# Migration from a Waterfall/Non SEI CMM Compliant Process to a RUP/SEI CMM Compliant Process

by

Chad Amos Chamberlin

A thesis submitted to the
School of Extended Graduate Studies at
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Software Engineering

Melbourne, Florida
May 2003

# Permission to Copy

We the undersigned committee hereby recommend that the attached document be accepted as fulfilling in part the requirements for the degree of Master of Science in Software Engineering.

"Migration from a Waterfall/Non SEI CMM Compliant Process to a RUP/SEI CMM Compliant Process,"
a thesis by Chad Amos Chamberlin

Mr. David Clay,
Director, Computer Programs
School of Extended Graduate Studies Spaceport, KSC
Thesis Advisor

Art Dickinson, Ph.D.
Assistant Director, Computer Programs
School of Extended Graduate Studies Spaceport, KSC

Michael Shaw, Ph.D.
Associate Professor and Associate Head
Department of Mathematical Sciences

William David Shoaff, Ph.D.
Associate Professor and Head
Department of Computer Sciences

# Abstract

Title:
Migration from
a Waterfall/Non SEI CMM Compliant Process to
a RUP/SEI CMM Compliant Process

Author:
Chad Amos Chamberlin

Thesis Advisor:
Mr. David Clay

Software Life Cycle processes play a key role in the successful development of software products. Software processes continue to evolve as the field of software development progresses toward being a true engineering discipline. This thesis has two objectives:

(1) To apply software engineering knowledge gained during the pursuit of this degree toward the design and development of a complete, Software Engineering Institute (SEI) Capability Maturity Model (CMM) compliant process for the design phase of the software lifecycle.

(2) To evaluate a Waterfall Lifecycle Model vs. an Iterative Lifecycle Model and compare SEI CMM compliant processes to non-SEI CMM compliant process using a "real world" software development project.

# Table of Contents

# Table of Figures

# Table of Acronyms

| | |
|---|---|
| APS | Application Software |
| CDR | Critical Design Review |
| CLCS | Checkout Launch and Control System |
| CMM | Capability Maturity Model |
| COTS | Commercial off the Shelf |
| CSC | Computer Software Component |
| CSCI | Computer Software Configuration Item |
| CSU | Computer Software Unit |
| DOD | Department of Defense |
| FPA | Function Point Analysis |
| IPT | Integrated Product Team |
| KPA | Key Process Area |
| KSC | Kennedy Space Center |
| OO | Object Oriented |
| PDR | Preliminary Design Review |
| RUP | Rational Unified Process |
| SDD | Software Design Document |
| SDP | Software Development Plan |
| SDR | Software Design Review |
| SEI | Software Engineering Institute |
| SQM | Software Quality Management |
| SRR | System Requirements Review |
| SSR | Software Specification Review |
| STD | Standard |
| UML | Unified Modeling Language |

# Acknowledgements

I would like to thank David Clay for serving as my thesis advisor. Special thanks to Jonsie Ivey for clearing the thesis path so that others at SEGS KSC could follow.

# 1 Introduction

It is commonly accepted throughout the software industry that the road to a successful software development project follows a software development process that is implemented inside the general confines of a software life cycle model.  What is not commonly accepted is which life cycle model and which development processes are the "ones" to follow for a particular development effort.  As the software industry continues to evolve so do the models and processes by which software is developed.  Over the years, the evolution of software development has produced many different types of software life cycle models and even more software development processes, each with their list of pros and cons, enthusiast and naysayers.

By choosing a life cycle model to follow and then instituting development processes, a software project begins the effort of developing software.  The ability for these processes to aid in the successful completion of a software product can be subjective.  How does one know if the processes are worth following?  Will they aid in a successful completion?

In 1986 the Software Engineering Institute (SEI) began work on a way to evaluate software development processes.  By evaluating the effectiveness of a software process a rating could be applied which in turn could help indicate the maturity of the software process and it's likelihood of successfully producing software.  This software process evaluation is known as the Capability Maturity Model (CMM).

This background work for this thesis was conducted over a period of twelve months.  The data collected are from a real world software engineering project; the Checkout and Launch Control System (CLCS), Application Software (APS) at NASA Kennedy Space Center (KSC). The purpose of CLCS was to replace the Checkout, Control and Monitoring System used to process and launch the United States Space Shuttle at KSC

For three years, September 1998 to September 2001, the CLCS project followed a Waterfall life cycle model with minimal process descriptions. During the three years of following Process A the CLCS project continually failed to meet schedule milestones and cost milestones. Not only where the milestones missed many of them were never achieved. A majority of the software produced was not progressing past unit test.

A corrective solution was needed to ensure the continuation of the project. It became evident that a radical redirection from Process A was necessary to get the CLCS project moving in a forward direction. The decision was made to bring the CLCS software life cycle into line with more modern life cycle models and SEI CMM compliant processes. Although the Waterfall life cycle model cannot be blamed in and of itself for the inadequacies of this development effort, the combination of this model and the lack of CMM compliant processes did produce disastrous results. The lock step method by which most organizations institute the Waterfall model prevented the discovery of severe problems until late in the life cycle. This model and minimal process definition will be referred to as "Process A" throughout the thesis.

The new life cycle model chosen iterative and implemented using the Rational Unified Process (RUP). Detailed processes were also created and modeled in the Unified Modeling Language (UML). In September of 2001, the CLCS project left Process A and began using the new process at the requirement analysis phase. This new model and process definition will be referred to as "Process B" throughout the thesis.

Software performance data were captured over the three-year period that CLCS followed Process A and over the twelve-month period that CLCS followed process B. The data taken capture the performance of one hundred plus software engineers assigned to approximately twenty Integrated Product Teams (IPTs).

Both process A and B were working to the same requirements as defined by the legacy system that was being replaced. The software engineering personnel were the same, the management structure was the same, and the development and target platforms were the same. Both Process A and Process B were followed using Object Oriented (OO) methodologies. This resulted in a unique environment in which the study of different life cycle models and development processes could be undertaken. It was this unique environment that created the opportunity for this thesis.

Throughout the pursuit of the master of software engineering degree the author was introduced to the idea that following the iterative lifecycle model and SEI CMM compliant process was the most efficient way to develop software. Although this concept was stated in many different sources and touted by experts in the industry, the author did not come across substantial evidence from a software development project that supported these claims.

The CLCS project provided the ability to validate these statements:

There is an improvement in the performance of a software development team when:

- Following an iterative lifecycle model versus a waterfall lifecycle model.
- Following SEI CMM compliant processes versus non-SEI CMM compliant processes.

In this thesis, background information on both the Waterfall and Iterative software life cycle models is presented. An explanation of the RUP is covered, as well as the history and background of the SEI CMM, including the key elements for each CMM Level covered in this thesis. The details of Process A and Process B are explained as well as the CMM evaluation of both processes. The metrics collected from each process will be presented along with an interpretation of the results. Finally, a hypothesis summary and areas of potential future research on this topic are discussed.

# 2 Evaluation of Lifecycle Models

## 2.1 Waterfall

### 2.1.1 Definition

Dr. Winston Royce first described the Waterfall lifecycle model in 1970. His paper titled, *Managing the Development of Large Software Systems: Concepts and* Techniques, "…set the roots for the 'Waterfall' model. Although Royce didn't mentioned the word 'waterfall' in it, his methodology became later known for it because of the layout of the boxes in his diagrams which looked like stones in a waterfall."[6]

This model is defined by a set of sequential phases that are strictly performed in order. Each phase is accompanied by a verification of the work in the phase that ensures that the work products are complete which in turn allows the next phase in the sequence to begin. These verifications take the form of formal reviews in which management, engineering peers, customers and even end user may be involved.

It is important that the work products in each phase are developed with due diligence, as the formal reviews are meant as a last check and verification that the work products are ready for promotion to the next phase. Although Royce allows phases to be revisited, in general practice the formal reviews indicate a firm completion of a phase and returning to phases previously reviewed is rarely allowed. Due to schedule and budget constraints there exists great pressure to "lock-down" one phase to move to the next. If issues are raised during the reviews work products may have to be revisited to solve any identified problems and the completion of a phase postponed.

Managers have been persuaded by the name "waterfall" to never move backwards in the software life cycle, trying to complete each step right the first time.[14]

The Waterfall model has been widely used in industry for many years and continues to be used today. The United States Department of Defense (DoD) developed a military software development standard known as DoD-Std-2167A. The DoD and its government contractors used this standard extensively. For its military projects the DoD mandated the use of this military software development standard for a number of years. There has been much debate on whether or not DoD 2167A dictated that the Waterfall Model be followed. Regardless of these debates, many defense contractors and those required to follow DoD 2167A used a Waterfall lifecycle model approach during their software development projects.

Although this standard has been superceded by MIL Standard 498 and other IEEE and ISO standards, the following description describes DoD 2167A in conjunction with a Waterfall lifecycle model, as it is this standard that was used as a model for Process A. Figure 1 shows the DoD-2167A life cycle model, its resemblance to waterfall model diagrams can easily be seen.

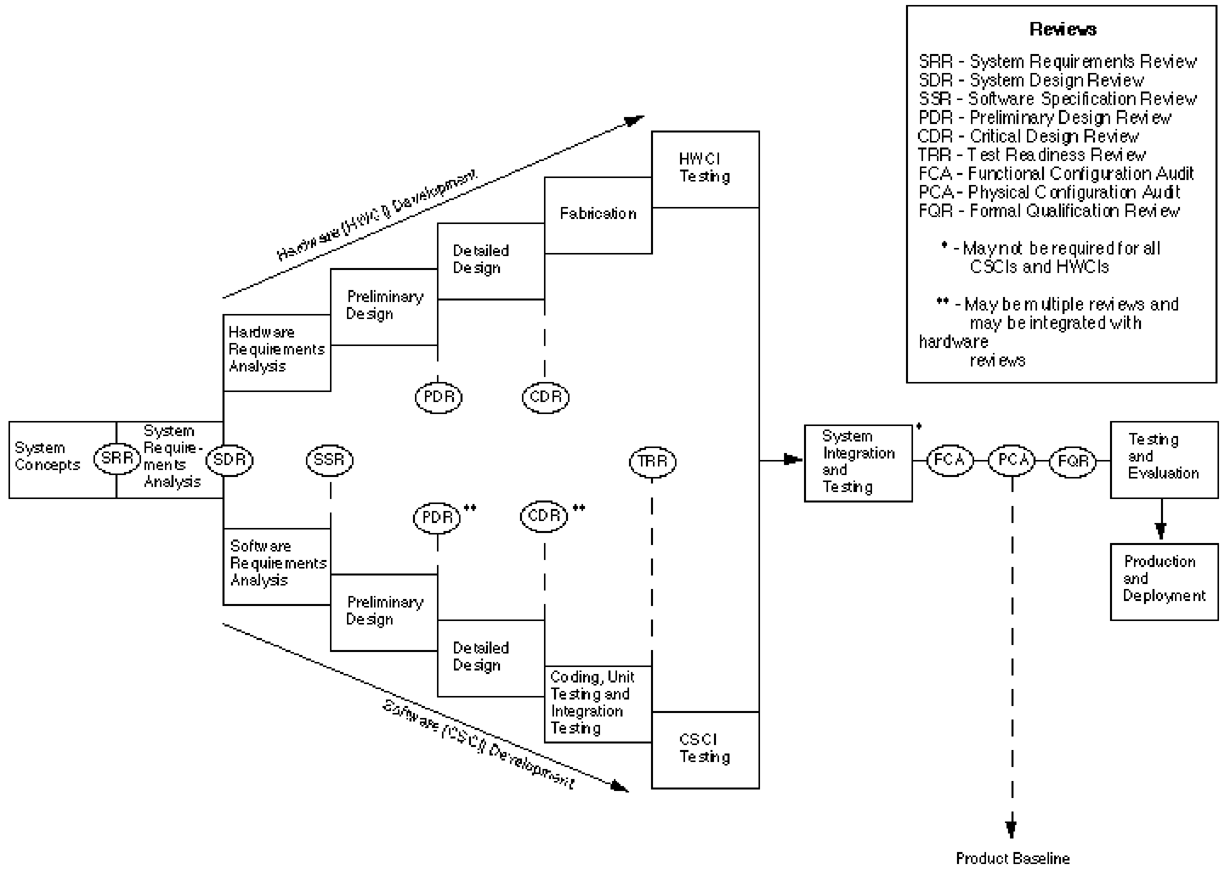

Reviews
SRR - System Requirements Review
SDR - System Design Review
SSR - Software Specification Review
PDR - Preliminary Design Review
CDR - Critical Design Review
TRR - Test Readiness Review
FCA - Functional Configuration Audit
PCA - Physical Configuration Audit
FQR - Formal Qualification Review
* - May not be required for all CSCIs and HWCIs
** - May be multiple reviews and may be integrated with hardware reviews
Hardware (HWCI) Development
Software (CSCI) Development
System Concepts
SRR
System Require-ments Analysis
SDR
SSR
Hardware Requirements Analysis
Preliminary Design
Detailed Design
Fabrication
HWCI Testing
PDR
CDR
TRR
PDR **
CDR **
Software Requirements Analysis
Preliminary Design
Detailed Design
Coding, Unit Testing and Integration Testing
CSCI Testing
System Integration and Testing *
FCA
PCA
FQR
Testing and Evaluation
Production and Deployment
Product Baseline

Figure 1: DoD 2167A.[5]

### 2.1.2 DOD 2167A

#### 2.1.2.1 System Requirements

The system requirements phase is used to define and/or analyze the requirements levied on the entire system being developed. These requirements are generally written at a high level and must be refined so that they are as clear and unambiguous as possible. This is a difficult task and can set the stage for the entire life of the project.

The end of this phase is marked by the first verification in which the system requirements documents are formally examined in the System Requirements Review (SRR). It is important that the system engineers understand the requirements prior to the SRR and that they have used this phase to resolve any misunderstandings or discrepancies in the system requirements.

Once the system requirements have been verified in the SRR they are ready to be used during the system design phase. At this point all parties involved have agreed upon the requirements and the ability to change requirements is strictly limited.

#### 2.1.2.2 System Design

The system design phase involves taking the verified system requirements and designing the system. The design of the system includes which hardware platforms, operating systems, networks, commercial off the shelf products (COTS), etc. must be included in the system. Also examined are custom hardware development and custom software development.

Those requirements that fall into the realm of custom software development are levied upon the software development organization. These requirements will be the foundation of the next software development phase. The conclusion of this phase comes with the verification of the system design in the system design review (SDR).

It is very important that those responsible for the software development agree with the requirements that system engineering has allocated to them as this will be the foundation of the software development effort.

#### 2.1.2.3 Software Requirements

Once the system designers allocate requirements to the software organization these requirements become the responsibility of that organization. Like the systems requirements phase, this phase requires a scrutiny of the requirements, the difference being that these requirements pertain to the software being developed not the entire system. Often times the initial requirements are at such a high level that they must be broken down into smaller subsets of requirements. This results in what are commonly referred to as grandparent, parent and child requirements.

It is imperative that the software development team understands the software requirements. It is this foundation that will dictate success or failure of the software end product. This phase concludes with the verification of the software requirements, both those allocated by the system design phase and those derived requirements created during this phase. This verification occurs via the Software Specification Review (SSR). At the conclusion of this review the software implementation effort is ready to begin.

### 2.1.2.4 Software Preliminary Design

During the preliminary design phase the software requirements are used to begin the design of the software system.  This includes identification of Computer Software Configuration Items (CSCI), and subsets of CSCIs Computer Software Components (CSCs).

Regardless of the design methodology used, Object Oriented, Structured or Functional Decomposition, the work products of this phase need to describe the "framework" or beginnings of the software design.  The use of two design phases allows the verification of the framework and is an attempt to prevent design mistakes at an early phase. This phase is verified during the Preliminary Design Review (PDR).

### 2.1.2.5 Software Detailed Design

The detailed design phase builds on the preliminary design to flush out the remaining detail and address remaining design issues. This phase is the last before code will actually be written and thus the work products produced must be accurate and complete.

In this phase additional CSCs may be identified as well as Computer Software Units (CSUs).  CSCs and CSUs make up the detailed design of each CSCI.  It is the design work products of these software entities that will directly translate into software code.  As a result the accurate verification of these work products is crucial and is conducted through via the Critical Design Review (CDR).

#### 2.1.2.6 Test and Integration

Test and Integration typically involves more then one phase. The testing of CSUs is generally referred to as Unit Test and is performed to ensure the correct functionality of each CSU. CSCs are comprised of CSUs and are tested after the CSUs have been integrated to function together. Subsequently each CSCI integrates its CSCs during process level testing and System level test and integration completes the testing phases by ensuring all CSCIs within a system function together.

The obvious work product of one or more test and integration phases is successful software. These tests attempt to ensure that not only does the software execute without failure but that it also performs what was dictated by the software requirements. Traceability of requirements from the requirements phase through to the testing phases is important to insure Additional work products include test plans and procedures that are followed to conduct the software tests.

Verification of the work products from this phase is generally the working software. However, other work products such as the test plans and procedures are verified for correctness as well.

## 2.2 Iterative

### 2.2.1 Definition

The waterfall process has given way to an iterative development approach.
"There are two fundamental flaws in the traditional (waterfall or modified waterfall) software development life cycle model: the information flow is unidirectional with inadequate provisions for feedback and user involvement

is focused primarily only at the beginning and end of the project. The iterative process addresses these flaws by using a shorter life cycle and allowing efficiently for feedback from later stages to earlier ones."[8]

The complexity of software products no longer allows the lock step process of fully completing each life cycle phase before moving on to the next. Iterative development allows requirements to be understood, designs to be flushed out, implementations to be created and errors to be found early.

This is accomplished by breaking the software project up into definable pieces of functionality or iterations. This is not a quantitative process and is usually determined based on the customer’s needs and what pieces of functionality are considered most important or are needed first. The complete life cycle is followed for each iteration, including delivery to the customer. As each piece of functionality, or iteration is completed the end product progressively grows. This is known as incremental delivery.

The iterative model still closely follows the phases as defined in the Waterfall Model, requirements analysis, design, implementation, test and delivery. However, by allowing iterations through each phase of the life cycle, the engineering process is enhanced from the education of the developers about the problem domain, and from the education of the users about the software product being produced.

The CLCS project chose to follow Rational Corporation's implementation of the iterative life cycle model for Process B.

### 2.2.2 Rational Unified Process (RUP)

The Rational Unified Process is an engineering process used for developing software. The RUP is a process that can be implemented for both large and small sized software projects. It is designed to produce quality software on time and within budget by increasing productivity, facilitating communication between developers and end users, being configurable, and taking advantage of the software engineering industries 'best practices'. RUP has evolved over several years and is the accumulation of these best practices as described in Section 3.2.2.1.

#### 2.2.2.1 Industry Best Practices

"Best Practices" are those that have proven to be successful in industry among many different software development organizations. Rational based the RUP on the "best practice" of an iterative life cycle model. In addition to the iterative model the RUP also incorporates other industry best practices.

##### 2.2.2.1.1 Management of requirements

If requirements are not managed correctly a software project has no chance of success. The ability to understand, control and track requirements is essential to the successful completion of any software project. The RUP, using the UML, facilitates the requirements analysis phase with use cases and scenarios. These tools help drive the users requirements through software design, implementation, test and delivery. Use cases are often used to define the iterations of a software development effort.

#### 2.2.2.1.2 Component Based

The ability to develop modules, subsystems or components has proven to reduce complexity and facilitate reuse. The break out of system objects and functions greatly improves the ability to understand, communicate and develop software products. The RUP encourages the practice of identifying architectural components within a software project and those that can be incorporated from outside of the project.

#### 2.2.2.1.3 Visually modeling

“A picture is worth a thousand words.” Visual modeling greatly improves the communication of ideas about a software design and architecture. Visual modeling contributes to the developer's ability to abstract the software's architectural components, leaving the details to the implementation phase. The RUP promotes the use of the UML for the visual modeling of a software product.

#### 2.2.2.1.4 Built-in Quality

If a quality assurance organization puts its stamp of approval on a software product at the end of the testing phase, chances are there is not very much quality in the software product. Quality is not something that can be tested for at the end of any software project. The people building the product must build quality into the product. Quality begins with requirement solicitation and must be present during every phase of the project life cycle. "A development process that does not address requirements quality is bound to produce poor-quality software."[10 p. 5] At the early stages of development a project must define how quality will be measured for all work products produced.

#### 2.2.2.1.5 Change Control

Changes come at all times during the software lifecycle and effect all artifacts of the software lifecycle.  Successful management of these changes is imperative for a software project to be successful.  The process of managing changes to the software work products as well as changes to the process itself must be defined during the preliminary stages of development.

#### 2.2.2.1.6 Commercial Viability

The force behind the creation of the RUP is a commercial entity, The Rational Software Corporation.  As such the need to generate profit and sell products comes into play.  The RUP is designed to be used in conjunction with the RUP Product.  This ‘product’ includes a central, searchable knowledge base that is used for guidelines, templates, tools, process improvement, etc.  In addition to the RUP Product, the Rational Corporation also has several development tools available to help facilitate software development.

The RUP is also meant to be used with the widely accepted Unified Modeling Language (UML).  UML was originally created by Rational and is now maintained by the Object Management Group.  UML is a visual syntax used to model and convey information (requirements, designs, and implementations).  Figure 2 shows the evolutionary history of the RUP and Figure 3 shows the RUP Iterative Model.

### 2.2.2.2 History of RUP

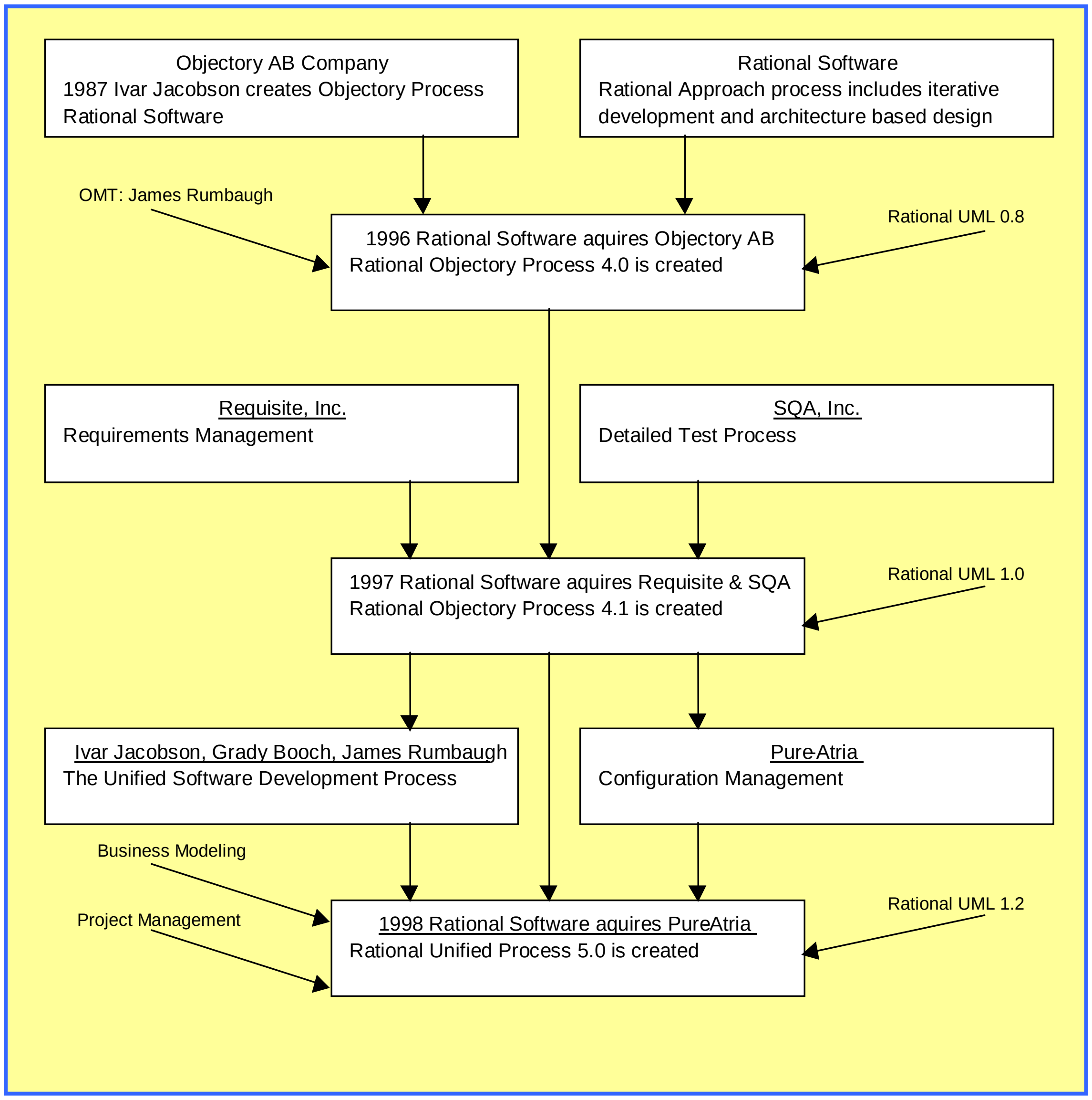

Objectory AB Company
1987 Ivar Jacobson creates Objectory Process
Rational Software
Rational Software
Rational Approach process includes iterative development and architecture based design
OMT: James Rumbaugh
1996 Rational Software aquires Objectory AB
Rational Objectory Process 4.0 is created
Rational UML 0.8
Requisite, Inc.
Requirements Management
SQA, Inc.
Detailed Test Process
1997 Rational Software aquires Requisite & SQA
Rational Objectory Process 4.1 is created
Rational UML 1.0
Ivar Jacobson, Grady Booch, James Rumbaugh
The Unified Software Development Process
Pure-Atria
Configuration Management
Business Modeling
Project Management
1998 Rational Software aquires PureAtria
Rational Unified Process 5.0 is created
Rational UML 1.2

Figure 2: The History of RUP Development

### 2.2.2.3 The Process

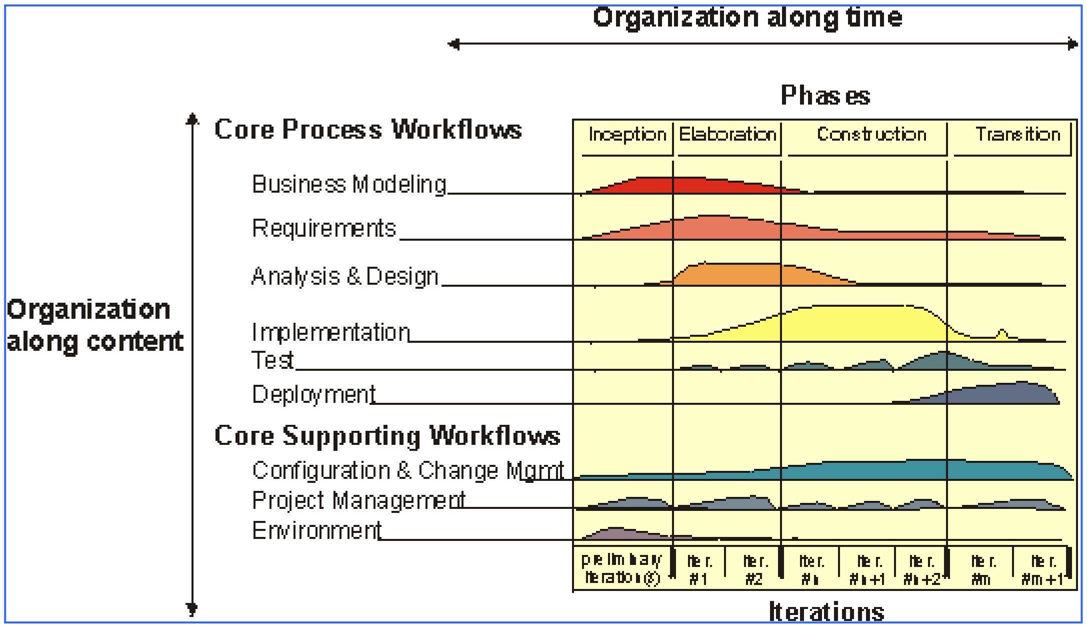


Figure 3: The RUP Implementation of the Iterative Model [7 p. 2]

#### 2.2.2.3.1 Cycles

A software project should consist of many cycles through the phases of the RUP. Each cycle results in an incremental delivery to the customer. A cycle involves traversing each phase in the model: Inception, Elaboration, Construction and Transition. There may be many iterations of each phase within a cycle. As each cycle occurs the Inception and Elaboration phases may become shorter as the understanding of the scope of the work required becomes more defined.

### 2.2.2.3.2 Phases

The phases of each cycle are the same, inception, elaboration, construction and transition. During each phase multiple iterations may be performed to refine the artifacts that will be produced.

#### 2.2.2.3.2.1 Inception

The inception phase defines the scope of the project and the business case for the project. This is done via identification of 10-20 percent of the use cases, the risks, the success criteria, scheduling and necessary resources. This information is used to determine the viability of the software project. Addressing the following objectives marks the conclusion of the Inception phase:

- What is the scope?
  - Core Requirements
  - Key deliverables
  - Project Constraints
- Do we understand the requirements?
  - Use Case model 10-20 percent complete.
- Can it be done? Will it be profitable?
  - Business case
    - Success criteria
    - Profitability
    - Financial goals
  - Risk assessment
  - Project Plan
    - Schedule with major milestones
    - Cycles
    - Phases
    - Iterations
    - Prototypes

#### 2.2.2.3.2.2 Elaboration

The elaboration phase is where the high level abstraction of the system is developed. This phase creates an overall architecture, identifying the full scope of the project, the majority of the use cases, the major functions of the system, and the major risks.  The development of the use cases helps to identify the full scope of the system including the complexity and those areas that could be critical paths due to any associated development risks. Addressing the following objectives marks the conclusion of the Elaboration phase:

- Is the ‘idea’ behind the product stable?
    - Requirements are under control.
- Are the risks addressed and can they be resolved?
- Is the Construction planned and viable?
- Can the product be produced from the use case model?

#### 2.2.2.3.2.3 Construction

The Construction phase is where the concept modeled during Inception and Elaboration is actually realized.  This phase will generally include many iterations and may have iterations occur in parallel.  The details of the system are finalized and created and all parts of the system are tested for completeness and correctness. The Construction phase includes all levels of testing.  The Construction phase does not just produce code and an executable.  Other artifacts of this phase include test plans, user manuals, product descriptions, etc. Addressing the following objectives marks the conclusion of the Construction phase:

- Initial Operational Capability:
    - Is the product ready for release to the user community?
    - Can a beta version be released?

#### 2.2.2.3.2.4 Transition

The Transition phase is used to, as the name suggests, transition the software product over to the control of the user community.  This may include required user training, simultaneous operation with an existing legacy system, establishment of maintenance organizations and processes, issue resolution, etc. The transition phase should only be entered after major issues from the Construction phases' Initial Operations Capability milestone have been resolved. Addressing the following objectives marks the conclusion of the Transition phase:

- Product Release:
  - Were the project objectives meet?
  - Can the next cycle begin?

#### 2.2.2.3.3 Iterations

The RUP is built around cycling through all RUP phases with each phase containing as much iteration as necessary. "The Unified Process assumes that the activities *Requirements, Analysis, Design, Implementation,* and *Testing* participate in each of these iterations."[2 p. 482] An iteration is the process of actually performing a particular phase more then once during a cycle. Iterating within a phase for a particular cycle allows lessons learned to be incorporated into the software products as the development is occurring.

#### 2.2.2.3.4 Modeling elements

For the RUP to work, process definition beyond cycles, phases and iterations is required. The 'who', 'what', 'when' and 'how' of the process is needed. Definitions are needed of who will do the work, what the work is comprised of and how and when the work will be performed. It is during the project-planning portion of the inception phase that these elements are identified.

- Who = Workers
- What = Artifacts
- How = Activities
- When = Workflow

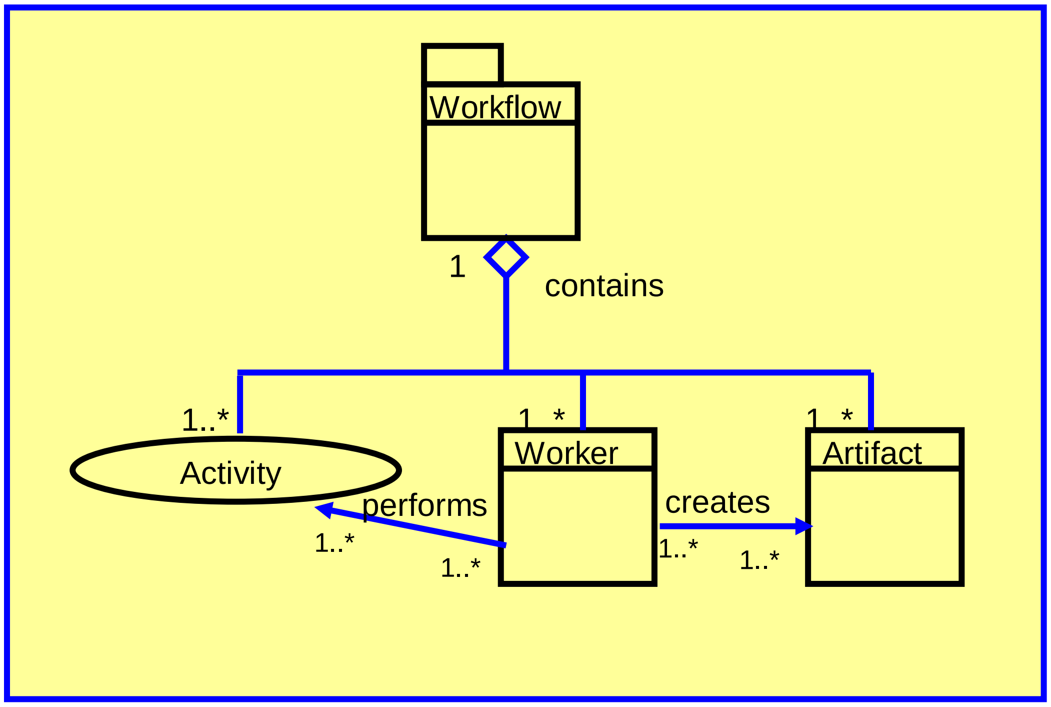


Figure 4: RUP Modeling Elements

#### 2.2.2.3.4.1 Who = Workers

Workers are the ‘who’. Workers can be individuals or entire teams and are logical entities that are responsible for assigned activities and the artifacts those activities produce. For example, a designer produces an Object Oriented Design and an Implementer produces an implementation of the design. Throughout the life cycle a person or team may fulfill many different worker responsibilities.

#### 2.2.2.3.4.2 What = Artifacts

Artifacts are the ‘what’. Artifacts are the actual products produced by the workers. These products can be entry or exit criteria to a particular phase. They can be internal or external deliverables or even end products.

Artifacts are produced in every cycle for every phase and for each iteration. Examples of artifacts include, Use-Case Models, Design Model, Class Diagram, Code, Test Plans, User Guide.  Any item, which is produced during the software lifecycle, is an artifact and had to be produced by a worker during an activity.

#### 2.2.2.3.4.3 How = Activities

Activities are the 'how'.  Activities are how the worker actually produces an artifact.  Although phases are actually "activities" these are too large to handle for the worker and must be broken down into more manageable activities.  Generally an activity should take no longer then a few days. Activities during the Elaboration phase might include the creation of a specific Use Case, or the reviewing of the System Architecture Diagram.

#### 2.2.2.3.4.4 When = Workflows

Workflows are the 'when'.  Workflows are the essential identification of when Workers should perform certain Activities to produce the appropriate Artifacts.  Although the RUP is not a waterfall methodology, there is a required ordering to the Workflows.  Implementation before Design would be disastrous; it is the ordering of Workflows that make the RUP a process by which software can be produced.  The RUP defines two types of Workflows, core process workflows and supporting workflows.

<u>Core Process Workflows</u>

The Core Process Workflows are used in the actual production of the end product.  The artifacts created during these Workflows are the engineering artifacts created by Workers who are generally part of the actual software engineering process.  These Workflows can be visited throughout each cycle, phase and iteration. Although different phases place more emphasis on certain Workflows then others, the Workflows are not performed in a sequential fashion and can be on going during each phase of a cycle. As shown in Figure 3 the Core Process Workflows consist of the following:

- - Business Modeling
- - Requirements
- - Analysis and Design
- - Implementation
- - Test
- - Deployment

Core Supporting Workflows

Core Supporting Workflows do not produce software artifacts, however, they are vital to the success of the RUP. These Workflows, as the heading suggest, “support” the Core Process Workflows. Like the Core Process Workflows, the Core Supporting Workflows can have more emphasis placed on them during certain phases but are continually active during all phases. The Workers involved with the Supporting Workflows are generally not part of the software engineering team. They fulfill separate yet equally important roles. As shown in Figure 3 the Core Supporting Workflows consist of the following:

- - Project Management
- - Configuration and Change Management
- - Environment Support

# 3 Discussion of SEI CMM

## 3.1 History of CMM

The Capability Maturity Model for developed by the SEI is a framework that describes the key elements of an effective software process. The CMM describes an evolutionary improvement path for software organizations from an ad hoc, immature process to a mature, disciplined one. This path follows five levels of maturity. Figure 4 shows the evolutionary history of the SEI CMM.

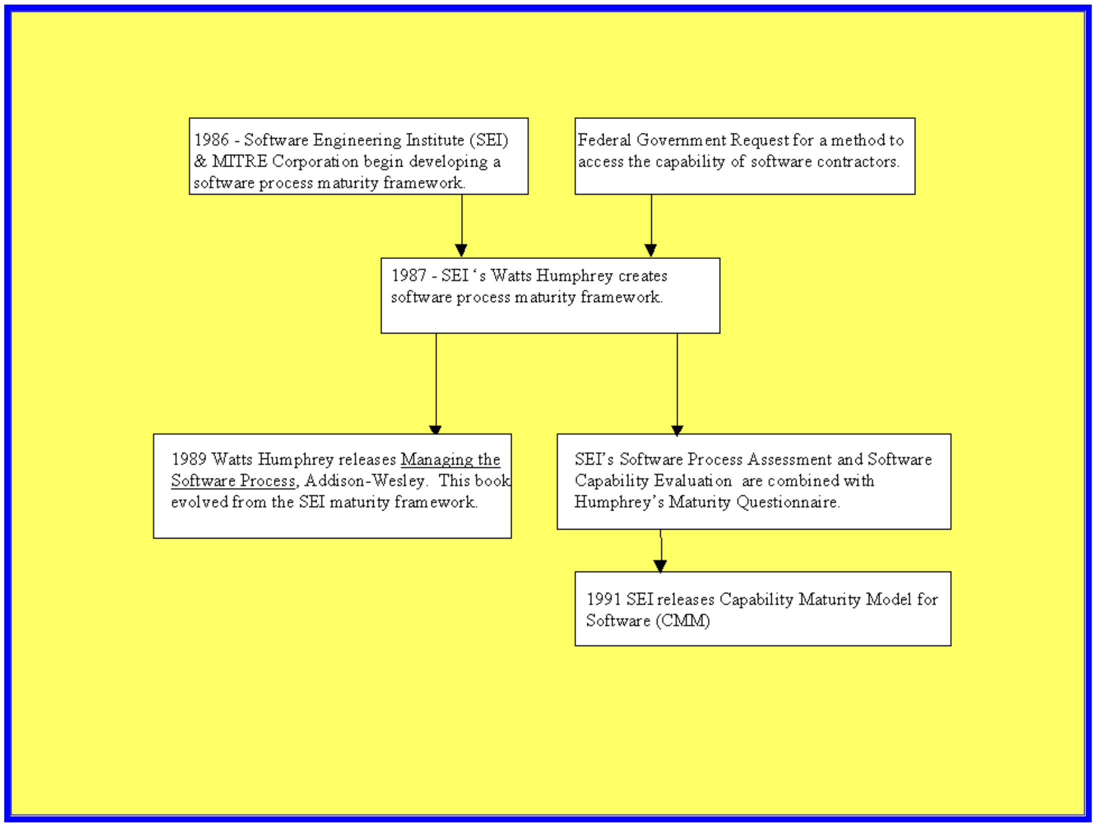

1986 - Software Engineering Institute (SEI) & MITRE Corporation begin developing a software process maturity framework.
Federal Government Request for a method to access the capability of software contractors.
1987 - SEI 's Watts Humphrey creates software process maturity framework.
1989 Watts Humphrey releases Managing the Software Process, Addison-Wesley. This book evolved from the SEI maturity framework.
SEI's Software Process Assessment and Software Capability Evaluation are combined with Humphrey's Maturity Questionnaire.
1991 SEI releases Capability Maturity Model for Software (CMM)

Figure 4: The History of the SEI CMM

## 3.2 CMM Levels

The CMM is comprised of five different levels indicating process maturity. Each level has its own criteria by which a software process is evaluated. These criteria are used to determine an organization's process maturity. The level in which an organization meets all of the criteria is assigned to that organization. This allows the organization to advertise that they are compliant with that particular SEI CMM Level.

The CMM levels are obtained through sequential achievement. For an organization to obtain CMM Level 4 it must have also achieved Level 2 and Level 3.

### 3.2.1 Level 1 - Initial

Level 1 is the initial CMM level. All software organizations are given a Level 1 without evaluation. If an organization is developing software they are at least at a Level 1. This is considered an immature process level from which an organization must quickly strive to surpass. Level 1 is frequently defined as chaos.

### 3.2.2 Level 2 - Repeatable

Level 2 is defined as repeatable. An organization must have the ability to repeat the processes by which they develop software. In order for this to occur the organization must have processes that are "defined, documented, practiced, trained, measured, enforced and improvable."[1 p.18]

These defined processes must include requirements management, project planning and tracking, subcontract management, quality assurance (QA), and software configuration management. This definition of an organization's process provides a framework of project management that allows the organization to set and follow realistic objectives and goals based on previous software project successes.

### 3.2.3 Level 3 - Defined

Level 3 indicates that an organization's process is defined. By defined, the CMM requires that the software processes and management processes be documented for an entire organization. Also required is the ability for personnel to understand these processes across all projects and that training is in place to ensure this comprehension.

In addition to an organization-wide defined set of software and management standards, the projects within an organization are allowed to tailor these processes to meet a project's specific needs. These changes would only be incorporated with appropriate training at the project level.

Two other larger factors for Level 3 are the tracking of quality metrics and verification of work products. The tracking of quality metrics must be in place at the project level as well as defined at the organization level. The verification of work products usually materializes in the form of Peer Reviews. Peer Reviews are performed to verify that the products being produced throughout the software process are accurate and complete. Documents describing the processes for conducting peer reviews are common artifacts in a Level 3 certified organization.

The main objective of Level 3 is the standardization and consistency of processes for both software and management across an organization's projects.  This is obtained by having an organization wide set of processes that can be tailored for each individual project as circumstances dictate.

### 3.2.4 Level 4 - Managed

Level 4 moves beyond the process definition of Level 3 by requiring the ability to measure the software products and processes.  The CMM requires the establishment of quality measurements across an organization.  These measurements must be collected into a database for analysis.

This organizational database of process and software quality metrics allows an organization to predict the results of their processes and the quality of both process and product.  Software Quality Management (SQM) allows predictive analysis of future projects as well as corrective action to be taken on projects that stray outside of the organizational process trends.

### 3.2.5 Level 5 - Optimizing

Level 5 builds on the SQM requirements of Level 4 to institute defect prevention and process improvement.  The quality metrics are taken a step further and analysis is performed to determine the cause of defects.  The identification of defects along with the identifications of strengths in both process and product allow corrective action to be taken to improve the product and the incorporation of process improvements to improve the process.

Process improvement is performed to remove common causes of defects found throughout an organization's projects. These improvements are distributed throughout the entire organization. Improvements can come not only in the form of process change but also technology change. Changes in technology are analyzed to determine their effect on the organization. Both process and technology changes are encouraged, planned for and managed as to not disrupt the stable development environment.

## 3.3 Key Process Areas (KPA)

Key Process Areas (KPA) govern each level in the CMM, except for level 1. The KPAs are obtained by achieving all of the goals within each KPA. When all KPAs have been addressed within a CMM Level then the organization has achieved that CMM level rating.

KPAs are organized by common features. The common features include:

- A commitment to perform
- Ability to perform
- Activities performed
- Measurement and analysis

Common features indicate whether the processes by which a software organization has implemented a particular KPA are effective. Each common feature contains key practices, which describe those activities that contribute most to the implementation of a common feature and in turn a KPA. The key practices are unique and specific to each key process area.

The CMM Levels and the relationships between key process areas, common features and key practices are shown in Figure 5.

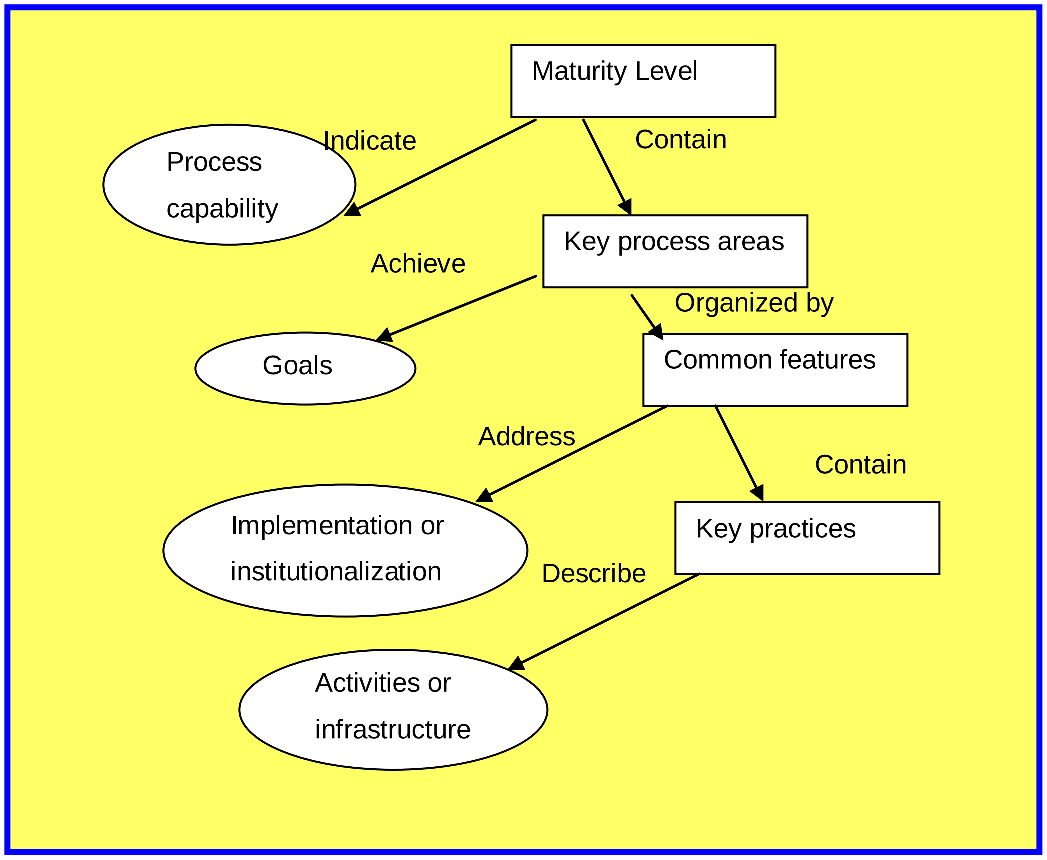


Figure 5: The CMM Structure [1 p. 31]

Although all KPAs were evaluated for both Process A and Process B, the following sections describe the KPAs that are addressed in this thesis.

### 3.3.1 Level 2 KPA Software Project Planning

This KPA has fifteen different activities defined to support three goals of the KPA. The first goal of this KPA is stated as follows, "Software estimates are documented for use in planning and tracking the software project."[1 p.134] The common feature "Activities Performed" contains activity nine. This activity states, "Estimates for the size of the software work products (or changes to the size of software work products) are derived according to a documented procedure." [1 p. 142]  This activity is mapped to the first goal of this KPA.

**Level 2** - Repeatable

**Key Process Area** - Software Project Planning

**Common Feature** - Activities Performed

**Key Practice** - Estimation for work product size

The common feature "Ability to Perform contains ability 4.  This ability states, "The software managers, software engineers, and other individuals involved in the software project planning are trained in the software estimating and planning procedures applicable to their areas of responsibility." [1 p.138]   This ability is mapped to all three goals of this KPA.

**Level 2** - Repeatable

**Key Process Area** - Software Project Planning

**Common Feature** - Ability to Perform

**Key Practice** - Software estimation training

## 3.3.2 Level 3 KPA Peer Reviews

This KPA has one commitment defined to support the two goals of the KPA. The first goal of this KPA is stated as follows, "Peer review activities are planned."[1 p.270] The common feature "Commitment to Perform" contains commitment 1, this commitment states, "The project follows a written organizational policy for performing peer reviews." [1 p. 271] This commitment is mapped to both goals of this KPA.

**Level 3** - Defined

**Key Process Area** -Peer Reviews

**Common Feature** - Commitment to Perform

**Key Practice** - Policy for peer reviews

# 4 Comparison of CLCS Processes

This section will discuss both Process A and Process B. For a comparison the SEI CMM KPAs covered under both processes are kept the same. This allows the reader to see a side-by-side comparison of the two processes. The magnitude of information required in evaluating a process for CMM and the amount of data produced from the evaluation prevents addressing all such data in this section. Instead, major violations of CMM in Process A will be pointed out and described.

The same KPAs detected as major violations in Process A will be addressed in the Process B section. These major violations where used to initiate the redirection which eventually led to a migration away from Process A to Process B. Although the SEI CMM evaluations of each process were performed on the entirety of Process A and Process B it should be kept in mind that this thesis focused on the Design Phase of each process.

## 4.1 Description of CLCS Process A

### 4.1.1 Process Overview

As stated in the introduction the CLCS project followed a Waterfall life cycle model with minimal process descriptions for a period of approximately three years. The lack of formal processes that qualified as CMM compliant along with the use of the Waterfall model resulted in a severe overrun of both schedule and budgets. Much of the blame for the inadequacies of this development effort lie with the lack of well written process, however, the use of the Waterfall Model prevented early detection of severe problems.

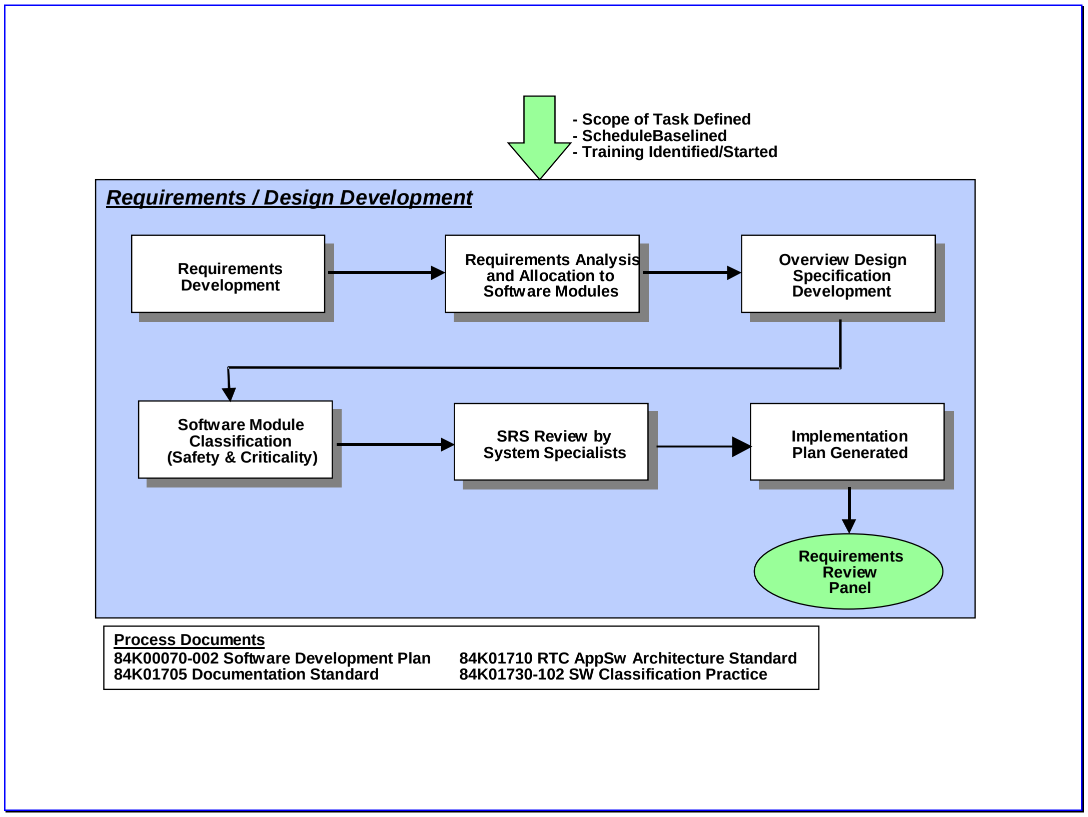
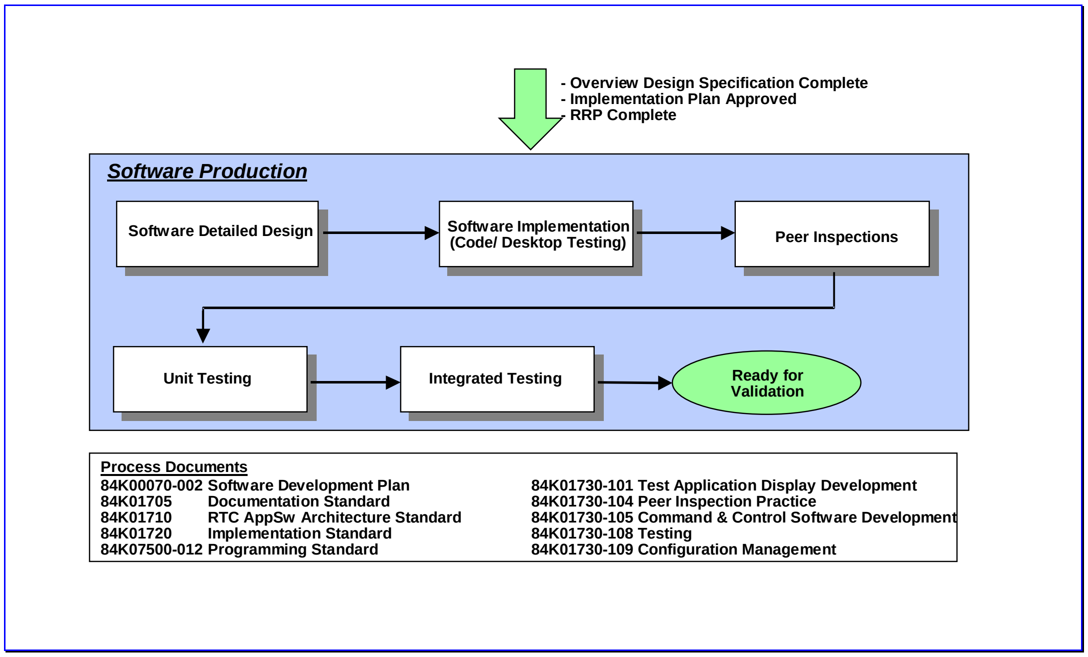

- Scope of Task Defined
- ScheduleBaselined
- Training Identified/Started
Requirements / Design Development
Requirements Development
Requirements Analysis and Allocation to Software Modules
Overview Design Specification Development
Software Module Classification (Safety & Criticality)
SRS Review by System Specialists
Implementation Plan Generated
Requirements Review Panel
Process Documents
84K00070-002 Software Development Plan
84K01705 Documentation Standard
84K01710 RTC AppSw Architecture Standard
84K01730-102 SW Classification Practice

Figure 6: Process A –Requirements / Design Development [17 p. 26]

As seen in Figure 6, Process A followed a Requirements Analysis, Design, and Implementation order of phases.  Although the phases were renamed the basic Waterfall approach can be easily detected.

The implementation portion of the model is shown in Figure 7.  Again, the Waterfall model can be seen with the lock step progression through Software Production phases.



Process Documents

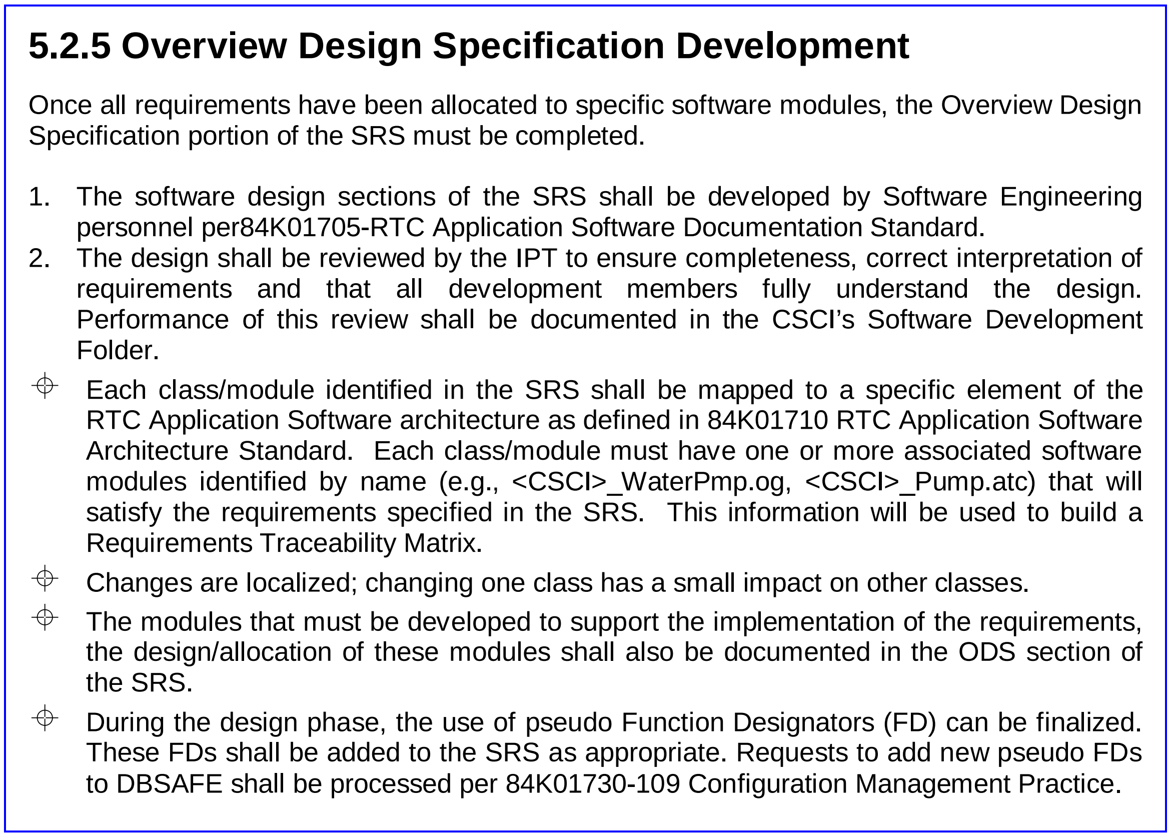
| | | | |
|---|---|---|---|
| 84K00070-002 | Software Development Plan | 84K01730-101 | Test Application Display Development |
| 84K01705 | Documentation Standard | 84K01730-104 | Peer Inspection Practice |
| 84K01710 | RTC AppSw Architecture Standard | 84K01730-105 | Command & Control Software Development |
| 84K01720 | Implementation Standard | 84K01730-108 | Testing |
| 84K07500-012 | Programming Standard | 84K01730-109 | Configuration Management |

Figure 7: Process A - Software Production Activities [17 p. 28]

Figures 6 and 7 show the basic Waterfall Model and appear to be the beginnings of a standard software development lifecycle. However, a software development effort requires more then just a model to follow. Software processes must be defined to guide the developers through the life cycle. In particular the industry has migrated toward following CMM compliant process and many of the features required to be present in such a model were previously presented in the Summary of SEI CMM section.

### 4.1.2 CMM Evaluation of Design Phase

This section describes the Process A design and its evaluation under SEI CMM. Process A falls short when it comes to defining processes and specifically CMM compliant processes. One of the reasons for this may lie in the brevity of the processes themselves. The main focus of this thesis was the design phase of software development, Figure 8 and Figure 9 are the complete preliminary and detailed design phases of Process A as defined in NASA's Software Development Plan, Volume II Revision B. [17]



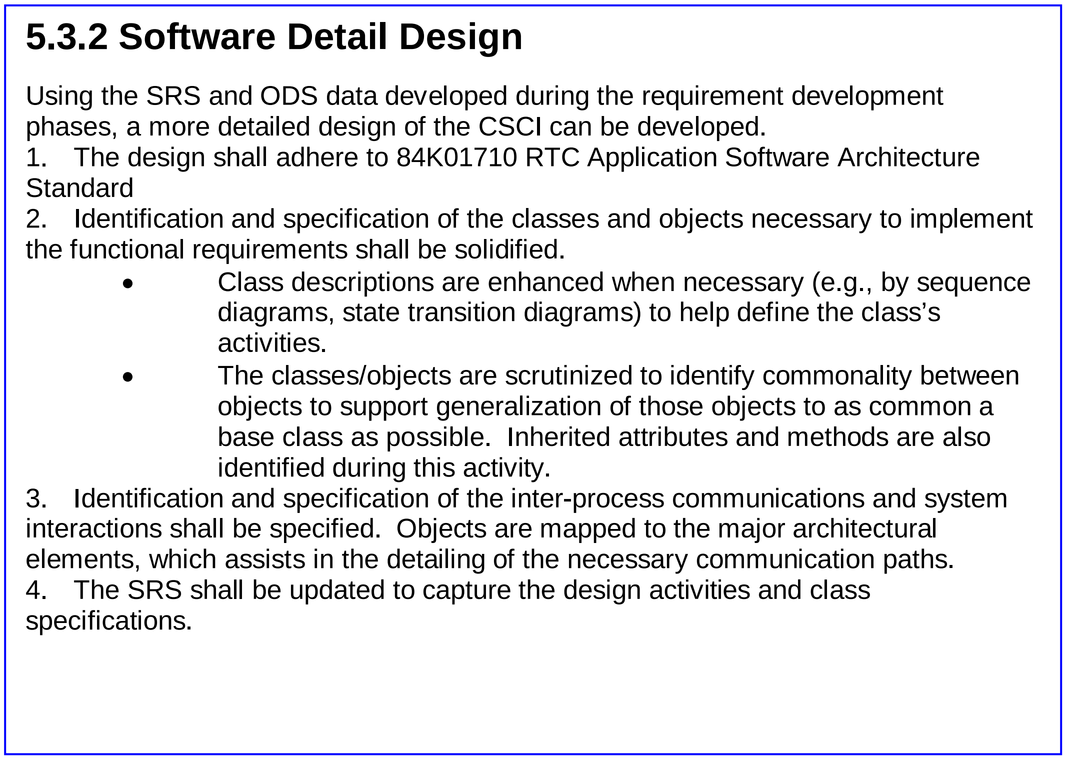
**5.2.5 Overview Design Specification Development**

Once all requirements have been allocated to specific software modules, the Overview Design Specification portion of the SRS must be completed.

1. The software design sections of the SRS shall be developed by Software Engineering personnel per84K01705-RTC Application Software Documentation Standard.
2. The design shall be reviewed by the IPT to ensure completeness, correct interpretation of requirements and that all development members fully understand the design. Performance of this review shall be documented in the CSCI's Software Development Folder.

- ⌖ Each class/module identified in the SRS shall be mapped to a specific element of the RTC Application Software architecture as defined in 84K01710 RTC Application Software Architecture Standard. Each class/module must have one or more associated software modules identified by name (e.g., <CSCI>_WaterPmp.og, <CSCI>_Pump.atc) that will satisfy the requirements specified in the SRS. This information will be used to build a Requirements Traceability Matrix.
- ⌖ Changes are localized; changing one class has a small impact on other classes.
- ⌖ The modules that must be developed to support the implementation of the requirements, the design/allocation of these modules shall also be documented in the ODS section of the SRS.
- ⌖ During the design phase, the use of pseudo Function Designators (FD) can be finalized. These FDs shall be added to the SRS as appropriate. Requests to add new pseudo FDs to DBSAFE shall be processed per 84K01730-109 Configuration Management Practice.

Figure 8: Process A - Preliminary Design [17 p.30]



**5.3.2 Software Detail Design**

Using the SRS and ODS data developed during the requirement development phases, a more detailed design of the CSCI can be developed.

1. The design shall adhere to 84K01710 RTC Application Software Architecture Standard
2. Identification and specification of the classes and objects necessary to implement the functional requirements shall be solidified.
    - Class descriptions are enhanced when necessary (e.g., by sequence diagrams, state transition diagrams) to help define the class's activities.
    - The classes/objects are scrutinized to identify commonality between objects to support generalization of those objects to as common a base class as possible. Inherited attributes and methods are also identified during this activity.
3. Identification and specification of the inter-process communications and system interactions shall be specified. Objects are mapped to the major architectural elements, which assists in the detailing of the necessary communication paths.
4. The SRS shall be updated to capture the design activities and class specifications.

Figure 9: Process A - Detail Design [17 p.31]

### 4.1.2.1 Level 2 KPA Software Project Planning

#### 4.1.2.1.1 Activities Performed (Work Products)

The SEI CMM Level 2 KPA, Software Project Planning (KPA-SPP), is one of five KPAs for Level 2. KPA-SPP directly effects the design phase of a software process. The Process A design phase does not satisfy all three of the goals required to meet this KPA.

The first major discrepancy is the lack of defined work products. The CMM requires under Goal 1, Activity 9 of KPA-SPP that the estimation of the size of work products be defined. Process A not only lacks a defined method of estimation it also lacks defined work products. Process A attempts to define Preliminary Design work products in Figure 8 Item 3: which indicates the need for classes and in Item 4 the documentation of these classes in the Overview Design Specification (ODS). In Figure 9 an attempt is made at identifying Detailed Design work products; class attributes and methods, and Inter-Process Communications (IPC).

Although these items are called out they fall short of being complete work products. Where is this information captured? What format is it in? Are there tools to help develop these items? The analysis of this KPA in relation to Process A indicated a need to identify all work products, a modeling language, and a tool to create the final design phase work artifact; a design document.

The CMM also requires, under Goal 2 Activity 8 of KPA-SPP, that the identification of software work products necessary to establish and maintain control of the software project be addressed and defined. Due to the lack of work products described Process A also falls short of meeting this goal.

#### 4.1.2.1.2 Ability to Perform (Estimation training)

KPA SPP Ability to Perform 4 requires training for all software engineers and managers to be trained in software estimating and planning procedures as stated in the Software Development Plan (SDP). Process A lacked any software estimation what so ever. Neither Figure 8 nor Figure 9 indicate any software estimation, this resulted directly in Process A lacking software estimation techniques. The involvement of Software Quality is also left out of the process and as a result Software Quality was not involved in the design phase and did not appear until later in the testing phases.

#### 4.1.2.2 Level 3 KPA Peer Reviews

Another serious flaw with Process A is the lack of review definition. Although Figure 8 Item 2 indicates that the preliminary design shall be reviewed there is no indication as to how to perform such a review. CMM Level 3 KPA ***Peer Reviews*** Commitment #1 requires projects to follow a written organizational policy for performing peer reviews. Absent in the Process A design phase are any instructions concerning how to perform the preliminary design peer review called for in Figure 8 Item 2. In addition, Figure 6 does not indicate a peer review of the preliminary design. This causes confusion as to which process definition is correct, the pictorial view or the textual description.

In addition there in no mention of a review of the detailed design created via Figure 9. As a result there is not a review of the work products produced by the detailed design phase. This results in inconsistency between the preliminary design phase and the detailed design phase and creates chaos in the process definition itself. The lack of peer review definition is a serious flaw in the Process A and was identified as a major reason to migrate to Process B. Interesting to note is the fact that a peer review does not occur in Process A until after the software has been implemented. This can be seen in Figure 7. This allowed the entire design and implementation phases to proceed without a single peer review.

#### 4.1.2.3 Process A CMM Evaluation Summary

Although many violations of KPA common features exists the aforementioned are serious enough on their own that Process A does not even qualify for SEI CMM Level 2. Process A did not meet any of the KPAs addressed in this thesis. The absence of Peer Review processes and the confusion between the textual and pictorial descriptions of the process prevent Process A from fulfilling the CMM requirements for Peer Reviews.

## 4.2 Description of CLCS Process B

### 4.2.1 Process Overview

As a result of the problems produced by Process A, the CLCS project had to find solutions to their development dilemma. After years of complaints from software developers more familiar with the latest software development models and processes, the opportunity to implement alternative solutions was at hand. Project management tasked a few individuals including the author to come up with a better process that could be followed. The result was Process B, Figure 10.

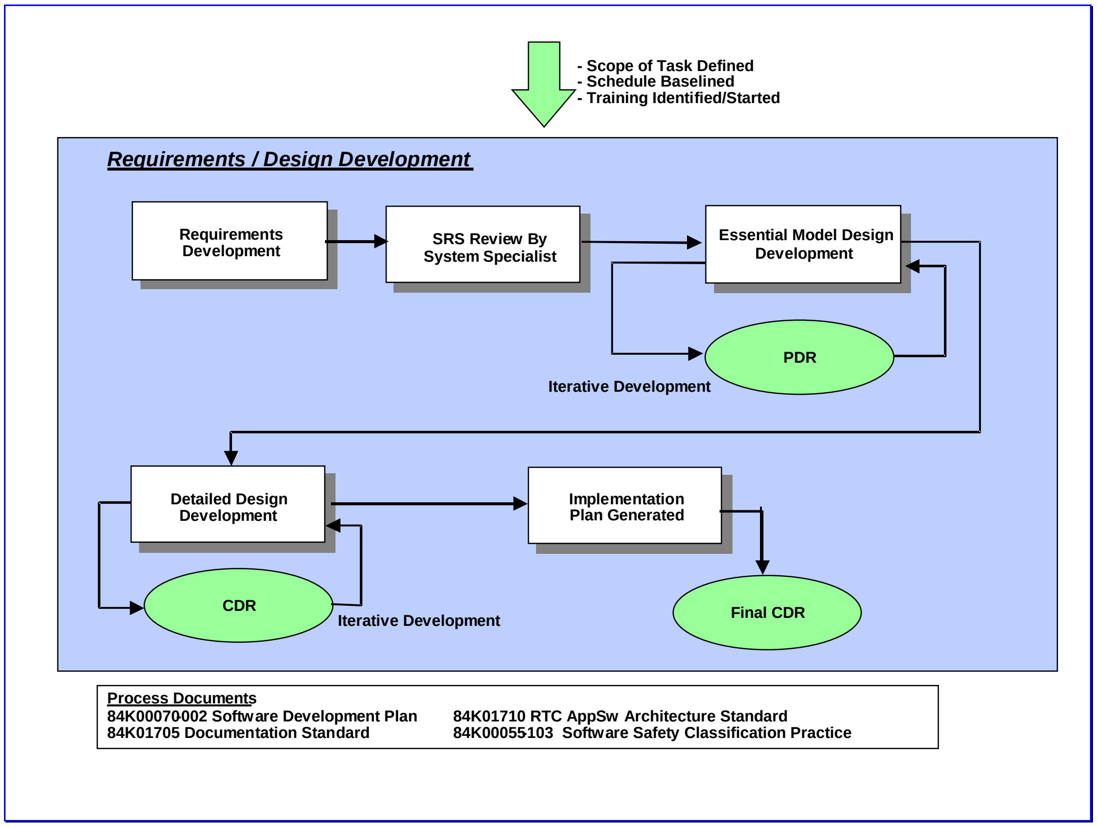

- Scope of Task Defined
- Schedule Baselined
- Training Identified/Started
Requirements / Design Development
Requirements Development
SRS Review By System Specialist
Essential Model Design Development
PDR
Iterative Development
Detailed Design Development
Implementation Plan Generated
CDR
Iterative Development
Final CDR
Process Documents
84K00070002 Software Development Plan
84K01705 Documentation Standard
84K01710 RTC AppSw Architecture Standard
84K00055103 Software Safety Classification Practice

Figure 10: Process B - Design Development [18 p. 26]

Process B created major changes for the CLCS development team. The most significant change being the migration from a Waterfall model to an Iterative Model and from an ill-defined process to a very detailed process. The CMM deficiencies in Process A were addressed in Process B to bring CLCS into CMM compliance. Due to the familiarity of terminology used during Process A many of the same terms and acronyms were carried over to Process B. This was to facilitate the learning curve of a new process and to create as little impact to the budget and schedule as possible.

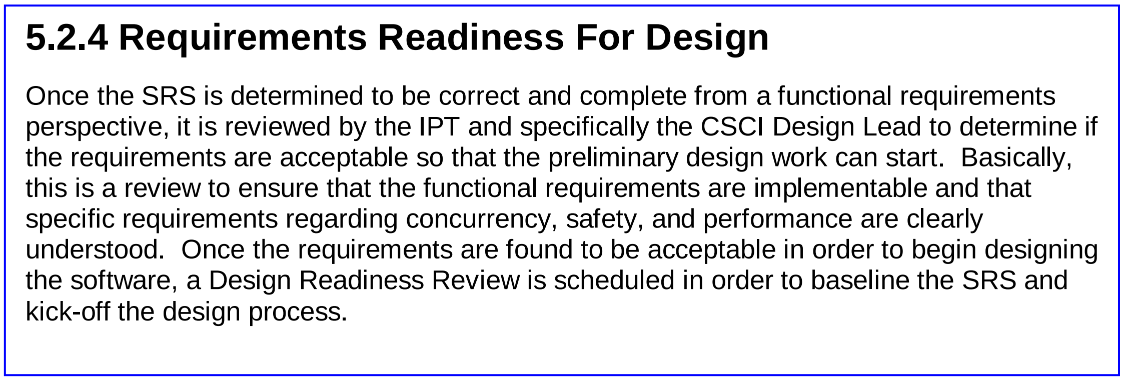
### 4.2.2 CMM Evaluation of Design Phase

Figure 10 shows the iterative design phase of Process B. Entry into this phase occurred via a review of the software requirements. The CSCI team was able to iterate through the preliminary or essential design with each pass going through a preliminary design review. The detailed design could be iteratively developed in the same manner with each iteration being presented to a critical design review. The presentation of design products to a formal review team allowed corrections to the design before implementation occurred. Each CSCI passed though multiple cycles of the RUP. Each design phase was encompassed within a cycle and could include multiple iterations, this allowed for iterative development and incremental deliveries to the end user.

For the CLCS software engineer, another significant difference between Process A and Process B was the sheer amount of information in the form of guidance. Although large of amounts of information do not indicate a well-written process and certainly do not guarantee CMM compliance the less then four hundred words and two diagrams in Process A left much room for improvement. Process B added information in the several ways.

1) To support this migration and process change an entirely new document was created. In Process A the entire scope of the design phase was included in the SDP. In Process B the design phase was expanded in the SDP and the details of the work products and peer reviews were created in a separate document entitled, **SDP Supplement: Technical Review Practice**.

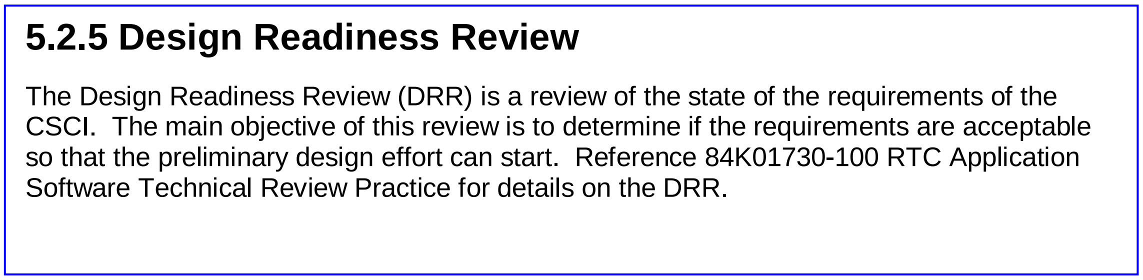
2) The CSCI's software lead was given the responsibility for determining the state of the requirements they were responsible to design from. In Process A this was not specifically spelled out and in most cases the CSCI received requirements that could not be understood. Figure 11 and Figure 12 explicitly state the role of the CSCI lead.



**5.2.4 Requirements Readiness For Design**

Once the SRS is determined to be correct and complete from a functional requirements perspective, it is reviewed by the IPT and specifically the CSCI Design Lead to determine if the requirements are acceptable so that the preliminary design work can start. Basically, this is a review to ensure that the functional requirements are implementable and that specific requirements regarding concurrency, safety, and performance are clearly understood. Once the requirements are found to be acceptable in order to begin designing the software, a Design Readiness Review is scheduled in order to baseline the SRS and kick-off the design process.

Figure 11: Process B - Requirements Readiness [18 p. 27]



**5.2.5 Design Readiness Review**

The Design Readiness Review (DRR) is a review of the state of the requirements of the CSCI. The main objective of this review is to determine if the requirements are acceptable so that the preliminary design effort can start. Reference 84K01730-100 RTC Application Software Technical Review Practice for details on the DRR.

Figure 12: Process B - DRR [18 p. 28]

3) Two quick reference cards were created to aid during training of Process B and Microsoft PowerPoint was used to create peer review presentation templates.

A Design Workflow Quick Reference card, Figure 13, that highlighted the new design process was used during process training and subsequently during the actual design phase of each CSCI. Details of Figure 13 can be found in Appendix A. The second quick reference card created, Figure 14, was used to highlight the usage of the Unified Modeling Language (UML). Although Process A was to be used with UML as the design modeling language, the process itself did not indicate this. In addition, Process A provided no training for the usage of UML. This was corrected under Process B, which provided training as well as the quick reference card for use during actual design development.

The presentation templates forced each CSCI to address and present the same information at each requirements review, preliminary design review and critical design review. Not only did this create consistency within a CSCI but it required each CSCI within the CLCS project to produce the same types of artifacts as required by Process B.

## 4.2.2.1 Level 2 KPA Software Project Planning

### 4.2.2.1.1 Activities Performed ( Work Products )

In addressing the deficiencies of Process A the first priority was to define the work products which needed to be produced. The lack of defined work products in Process A created great diversity among CSCIs and to be able to be CMM compliant at any level this needed to be brought under control.

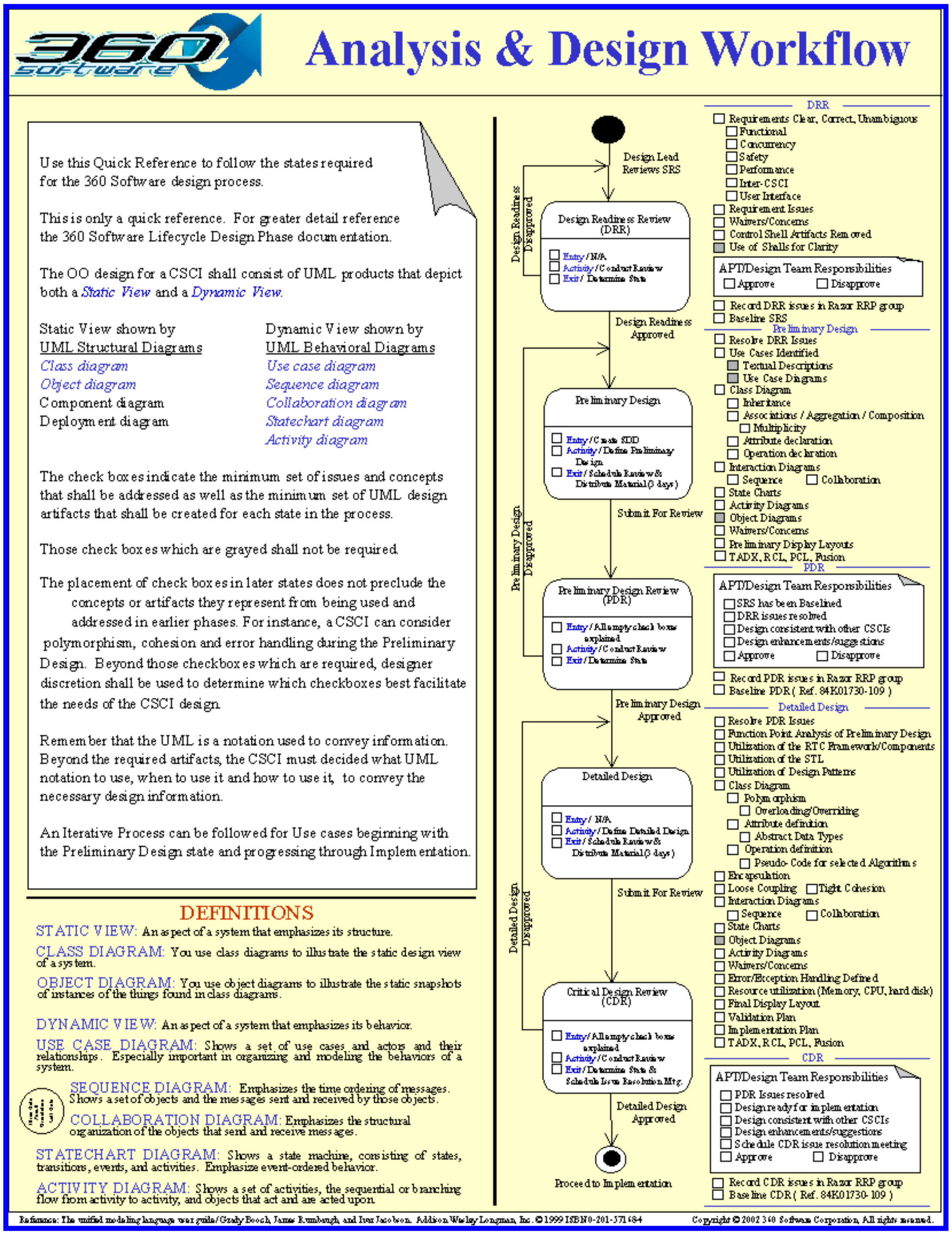
Process B defined work products for each Process Workflow. In particular work products were identified for the Design Workflow for both the Elaboration and Construction phases. The work products are outlined for the Requirements Analysis, Preliminary Design and Detailed Design.

General descriptions are found in the SDP, Figures 11, 12, 15 and 16, and specific detail is provided in the SDP Supplement.

These work products include entry/exit criteria for each phase and verification review. The work products are also found in the Quick Reference Card for the Design Workflow found in Figure 13. The detailed specification of what is required for each phase and at each review provides no room for interpretation. The CSCIs had to produce all of the required work products, this resulted in consistency across the entire project.

The greatest improvement in Process B came in the form of a Software Design Document (SDD). This document had to be created by each CSCI from a project level template. This template ensured consistency across all CSCIs. The SDD contained all of the UML work products defined in the process as required. The UML quick reference guide was used to aid software engineers in the development of the UML work products. By mandating the use of the SDD template and the UML quick reference guide Process B not only enforced consistency in the look of the SDD but also in the actual UML artifacts contained in it.

The real point of this KPA is to provide ways to estimate the size of work products. In Process A there were no defined work products so the estimation process was moot. In Process B the work products are defined in detail, therefore the processes for software estimation and planning could be defined.



# Analysis & Design Workflow

Use this Quick Reference to follow the states required for the 360 Software design process.

This is only a quick reference. For greater detail reference the 360 Software Lifecycle Design Phase documentation.

The OO design for a CSCI shall consist of UML products that depict both a *Static View* and a *Dynamic View*.

| Static View shown by | Dynamic View shown by |
|---|---|
| UML Structural Diagrams | UML Behavioral Diagrams |
| *Class diagram* | *Use case diagram* |
| *Object diagram* | *Sequence diagram* |
| Component diagram | *Collaboration diagram* |
| Deployment diagram | *Statechart diagram* |
| | *Activity diagram* |

The check boxes indicate the minimum set of issues and concepts that shall be addressed as well as the minimum set of UML design artifacts that shall be created for each state in the process.

Those check boxes which are grayed shall not be required.

The placement of check boxes in later states does not preclude the concepts or artifacts they represent from being used and addressed in earlier phases. For instance, a CSCI can consider polymorphism, cohesion and error handling during the Preliminary Design. Beyond those checkboxes which are required, designer discretion shall be used to determine which checkboxes best facilitate the needs of the CSCI design.

Remember that the UML is a notation used to convey information. Beyond the required artifacts, the CSCI must decided what UML notation to use, when to use it and how to use it, to convey the necessary design information.

An Iterative Process can be followed for Use cases beginning with the Preliminary Design state and progressing through Implementation.

## DEFINITIONS

STATIC VIEW: An aspect of a system that emphasizes its structure.

CLASS DIAGRAM: You use class diagrams to illustrate the static design view of a system.

OBJECT DIAGRAM: You use object diagrams to illustrate the static snapshots of instances of the things found in class diagrams.

DYNAMIC VIEW: An aspect of a system that emphasizes its behavior.

USE CASE DIAGRAM: Shows a set of use cases and actors and their relationships. Especially important in organizing and modeling the behaviors of a system.

SEQUENCE DIAGRAM: Emphasizes the time ordering of messages. Shows a set of objects and the messages sent and received by those objects.

COLLABORATION DIAGRAM: Emphasizes the structural organization of the objects that send and receive messages.

STATECHART DIAGRAM: Shows a state machine, consisting of states, transitions, events, and activities. Emphasize event-ordered behavior.

ACTIVITY DIAGRAM: Shows a set of activities, the sequential or branching flow from activity to activity, and objects that act and are acted upon.

Design Lead Reviews SRS
Design Readiness Review (DRR)
☐ Entry / N/A
☐ Activity / Conduct Review
☐ Exit / Determine State
Design Readiness Disapproved
Design Readiness Approved
Preliminary Design
☐ Entry / Create SDD
☐ Activity / Define Preliminary Design
☐ Exit / Schedule Review & Distribute Material (3 days)
Submit For Review
Preliminary Design Disapproved
Preliminary Design Review (PDR)
☐ Entry / All empty check boxes explained
☐ Activity / Conduct Review
☐ Exit / Determine State
Preliminary Design Approved
Detailed Design
☐ Entry / N/A
☐ Activity / Define Detailed Design
☐ Exit / Schedule Review & Distribute Material (3 days)
Submit For Review
Detailed Design Disapproved
Critical Design Review (CDR)
☐ Entry / All empty check boxes explained
☐ Activity / Conduct Review
☐ Exit / Determine State & Schedule Issue Resolution Mtg.
Detailed Design Approved
Proceed to Implementation

DRR

- ☐ Requirements Clear, Correct, Unambiguous
  - ☐ Functional
  - ☐ Concurrency
  - ☐ Safety
  - ☐ Performance
  - ☐ Inter-CSCI
  - ☐ User Interface
- ☐ Requirement Issues
- ☐ Waivers/Concerns
- ☐ Control Shell Artifacts Removed
- ▨ Use of Shalls for Clarity

APT/Design Team Responsibilities
☐ Approve ☐ Disapprove

- ☐ Record DRR issues in Razor RRP group
- ☐ Baseline SRS

Preliminary Design

- ☐ Resolve DRR Issues
- ☐ Use Cases Identified
  - ▨ Textual Descriptions
  - ▨ Use Case Diagrams
- ☐ Class Diagram
  - ☐ Inheritance
  - ☐ Associations / Aggregation / Composition
    - ☐ Multiplicity
  - ☐ Attribute declaration
  - ☐ Operation declaration
- ☐ Interaction Diagrams
  - ☐ Sequence ☐ Collaboration
- ☐ State Charts
- ☐ Activity Diagrams
- ▨ Object Diagrams
- ☐ Waivers/Concerns
- ☐ Preliminary Display Layouts
- ☐ TADX, RCL, PCL, Fusion

PDR

APT/Design Team Responsibilities
- ☐ SRS has been Baselined
- ☐ DRR issues resolved
- ☐ Design consistent with other CSCIs
- ☐ Design enhancements/suggestions
- ☐ Approve ☐ Disapprove

- ☐ Record PDR issues in Razor RRP group
- ☐ Baseline PDR ( Ref. 84K01730-109 )

Detailed Design

- ☐ Resolve PDR Issues
- ☐ Function Point Analysis of Preliminary Design
- ☐ Utilization of the RTC Framework/Components
- ☐ Utilization of the STL
- ☐ Utilization of Design Patterns
- ☐ Class Diagram
  - ☐ Polymorphism
    - ☐ Overloading/Overriding
  - ☐ Attribute definition
    - ☐ Abstract Data Types
  - ☐ Operation definition
    - ☐ Pseudo-Code for selected Algorithms
- ☐ Encapsulation
- ☐ Loose Coupling ☐ Tight Cohesion
- ☐ Interaction Diagrams
  - ☐ Sequence ☐ Collaboration
- ☐ State Charts
- ▨ Object Diagrams
- ☐ Activity Diagrams
- ☐ Waivers/Concerns
- ☐ Error/Exception Handling Defined
- ☐ Resource utilization (Memory, CPU, hard disk)
- ☐ Final Display Layout
- ☐ Validation Plan
- ☐ Implementation Plan
- ☐ TADX, RCL, PCL, Fusion

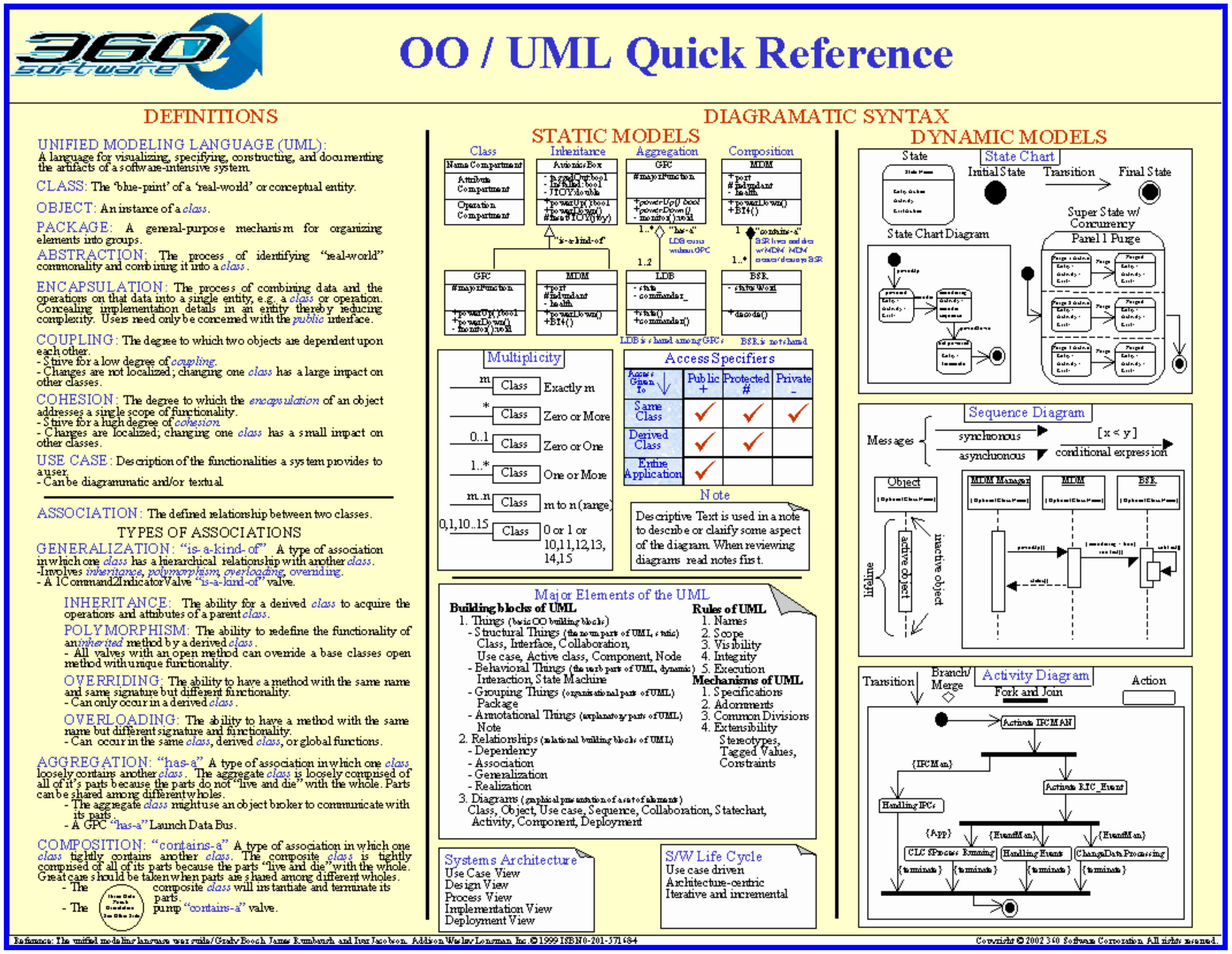
CDR

APT/Design Team Responsibilities
- ☐ PDR Issues resolved
- ☐ Design ready for implementation
- ☐ Design consistent with other CSCIs
- ☐ Design enhancements/suggestions
- ☐ Schedule CDR issue resolution meeting
- ☐ Approve ☐ Disapprove

- ☐ Record CDR issues in Razor RRP group
- ☐ Baseline CDR ( Ref. 84K01730-109 )

Reference: The unified modeling language user guide/Grady Booch, James Rumbaugh, and Ivar Jacobson. Addison Wesley Longman, Inc. ©1999 ISBN0-201-571684

Figure 13: Process B - Analysis & Design Workflow [4]



OO / UML Quick Reference

DEFINITIONS

UNIFIED MODELING LANGUAGE (UML): A language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system.

CLASS: The 'blue-print' of a 'real-world' or conceptual entity.

OBJECT: An instance of a *class*.

PACKAGE: A general-purpose mechanism for organizing elements into groups.

ABSTRACTION: The process of identifying "real-world" commonality and combining it into a *class*.

ENCAPSULATION: The process of combining data and the operations on that data into a single entity, e.g. a *class* or operation. Concealing implementation details in an entity thereby reducing complexity. Users need only be concerned with the *public* interface.

COUPLING: The degree to which two objects are dependent upon each other.
- Strive for a low degree of *coupling*.
- Changes are not localized; changing one *class* has a large impact on other classes.

COHESION: The degree to which the *encapsulation* of an object addresses a single scope of functionality.
- Strive for a high degree of *cohesion*.
- Changes are localized; changing one *class* has a small impact on other classes.

USE CASE: Description of the functionalities a system provides to a user.
- Can be diagrammatic and/or textual.

ASSOCIATION: The defined relationship between two classes.

TYPES OF ASSOCIATIONS

GENERALIZATION: "is-a-kind-of" A type of association in which one *class* has a hierarchical relationship with another *class*.
-Involves *inheritance*, *polymorphism*, *overloading*, *overriding*.
- A 1Command2IndicatorValve "is-a-kind-of" valve.

INHERITANCE: The ability for a derived *class* to acquire the operations and attributes of a parent *class*.

POLYMORPHISM: The ability to redefine the functionality of an *inherited* method by a derived *class*.
- All valves with an open method can override a base classes open method with unique functionality.

OVERRIDING: The ability to have a method with the same name and same signature but different functionality.
- Can only occur in a derived *class*.

OVERLOADING: The ability to have a method with the same name but different signature and functionality.
- Can occur in the same *class*, derived *class*, or global functions.

AGGREGATION: "has-a" A type of association in which one *class* loosely contains another *class*. The aggregate *class* is loosely comprised of all of it's parts because the parts do not "live and die" with the whole. Parts can be shared among different wholes.
- The aggregate *class* might use an object broker to communicate with its parts.
- A GPC "has-a" Launch Data Bus.

COMPOSITION: "contains-a" A type of association in which one *class* tightly contains another *class*. The composite *class* is tightly comprised of all of its parts because the parts "live and die" with the whole. Great care should be taken when parts are shared among different wholes.
- The composite *class* will instantiate and terminate its parts.
- The pump "contains-a" valve.

DIAGRAMATIC SYNTAX

STATIC MODELS

Multiplicity

| | | |
|---|---|---|
| m | Class | Exactly m |
| * | Class | Zero or More |
| 0..1 | Class | Zero or One |
| 1..* | Class | One or More |
| m..n | Class | m to n (range) |
| 0,1,10..15 | Class | 0 or 1 or 10,11,12,13, 14,15 |

Access Specifiers

| Access Given To | Public + | Protected # | Private - |
|---|---|---|---|
| Same Class | ✓ | ✓ | ✓ |
| Derived Class | ✓ | ✓ | |
| Entire Application | ✓ | | |

Note

Descriptive Text is used in a note to describe or clarify some aspect of the diagram. When reviewing diagrams read notes first.

Major Elements of the UML

**Building blocks of UML**
1. Things (basic OO building blocks)
   - Structural Things (the noun part of UML, static)
     Class, Interface, Collaboration, Use case, Active class, Component, Node
   - Behavioral Things (the verb part of UML, dynamic)
     Interaction, State Machine
   - Grouping Things (organizational part of UML)
     Package
   - Annotational Things (explanatory parts of UML)
     Note
2. Relationships (relational building blocks of UML)
   - Dependency
   - Association
   - Generalization
   - Realization
3. Diagrams (graphical presentation of a set of elements)
   Class, Object, Use case, Sequence, Collaboration, Statechart, Activity, Component, Deployment

**Rules of UML**
1. Names
2. Scope
3. Visibility
4. Integrity
5. Execution

**Mechanisms of UML**
1. Specifications
2. Adornments
3. Common Divisions
4. Extensibility
   Stereotypes, Tagged Values, Constraints

Systems Architecture
Use Case View
Design View
Process View
Implementation View
Deployment View

S/W Life Cycle
Use case driven
Architecture-centric
Iterative and incremental

DYNAMIC MODELS

Reference: The unified modeling language user guide/Grady Booch, James Rumbaugh, and Ivar Jacobson. Addison Wesley Longman, Inc. ©1999 ISBN 0-201-57168-4

Copyright © 2002 360 Software Corporation. All rights reserved.

Figure 14: Process B - OO/UML [4]

#### 4.2.2.1.2 Ability to Perform (Estimation training)

For this KPA the Ability to Perform 4 requires all software engineers and managers to be trained in software estimating and planning procedures. In Process B it was decided to perform Function Point Analysis (FPA) for software estimation and planning. This approach was documented and detailed training was provided to those responsible for performing the FPA.

In addition to providing training the process also dictated that FPA be part of the entry criteria to the Design Readiness Review (DRR). This enforced the process and ensured that FPA had actually been performed and that it was reviewed to ensure accuracy. The DRR was a review required for the transition from Process A to Process B and will be described in the next section. Figure 13 fails to indicate the requirement of the FPA as entry criteria to the DRR.

### 4.2.2.2 Level 3 KPA Peer Reviews

Commitment 1 for this KPA requires projects to follow a written organizational policy for performing peer reviews. Process B introduced three new peer reviews in addition to what Process A already had in place. Each of these reviews required specific work products to be reviewed, forcing each CSCI to become consistent. Each CSCI was also required to use a PowerPoint review template that forced consistency in the manner in which the now consistent work products were presented.

#### 4.2.2.2.1 Design Readiness Review (DRR)

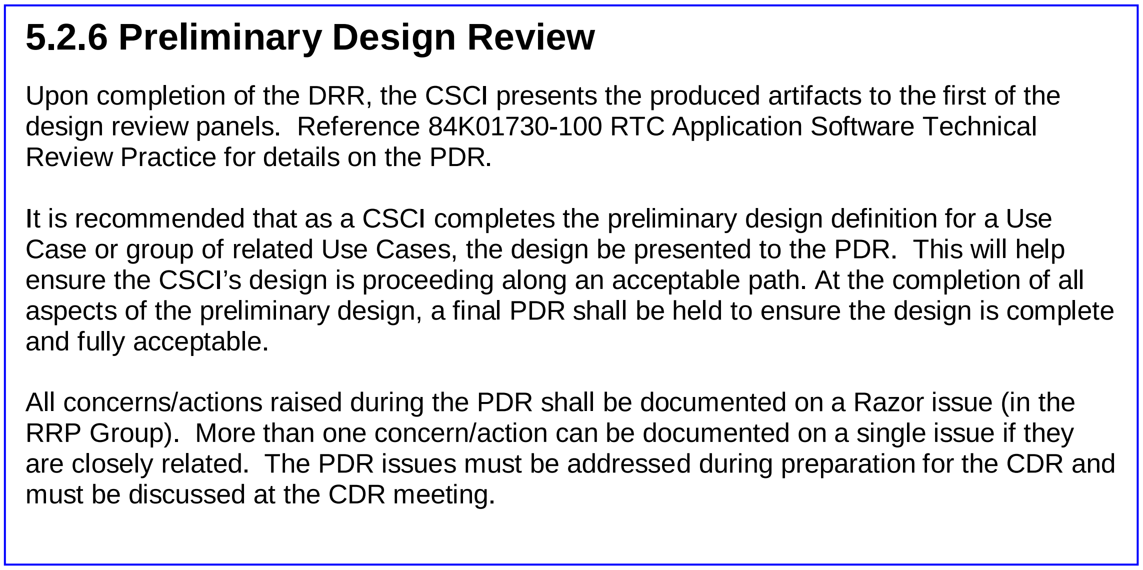
The first review added was the Design Readiness Review, Figure 12. This was in effect a Software Requirements Review and was performed during the Transition phase of the Analysis & Design Workflow. This can be seen on Figure 13. This review has a non-traditional name due to the fact the during Process A a requirements review had already taken place. As in all cases of real world software development a process has to take into account real world schedule and politics. Process B could not "repeat" a requirements review due to political reasons, thus the design readiness review was created.

This review allowed the CSCI design lead to give a final acceptance of the requirements. Since many of the CSCI requirements were not complete this allowed the software organization to request additional analysis of the requirements without indicating on the schedules that the original requirements review under Process A was null and void.

#### 4.2.2.2.2 Preliminary Design Review (PDR)

The PDR is to be performed during the Elaboration Phase of the Design Workflow for Process B. This review serves the same purpose of the Waterfall PDR in Process A, but can be performed multiple times during the iterations through the system.

The highlights of this Review are seen in Figure 15 from the Process B SDP, with the specific details covered in the SDP Supplement.



**5.2.6 Preliminary Design Review**

Upon completion of the DRR, the CSCI presents the produced artifacts to the first of the design review panels. Reference 84K01730-100 RTC Application Software Technical Review Practice for details on the PDR.

It is recommended that as a CSCI completes the preliminary design definition for a Use Case or group of related Use Cases, the design be presented to the PDR. This will help ensure the CSCI's design is proceeding along an acceptable path. At the completion of all aspects of the preliminary design, a final PDR shall be held to ensure the design is complete and fully acceptable.

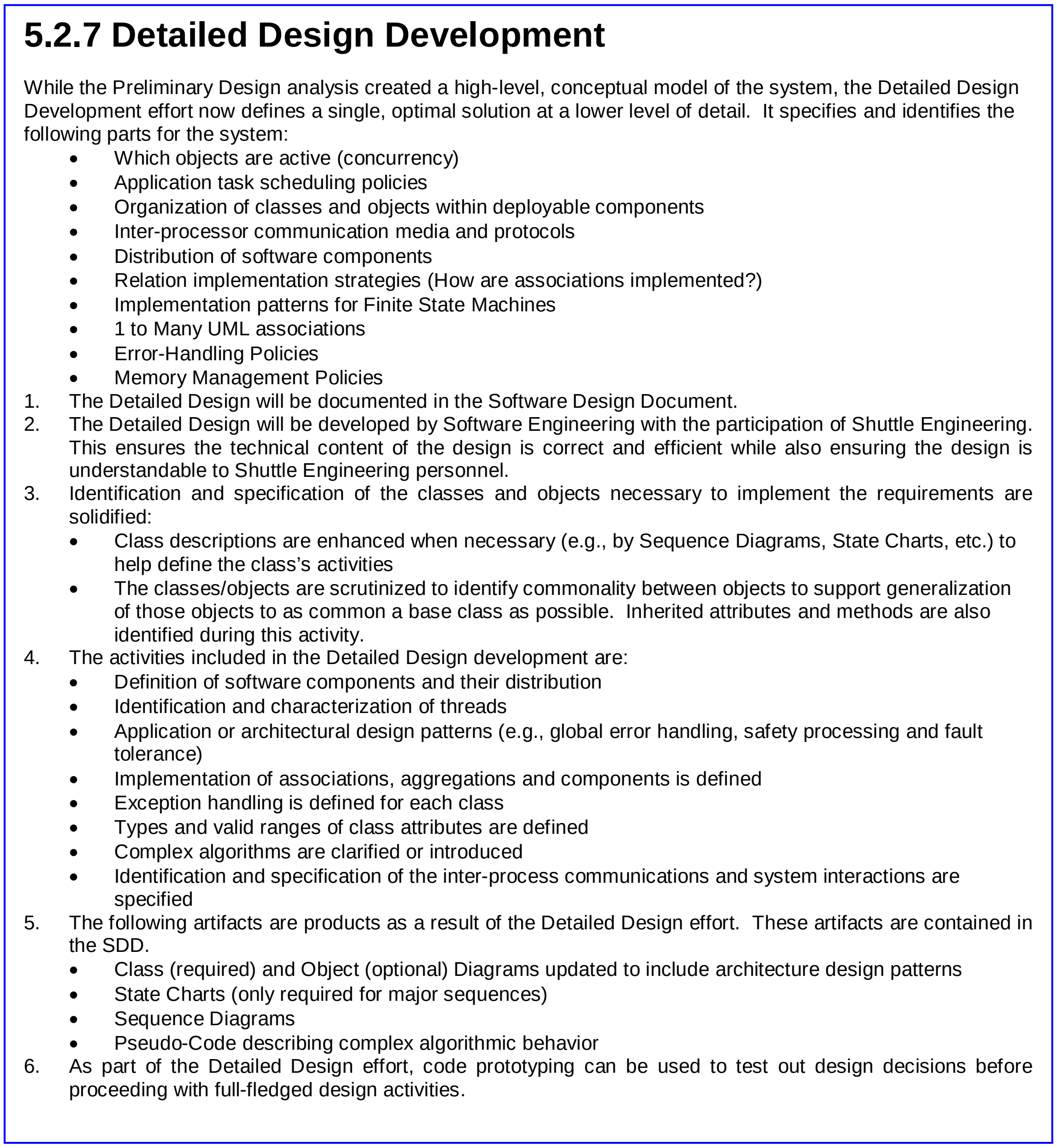
All concerns/actions raised during the PDR shall be documented on a Razor issue (in the RRP Group). More than one concern/action can be documented on a single issue if they are closely related. The PDR issues must be addressed during preparation for the CDR and must be discussed at the CDR meeting.

Figure 15: Process B - PDR [18 p. 28]

#### 4.2.2.2.3 Critical Design Review (CDR)

Like the PDR the CDR was added to the Design Workflow and is to be performed during the Elaboration Phase. This review serves the same purpose of the Waterfall CDR in Process A, but can be performed multiple time during the iterations through the system.

The highlights of this Review are seen in Figure 16 from the Process B SDP, with the specific details covered in the SDP Supplement.



### 5.2.7 Detailed Design Development

While the Preliminary Design analysis created a high-level, conceptual model of the system, the Detailed Design Development effort now defines a single, optimal solution at a lower level of detail. It specifies and identifies the following parts for the system:

- Which objects are active (concurrency)
- Application task scheduling policies
- Organization of classes and objects within deployable components
- Inter-processor communication media and protocols
- Distribution of software components
- Relation implementation strategies (How are associations implemented?)
- Implementation patterns for Finite State Machines
- 1 to Many UML associations
- Error-Handling Policies
- Memory Management Policies

1. The Detailed Design will be documented in the Software Design Document.
2. The Detailed Design will be developed by Software Engineering with the participation of Shuttle Engineering. This ensures the technical content of the design is correct and efficient while also ensuring the design is understandable to Shuttle Engineering personnel.
3. Identification and specification of the classes and objects necessary to implement the requirements are solidified:
   - Class descriptions are enhanced when necessary (e.g., by Sequence Diagrams, State Charts, etc.) to help define the class's activities
   - The classes/objects are scrutinized to identify commonality between objects to support generalization of those objects to as common a base class as possible. Inherited attributes and methods are also identified during this activity.
4. The activities included in the Detailed Design development are:
   - Definition of software components and their distribution
   - Identification and characterization of threads
   - Application or architectural design patterns (e.g., global error handling, safety processing and fault tolerance)
   - Implementation of associations, aggregations and components is defined
   - Exception handling is defined for each class
   - Types and valid ranges of class attributes are defined
   - Complex algorithms are clarified or introduced
   - Identification and specification of the inter-process communications and system interactions are specified
5. The following artifacts are products as a result of the Detailed Design effort. These artifacts are contained in the SDD.
   - Class (required) and Object (optional) Diagrams updated to include architecture design patterns
   - State Charts (only required for major sequences)
   - Sequence Diagrams
   - Pseudo-Code describing complex algorithmic behavior
6. As part of the Detailed Design effort, code prototyping can be used to test out design decisions before proceeding with full-fledged design activities.

Figure 16: Process B - CDR [18 p. 28]

#### 4.2.2.3 Process B CMM Evaluation Summary

The KPAs discussed above are only a few of what is required by the CMM. These were pointed out in this thesis because of the severity and consequences of not meeting them in Process A. All KPAs are critical to a CMM evaluation, however, these directly affected the design phase and the focus of the thesis.

The creation of Process B addressed the many CMM deficiencies in Process A, specifically the KPAs addressed in this thesis. The definitions of work products as well as an entirely new document outlining entry/exit criterion for each phase and verifications review were additions to Process B. Software estimation activity via FPA and the complete training of how to perform this activity was also added. In addition to providing training to satisfy Level 2 KPA - Ability to Perform 4, Process B instituted training in all areas of the life cycle.

Finally the peer review process was greatly enhanced. Process B took what amounted to a review before design and a review after implementation and added three additional reviews in between. This brought the CLCS process more in line with mainstream review processes.

# 5 Supporting Evidence for Hypothesis

The CLCS project was a "real world" software development project and as a result the data collected are based on actual work performed by the project's software engineers. Part of the improvements made for Process B included the collections of metrics. Process B instituted function point analysis, and gathered metrics based on the function points. OO Classes, Source Lines of Code (SLOC), Man Months, defects, and test results were all items included in the metrics collecting. Process A did not included such detailed metrics collecting. As a result the basis for process comparison is actual performance data.

This performance data is schedule based and is used by this thesis as the criteria for comparing Process A against Process B. This data includes the "Implementation Actual Percentage Complete" and the "Actual Duration" elapsed in terms of days. Both Process A and Process B scheduling included preliminary design, detailed design, coding, and unit test under the category of implementation. Although this could have been done on a finer level of granularity it does provide for an easy direct comparison between the two processes.

Using the given data a total projected duration time can be calculated. This is based on past performance and assumes the rate of performance will continue at a steady pace. The following formula is used to calculate the total projected duration implementation time per CSCI:

$$\frac{actual\%Complete}{100} \equiv \frac{actualDuration}{total\,\mathrm{Pr}\,ojectedDuration}$$

or

$$\frac{(actualDuration \times 100)}{actual\%Complete} \equiv total\,\mathrm{Pr}\,ojectedDuration$$

In addition a comparable duration is calculated at 25%, 50% and 75% complete for Process A and Process B based on the total projected duration. The following formula is used to calculate the comparable duration time.

$$((.25\,|\,.50\,|\,.75) \times total\,\mathrm{Pr}\,ojectedDuration) \equiv comparableDuration$$

Finally, a change in development time percentage is calculated for Process B. This shows a percentage change in the amount of development time following Process B as opposed to following Process A. The following formula is used for this calculation.

$$\frac{(processBActualDuration \times 100)}{processB\%Complete} \bigg/ \frac{(processAActualDuration \times 100)}{processA\%Complete} \equiv$$

$$\%changeInDevelopmentTime$$

Figure 17 shows the direct comparisons between Process A and Process B for each CSCI. The table contains the formula calculations as stated above. Detailed Microsoft Project schedules from the CLCS project and data for each individual CSCI are located in Appendix B.

| CSCI | Process | Actual Percentage Complete | Actual Duration (Days) | Duration for common percentages complete at existing rate of completion in Days. | | | | %reduction |
|---|---|---|---|---|---|---|---|---|
| | | | | 25% | 50% | 75% | 100% | |
| CRYO | A | 100 | 315 | 78.75 | 157.50 | 236.25 | 315.00 | 48.25% |
| | B | 91 | 148.33 | 40.75 | 81.5 | 122.25 | 163.00 | |
| DPS | A | 66 | 521.87 | 197.68 | 395.36 | 593.03 | 790.71 | 48.72 |
| | B | 34 | 137.86 | 101.37 | 202.74 | 304.10 | 405.47 | |
| ECL | A | 82 | 519.88 | 158.50 | 317.00 | 475.50 | 634.00 | 50.00 |
| | B | 62 | 196.54 | 79.25 | 158.5 | 237.75 | 317.00 | |
| EPD | A | 46 | 181.24 | 98.50 | 197.00 | 295.50 | 394.00 | 11.21 |
| | B | 26 | 90.96 | 87.46 | 174.93 | 262.39 | 349.85 | |
| HWS | A | 100 | 457 | 114.25 | 228.50 | 342.75 | 457.00 | 70.90 |
| | B | 100 | 133 | 33.25 | 66.5 | 99.75 | 133.00 | |
| HYD | A | 97 | 452.99 | 116.75 | 233.50 | 350.25 | 467.00 | 63.81 |
| | B | 88 | 148.72 | 42.25 | 84.5 | 126.75 | 169.00 | |
| INS | A | 69 | 523.02 | 189.50 | 379.00 | 568.50 | 758.00 | 57.39 |
| | B | 36 | 116.28 | 80.75 | 161.5 | 242.25 | 323.00 | |
| MEQ | A | 63 | 253.89 | 100.75 | 201.50 | 302.25 | 403.00 | 65.01 |
| | B | 60 | 84.6 | 35.25 | 70.5 | 105.75 | 141.00 | |
| PLD | A | 27 | 49.95 | 46.25 | 92.50 | 138.75 | 185.00 | 64.86 |
| | B | 20 | 13.00 | 16.25 | 32.5 | 48.75 | 65.00 | |
| RMS | A | 49 | 69.09 | 35.25 | 70.50 | 105.75 | 141.00 | 31.21 |
| | B | 80 | 77.60 | 24.25 | 48.5 | 72.75 | 97.00 | |
| OMS | A | 84 | 418.32 | 124.50 | 249.00 | 373.50 | 498.00 | 75.70 |
| | B | 65 | 78.65 | 30.25 | 60.5 | 90.75 | 121.00 | |

Figure 17: CSCI Comparable durations

# 6 Hypothesis Summary

This thesis involved using the information learned in pursuit of the master of software engineering degree to create an iterative lifecycle model using SEI CMM compliant processes for a real world software project. In addition, the CLCS Project created a unique environment in which the study of different life cycle models and development processes could be undertaken. It created an environment in which one could analyze whether or not there is an improvement in the performance of a software development team under the following conditions:

1) When the team follows an iterative lifecycle model versus a waterfall lifecycle model
2) When the team follows SEI CMM compliant processes versus non-SEI CMM compliant processes.

The Process A SDP shows exactly why Process A was struggling for three years to produce software. The lack of direction from the written design process left each CSCI struggling to determine what was required during the design phase, including which work products were to be produced. The work products themselves were also left undefined by the process. As a result each of the twenty CSCIs proceeded to define what was needed for them to produce software. There was no regard for the project as a whole. The lack of definition in Process A actually forced the creation of twenty different software development processes, one for each CSCI.

The inability for Process A to meet the SEI CMM KPAs identified in this thesis proved to NASA managers that Process A was at a Level 1 and needed to be improved. It was this chaotic Level 1 environment that created an atmosphere of non-productivity. This environment led to schedule and budget slips that eventually became unbearable for upper management. It was under the threat of project cancellation that the decision was made to search for a solution.

In addition to addressing the KPA deficiencies in Process A, Process B introduced a completely new direction in terms of a software life cycle model. The use of an iterative model allowed for constant process and product improvement through lessons learned in each iteration.

Evaluation of the supporting Evidence in Figure 17 shows that as a result of migrating from Process A to Process B a CSCI's development time was greatly reduced. In some cases Process B created a development timesaving of over seventy-percent. There does exists one CSCI which appears to be an anomaly, the EPD CSCI only showed a 11.21% reduction in development time. The average savings is 57.58% if EPD is thrown out of the calculation, but still a respectable 53.36% if EPD is used in the calculation.

As a result of this redirection, the CLCS was able to surpass their three-year performance using Process A in one year using this new process. An Integrated Process Team (IPT) following Process A took approximately three years from requirements analysis to unit test. An IPT following the new process took approximately one year from requirements analysis to, in some cases, user validation and acceptance. It is this analysis that validates the two thesis statements. According to the data gathered there is an improvement in performance when following an iterative lifecycle model with SEI CMM compliant processes as opposed to a waterfall lifecycle model with non-SEI CMM compliant processes.

# 7 Future Research on this Topic

This thesis concentrated primarily on the design phase of the software life cycle. One phase does not make a successful project. It is unknown whether or not the performance improvement would continue through the test and deployment process workflows. Further research could be conducted on the impact of this lifecycle and process migration during these later phases. In addition, CLCS was only one project. Further research could be conducted on similar projects in an effort to expand the thesis across different types of software development efforts.

Other areas of research could expand the usage of the iterative model using processes other then Rational Corporations RUP. Do other processes create the same level of improvements as the RUP did. Do other lifecycle models, e.g. extreme programming, rapid prototyping create improvements over iterative?

It would also be interesting to research what effect the iterative model had in comparison to the SEI CMM compliant processes. What percentage of the improvements in performance can be attributed to the change in lifecycle models vs. the change in software processes. Could the same improvements occurred with just a change in process or just a change in lifecycle models.

# 8 References

1. The Capability Maturity Model: Guidelines for Improving Software Process, Carnegie Mellon University Software Engineering Institute, Addison-Wesley, Reading, Massachusetts, 1999.

2. Bernd Bruegge and Allen H. Dutoit, *Object-Oriented Software Engineering*, Prentice Hall, Upper Saddle River, NJ, 2000.

3. Bryan Campbell and Dr. Glenn Ray, *Iterative Development Testing Approaches*, http://www.bryancampbell.com/Articles/Test_strategy_long.htm, (April 2003).

4. Chad A. Chamberlin, 360 Software Corporation, 2002.

5. DOD-STD-2167A (1988, February) *Military Standard: Defense System Software Development*, Washington, D.C.: Department of Defense

6. Graphical Development Process Assistant, http://www.informatik.uni-bremen.de/gdpa/def_w/WATERFALL.htm, (April 2003).

7. Maria Ericsson, Developing Large-scale Systems with the Rational Unified Process, http://www.rational.com/products/whitepapers/sis.jsp Rational Software, 2000.

8. *Iterative Life Cycle*, http://www.accelerasoftware.com/notes/200207_073.html, (April 2003).

9. *Iterative vs. Waterfall Approach*, CDC Technologies, http://www.cdc-technologies.com/method/it_vs_wat.htm, (April 2003).

10. Stephen H. Kan, *Metrics and Models in Software Quality Engineering*, Addison-Wesley, Reading, Massachusetts, 1997.

11. Philippe Kruchten, *A Rational Development Process*, http://www.rational.com/products/whitepapers/334.jsp Rational Software Corporation, 1996.

12. Philippe Kruchten, *From Waterfall to Iterative Lifecycle - A tough transition for project managers*, http://www.rational.com/products/whitepapers/334.jsp Rational Software Corporation, 2000.

13. Jack R. Meredith and Samuel J. Mantel, Jr., *Project Management: A Managerial Approach*, Fourth Edition, John Wiley & Sons, New York, New York, 2000.

14. James W. Moore *Software Engineering Standars: A User's Road Map*, 1st Edition Wiley IEEE Press Nov. 1997.

15. Jim Pietrocarlo, *Managemening Iterative Development*, Rational Software http://www.cooug.org/managing_iterative_development4%5B1%5D.ppt, (April 2003).

16. Leslee Probasco, *The Ten Essentials of RUP, The Essence of an Effective Development Process,* http://www.rational.com/products/whitepapers/413.jsp Rational Software, 2000.

17. National Aeronautics and Space Administration, Kennedy Space Center, FL. Document 84K00070-002 Software Development Plan Volume II Revision B, Aug. 2001.

18. National Aeronautics and Space Administration, Kennedy Space Center, FL. Document 84K00070-002 Software Development Plan Volume II Revision D, Apr. 2002.

19. *Rational Unified Process: Best Practices for Software Development Teams*, http://www.rational.com/products/whitepapers/100420.jsp Rational Software, 2001.

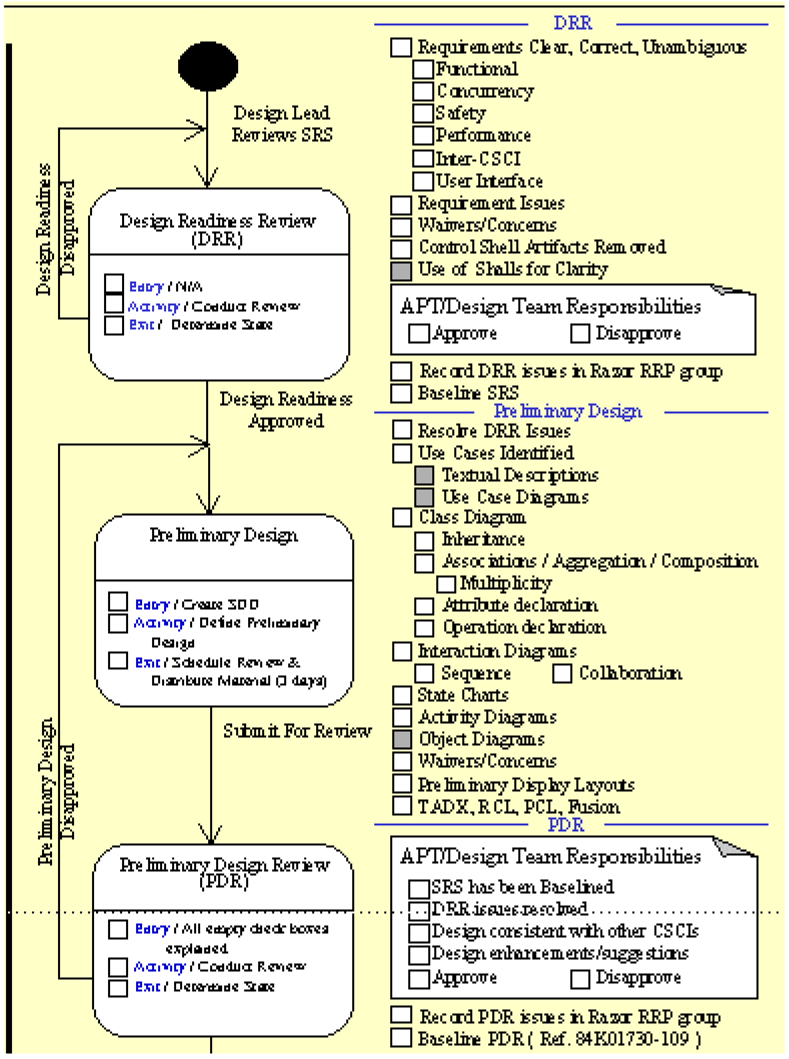
20. W. W. Royce: Managing the Development of Large Software Systems: Concepts and Techniques. ICSE 1987: 328-339.

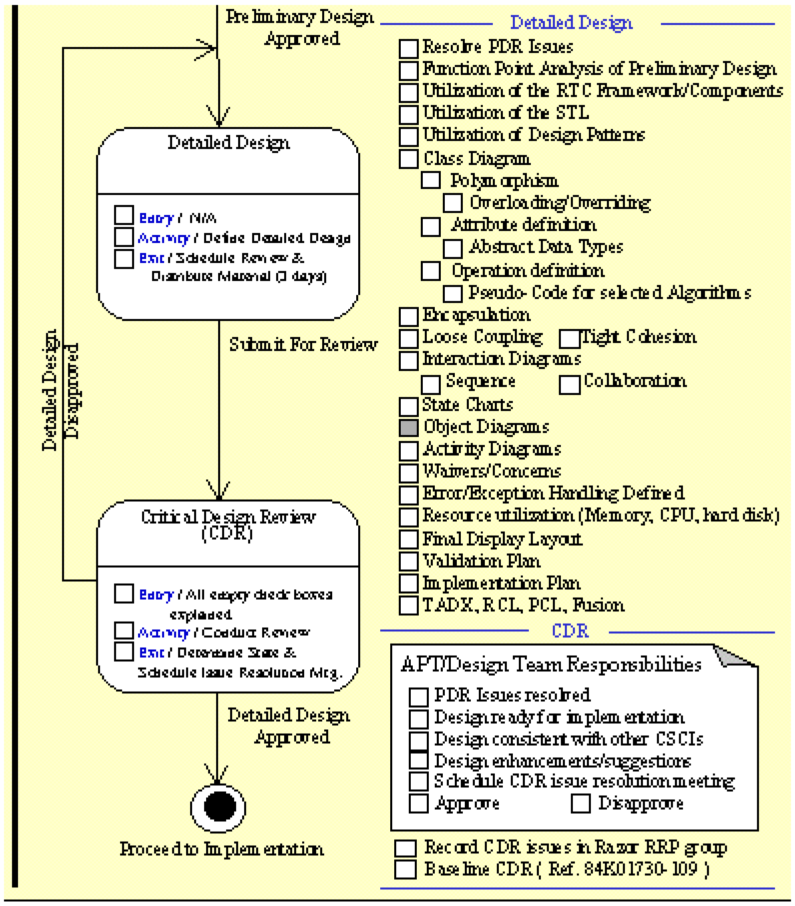
21. Ian Sommerville and Pete Sawyer, *Requirements Engineering: A good practice guide*, John Wiley & Sons, New York, New York, 1997.

22. Thesis Manual and Style Guide for Use at Florida Institute of Technology, Second Edition, Florida Institute of Technology, Melbourne, FL, 2001.

23. J.A. Whittaker, *Introduction to Software Engineering*, SES Press, Melbourne, FL, 1998.

# 9 Appendix A


Design Lead Reviews SRS
Design Readiness Disapproved
Design Readiness Review (DRR)
Entry / N/A
Activity / Conduct Review
Exit / Determine State
Design Readiness Approved
Preliminary Design
Entry / Create SDD
Activity / Define Preliminary Design
Exit / Schedule Review & Distribute Material (3 days)
Submit For Review
Preliminary Design Disapproved
Preliminary Design Review (PDR)
Entry / All empty check boxes explained
Activity / Conduct Review
Exit / Determine State
DRR
Requirements Clear, Correct, Unambiguous
Functional
Concurrency
Safety
Performance
Inter-CSCI
User Interface
Requirement Issues
Waivers/Concerns
Control Shell Artifacts Removed
Use of Shalls for Clarity
APT/Design Team Responsibilities
Approve
Disapprove
Record DRR issues in Razor RRP group
Baseline SRS
Preliminary Design
Resolve DRR Issues
Use Cases Identified
Textual Descriptions
Use Case Diagrams
Class Diagram
Inheritance
Associations / Aggregation / Composition
Multiplicity
Attribute declaration
Operation declaration
Interaction Diagrams
Sequence
Collaboration
State Charts
Activity Diagrams
Object Diagrams
Waivers/Concerns
Preliminary Display Layouts
TADX, RCL, PCL, Fusion
PDR
APT/Design Team Responsibilities
SRS has been Baselined
DRR issues resolved
Design consistent with other CSCIs
Design enhancements/suggestions
Approve
Disapprove
Record PDR issues in Razor RRP group
Baseline PDR ( Ref. 84K01730-109 )


Preliminary Design Approved
Detailed Design
☐ Entry / N/A
☐ Activity / Define Detailed Design
☐ Exit / Schedule Review & Distribute Material (3 days)
Submit For Review
Detailed Design Disapproved
Critical Design Review (CDR)
☐ Entry / All empty check boxes explained
☐ Activity / Conduct Review
☐ Exit / Determine State & Schedule Issue Resolution Mtg.
Detailed Design Approved
Proceed to Implementation

Detailed Design

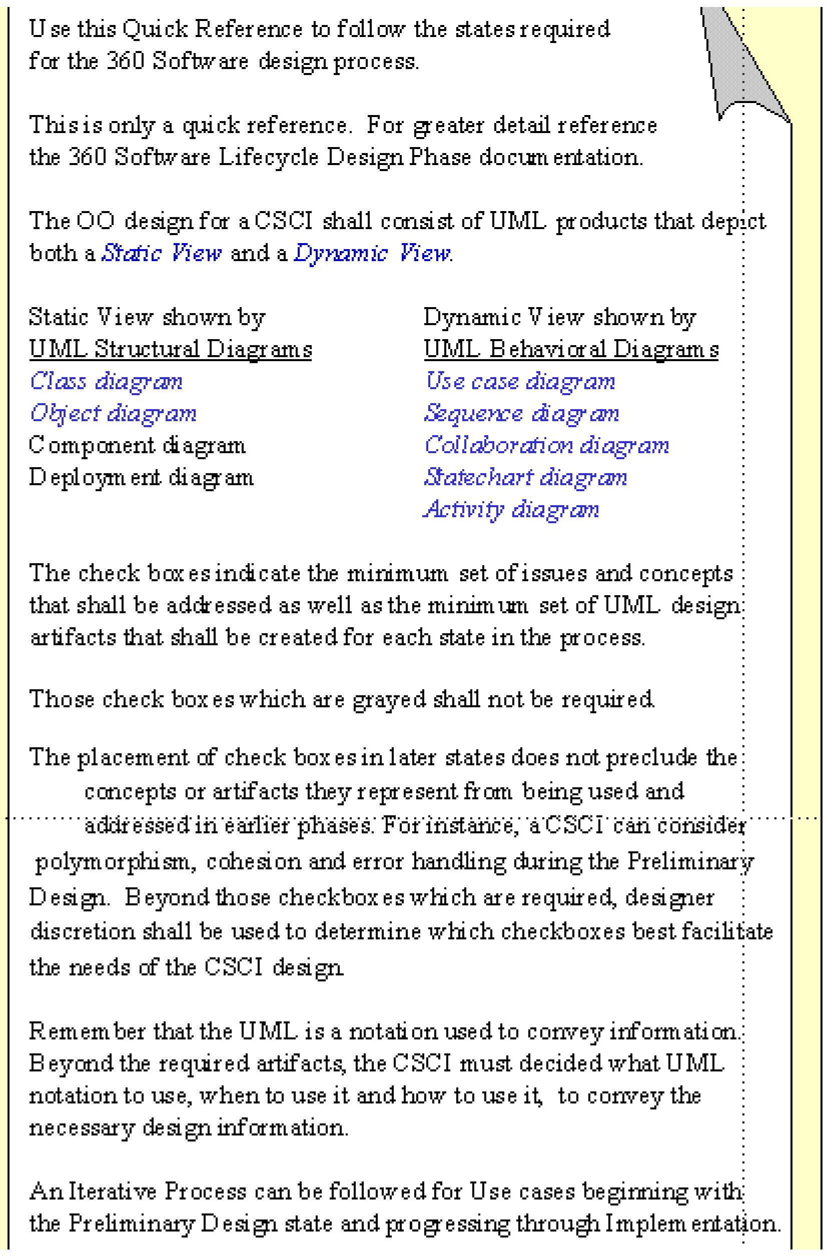
- ☐ Resolve PDR Issues
- ☐ Function Point Analysis of Preliminary Design
- ☐ Utilization of the RTC Framework/Components
- ☐ Utilization of the STL
- ☐ Utilization of Design Patterns
- ☐ Class Diagram
  - ☐ Polymorphism
    - ☐ Overloading/Overriding
  - ☐ Attribute definition
    - ☐ Abstract Data Types
  - ☐ Operation definition
    - ☐ Pseudo-Code for selected Algorithms
- ☐ Encapsulation
- ☐ Loose Coupling ☐ Tight Cohesion
- ☐ Interaction Diagrams
  - ☐ Sequence ☐ Collaboration
- ☐ State Charts
- ▨ Object Diagrams
- ☐ Activity Diagrams
- ☐ Waivers/Concerns
- ☐ Error/Exception Handling Defined
- ☐ Resource utilization (Memory, CPU, hard disk)
- ☐ Final Display Layout
- ☐ Validation Plan
- ☐ Implementation Plan
- ☐ TADX, RCL, PCL, Fusion

CDR

APT/Design Team Responsibilities

- ☐ PDR Issues resolved
- ☐ Design ready for implementation
- ☐ Design consistent with other CSCIs
- ☐ Design enhancements/suggestions
- ☐ Schedule CDR issue resolution meeting
- ☐ Approve ☐ Disapprove

- ☐ Record CDR issues in Razor RRP group
- ☐ Baseline CDR ( Ref. 84K01730-109 )



Use this Quick Reference to follow the states required for the 360 Software design process.

This is only a quick reference. For greater detail reference the 360 Software Lifecycle Design Phase documentation.

The OO design for a CSCI shall consist of UML products that depict both a *Static View* and a *Dynamic View*.

| Static View shown by<br>UML Structural Diagrams | Dynamic View shown by<br>UML Behavioral Diagrams |
|---|---|
| *Class diagram* | *Use case diagram* |
| *Object diagram* | *Sequence diagram* |
| Component diagram | *Collaboration diagram* |
| Deployment diagram | *Statechart diagram* |
| | *Activity diagram* |

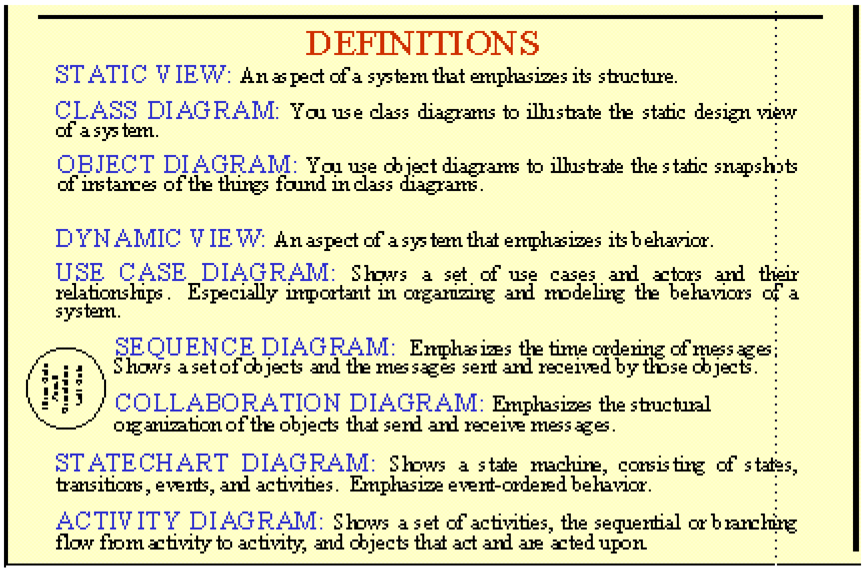
The check boxes indicate the minimum set of issues and concepts that shall be addressed as well as the minimum set of UML design artifacts that shall be created for each state in the process.

Those check boxes which are grayed shall not be required.

The placement of check boxes in later states does not preclude the concepts or artifacts they represent from being used and addressed in earlier phases. For instance, a CSCI can consider polymorphism, cohesion and error handling during the Preliminary Design. Beyond those checkboxes which are required, designer discretion shall be used to determine which checkboxes best facilitate the needs of the CSCI design.

Remember that the UML is a notation used to convey information. Beyond the required artifacts, the CSCI must decided what UML notation to use, when to use it and how to use it, to convey the necessary design information.

An Iterative Process can be followed for Use cases beginning with the Preliminary Design state and progressing through Implementation.



## DEFINITIONS

STATIC VIEW: An aspect of a system that emphasizes its structure.

CLASS DIAGRAM: You use class diagrams to illustrate the static design view of a system.

OBJECT DIAGRAM: You use object diagrams to illustrate the static snapshots of instances of the things found in class diagrams.

DYNAMIC VIEW: An aspect of a system that emphasizes its behavior.

USE CASE DIAGRAM: Shows a set of use cases and actors and their relationships. Especially important in organizing and modeling the behaviors of a system.

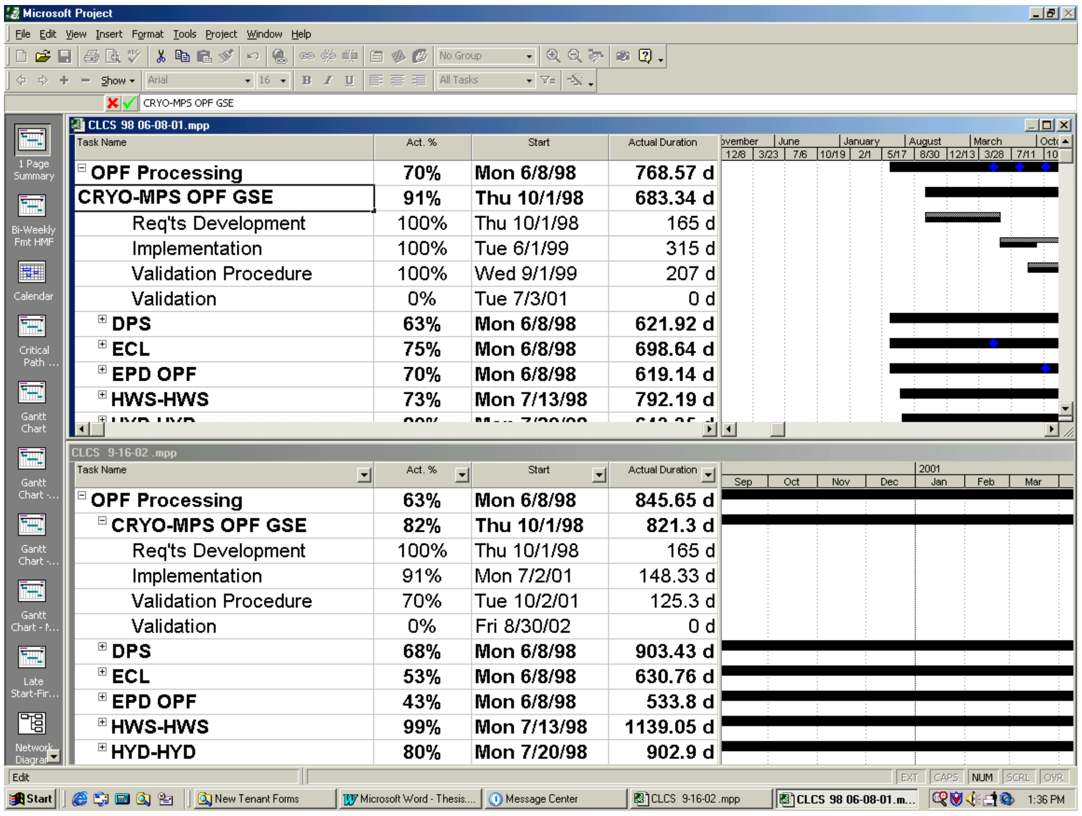
SEQUENCE DIAGRAM: Emphasizes the time ordering of messages. Shows a set of objects and the messages sent and received by those objects.

COLLABORATION DIAGRAM: Emphasizes the structural organization of the objects that send and receive messages.

STATECHART DIAGRAM: Shows a state machine, consisting of states, transitions, events, and activities. Emphasize event-ordered behavior.

ACTIVITY DIAGRAM: Shows a set of activities, the sequential or branching flow from activity to activity, and objects that act and are acted upon.

# 10 Appendix B



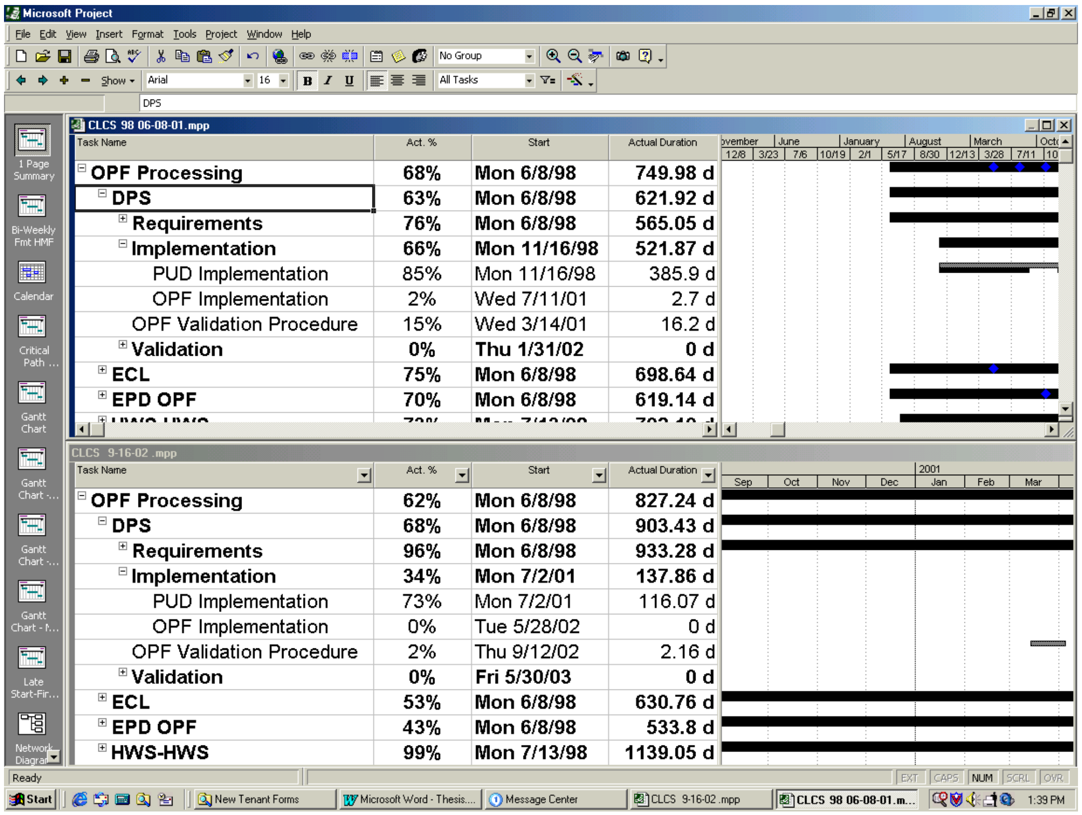
CLCS 98 06-08-01.mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 70% | Mon 6/8/98 | 768.57 d |
| CRYO-MPS OPF GSE | 91% | Thu 10/1/98 | 683.34 d |
| Req'ts Development | 100% | Thu 10/1/98 | 165 d |
| Implementation | 100% | Tue 6/1/99 | 315 d |
| Validation Procedure | 100% | Wed 9/1/99 | 207 d |
| Validation | 0% | Tue 7/3/01 | 0 d |
| DPS | 63% | Mon 6/8/98 | 621.92 d |
| ECL | 75% | Mon 6/8/98 | 698.64 d |
| EPD OPF | 70% | Mon 6/8/98 | 619.14 d |
| HWS-HWS | 73% | Mon 7/13/98 | 792.19 d |

CLCS 9-16-02 .mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 63% | Mon 6/8/98 | 845.65 d |
| CRYO-MPS OPF GSE | 82% | Thu 10/1/98 | 821.3 d |
| Req'ts Development | 100% | Thu 10/1/98 | 165 d |
| Implementation | 91% | Mon 7/2/01 | 148.33 d |
| Validation Procedure | 70% | Tue 10/2/01 | 125.3 d |
| Validation | 0% | Fri 8/30/02 | 0 d |
| DPS | 68% | Mon 6/8/98 | 903.43 d |
| ECL | 53% | Mon 6/8/98 | 630.76 d |
| EPD OPF | 43% | Mon 6/8/98 | 533.8 d |
| HWS-HWS | 99% | Mon 7/13/98 | 1139.05 d |
| HYD-HYD | 80% | Mon 7/20/98 | 902.9 d |

| **CRYO** | **Actual Percentage Complete** | **Actual Duration (Days)** | **Duration for common percentages complete at existing rate of completion** | | | |
|---|---|---|---|---|---|---|
| | | | 25% | 50% | 75% | Calculated Total Duration |
| **Process A** | 100 | 315 | 78.75 | 157.50 | 236.25 | 315.00 |
| **Process B** | 91 | 148.33 | 40.75 | 81.5 | 122.25 | 163.00 |
| | | | | | | |
| **Process B** | Provided a | 48.25% | reduction in development time. | | | |



DPS

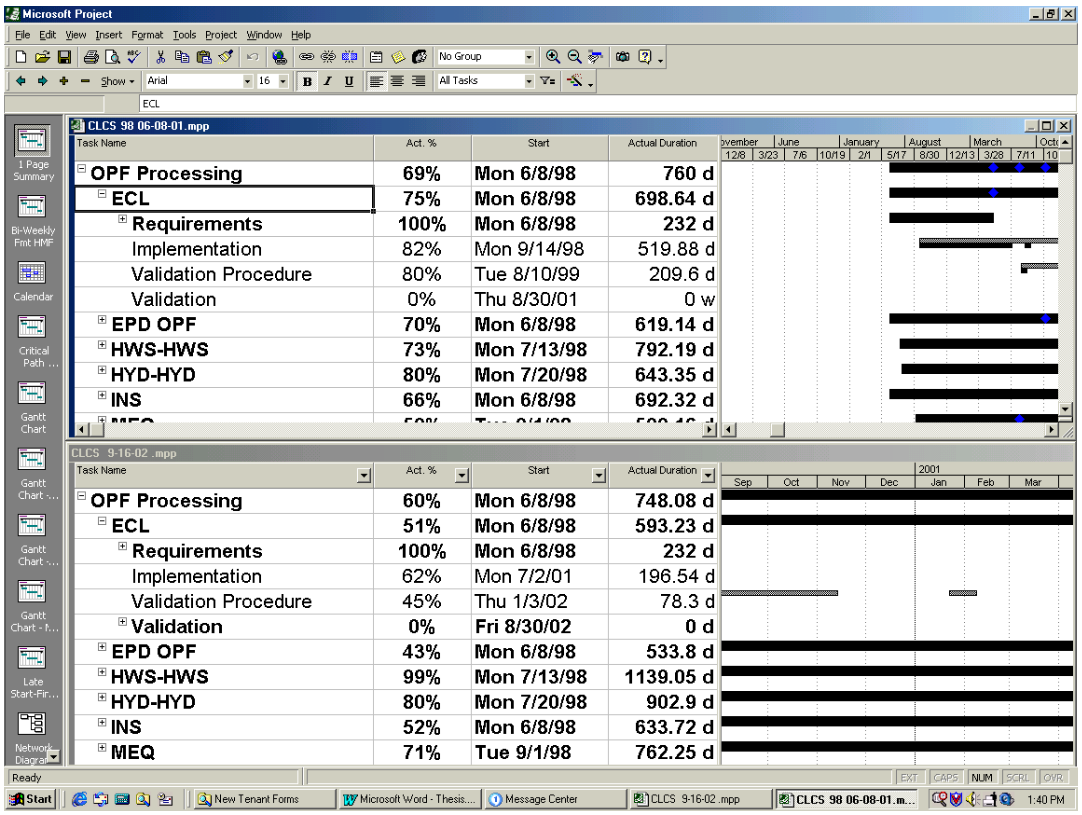
CLCS 98 06-08-01.mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 68% | Mon 6/8/98 | 749.98 d |
| DPS | 63% | Mon 6/8/98 | 621.92 d |
| Requirements | 76% | Mon 6/8/98 | 565.05 d |
| Implementation | 66% | Mon 11/16/98 | 521.87 d |
| PUD Implementation | 85% | Mon 11/16/98 | 385.9 d |
| OPF Implementation | 2% | Wed 7/11/01 | 2.7 d |
| OPF Validation Procedure | 15% | Wed 3/14/01 | 16.2 d |
| Validation | 0% | Thu 1/31/02 | 0 d |
| ECL | 75% | Mon 6/8/98 | 698.64 d |
| EPD OPF | 70% | Mon 6/8/98 | 619.14 d |

CLCS 9-16-02 .mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 62% | Mon 6/8/98 | 827.24 d |
| DPS | 68% | Mon 6/8/98 | 903.43 d |
| Requirements | 96% | Mon 6/8/98 | 933.28 d |
| Implementation | 34% | Mon 7/2/01 | 137.86 d |
| PUD Implementation | 73% | Mon 7/2/01 | 116.07 d |
| OPF Implementation | 0% | Tue 5/28/02 | 0 d |
| OPF Validation Procedure | 2% | Thu 9/12/02 | 2.16 d |
| Validation | 0% | Fri 5/30/03 | 0 d |
| ECL | 53% | Mon 6/8/98 | 630.76 d |
| EPD OPF | 43% | Mon 6/8/98 | 533.8 d |
| HWS-HWS | 99% | Mon 7/13/98 | 1139.05 d |

| DPS | Actual Percentage Complete | Actual Duration (Days) | Duration for common percentages complete at existing rate of completion | | | |
|---|---|---|---|---|---|---|
| | | | 25% | 50% | 75% | Calculated Total Duration |
| Process A | 66 | 521.87 | 197.68 | 395.36 | 593.03 | 790.71 |
| Process B | 34 | 137.86 | 101.37 | 202.74 | 304.10 | 405.47 |
| | | | | | | |
| Process B | Provided a | 48.72% | reduction in development time. | | | |



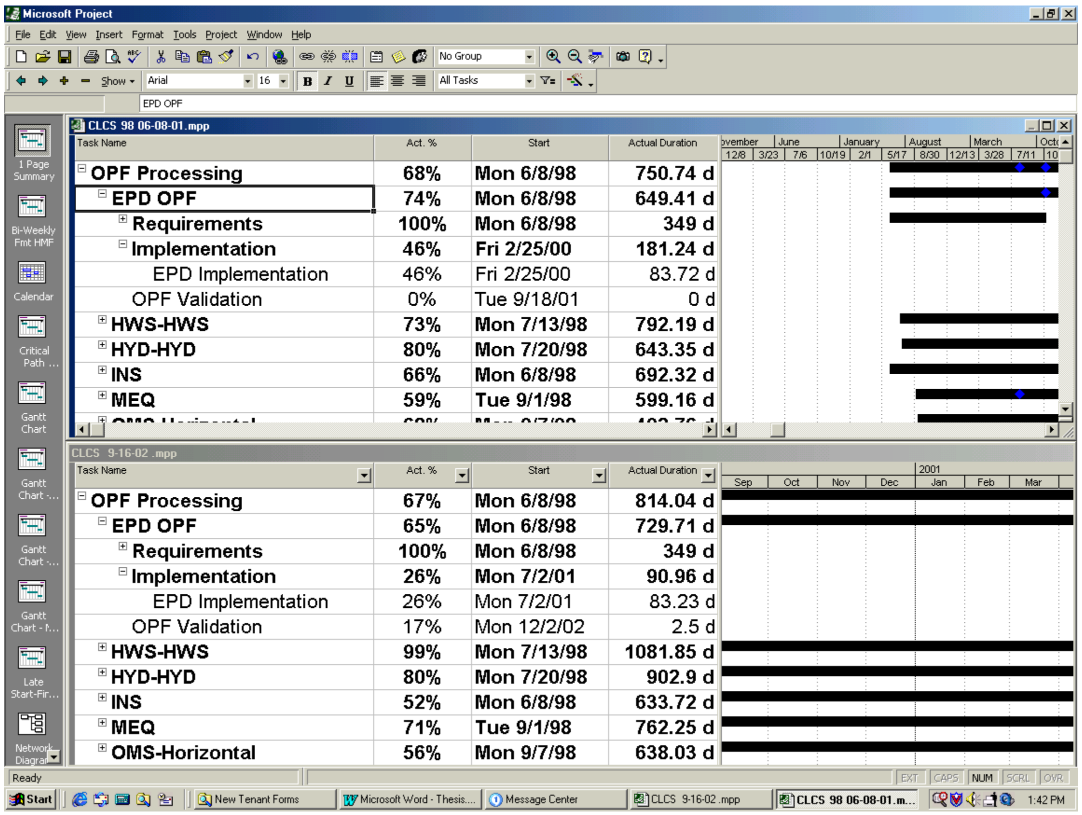
CLCS 98 06-08-01.mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 69% | Mon 6/8/98 | 760 d |
| ECL | 75% | Mon 6/8/98 | 698.64 d |
| Requirements | 100% | Mon 6/8/98 | 232 d |
| Implementation | 82% | Mon 9/14/98 | 519.88 d |
| Validation Procedure | 80% | Tue 8/10/99 | 209.6 d |
| Validation | 0% | Thu 8/30/01 | 0 w |
| EPD OPF | 70% | Mon 6/8/98 | 619.14 d |
| HWS-HWS | 73% | Mon 7/13/98 | 792.19 d |
| HYD-HYD | 80% | Mon 7/20/98 | 643.35 d |
| INS | 66% | Mon 6/8/98 | 692.32 d |

CLCS 9-16-02 .mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 60% | Mon 6/8/98 | 748.08 d |
| ECL | 51% | Mon 6/8/98 | 593.23 d |
| Requirements | 100% | Mon 6/8/98 | 232 d |
| Implementation | 62% | Mon 7/2/01 | 196.54 d |
| Validation Procedure | 45% | Thu 1/3/02 | 78.3 d |
| Validation | 0% | Fri 8/30/02 | 0 d |
| EPD OPF | 43% | Mon 6/8/98 | 533.8 d |
| HWS-HWS | 99% | Mon 7/13/98 | 1139.05 d |
| HYD-HYD | 80% | Mon 7/20/98 | 902.9 d |
| INS | 52% | Mon 6/8/98 | 633.72 d |
| MEQ | 71% | Tue 9/1/98 | 762.25 d |

| ECL | Actual Percentage Complete | Actual Duration (Days) | Duration for common percentages complete at existing rate of completion | | | |
|---|---|---|---|---|---|---|
| | | | 25% | 50% | 75% | Calculated Total Duration |
| **Process A** | 82 | 519.88 | 158.50 | 317.00 | 475.50 | 634.00 |
| **Process B** | 62 | 196.54 | 79.25 | 158.5 | 237.75 | 317.00 |
| | | | | | | |
| **Process B** | Provided a | 50.00% | reduction in development time. | | | |



Microsoft Project

EPD OPF

CLCS 98 06-08-01.mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 68% | Mon 6/8/98 | 750.74 d |
| EPD OPF | 74% | Mon 6/8/98 | 649.41 d |
| Requirements | 100% | Mon 6/8/98 | 349 d |
| Implementation | 46% | Fri 2/25/00 | 181.24 d |
| EPD Implementation | 46% | Fri 2/25/00 | 83.72 d |
| OPF Validation | 0% | Tue 9/18/01 | 0 d |
| HWS-HWS | 73% | Mon 7/13/98 | 792.19 d |
| HYD-HYD | 80% | Mon 7/20/98 | 643.35 d |
| INS | 66% | Mon 6/8/98 | 692.32 d |
| MEQ | 59% | Tue 9/1/98 | 599.16 d |

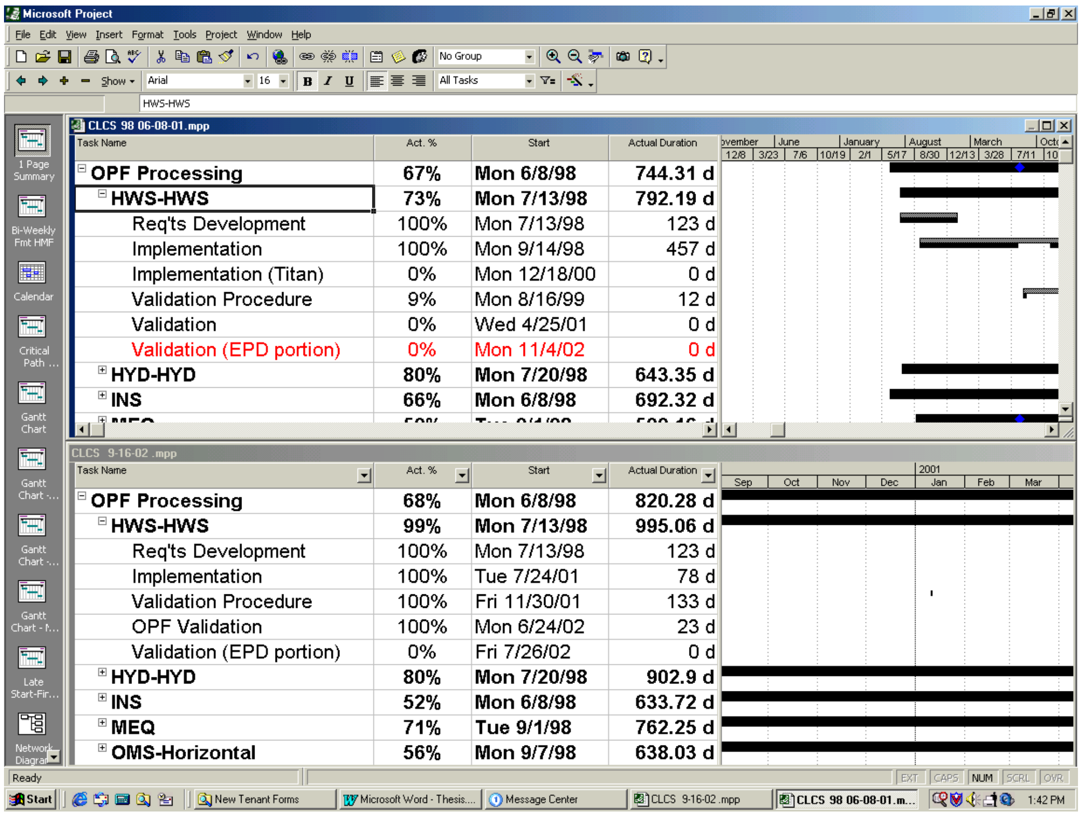
CLCS 9-16-02 .mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 67% | Mon 6/8/98 | 814.04 d |
| EPD OPF | 65% | Mon 6/8/98 | 729.71 d |
| Requirements | 100% | Mon 6/8/98 | 349 d |
| Implementation | 26% | Mon 7/2/01 | 90.96 d |
| EPD Implementation | 26% | Mon 7/2/01 | 83.23 d |
| OPF Validation | 17% | Mon 12/2/02 | 2.5 d |
| HWS-HWS | 99% | Mon 7/13/98 | 1081.85 d |
| HYD-HYD | 80% | Mon 7/20/98 | 902.9 d |
| INS | 52% | Mon 6/8/98 | 633.72 d |
| MEQ | 71% | Tue 9/1/98 | 762.25 d |
| OMS-Horizontal | 56% | Mon 9/7/98 | 638.03 d |

Ready

1:42 PM

| EPD | Actual Percentage Complete | Actual Duration (Days) | Duration for common percentages complete at existing rate of completion | | | |
|---|---|---|---|---|---|---|
| | | | 25% | 50% | 75% | Calculated Total Duration |
| **Process A** | 46 | 181.24 | 98.50 | 197.00 | 295.50 | 394.00 |
| **Process B** | 26 | 90.96 | 87.46 | 174.93 | 262.39 | 349.85 |
| | | | | | | |
| **Process B** | Provided a | 11.21% | reduction in development time. | | | |



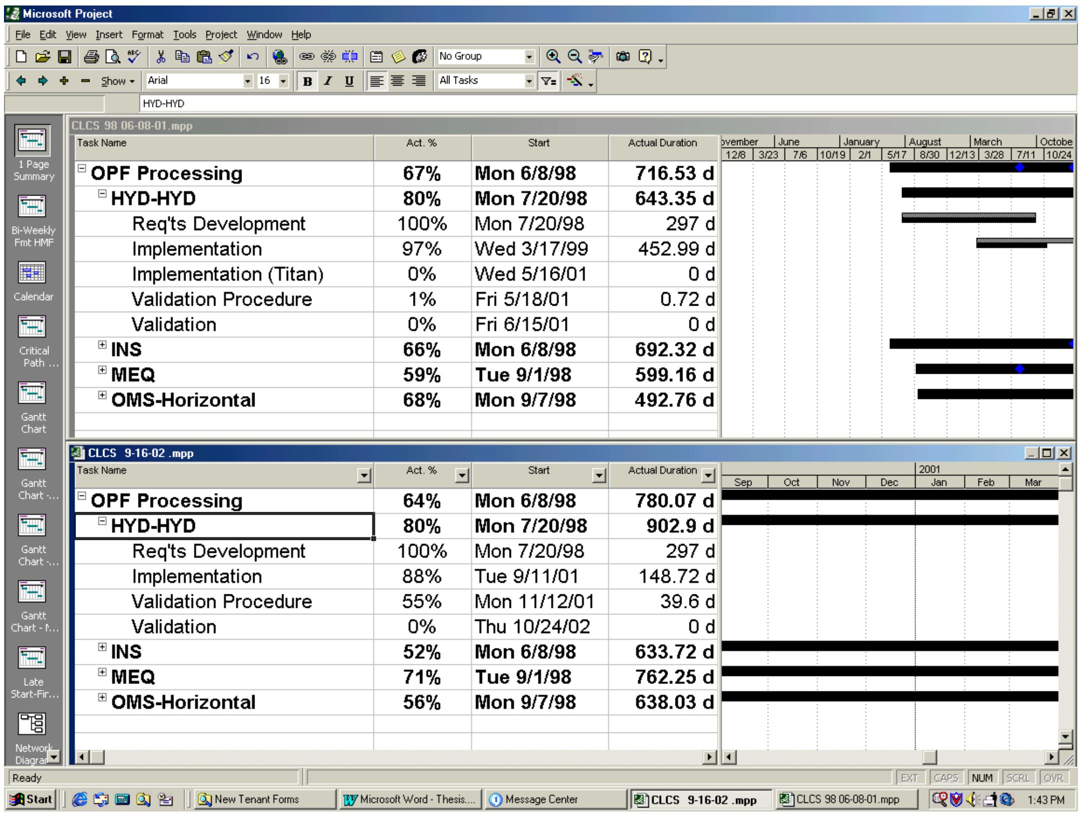
CLCS 98 06-08-01.mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| **OPF Processing** | **67%** | **Mon 6/8/98** | **744.31 d** |
| **HWS-HWS** | **73%** | **Mon 7/13/98** | **792.19 d** |
| Req'ts Development | 100% | Mon 7/13/98 | 123 d |
| Implementation | 100% | Mon 9/14/98 | 457 d |
| Implementation (Titan) | 0% | Mon 12/18/00 | 0 d |
| Validation Procedure | 9% | Mon 8/16/99 | 12 d |
| Validation | 0% | Wed 4/25/01 | 0 d |
| Validation (EPD portion) | 0% | Mon 11/4/02 | 0 d |
| **HYD-HYD** | **80%** | **Mon 7/20/98** | **643.35 d** |
| **INS** | **66%** | **Mon 6/8/98** | **692.32 d** |

CLCS 9-16-02 .mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| **OPF Processing** | **68%** | **Mon 6/8/98** | **820.28 d** |
| **HWS-HWS** | **99%** | **Mon 7/13/98** | **995.06 d** |
| Req'ts Development | 100% | Mon 7/13/98 | 123 d |
| Implementation | 100% | Tue 7/24/01 | 78 d |
| Validation Procedure | 100% | Fri 11/30/01 | 133 d |
| OPF Validation | 100% | Mon 6/24/02 | 23 d |
| Validation (EPD portion) | 0% | Fri 7/26/02 | 0 d |
| **HYD-HYD** | **80%** | **Mon 7/20/98** | **902.9 d** |
| **INS** | **52%** | **Mon 6/8/98** | **633.72 d** |
| **MEQ** | **71%** | **Tue 9/1/98** | **762.25 d** |
| **OMS-Horizontal** | **56%** | **Mon 9/7/98** | **638.03 d** |

| **HWS** | **Actual Percentage Complete** | **Actual Duration (Days)** | **Duration for common percentages complete at existing rate of completion** | | | |
|---|---|---|---|---|---|---|
| | | | 25% | 50% | 75% | Calculated Total Duration |
| **Process A** | 100 | 457 | 114.25 | 228.50 | 342.75 | 457.00 |
| **Process B** | 100 | 133 | 33.25 | 66.5 | 99.75 | 133.00 |
| | | | | | | |
| **Process B** | Provided a | 70.90% | reduction in development time. | | | |



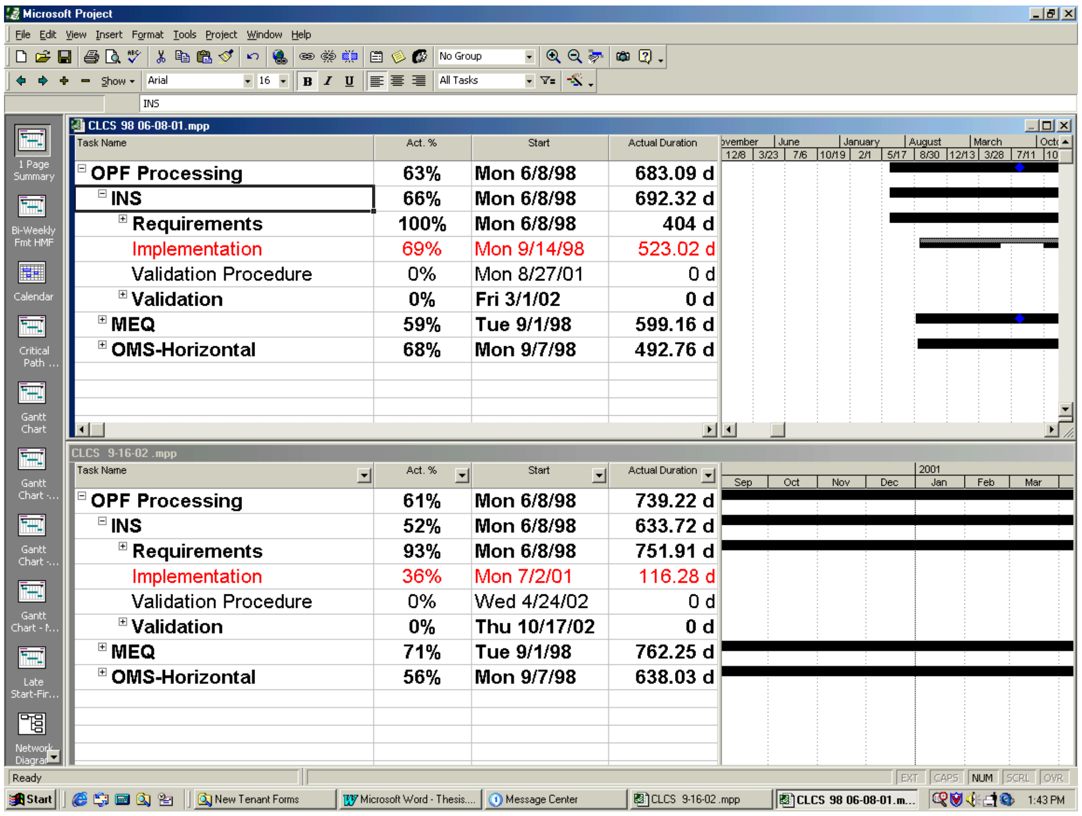
CLCS 98 06-08-01.mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| **OPF Processing** | **67%** | **Mon 6/8/98** | **716.53 d** |
| **HYD-HYD** | **80%** | **Mon 7/20/98** | **643.35 d** |
| Req'ts Development | 100% | Mon 7/20/98 | 297 d |
| Implementation | 97% | Wed 3/17/99 | 452.99 d |
| Implementation (Titan) | 0% | Wed 5/16/01 | 0 d |
| Validation Procedure | 1% | Fri 5/18/01 | 0.72 d |
| Validation | 0% | Fri 6/15/01 | 0 d |
| **INS** | **66%** | **Mon 6/8/98** | **692.32 d** |
| **MEQ** | **59%** | **Tue 9/1/98** | **599.16 d** |
| **OMS-Horizontal** | **68%** | **Mon 9/7/98** | **492.76 d** |

CLCS 9-16-02 .mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| **OPF Processing** | **64%** | **Mon 6/8/98** | **780.07 d** |
| **HYD-HYD** | **80%** | **Mon 7/20/98** | **902.9 d** |
| Req'ts Development | 100% | Mon 7/20/98 | 297 d |
| Implementation | 88% | Tue 9/11/01 | 148.72 d |
| Validation Procedure | 55% | Mon 11/12/01 | 39.6 d |
| Validation | 0% | Thu 10/24/02 | 0 d |
| **INS** | **52%** | **Mon 6/8/98** | **633.72 d** |
| **MEQ** | **71%** | **Tue 9/1/98** | **762.25 d** |
| **OMS-Horizontal** | **56%** | **Mon 9/7/98** | **638.03 d** |

| **HYD** | **Actual Percentage Complete** | **Actual Duration (Days)** | **Duration for common percentages complete at existing rate of completion** | | | |
|---|---|---|---|---|---|---|
| | | | 25% | 50% | 75% | Calculated Total Duration |
| **Process A** | 97 | 452.99 | 116.75 | 233.50 | 350.25 | 467.00 |
| **Process B** | 88 | 148.72 | 42.25 | 84.5 | 126.75 | 169.00 |
| | | | | | | |
| **Process B** | Provided a | 63.81% | reduction in development time. | | | |



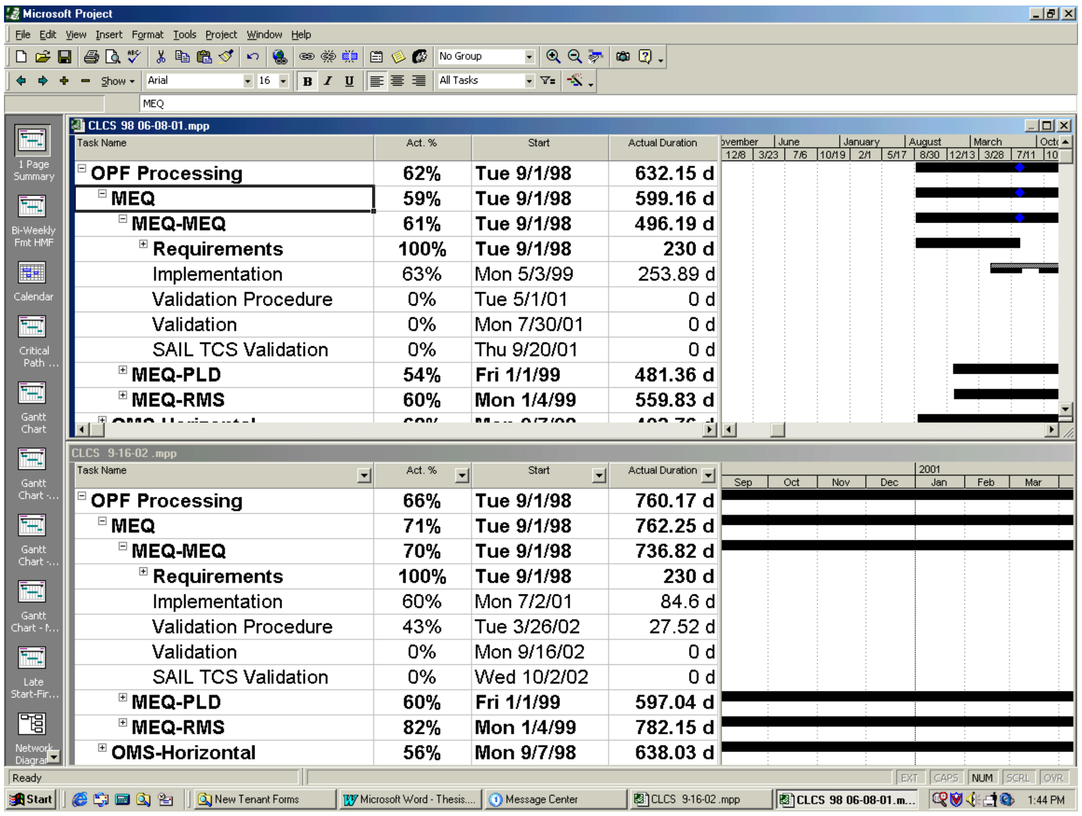
CLCS 98 06-08-01.mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 63% | Mon 6/8/98 | 683.09 d |
| INS | 66% | Mon 6/8/98 | 692.32 d |
| Requirements | 100% | Mon 6/8/98 | 404 d |
| Implementation | 69% | Mon 9/14/98 | 523.02 d |
| Validation Procedure | 0% | Mon 8/27/01 | 0 d |
| Validation | 0% | Fri 3/1/02 | 0 d |
| MEQ | 59% | Tue 9/1/98 | 599.16 d |
| OMS-Horizontal | 68% | Mon 9/7/98 | 492.76 d |

CLCS 9-16-02 .mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 61% | Mon 6/8/98 | 739.22 d |
| INS | 52% | Mon 6/8/98 | 633.72 d |
| Requirements | 93% | Mon 6/8/98 | 751.91 d |
| Implementation | 36% | Mon 7/2/01 | 116.28 d |
| Validation Procedure | 0% | Wed 4/24/02 | 0 d |
| Validation | 0% | Thu 10/17/02 | 0 d |
| MEQ | 71% | Tue 9/1/98 | 762.25 d |
| OMS-Horizontal | 56% | Mon 9/7/98 | 638.03 d |

| **INS** | **Actual Percentage Complete** | **Actual Duration (Days)** | **Duration for common percentages complete at existing rate of completion** | | | |
|---|---|---|---|---|---|---|
| | | | 25% | 50% | 75% | Calculated Total Duration |
| **Process A** | 69 | 523.02 | 189.50 | 379.00 | 568.50 | 758.00 |
| **Process B** | 36 | 116.28 | 80.75 | 161.5 | 242.25 | 323.00 |
| | | | | | | |
| **Process B** | Provided a | 57.39% | reduction in development time. | | | |



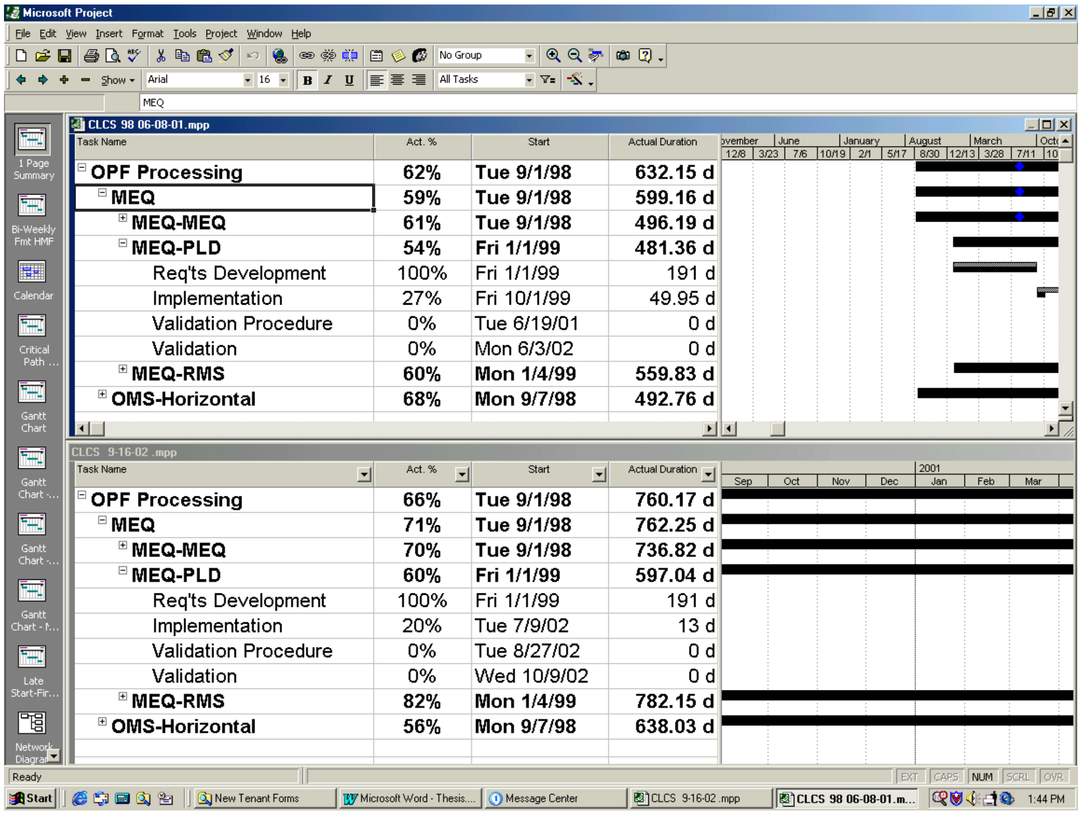
CLCS 98 06-08-01.mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 62% | Tue 9/1/98 | 632.15 d |
| MEQ | 59% | Tue 9/1/98 | 599.16 d |
| MEQ-MEQ | 61% | Tue 9/1/98 | 496.19 d |
| Requirements | 100% | Tue 9/1/98 | 230 d |
| Implementation | 63% | Mon 5/3/99 | 253.89 d |
| Validation Procedure | 0% | Tue 5/1/01 | 0 d |
| Validation | 0% | Mon 7/30/01 | 0 d |
| SAIL TCS Validation | 0% | Thu 9/20/01 | 0 d |
| MEQ-PLD | 54% | Fri 1/1/99 | 481.36 d |
| MEQ-RMS | 60% | Mon 1/4/99 | 559.83 d |
| [illegible] | [illegible] | [illegible] | [illegible] |

CLCS 9-16-02 .mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 66% | Tue 9/1/98 | 760.17 d |
| MEQ | 71% | Tue 9/1/98 | 762.25 d |
| MEQ-MEQ | 70% | Tue 9/1/98 | 736.82 d |
| Requirements | 100% | Tue 9/1/98 | 230 d |
| Implementation | 60% | Mon 7/2/01 | 84.6 d |
| Validation Procedure | 43% | Tue 3/26/02 | 27.52 d |
| Validation | 0% | Mon 9/16/02 | 0 d |
| SAIL TCS Validation | 0% | Wed 10/2/02 | 0 d |
| MEQ-PLD | 60% | Fri 1/1/99 | 597.04 d |
| MEQ-RMS | 82% | Mon 1/4/99 | 782.15 d |
| OMS-Horizontal | 56% | Mon 9/7/98 | 638.03 d |

| **MEQ-MEQ** | **Actual Percentage Complete** | **Actual Duration (Days)** | **Duration for common percentages complete at existing rate of completion** | | | |
|---|---|---|---|---|---|---|
| | | | 25% | 50% | 75% | Calculated Total Duration |
| **Process A** | 63 | 253.89 | 100.75 | 201.50 | 302.25 | 403.00 |
| **Process B** | 60 | 84.6 | 35.25 | 70.5 | 105.75 | 141.00 |
| | | | | | | |
| **Process B** | Provided a | 65.01% | reduction in development time. | | | |



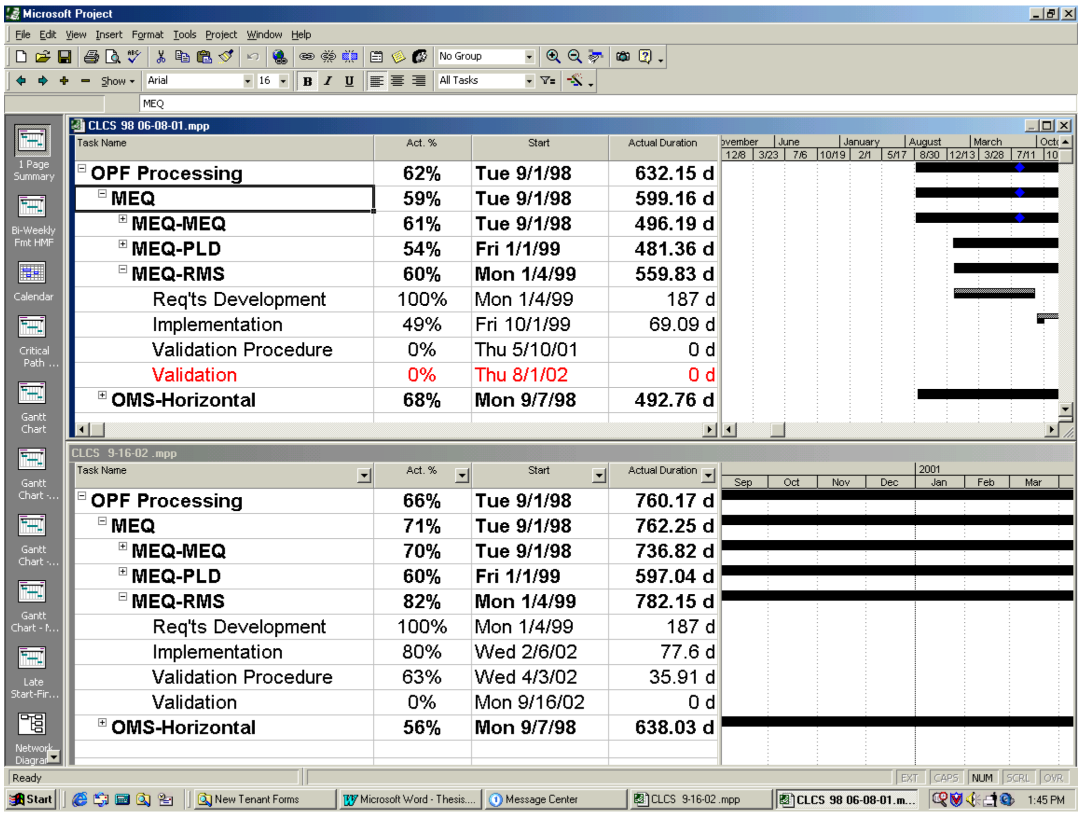
CLCS 98 06-08-01.mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 62% | Tue 9/1/98 | 632.15 d |
| MEQ | 59% | Tue 9/1/98 | 599.16 d |
| MEQ-MEQ | 61% | Tue 9/1/98 | 496.19 d |
| MEQ-PLD | 54% | Fri 1/1/99 | 481.36 d |
| Req'ts Development | 100% | Fri 1/1/99 | 191 d |
| Implementation | 27% | Fri 10/1/99 | 49.95 d |
| Validation Procedure | 0% | Tue 6/19/01 | 0 d |
| Validation | 0% | Mon 6/3/02 | 0 d |
| MEQ-RMS | 60% | Mon 1/4/99 | 559.83 d |
| OMS-Horizontal | 68% | Mon 9/7/98 | 492.76 d |

CLCS 9-16-02 .mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| OPF Processing | 66% | Tue 9/1/98 | 760.17 d |
| MEQ | 71% | Tue 9/1/98 | 762.25 d |
| MEQ-MEQ | 70% | Tue 9/1/98 | 736.82 d |
| MEQ-PLD | 60% | Fri 1/1/99 | 597.04 d |
| Req'ts Development | 100% | Fri 1/1/99 | 191 d |
| Implementation | 20% | Tue 7/9/02 | 13 d |
| Validation Procedure | 0% | Tue 8/27/02 | 0 d |
| Validation | 0% | Wed 10/9/02 | 0 d |
| MEQ-RMS | 82% | Mon 1/4/99 | 782.15 d |
| OMS-Horizontal | 56% | Mon 9/7/98 | 638.03 d |

| MEQ-PLD | Actual Percentage Complete | Actual Duration (Days) | Duration for common percentages complete at existing rate of completion | | | |
|---|---|---|---|---|---|---|
| | | | 25% | 50% | 75% | Calculated Total Duration |
| **Process A** | 27 | 49.95 | 46.25 | 92.50 | 138.75 | 185.00 |
| **Process B** | 20 | 13.00 | 16.25 | 32.5 | 48.75 | 65.00 |
| | | | | | | |
| **Process B** | Provided a | 64.86% | reduction in development time. | | | |



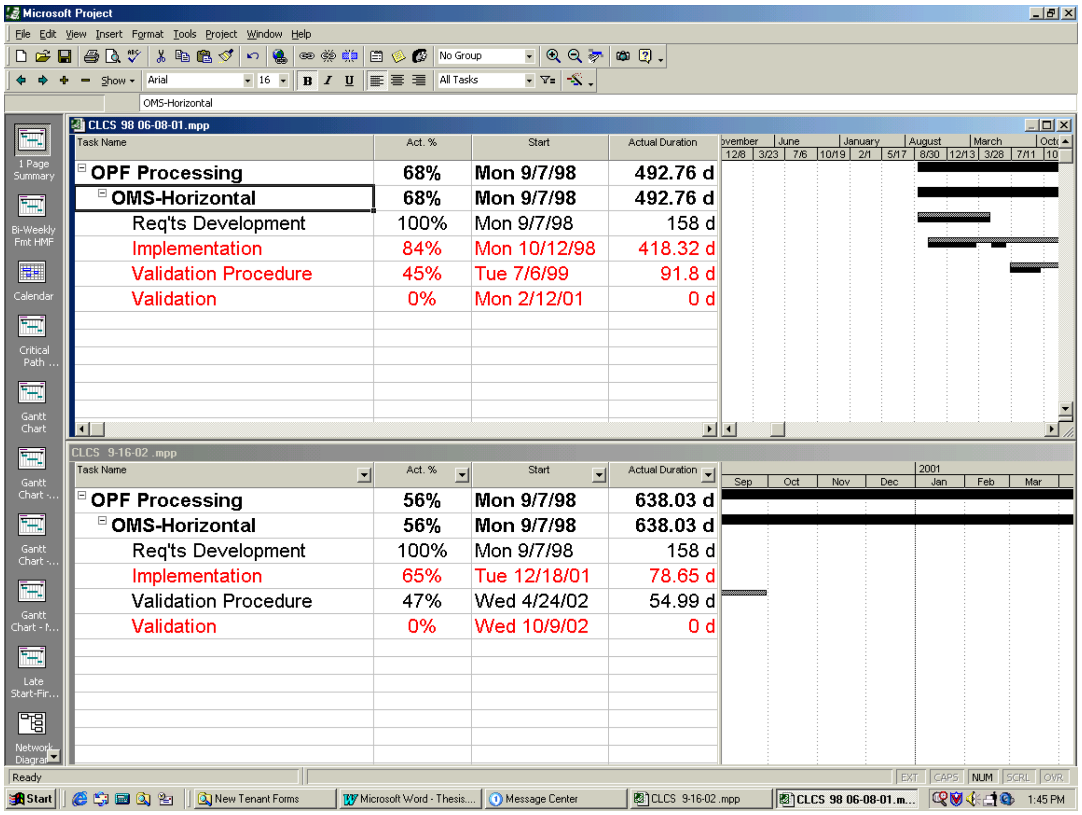
CLCS 98 06-08-01.mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| **OPF Processing** | **62%** | **Tue 9/1/98** | **632.15 d** |
| **MEQ** | **59%** | **Tue 9/1/98** | **599.16 d** |
| **MEQ-MEQ** | **61%** | **Tue 9/1/98** | **496.19 d** |
| **MEQ-PLD** | **54%** | **Fri 1/1/99** | **481.36 d** |
| **MEQ-RMS** | **60%** | **Mon 1/4/99** | **559.83 d** |
| Req'ts Development | 100% | Mon 1/4/99 | 187 d |
| Implementation | 49% | Fri 10/1/99 | 69.09 d |
| Validation Procedure | 0% | Thu 5/10/01 | 0 d |
| Validation | 0% | Thu 8/1/02 | 0 d |
| **OMS-Horizontal** | **68%** | **Mon 9/7/98** | **492.76 d** |

CLCS 9-16-02 .mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| **OPF Processing** | **66%** | **Tue 9/1/98** | **760.17 d** |
| **MEQ** | **71%** | **Tue 9/1/98** | **762.25 d** |
| **MEQ-MEQ** | **70%** | **Tue 9/1/98** | **736.82 d** |
| **MEQ-PLD** | **60%** | **Fri 1/1/99** | **597.04 d** |
| **MEQ-RMS** | **82%** | **Mon 1/4/99** | **782.15 d** |
| Req'ts Development | 100% | Mon 1/4/99 | 187 d |
| Implementation | 80% | Wed 2/6/02 | 77.6 d |
| Validation Procedure | 63% | Wed 4/3/02 | 35.91 d |
| Validation | 0% | Mon 9/16/02 | 0 d |
| **OMS-Horizontal** | **56%** | **Mon 9/7/98** | **638.03 d** |

| **MEQ-RMS** | **Actual Percentage Complete** | **Actual Duration (Days)** | **Duration for common percentages complete at existing rate of completion** | | | |
|---|---|---|---|---|---|---|
| | | | 25% | 50% | 75% | Calculated Total Duration |
| **Process A** | 49 | 69.09 | 35.25 | 70.50 | 105.75 | 141.00 |
| **Process B** | 80 | 77.60 | 24.25 | 48.5 | 72.75 | 97.00 |
| | | | | | | |
| **Process B** | Provided a | 31.21% | reduction in development time. | | | |



CLCS 98 06-08-01.mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| **OPF Processing** | **68%** | **Mon 9/7/98** | **492.76 d** |
| **OMS-Horizontal** | **68%** | **Mon 9/7/98** | **492.76 d** |
| Req'ts Development | 100% | Mon 9/7/98 | 158 d |
| Implementation | 84% | Mon 10/12/98 | 418.32 d |
| Validation Procedure | 45% | Tue 7/6/99 | 91.8 d |
| Validation | 0% | Mon 2/12/01 | 0 d |

CLCS 9-16-02 .mpp

| Task Name | Act. % | Start | Actual Duration |
|---|---|---|---|
| **OPF Processing** | **56%** | **Mon 9/7/98** | **638.03 d** |
| **OMS-Horizontal** | **56%** | **Mon 9/7/98** | **638.03 d** |
| Req'ts Development | 100% | Mon 9/7/98 | 158 d |
| Implementation | 65% | Tue 12/18/01 | 78.65 d |
| Validation Procedure | 47% | Wed 4/24/02 | 54.99 d |
| Validation | 0% | Wed 10/9/02 | 0 d |

| **OMS** | **Actual Percentage Complete** | **Actual Duration (Days)** | **Duration for common percentages complete at existing rate of completion** | | | |
|---|---|---|---|---|---|---|
| | | | 25% | 50% | 75% | Calculated Total Duration |
| **Process A** | 84 | 418.32 | 124.50 | 249.00 | 373.50 | 498.00 |
| **Process B** | 65 | 78.65 | 30.25 | 60.5 | 90.75 | 121.00 |
| | | | | | | |
| **Process B** | Provided a | 75.70% | reduction in development time. | | | |