified. This often helps make code easier to maintain and evolve for present and future developers of the system.

Design and coding standards should promote self-documenting artifacts. Having source files in a single, homogeneous format consistent and integrated with the SSAD, will avoid having separate documents that inevitably diverge from the implementation.

Each source file should contain where further elaboration beyond the SSAD and SSRD (due to implementation issues) the following as applicable:

- DEVELOPER UPDATES & REVISIONS
- PROJECT UPDATES & REVISIONS
- BASIC ASSUMPTIONS & DEFINITIONS

- Having well-commented source files goes a long way towards improving the maintainability of the delivered systems.

2.3 Testing Guidelines

2.3.1 Testing Standards

- Identify and describe the major guidelines and standards to be used in the test-related activities (planning, design, execution and evaluation of tests on the software system):
 - **Test Case Standards:** the types of test cases that should be developed for testing, such as valid, invalid, boundary, etc.
 - **Test Naming Convention:** how each kind of entity (such as test case and test procedure) should be named.
 - **Test Design Guidelines:** how test procedures and, or scripts will be developed, e.g., with the underlying goals for modularity, for reuse and maintenance.
 - **Test Data Standards:** how data will be selected or created and restored to support testing.

2.3.2 Deliverables

- Identify the expected deliverable results from the various tests (e.g. test reports, problem reports, ...)

2.3.3 Tools

- Describe tools to be used or built for testing

2.4 Inspections and Reviews

<Winsor's and CMMI stuff here>

2.5 Configuration Management

Projects deal with many evolving items produced and used by many people sometimes involving complex dependencies. In particular during the project various versions of a software product are created. In order to avoid costly re-work, hidden defects, and deliver a software system consistent with the stage of development and traceable to the needs of the customer, configuration management is needed.

Configuration management involves the development and application of procedures and standards for managing the evolution of a software product.

2.5.1 Configuration Item and Rationale

Each project churns out a number of documents, but not all are required for continued development and system maintenance. Only some baselined documents are to be kept under configuration control. This section should provide the rationale behind selection of those artifacts that would be managed for changes. Artifacts suited for configuration control include plans, specifications, designs, code, tests, manuals, object code, defect reports and change requests. Artifacts that are bad choices for configuration control include test results, project tracking results, working documents and samples.

The fewer the number the better, but all artifacts that are prone to frequent changes, have many dependencies and/or affect project progress should be considered possible configuration management items (CIs). All CIs should be classified into categories for ease of management. Categories should be based on the typical rate of change, impact of changes and/or project structure. Typical categories include project subsystems, source code, objects, tests and documents. Categorization helps in proper organization of software artifacts and enables quicker updates and retrieval.

2.5.2 Identification System

It is essential to be able to unambiguously identify a particular version of a particular artifact. A naming scheme also allows easy retrieval of CI for maintenance and reuse. The naming scheme should be general enough to cover all CIs, flexible enough to allow new CIs to be added and simple enough to be navigated and understood. An example scheme consists of the project name followed by the subsystem name followed by the specific items name and version. E.g. HVM/VT/code/generator/1.0

Another example scheme is to use numbers and allocate ranges of numbers to each CI category. E.g. Items can be identified by a five-digit code followed by a two-digit version number. 102-06-10, perhaps involving the date and responsible party. Use of numbers is difficult for subsequent retrieval but is flexible and lends itself easily for organization.

2.5.3 Storage of Configuration Items

The storage location for each CI should be identified. A directory structure for the CI storage should be formulated based on the directory structure used for other development artifacts (e.g. other MBASE models) and specific disks/machines should be identified for storage. A mechanism to archive previous versions should be identified and a plan for backup of project artifacts should be defined. Specific personnel assigned for the tasks of backup and archival should be identified.

2.5.4 Configuration Control

Projects should identify the possible range of changes, and define a process or policy to carry out the change requests. Typically, change control requires:

- An understanding and analysis of the requested change.
- Impact and Feasibility of carrying out the change
- Authorization to carry out the change
- Implementation of the change
- Verification of the changes implemented
- Reconfiguration and baseline of software product.

The policy should address technical, management and quality issues in implementing the changes. Overall, the policy should identify steps for change analysis and change execution. Note that all the steps may not be required for every instance. It is sufficient to identify the steps for each category of possible change. Generally a configuration team conducts the analysis for a possible (complex) change, whereas the development/maintenance team carries out the change. A representation of the policy in the form of a workflow diagram is often useful.

2.5.5 Status and Accounting

Accounting activities record and report the status of project CIs. Specifically the following should be addressed:

- The CIs to be tracked and reported for changes
- The types of reports to be generated along with the frequency of generating them

At least the initial version, status of requested changes and the implementation of approved changes should be tracked. This can be reported in a simple table within a document/file, or with more sophisticated revision control tools such as CVS. Whatever is used should be clearly documented and used consistently and inspected regularly.

2.5.6 Baselining Events

Each document or software artifact is baselined at certain major project milestones. These milestones mark the overall consensus of the project's progress and satisfaction or achievement of certain quality goals. Baselining documents puts these under configuration control when the configuration personnel take over ownership of the artifact from the developer/author. This event should be defined for each CI, as it is critical in identification of responsibility in the event of a change. The milestones should match process anchor points.

2.5.7 Resources and Personnel

Identify the software and human resources required to perform configuration management (CM) functions. Designate specific project personnel to perform the tasks of CM and main project librarian or archivist.

2.5.8 Tools

Each of the tasks mentioned above may necessitate the use of tools for efficiency and effectiveness. Simple tools such as spreadsheets, databases, flat files, file names, groups, and permissions can be used. Specifically, identify the structure of worksheets and processing required to achieve the objectives set out in the previous sections. E.g. versions of an item can be maintained in the form of suffixed version numbers. Also a table describing the CIs can be maintained as a spreadsheet or in a table at the top of a document. The GNU foundation provides a standard configuration management tool CVS supported on several major platforms (see <http://www.wincvs.com> for example). Additionally many UNIX systems support SCCS. More information on this can be obtained using the command UNIX "man sccs." Use of flat files or directory structures can meet the quality management needs. Word documents with templates and spreadsheets can be useful to record change requests and defect records. Charts are helpful in analyzing the changes.

2.6 Defect and Change Management

Changes are a normal part of software development. It is required to have a plan in place for managing changes before they are actually required. Defect and changes often arise from several stakeholders. This section describes the practices and procedures for reporting, tracking and resolving the problems identified in inspection and testing and also in the maintenance process.

2.6.1 Reporting procedure

Identify the means by which problems and changes get reported. Problems and changes could be identified by the developers, test team or the customer. There should be a single interface for all stakeholders and by unifying the change reporting with defect reporting, the change traffic can be managed under a single umbrella.

Provide the structure and layout of the Change report form along with the process of communicating this information. It should be sufficient to meet the needs of the change control policy described in section 7.4. Document the rationale for each decision taken.

2.6.2 Tracking

The purpose of a problem and change tracking system is to ensure that these get adequately addressed within the scope of the project and at the same time serve as a means to evaluate the progress and quality of work in the project. It also provides the developer and customer feedback about the status of a problems resolution. There are many possible effective tracking systems and several commercial tracking systems. In particular the GNU foundation provides a popular free tracking system called GNATS (see <http://www.gnu.org>).

2.6.3 Resolution

Define the procedure whereby valid problems and changes are implemented in a time bound fashion or a justification is provided for why the changes cannot be carried out. The process should result in a WinWin for all involved stakeholders.

3 Appendix

- The appendix should list the vendor documents, user manuals and hyperlinks for tools to be used in quality management. It should also describe the layout and structure of the forms to be used for quality management.
- Change reporting form
- Inspection form

Also listed should be the coding standards defined previously in section 2.2.

Test Plan

Purpose

- To prescribe the scope, approach, resources, and schedule of testing activities. To identify the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with testing.
- To detail the activities required to prepare for and conduct the system test
- To communicate to all responsible parties the tasks which they are to perform, and the schedule to be followed in performing the tasks
- To define the sources of the information used to prepare the plan
- To define the test tools and environment needed to conduct the system test

Timing

An initial test plan is created during or just after LCA, then refined during each construction iteration based on need.

Intended Audience

Participants

Responsibility

The test designer is responsible for planning the test effort during the project and during each iteration.

Completion Criteria

Additional Information

CS 577b Guidelines:

High-Level Dependencies

- Life Cycle Plan
- Quality Management Plan
- System and Software Requirements Definition (SSRD)
- Configuration Management Plan
- System and Software Architecture Description (SSAD)
- Relevant organizational policies
- Relevant organizational standards

| OCD | SSRD | SSAD | LCP | FRD | Prototype |
|-----|------|------|-----|-----|-----------|
| - | 0 | 0 | 0 | - | + |

Outline

1. Introduction

- Provide the purpose, background, and scope of testing within this project.
- References to the following documents, when they exist, are required in the highest-level test plan (LCP 2., SSRD, SSAD)

2. Test Strategy

2.1 Approach

Describe the overall approach to testing. For each major group of features, specify the approach which will ensure that these feature groups are adequately tested. Specify the major activities, techniques and tools, which are used to test the designated features.

The approach should be described in sufficient detail to permit identification of the major testing tasks, their dependencies and estimation of the time required to do each one.

Examples include:

- Interface Testing (alpha, beta user testing)
- Performance Testing
- Security Testing
- Regression Testing

2.2 Deliverables

- Identify the deliverable documents. The following documents should always be included: test reports to use for the test work, test tools used.

2.3 Requirements Verification

Include a Requirements Verification Matrix specifying how each requirement from SSRD will be verified:

- Testing
- Analysis
- Simulation
- Inspection
- ...

3. Environment Preparation

Specify how to prepare test environment for testing.

3.1 Hardware preparation

Describe the procedures needed to prepare the hardware for the test, including support hardware (e.g., test equipment). Reference any operating manuals, if applicable.

Provide the following as applicable:

- a. The specific hardware to be used, identified by name and, if applicable, number
- b. Any particular settings and cabling needed to connect the hardware
- c. One or more diagrams to show hardware, interconnecting control, and data paths
- d. Step-by-step instructions for placing the hardware in a state of readiness for testing

3.2 Software preparation

Describe the procedures needed to prepare the software for the test, including support software (e.g., simulators, data recording/reduction software). Reference any software manuals, if applicable.

Provide the following as applicable:

- a. The specific software to be used, identified by name and, if applicable, version number
- b. The storage medium of the software (magnetic tape, diskette)
- c. Instructions for loading the software, including required sequence
- d. Instructions for software initialization common to more than one test case

3.3 Other pre-test preparations

Describe any other pre-test personnel actions, preparations, or procedures needed to perform the test not accounted for in 3.1 or 3.2.

4. Resources

Specify the people , time, budget , resources allocated for testing, etc..

4.1 Responsibilities

Identify the groups responsible for managing, designing, preparing, executing, witnessing, inspecting and resolving test items. In addition, provide the groups responsible for providing items to be tested.

4.2 Staffing and Training Needs

Specify test staffing needs by skill level. Identify training options for providing necessary skills.

4.3 Schedule for Testing Activities

Include test milestones. Estimate the time required to do each testing task, and a schedule for the testing activities.

4.4 Other resource allocations

5. Key Measures

Describe what kind of measures you will use to determine the progress of test activities (what type of defect counts are going to be used, how to measure successfully executed test cases).

6. Test Completion Criteria

A statement identifying recommended test completion and evaluation criteria. Overall, individual tests may have separate completion criteria.

7. Defect Management Guidelines

Statement identifying how defects will be identified, reported and handled.

8. Change Management Criteria

A statement identifying how test artifacts will be managed, maintained and controlled.

Test Description and Results

Purpose

- Detail the activities required to prepare for and conduct specific system tests and document their results.
- Communicate to all responsible parties the tasks which they are to perform, and the schedule to be followed in performing the tasks
 - Test procedure specification. A document specifying a sequence of actions for the execution of a test.
 - Tester. Identify the person who prepared the test description.
 - Test Preparation. Identify the test tools to be used; the names of files in which test cases and/or data reside; the hardware and software required for the test.
 - Test Initialization. Describe how to set up the conditions for the test; identify any flags, breakpoints, or data to be set/reset prior to starting the test.
 - Test Procedure.

Timing

Intended Audience

Participants

Responsibility

The test designer/manager is responsible for planning the test effort during the project and during each iteration.

Completion Criteria

Additional Information

CS 577b Guidelines:

High Level Dependencies

Test cases are obtained from the following sources:

- CDC Test Plan
- System Requirements (SSRD)
- Behavior Model (SSAD 2,3,5)

Outline

1. Test Identification

Describe the test items including their version/revision level. A test consists of a set of one or more test cases, or a set of one or more test procedures, or a set of one or more test cases and procedures.

2. Test Preparation

Provide an assessment of the manner in which the test environment may be different from the operational environment and the possible effects on the test results.

2.1 Hardware preparation

Describe the procedures needed to prepare the hardware for the test, including support hardware (e.g., test equipment). Reference any operating manuals, if applicable.

Provide the following as applicable:

- a. The specific hardware to be used, identified by name and, if applicable, number
- b. Any switch settings and cabling needed to connect the hardware
- c. One or more diagrams to show hardware, interconnecting control, and data paths
- d. Step-by-step instructions for placing the hardware in a state of readiness

2.2 Software preparation

Describe the procedures needed to prepare the software for the test, including support software (e.g., simulators, gages, data recording/reduction software). Reference any software manuals, if applicable.

Provide the following as applicable:

- a. The specific software to be used, identified by name and if applicable version number
- b. The storage medium of the software (magnetic tape, diskette)
- c. Instructions for loading the software, including required sequence
- d. Instructions for software initialization common to more than one test case

2.3 Other pre-test preparations

Describe any other pre-test personnel actions, preparations, or procedures needed to perform the test.

3. Test Case Specifications

A test case specification specifies inputs, predicted results, and a set of execution conditions for a test item. For each test case, create a sub-heading using the following structure:

Identify a test (one of the tests in the test set comprising the application testing addressed by this test description) by project-unique identifier and provide the information specified below for the test. The name includes the identification of the applicable unit.

Note: the “tests” in this paragraph are collections of test cases covering a specific area or function within the application test described by this test description.

3. Test Case x.

Each test case is assigned a project-unique identifier (x.)

3.1 Identifier

Specify the unique identifier assigned to this test-case specification.

3.2 Test Items

Identify and briefly describe the test items and features to be exercised by this test case. That may also include non-functional or extra-functional requirements, such as performance.

For each item, consider supplying references to the following documentation:

- (1) Requirements specification
- (2) Design specification
- (3) Users guide
- (4) Operations guide
- (5) Installation guide

3.3 Pre-conditions

Describe prerequisite conditions that must be established prior to performing the test case, such as flags, initial breakpoints, pointers, control parameters, or initial data to be set/reset prior to test commencement.

3.4 Post-conditions

Describe conditions that must be established after performing the test case such as resetting data, logging out, removing monitors, breakpoints, etc..

3.5 Input Specifications

Specify each input required to execute the test case. Some of the inputs may be specified by value (with tolerances or choices where appropriate), while others such as constant tables or transaction files, will be specified by name.

Describe the test input data and commands needed for the test case (Use items from Table 1 as applicable). This information can be included in the test procedure.

| |
|--|
| <ol style="list-style-type: none"> 1) Name, purpose, and description (e.g., range of values, accuracy) of each test input 2) Source of the test input and the method to be used for selecting the test input 3) Whether the test input is real or simulated 4) Time or event sequence of test input 5) The manner in which the input data will be controlled to: <ol style="list-style-type: none"> a) Test the item(s) with a minimum/reasonable number of data types and values b) Exercise the item(s) with a range of valid data types and values that test for overload, saturation, and other “worst case” effects c) Exercise the item(s) with invalid data types and values to test for appropriate handling of irregular inputs d) Permit retesting, if necessary |
|--|

Table 1: Test Inputs

3.6 Expected Output Specifications

Identify all expected test results for the test case, both intermediate and final test results, as applicable.

3.7 Pass/Fail Criteria

Identify the pass/fail criteria to be used for evaluating the intermediate and final results of this test case (Use items from Table 2 as applicable), i.e., the decision rules used to determine whether a software item or a software feature passes or fails a test.

| |
|---|
| <ol style="list-style-type: none"> a. The range or accuracy over which an output can vary and still be acceptable b. Minimum number of combinations or alternatives of input and output conditions that constitute an acceptable test result c. Maximum/minimum allowable test duration, in terms of time or number of events d. Maximum number of interrupts, halts, or other system breaks that may occur e. Allowable severity of processing errors f. Conditions under which the result is inconclusive and retesting is to be performed g. Conditions under which the outputs are to be interpreted as indicating irregularities in input test data, in the test database/data files, or in test procedures h. Allowable indications of the control, status, and results of the test and the readiness for the next test case (may be output of auxiliary test software) i. Additional criteria not mentioned above |
|---|

Table 2: Test Criteria

3.8 Test process

Define the test process for this test case. The test process is a series of individually numbered steps listed sequentially in the order in which the steps are to be performed. If the process is non-trivial, include a Procedure Template. You may reference a separate Test Procedure Specification (section 4.x) if the Procedure Specification applies to multiple Test Cases.

Test Procedure Template:

A test procedure provides detailed steps that carry out the test as defined by the associated test case(s). A sample test procedure template is shown below:

| Step No. | Step Description | Expected Result | Observed Result | Pass/Fail |
|----------|------------------|-----------------|-----------------|-----------|
| | | | | |

Provide the following for each test procedure, as applicable:

- a. Test operator actions and equipment operation required for each step, including commands, as applicable to:
 - 1) Initiate the test case and apply test inputs
 - 2) Inspect test conditions
 - 3) Perform interim evaluations of test results
 - 4) Record data
 - 5) Halt or interrupt the test case
 - 6) Request data dumps or other aids, if needed
 - 7) Modify the database/data files
 - 8) Repeat the test case, if unsuccessful
 - 9) Apply alternate modes as required by the test case
 - 10) Terminate the test case
- b. Test Inputs
- c. Expected result and evaluation criteria for each step
- d. If the test case addresses multiple requirements, identification of which test procedure step(s) address which requirements.
- e. Actions to follow in the event of a system stop or indicated error, such as:
 - 1) Recording of critical data from indicators for reference purposes
 - 2) Halting or pausing time-sensitive test-support software and test apparatus
 - 3) Collection of system and operator records of test results
- f. Procedures to be used to validate expected results, reduce and analyze test results to accomplish the following, as applicable:
 - 1) Detect whether an output has been produced
 - 2) Identify media and location of data produced by the test case
 - 3) Evaluate output as a basis for continuation of test sequence
 - 4) Evaluate test output against required output

3.9 Assumptions and constraints

Identify any assumptions made and constraints or limitations imposed in the description of this test case due to system or test conditions, such as limitations on timing, interfaces, equipment, personnel, database/data files.

3.10 Dependencies

List the identifiers of the test cases which must be executed prior to this test case or for that depend on the execution of this test. Summarize the nature of the dependencies.

3.11 Traceability

Include reference to:

- SSAD, OCD addressed
- Use Case/Scenario addressed

4. Test Procedure Specifications

In this section, specify the steps for executing a set of test cases or, more generally, the steps used to analyze a software item in order to evaluate a set of features.

For each test procedure, create a sub-heading using the following structure:

4. Test Procedure x.

Each test procedure is assigned a project-unique procedure section (x.)

4.1 Identifier

Specify the unique identifier assigned to this test-procedure specification. Include a description name such as TP-03 Edit User Profile.

4.2 Purpose

Describe the purpose of this procedure. If this procedure executes any test cases (section 3.x), provide a reference for each of them. In addition, provide references to relevant sections of the test item documentation (for example, references to software preparation, section 2.2).

4.3 Special Requirements

Identify any special requirements that are necessary for the execution of this procedure. These may include prerequisite procedures, special skills requirements, timing, and special environmental requirements.

4.4 Procedure Steps

Include the following steps as applicable:

Log

Describe any special methods or formats for logging the results of test execution, the incidents observed, and any other events pertinent to the test. You may decide to generate a Test Log (section 6.) and a Test Incident Report (section 5.).

Setup

Describe the sequence of actions necessary to prepare for the execution of the procedure

Start

Describe the actions necessary to begin execution of the procedure

Proceed

Describe any actions necessary during the execution of the procedure

Measure

Describe how test measurements will be made

Shutdown

Describe the actions necessary to suspend testing, when unscheduled events dictate them

Restart

Identify any procedural restart points and describe the actions necessary to restart the procedure at each of these points.

Stop

Describe the actions necessary to bring execution to an orderly halt

Wrap Up

Describe the actions necessary to restore the environment

Contingencies

Describe the actions necessary to deal with anomalous events which may occur during execution

5. Test Incident Reports

A test incident report is a document reporting on any event that occurs during the testing process which requires further investigation.

5.1 Identifier

Specify the unique identifier assigned to this test incident report.

5.2 Summary

Briefly summarize the incident. Identify the test items involved indicating their version/revision level. References to the appropriate test-procedure specification, test- case specification, and test log should be supplied.

5.3 Incident Description

Provide a detailed description of the incident. This description should include the following items:

- Inputs
- Expected results
- Actual results
- Anomalies
- Date and time
- Procedure step Environment
- Attempts to repeat
- Testers
- Observers

Related activities and observations that may help to isolate and correct the cause of the incident should be included. For example, describe any test-case executions that might have a bearing on this particular incident and any variations from the published test procedure.

5.4 Impact

If known, indicate what impact this incident will have on test plans, test- design specifications, test-procedure specifications, or test-case specifications.

6. Test Log

The purpose of the test logs is to provide a chronological record of relevant details about the execution of tests.

6.1 Test-Log Identifier

Specify the unique identifier and description name assigned to this test log.

6.2 Description

Information which applies to all entries in the log except as specifically noted in a log entry should be included here. The following information should be considered.

- 1) Identify the items being tested including their version/revision levels. For each of these items, supply a reference to its transmittal report, if it exists.

- 2) Identify the attributes of the environments in which the testing is conducted. Include facility identification, hardware being used (for example, amount of memory being model of tape drives, and/or mass storage de- vices), system software used, and resources available such as the amount of memory available.

6.3 Activity and Event Entries

For each event, including the beginning and end of activities, record the occurrence date and time along with the identity of the author. The following information should be considered:

Execution Description

Record the identifier of the test procedure being executed and supply a reference to its specification. Record all personnel present during the execution including testers, operators, and observers. Also indicate the function of each individual.

Procedure Results

For each execution, record the visually observable results (for example, error messages generated, aborts, and requests for operator action). Also record the location of any output (for example, reel number). Record the successful or unsuccessful execution of the test.

Environmental Information

Record any environmental conditions specific to this entry (for example, hardware substitutions).

Anomalous Events

Record what happened before and after an unexpected event occurred. For example, “A summary display was requested and the correct screen displayed, but response seemed unusually long. A repetition produced the same prolonged response”. Record circumstances surrounding the inability to begin execution of a test procedure or failure to complete a test procedure. For example, a power failure or system software problem.

Incident-Report Identifiers

Record the identifier of each test-incident report, whenever one is generated.

7. Test Summary

Summarize the results of the designated testing activities post testing

7.1 Summary

Summarize the overall evaluation of the test items covered by this test plan, including its limitations based on these results. This evaluation must be based upon the test result and the item level pass/fail criteria. An estimate of failure risk may be included. It also contains an evaluation of the corresponding test items. Identify the items tested, indicating their version/revision level. Indicate the environment in which the testing activities took place. For each test item, supply references to the following documents if they exist: test plan, test-design specifications, test-procedure specifications, test-item transmittal reports, test logs, and test-incident reports.

7.2 Variances

Report any variances of the test items from their design specifications. Indicate any variances from the test plan, test designs, or test procedures. Specify the reason for each variance.

7.3 Comprehensiveness Assessment

Evaluate the comprehensiveness of the testing process against the approach specified in the test plan (2.1) if the plan exists. Identify features or feature combinations which were not sufficiently tested and explain the reasons.

For each measure you have chosen to use, state the result. Compare with previous results and discuss trends.

7.4 Summary of Results and Consequences

Summarize the main results and consequences of testing. Identify all resolved incidents and summarize their resolutions. Identify all unresolved incidents and plan for their future resolution.

7.5 Evaluation

Provide an overall evaluation of each test item including its limitations. This evaluation must be based upon the test result and the item level pass/fail criteria. An estimate of failure risk may be included.

7.6 Summary of Activities

Summarize the major testing activities and events. Summarize resource consumption data, for example, total staffing level, total machine time, and total elapsed time used for each of the major testing activities.

8. Notes

Provide any general information and rationale pertinent to the information contained in this test description and results report. Include a list of acronyms and abbreviations, and a list of terms and their definitions used in this test report. This list contains definitions needed to understand this test report. Include additional definitions and delete those not applicable.

9. Appendix

Use appendixes to provide information that is published separately or that does not fit conveniently in the body of the document. Each appendix should be referenced in one of the above sections where data would normally have been provided.

A. Test log

Present a chronological record of software requirement verification activities, indicating the following:

- Date
- Time
- Location
- Participants
- Reference (Test Case/Test Report/Inspection Report, etc...)

Inspection Plan

Purpose

This inspection plan describes the inspection process, inspection checklists, the people involved and the schedule of inspections for the project.

Timing

Intended Audience

Participants

Responsibility

The project manager should create the inspection plan, designate inspection areas and facilitate persons inspecting.

Completion Criteria

Additional Information

CS 577b Guidelines:

High-Level Dependencies

| OCD | SSRD | SSAD | LCP | FRD | Prototype |
|-----|------|------|-----|-----|-----------|
| ~ | + | + | 0 | ~ | ~ |

Outline

1. Purpose of Inspections

Describe the intended purpose of the inspections and expected outcomes for this project.

2. Inspection Items, Participants and Roles

- Describe the artifacts (documents, etc...) that will be subjected to inspection, taking into account schedule or staffing constraints
- For each artifact, indicate the primary author, and identify who will play the various roles. This information will be required to help the participants involved prepare for their respective roles.
- Provide checklists for each of the artifacts to be used during the inspection process. An example is the JPL Inspection Process reference checklists.
- You may simply provide a reference to the various Inspection Announcements (to be included in the Appendix)

3. Inspection Milestones

Provide details about the schedules for the various activities for inspections:

- completion of planning
- overview meeting
- preparation
- inspection meeting (date, time, place)
- inspection data summary and results reporting

4. Inspection Process

- Describe the inspection process for each type of inspection item
- Reference any inspection process (e.g., Fagan Code Inspection): if that is the case, simply indicate any variations introduced particularly

5. Classification of Defects

All defects identified during the inspection meeting shall be classified by severity and type. This serves as a metric for future project. Example severity levels and type of defects are presented in the tables below. Consider using existing defect classifications such as Orthogonal Defect Classification (ODC). Describe the classification scheme to be used for this project. In addition to an overall description, define the form below from the project perspective.

5.1 Severity

a. Major

Define what the major defects are. You may wish to specialize the following:

- A condition that causes an operational failure, malfunction, or prevents attainment of an expected or specified result
- Information that would lead to an incorrect response or misinterpretation of the information by the user
- An instance of non-conformance that would lead to a discrepancy report if implemented as is

b. Minor

- A violation of standards, guidelines, or rules, but would not lead to a discrepancy report
- Information that is undesirable but would not cause a malfunction or unexpected results (bad workmanship)
- Information that, if left uncorrected, may decrease maintainability

5.2 Type

Define the type of defect

a. Missing

Information that is specified in the requirements or standard, but is not present in the document

b. Wrong

Information that is specified in the requirements or standards and is present in the document, but the information is incorrect

c. Extra

Information that is not specified in the requirements or standards but is present in the document

5.3 Category

- Refine the types of defects to the inspection items identified in Section 2
- Examples of defect types include:
 - Logic
 - Syntax
 - Clarity
 - Performance
 - Interface
 - No Error Handling

- ...

6. Appendix

Include in the appendix references to the forms to be used by the various participants in the preparation of Inspections. This include:

- Announcement
- Individual Preparation Log
- Defect List
- Detailed Report
- Summary Report

Inspection Report

Purpose

Timing

Intended Audience

Participants

Responsibility

Completion Criteria

Additional Information

CS 577b Guidelines:

High –Level Dependencies

Outline

1. Participants and Roles

- Provide an update of any changes in the Inspection Plan, section 2.
- Inspectors should include in the appendix the Summary Forms, Detailed Report and the Summary Report as specified in the Inspection Plan Appendix, section 6.

2. Inspection Items

- Describe the item(s) that were subjected to this inspection.
- Describe the exit criteria used for each Inspection Item (and whether they were updated from Inspection Plan, section 4.)

3. Pre-Inspection Defect Data

Report the pre-inspection data (time, majors and minors found per inspector)

4. Inspection Defect Data

Report the meeting defect data (size of artifact, major and minor counts, time of inspection, new defects found during the meeting not found in preparation).

5. Post-Inspection Defect Data

Report the post- meeting data (rework hours and final count of defects).

6. Inspection Statistics

From the above, compute the following inspection statistics:

- total effort consumed for each inspector
- defect density of the inspected artifacts
- defects asserted per minute of the meeting
- defect removal effectiveness

These are defined as follows:

Time Meeting Effort = Time Meeting * Persons

Total Defects found = Items from preparation + New items

Defects asserted per minute of meeting = Total Defects / Time Meeting

Inspection effort = Preparation effort + Meeting effort + rework effort

Defect removal effectiveness = Defects found / Inspection effort

Defect Density = Defects / SLOC or Pages

Inspection Rate = Inspected pages / Meeting time

7. Defect Correction and Rework

Provide the status of defect correction, including

- technical approach of outstanding rework
- due date of outstanding rework if any

Provide information for tracking that the defects were properly removed.

8. Appendix

Include in the Appendix the following forms, completed by the various participants in the Inspection Meeting.

- Announcement
- Individual Preparation Log
- Defect List
- Detailed Report
- Summary Report

Transition Plan

Purpose

The purpose of the transition plan is to ensure that the system's operational stakeholders (users, administrators, operators, maintainers) are able to successfully operate and maintain the system.

Timing

The plans are typically created during construction, to be used in the transition phase.

Intended Audience

The primary audiences are the transition participants (developers, operators, maintainers, operations managers, suppliers, user representatives, others as appropriate) and others involved in the Transition Readiness Review (customers, transition experts, senior management).

Participants

Responsibility

The Project Manager is responsible for creating and updating the transition plans, and for ensuring the commitment of the other stakeholders to them. These responsibilities may be delegated to other team members. You may consider forming a transition team.

Completion Criteria

The primary completion criteria are stakeholder concurrence, feasibility of execution, and assurance of successful transition. These are evaluated by the equivalent of an Architecture Review Board in a Transition Readiness Review. The plan should address the following transition success factors: (add success factors!!)

Additional Information

CS 577b Guidelines:

High-Level Dependencies

The Transition Plan draws on and elaborates the transition aspects of the Operational Concept (OCD 4); Milestones and Products (LCP 2.) and Responsibilities (LCP 3.) from the Life Cycle Plan. It prepares for the successful execution of the life cycle support activities and responsibilities in the Support Plan.

| OCD | SSRD | SSAD | LCP | FRD | Prototype |
|-----|------|------|-----|-----|-----------|
| ~ | - | - | 0 | + | - |

Outline

1. Transition Strategy

This section shall be divided into paragraphs as needed to describe the developer's plans for transitioning the deliverable software to the support agency. This section shall address the following:

1.1 Transition Objectives

Establish the various dimensions of success associated with the particular transition activity. These may include the following classes of transition choices:

- Extent of capability transitioned: limited pilot operation; full operation; intermediate.

- Number and nature of transition sites: one vs. many; homogeneous vs. heterogeneous.
- Degree of post-transition developer support: none (pure turnkey), full (facilities management by developer), intermediate.
- Degree of validation of operational satisfaction of stakeholder objectives.
- Nature of product transition: shrink-wrapped product; new system; improvement in existing operation.

1.2 Transition Process Strategy

This section establishes the major strategic transition choices:

- Phasing of cutover: instantaneous (cold turkey); incremental; parallel operation; other.
- Phasing of transition of multiple increments to multiple sites.
- Role of alpha-testing, beta-testing, independent operational testing and evaluation.

2. Preparing for Transition

Preparing for transition is a significant non trivial task critical to ultimate successful operation of the system. Careful attention should be paid to scheduling adequate time and resources. Anticipate problems and plan for them. Be careful of risky assumptions (e.g. the customer will hire an experienced database administrator to install Oracle and set up the proper database schema)

2.1 Hardware Preparation

- Indicate additional hardware that needs to be purchased if any
- Indicate any special or additional instructions for placing the hardware in a state of readiness
- Staff time required for hardware preparation

2.2 Software Preparation

This section should draw upon and extend LCP 2.3, Project Deliverables, to establish:

- Appropriate packaging of deliverable support software (tools; infrastructure software).
- Licenses for commercial software.
- Preparation and installation of necessary data.
- Conversion of legacy software for compatibility with the new system.

2.3 Site Preparation

Facilities may include:

- computer rooms, flooring, power supply
- computers, peripherals, and supplies
- data communications capabilities

2.4 Staff Preparation

This section shall be divided into paragraphs as appropriate to describe the developer's plans for training support personnel to support of the deliverable software.

2.4.1 Training Deliverables

- Hardware and Software Preparation
- Staff availability
- Impact on staff time

2.4.2 Training Schedule

- Number of training sessions
- Lengths of sessions
- Contents of each session
- Training materials used in each session

2.4.3 Measure of Success

- Evaluations of the results of the training

2.5 Operational Test and Evaluation

Assurance of adequate transition readiness is essential for successful transition. Tasks and events prior to transition help identify risks, problem areas, set baselines, and manage stakeholder expectations. Address the following items to estimate.

2.5.1 Evaluation Criteria

Carefully define appropriate metrics to be collected and analyzed addressing such criteria as improved efficiency, quality of service, mission effectiveness, stakeholder satisfaction, and return on investment (FRD 2.1.5)

2.5.2 Procedures

This section establishes the major operational test and evaluation procedure choices. Refer to the test plan or test description results if needed.

- Exercise procedures: operational scenarios, alpha test, beta test, other
- Exercise participants: number, qualifications, organizational representation.
- Instrumentation: performance measurement, problem reports, questionnaires, other.
- Analysis procedures with respect to evaluation criteria.

2.5.3 Outline of Operational Test and Evaluation Report

The report should include an update of the evaluation criteria and procedures above, and various summaries of the results, conclusions, and recommendations.

3. Stakeholder Roles and Responsibilities

This should be a transition-oriented elaboration of your version of Table 2, (LCP 3.1), and its refinements in LCP 3.2.2. Transition roles and responsibilities can vary significantly, depending on the nature of the product (shrink-wrap, turnkey, internally developed, ...) and the transition target (single vs. multiple, homogeneous vs. heterogeneous, ...).

Provide a subsection for each stakeholder category with a significant transition role. Besides developers, maintainers, users, customers, and interfacers, these could include operational testers, labor unions, and safety officials for example.

Where the LCP focuses on organizational roles, this section should focus on the roles of specific individuals (e.g., which maintainer individuals are going to perform representative product modifications to evaluate maintainability?). These commitments need to be coordinated with the stakeholder organizations and individuals.

4. Milestone Plan

The plan should include key preparation milestones leading up to the key Transition Readiness Review (TRR) milestone, and key transition milestones leading up to the Product Release milestone. There may be post-release milestones as well: for example, if user training is spread across time.

Any explicit or implicit client milestone commitments need to be coordinated with the clients. Implicit commitments can arise from dependency relations, e.g., data preparation ? training material preparation ? training ? operational use ? operational evaluation.

CS 577b Guidelines:

All developer activities are assumed to be complete at the Product Release milestone. The schedules and milestones should be compatible with those in the Life Cycle Plan.

5. Required Resources

This section should elaborate the transition portion of LCP 5. Resources should include the time required by developers and clients, and the necessary financial resources for hardware, software, facilities, supplies, data, training, and other possible needs.

Software User's Manual

Purpose

The purpose is to teach and guide the user how to use the product, i.e., the steps for running the software, describes the expected output(s), and describes the measures to be taken if errors occur.

Timing

Produce the manuals early in the development process and update through each iteration. This helps users evaluate each release, and ensures early and continuous user feedback. How early in the development cycle to begin producing the user manual depends on the type of system. Systems with complex interfaces or with a lot of user interaction will require early versions of the user manual and also early prototypes of the interface. Embedded systems with little human interface will probably not require an early start on user documentation.

Intended Audience

Participants

Responsibility

The Test Team or the Technical Writer is responsible for creating and updating the material. The user manual can be written by technical writers, with input from developers, or it can be written by the test team, whose members are likely to understand the user's perspective.

Completion Criteria

Additional Information

The end-user documentation gives instructions for using the software. Provide documentation for all types of users. You may want to consider a different user manual for each operational role or user class: e.g., User Manual, Administrator Manual. Use cases are the basis for the manual.

Use screen shots extensively.

A reason for allocating the user manual to the test team is that it can be generated in parallel with development and evolved early as a tangible and relevant perspective of evaluation criteria. Errors and poor solutions in the user interface and use-case model can be spotted and corrected during the early iterations of the project, when changes are cheaper.

By writing the user manual, the testers will get to know the system well before they start any full-scale testing. Furthermore, it provides a necessary basis for test plans and test cases, and for construction of automated test suites.

CS 577b Guidelines:

High-Level Dependencies

| OCD | SSRD | SSAD | LCP | FRD | Prototype |
|-----|------|------|-----|-----|-----------|
| + | + | ~ | - | ~ | + |

Outline

1. Introduction

1.1 System Overview

State the purpose of the library system and the software to which this manual applies.

1.2 System Requirements

Describe the minimum hardware and software (Operating System, etc.) requirements for the system.

2. Operational Procedures

Briefly describe functions available to the user and describe the step-by-step procedures for accessing these functions.

Include the following information, as applicable:

- a. Initialization. Describe any procedures needed to initialize the software. Identify any initialization options available to the user.
- b. User Functions. Describe the functions available to the user. For each function describe:
 1. User inputs including format, limitations, input method, etc.
 2. Operational results
- c. System Inputs. Describe the system inputs to the software that may occur while the software is in use and that may affect the interface with the user. Include format, frequency, allowable range, and units of measure, as applicable.
- d. Termination. Describe how to terminate software operation and how the user determines that normal termination has occurred.
- e. Restart. Describe the procedures for restarting the software.

You may want to split the description of the user manual into:

- Basic Features (*Getting Started*, or *Learning the Basics*)
- Advanced Features

3. Installation Procedures

In a system where the end user is expected to install the product, the Installation Instructions can be included in the user's guide. For a more complicated installation where qualified service staff are needed, the installation instructions would be described in a separate Installation Manual.

3.1 Initialization Procedures

Describe first-time installation procedures

3.2 Re-installation

Describe procedures for reinstalling the system (e.g., to recover from a corrupt installation)

3.3 De-installation

Describe procedures for removing the system

4. Troubleshooting

4.1 Frequently Asked Questions

List Frequently Asked Questions by operators, and answers to those questions.

4.2 Error Codes and Messages

List and identify all error codes and messages generated by the software, the meaning of each message, and the action to be taken when each message appears.

5. Notes

Include any general information that aids in the understanding of the document. All acronyms, abbreviations, and their meaning as used in this document should be listed.

6. Appendix

List any additional information, such as technical specifications, etc.

Support Plan

Purpose

The purpose of the transition plan is to ensure that the system's operational stakeholders (users, administrators, operators, maintainers) are able to successfully operate and maintain the system.

Timing

Intended Audience

Participants

Responsibility

Completion Criteria

Additional Information

CS 577b Guidelines:

High-Level Dependencies

| OCD | SSRD | SSAD | LCP | FRD | Prototype |
|-----|------|------|-----|-----|-----------|
| ~ | - | + | 0 | + | - |

Outline

1. Environment

Document the anticipated environmental concerns for the long term use of the system. These may include:

1.1 Facilities

Describe the facilities needed to maintain the deliverable software. These facilities may include special building features, such as cabling, special power requirements, etc...

1.2 Hardware

Describe the hardware and associated documentation needed to maintain the deliverable software. This hardware may include computers, peripheral equipment, and non-computer equipment.

1.3 Software

Identify and describe the software and associated documentation needed to maintain the deliverable software. This software may include computer-aided software engineering (CASE) tools, compilers, test tools, test data, utilities, configuration management tools, databases and other software. The description shall include:

- a) Specific names, identification numbers, version numbers, release numbers, and configurations, as applicable
- b) Rationale for the selected software
- c) Reference to user/operator manuals or instructions for each item, as applicable
- d) If items must be acquired, information about a current source of supply, including whether the item is currently available and whether it is expected to be available at the time of delivery

- e) Information about vendor support, licensing, and data rights, including whether the item is currently supported by the vendor, whether it is expected to be supported at the time of delivery, whether licenses will be assigned to the support agency, and the terms of such licenses
- f) Security and privacy considerations, limitations, or other items of interest

CS 577b Guidelines:

Include any special settings for various tools (such as compilers, database management systems), environment variables that the maintainer needs to be aware of to properly maintain the system.

1.4 Other documentation

Identify any other documentation needed to support the deliverable software. The list will include, for example, plans, reports, studies, specifications, design descriptions, test cases/procedures, test reports, user/operator manuals, and support manuals for the deliverable software. This paragraph shall provide:

- a. Names, identification numbers, version numbers, and release numbers, as applicable
- b. Rationale for including each document in the list
- c. Identification of each document as acquirer-furnished, an item that will be delivered to the support agency, an item the support agency is known to have, an item the support agency must acquire, or other description of status
- d. If a document must be acquired, information about where to acquire it
- e. Information about licensing and data rights, in particular license expiration dates
- f. Security and privacy considerations, limitations, or other items of interest

2. Resources

2.1 Personnel

Describe the personnel needed to support the deliverable software, including anticipated number of personnel, types and levels of skills and expertise, and security clearances. This paragraph shall cite, as applicable, actual staffing on the development project as a basis for the staffing needs cited.

2.2 Other resources

This paragraph shall identify any other resources needed to support the deliverable software. Included may be consumables such as magnetic tapes and diskettes, together with an estimate of the type and number that should be acquired.

3. Recommended tools and procedures

This section shall be divided into paragraphs as needed to describe any procedures, including advice and lessons learned, that the developer may wish to recommend to the support agency for supporting the deliverable software and associated support environment.

- Regression Test Cases
- Maintenance Package
- Tools and Procedures

4. Anticipated areas of change

Describe anticipated areas of change to the deliverable software. Account for need to implement evolution requirements (SSRD 6), possible or anticipated organizational changes (OCD 3.6), future major changes to COTS components the system depends on, anticipated new COTS packages that may replace current components and so forth. Address how anticipated changes might be managed (SSAD) and/or is already accommodated in design as evolution requirements (SSRD 6.).

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.

[Laprie 1992] J.C. Laprie. "For a product-in-a-process approach to software reliability evaluation", Proc. Third International Symposium on Software Reliability Engineering (ISSRE92), pp. 134-139, Research Triangle Park, North Carolina, 1992.

[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-1994] Potts, C., Takahashi, K., & Anton, A.: "Inquiry-Based Requirements Analysis"; *IEEE Software*, March 1994), 21-32

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/Robts/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.

[Thorp 1999] Thorp, J, The Information Paradox: Realizing the Business Benefits of Information Technology, McGraw Hill, 1999

Appendices

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

Appendix A. 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. Project Requirements)
 - 1.1 Budget Constraints
 - 1.2. Schedule Constraints
 - 1.3 Staffing Constraints
2. Application Capabilities (====>SSRD 3.2 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. Level of Services (====> SSRD 5. 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 Compatibility
 - 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
 - 3.7.3 Storage Efficiency
 - 3.7.4 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
- 3.9 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
- 5. Environment and Data (====> SSRD 2. Project Requirements)
 - 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 B. 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 C. 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.

ARB: Architectural Review Board

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.

CLI: Command Line Interface

CM: Configuration Management

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.

CTS: Construction Transition Support package

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.

FRD: Feasibility Rationale Description

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 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.

LCO: Life Cycle Objectives (milestone)

LCA: Life Cycle Architecture (milestone)

LCP: Life Cycle Plan

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.

OCD: Operational Concept Description

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.

Participating Agent: Non developer stakeholder in the system

Performing Agent: One that expends effort to realize the solution to the problem being solved in the project (manager, architect, analyst, designer)

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

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.

RLCA: Rebaselined LCA

ROI: Return on Investment

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.

SDP: Software Development Plan

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.

SSRD: System and Software Requirements Definition

SSAD: System and Software Architecture Description

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.

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’

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.

UTCR: Unit Test Completion REview

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

Weekly Effort Form

Week

Name:

Team:

| Activity | Mon | Tue | Wed | Thu | Fri | Sat | Sun |
|---|-----|-----|-----|-----|-----|-----|-----|
| Training/Learning | | | | | | | |
| Training/Learning: | | | | | | | |
| Application Research: | | | | | | | |
| People Interactions | | | | | | | |
| Client Interaction: | | | | | | | |
| Team meetings: | | | | | | | |
| Email: | | | | | | | |
| Telephone and Other Interactions: | | | | | | | |
| Requirements Management | | | | | | | |
| Updates to Operational Concept Description: | | | | | | | |
| Updates to System and Software Requirements Definition: | | | | | | | |
| Updates to Feasibility Rationale Description: | | | | | | | |
| Detailed Design | | | | | | | |
| Design Creation or Modification: | | | | | | | |
| Design Review/Inspection: | | | | | | | |
| Coding | | | | | | | |
| Code Generation or Modification: | | | | | | | |
| Code Review/Inspection: | | | | | | | |
| Testing | | | | | | | |
| Unit Testing: | | | | | | | |
| Integration Testing: | | | | | | | |
| Miscellaneous | | | | | | | |
| Prototyping: | | | | | | | |
| COTS Assessment: | | | | | | | |
| COTS Tailoring: | | | | | | | |
| COTS Integration: | | | | | | | |
| Project Management and Special Functions | | | | | | | |
| Planning and Control: | | | | | | | |
| Project Review preparation: | | | | | | | |
| Configuration Management and Quality Assurance: | | | | | | | |
| Transition Activities | | | | | | | |
| Transition Planning, Preparation and Execution: | | | | | | | |
| Training: | | | | | | | |
| User Documentation: | | | | | | | |
| Customer Deliverables: | | | | | | | |
| Other Effort Not Covered Here | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

Weekly Status Report

Team #

Team Members

Project Title

Week #

This is a suggested Status Report. Feel free to expand or collapse the various categories.

Progress

- Describe the progress on the project during the past week, both in qualitative and quantitative terms
- List any significant accomplishments since the last status report. This can include the completion/revision of project deliverables
- Include metrics to describe the progress

Problems

- List any problems the team is experiencing. This includes things that have caused delays, or any other roadblocks to anticipated progress.
- Provide a weekly Top-N risk items list as follows:

| Risk Items | Weekly Ranking | | | Risk Resolution Progress |
|------------|----------------|----------|---------|--------------------------|
| | Current | Previous | # Weeks | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Immediate Plans

List your team's plans (or TO DO items) for the next week. This section might be filled with significant action items that are mentioned during your team status meeting, to help remind team members of significant items discussed in the meeting. You are encouraged to maintain, separately, meeting minutes, to record important discussions and action items raised during your team meetings.

Comments

Include here any items not covered above.