GRAND SYSTEMS DEVELOPMENT TRAINING PROGRAM

VERSION 10.1

**The union of system engineering,
domain engineering, functional management,
and program management
for the greater good of the enterprise
and customer base.**

VOLUME 2R
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITH

Presented By
Jeffrey O. Grady

JOG SYSTEM ENGINEERING, Inc.

**6015 Charae Street
San Diego, California 92122
(858) 458-0121
(858) 456-0867 Fax
jgrady@ucsd.edu or jeff@jogse.com
http://www.jogse.com**

TABLE OF CONTENTS

TABLE OF CONTENTS

NOTE

Exhibit B available from the lecturer by sending an email to jgady@ucsd.edu and requesting it.

LIST OF ILLUSTRATIONS

LIST OF TABLES

An Effective Specification Development Algorithm

The first order of business on any new program is the identification of requirements that collectively define the problem to be solved. This work can be completed in a timely and affordable way producing a quality definition of the problem space in terms of the minimum collection of requirements each one of which defines an essential characteristic of the system, or item thereof, to be developed. The target we should shoot for is all of the essential characteristics identified (no missed essential characteristics) and no unessential characteristics identified. Success in this process can be encouraged by the enterprise developing a set of specification templates and a corresponding set of data item descriptions coordinated with models selected to accomplish the requirements work and an effective suite of tools within which to capture the results. Every requirement that appears in every specification should be traceable to a model artifact from which it was derived. Before embarking on a new program, the enterprise needs to have selected this preferred set of models and tools and trained its staff to employ those models effectively entering the results in the tools suite. Every specification released is printed from the tools suite and should pass through a formal review and approval process as should any subsequent changes to any specification. As the models create work products, commonly simple diagrams, they should be saved in an organized fashion in paper or computer medias and formally released so as to be available for future system phases and modification projects.

Figure 1 offers a view of the overall process within which the preparatory work and definition activity that is discussed in the first section falls. Before entering a program, the enterprise should have prepared itself for the requirements work that will have to be done. The enterprise needs models to apply for the cases where the product is going to be implemented in computer software and in hardware. Both cases are covered in this tutorial followed by a discussion of integrated modeling.

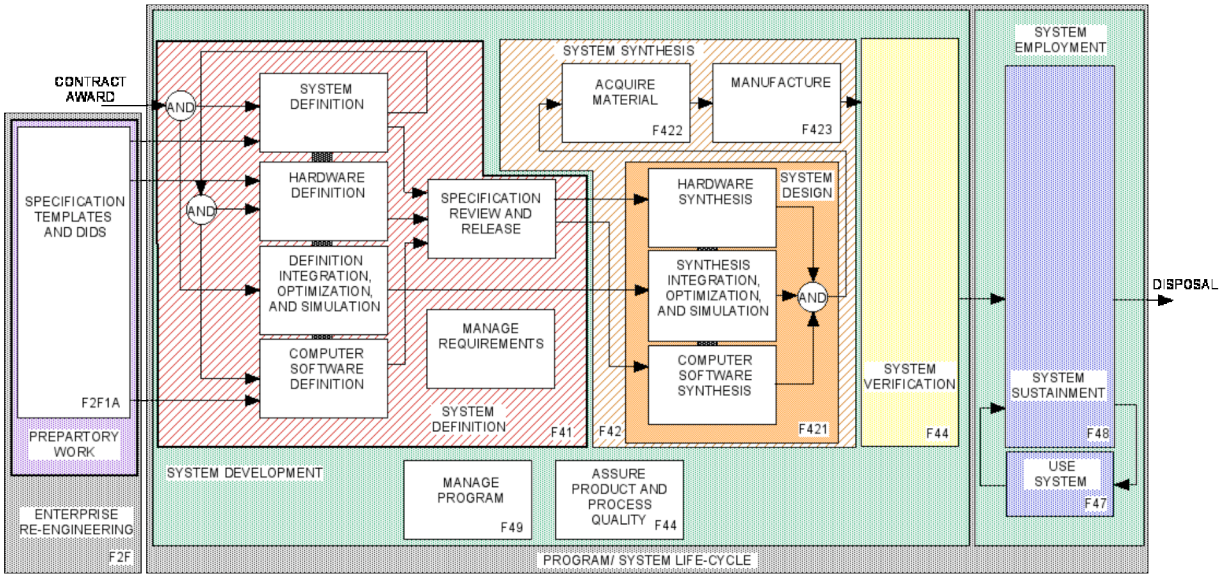


Figure 1 Overall Process

1. Specification Templates and DIDs

Many system development organizations experience some difficulty in clearly identifying appropriate requirements for inclusion in specifications they must develop and they find it difficult to accomplish the work in an affordable and timely fashion. Over a period of the last two years the author developed an algorithm for improving system development organization ability in this work using templates and specially developed data item descriptions (DIDs). It requires some work to prepare the functional organization to support programs and some work on the part of proposal teams to accomplish initial analyses and extend the templates made available from the functional system engineering department to provide program-ready data, and work by the program teams starting with contract award and running through the period of time while specifications are being developed.

The goal of this specification algorithm is to provide for affordable and timely enterprise and program definition and documentation of new product technical requirements, the management and maintenance of related data, and publication and subsequent configuration control of the resulting documents.

The work required to implement the algorithm can be described in three preparatory steps: (1) enterprise engineering work, (2) system engineering generic work, and (3) proposal team work. The first two steps are illustrated in Figure 2. The numbers in the blocks coordinate with the steps in the algorithm. Recommended functional department responsibility for accomplishing the indicated tasks is noted at the lower left corner of the task blocks.

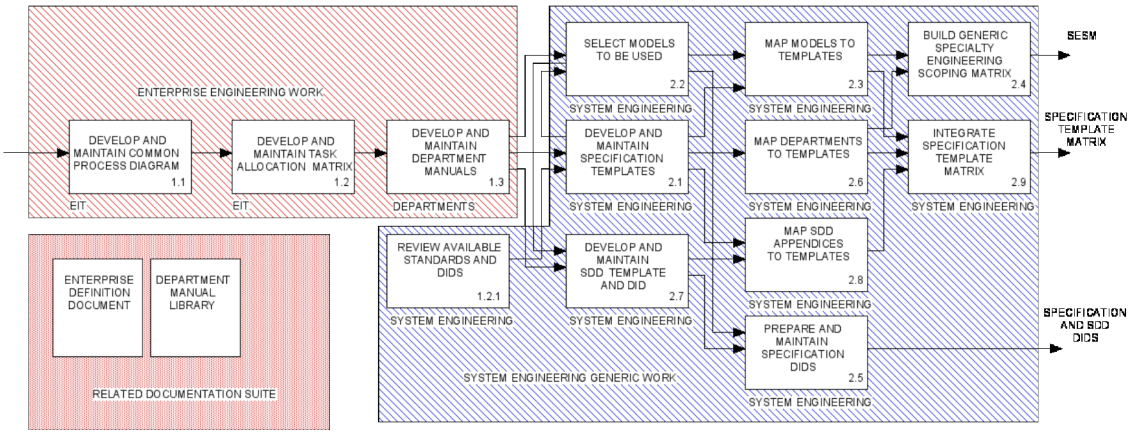


Figure 2 Preparatory Steps

1.1 Enterprise Engineering Work

Identify and staff an enterprise integration team (EIT) that is responsible for engineering the enterprise common process and acting as the process integration and optimization agent during its development and implementation on programs. The EIT should report to the enterprise executive.

1.1.1 The enterprise, through the efforts of the EIT, must develop a common process diagram that generically identifies all program work at a level of indenture that is adequate for making clear what work must be commonly done and allocating the corresponding work responsibilities to functional departments responsible for supplying programs with the necessary resources to accomplish that work well.

1.1.2 Allocate all work on the common process diagram to functional departments forming a task allocation matrix. This matrix establishes the requirements work that functional departments must be prepared to do on programs and any training that the functional departments are funded for and capable of performing must be focused on these tasks. This matrix covers the whole enterprise capability but in this section we are focused on doing the requirements work.

1.1.3 Functional departments collect all work allocated to them and build department manuals that explain what work must be done and provide links or descriptions for how to perform this work. One of the departments will be system engineering that will have responsibility for specification development and management on programs. For each task a functional department has responsibility, that department must identify work products that will result as a function of having accomplished the work on a program. EIT must integrate and optimize the evolving functional department work descriptions and work product identifications to ensure overall efficiency and effectiveness. All work products must have at least one user. Work products must be linked to the common process diagram tasks. Specifications are an example of a task work product and the work product of interest in this algorithm.

1.2 System Engineering Generic Work

1.2.1 To the extent that work products are documents, the responsible functional department must prepare a template containing the basic structure of the document in terms of generic paragraphing structure and calls for graphical images. In preparing for implementation of specification development and management work on programs in general, the functional system engineering department will select the specification standards that will be applied respecting the common customer base of the enterprise. They will review these standards and associated data item descriptions (DID), ensure that the system engineering department manual adequately covers specification standards the enterprise has chosen to respect, and build a set of specification templates (paragraph numbering and titles only), one for each kind of specification that will commonly have to be prepared on programs.

1.2.2 For each specification template defined, the system engineering department will determine one or more preferred modeling approaches for each kind of requirement in the template. The modeling approaches encouraged are the following relative to the primary kinds of specifications that will have to be developed:

System Specification	Traditional Structured Analysis
Hardware Performance Specification	Traditional Structured Analysis
Software Performance Specification, General	Unified Modeling Language (UML)
Software Performance Specification, Database	IDEF-1X
Software Performance Specification, DoD IS	DoDAF

1.2.3 Map models to templates such that for any of the specification s listed above there is a model identified for each paragraph of each specification. Ideally, all of the paragraphs of a particular kind of specification would employ models from the same family as suggested in the pairing above.

1.2.4 Map specialty engineering disciplines to specification templates in preparation for program mapping of these disciplines to specific product entities as a means of directing specialty engineering requirements analysis work.

1.2.5 For each template and model combination, prepare a DID communicating how the analyst will prepare the specification using a particular modeling approach and template. The DID must clearly show the connection between the modeling artifacts and the template paragraphing structure. The DID paragraphing structure must coordinate with the modeling components that are intended to yield derived requirements.

1.2.6 Map the functional departments that will be responsible for performing requirements analysis on programs to the template paragraphing structure for each kind of specification telling where the programs will acquire the analysts to accomplish the specification development work.

1.2.7 Prepare a template and DID for a system definition document (SDD) within which program structured analysis work products will be captured and configuration managed. These work products are to be captured in appendices of the document. An alternative is to capture the modeling artifacts within a computer tool that can be configuration managed.

1.2.8 Map the appendices of the SDD to the paragraphing structure of the templates telling in what appendix the corresponding work products will be captured.

1.2.9 Combine the functional department map (1.2.5), models map (1.2.3), and SDD appendix map (1.2.7) on a single matrix for each template and make these matrices available for program use.

1.2.10 EIT and the functional system engineering department must cooperate on selection of one or more computer tools or paper and pencil algorithms with which to accomplish requirements analysis on programs. Built a generic schema coordinated with preferred methods and models.

1.3 Proposal Work That Prepares the Program for Initiation

Proposal team work is illustrated in Figure 3. Blocks that do not have numbers coordinating them with the steps in this algorithm are not covered by the algorithm because they are not directly related to requirements analysis and specification development but these blocks add valuable context.

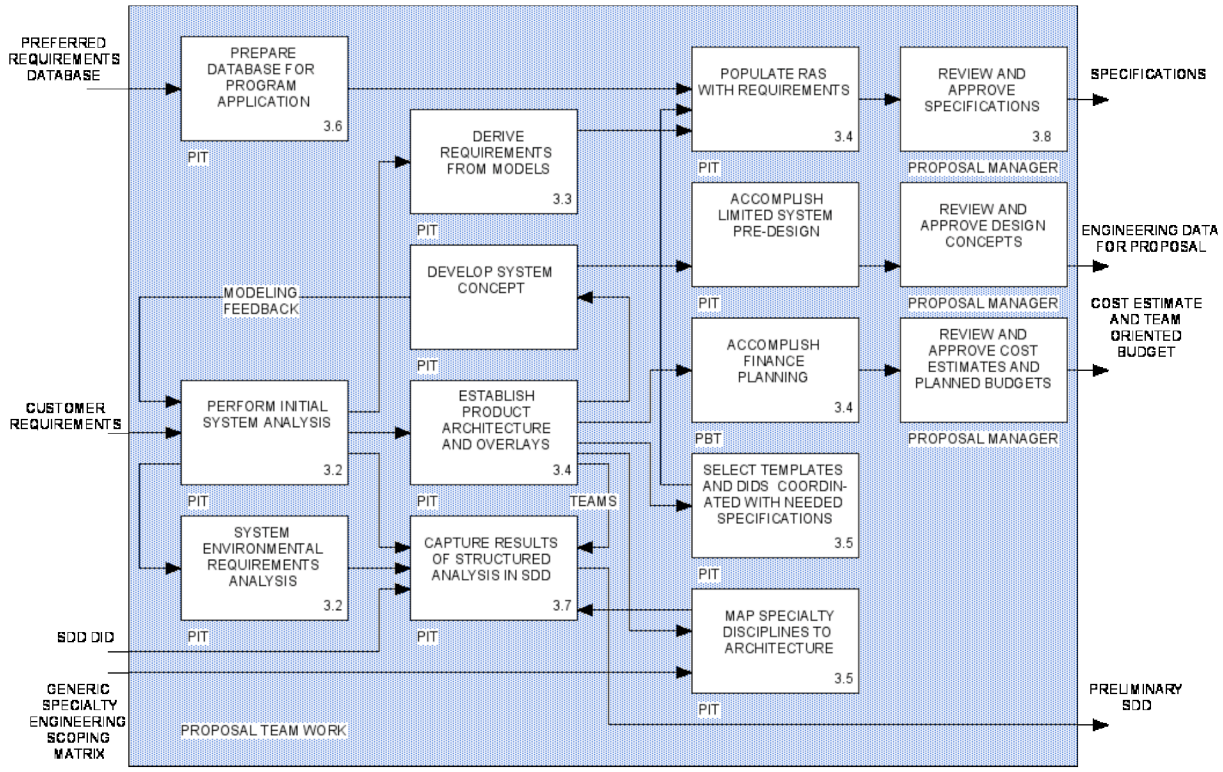


Figure 3 Proposal Team Work

1.3.1 When beginning the proposal or program work, the manager should establish a program integration team (PIT) staffed by engineering, manufacturing, verification, logistics, and quality and a program business team (PBT) staffed by finance, contracts, scheduling, business information systems, and administration. Both of these teams should report to the program manager. These two teams could be combined as a staff function to the proposal manager but they will have integrating and optimization roles to play across the product oriented teams.

1.3.2 The PIT will perform an initial system analysis that will result in a clear understanding of any requirements provided by the customer, formatting of those requirements into alignment with the enterprise DID for a system specification, and adding to those requirements the results of their own system analysis work. A system environmental requirements analysis activity will expose a set of tailored standards covering the natural environment corresponding with spaces within which the system shall have to operate. A threat analysis will lead to exposure of hostile requirements. An interface analysis will identify external and top level internal interfaces that will be characterized in requirements statements. The result will be a preliminary system specification for submission with a proposal. If possible, this analysis work will be continued to develop all of the immediately subordinate specifications each of which will be the development responsibility of one of the top level integrated product and process teams (IPPT) to be identified and staffed subsequent to contract award.

1.3.3 The modeling work described in paragraph 1.3.2 will yield modeling artifacts from which requirements may be derived. The preliminary system specification development and any other specifications developed during the proposal effort will follow the pattern described in the

next paragraph. The second tier specification development may be delayed until a contract award but should precede the formation of the top tier IPPT. In all cases, requirements are derived from a model.

1.3.4 Requirements flowing from the structured analysis work will flow into a requirements analysis sheet (RAS) implemented in a computer database tool. All requirements entered into the RAS must include a traceability reference to the model from which they were derived.

1.3.5 Out of the initial PIT system analysis work will also come the preferred product entity breakdown diagram upon which the PIT, working in concert with integrated business team personnel, will construct overlays for organization responsibility breakdown (IPPT assignments), specification tree breakdown, engineering drawing breakdown, work breakdown (WBS), and manufacturing breakdown The work breakdown will be handed off to the business team that will use it as the basis for building the program work definition, cost estimate, and IPPT work budgets. The IPPT will be assigned so as to align perfectly with the WBS making it possible to present to each IPPT leader as the teams are formed, a copy of the top level specification for which the team is responsible and the related budget and planning package for the whole WBS the team is responsible for as well as their top level schedule responsibilities encouraging the result that the IPPT leader may be held accountable for managing all aspects of the development of the entity assigned to the team.

1.3.6 The PIT will select a set of templates that correspond with the kinds of specifications that will have to be prepared on the program and the related DIDs that are coordinated with the models that will be applied in the analytical work. The PIT must also map specialty engineering disciplines to product entities to aid teams being formed to staff appropriately.

1.3.7 The PIT must take action to cause adequate computer tool seats to be allocated to the proposal team and accomplish any planning necessary for the subsequent program relative to the use of any requirements database tools and make any needed adjustments in database schema for the program.

1.3.8 The PIT shall capture the results of structured analysis work performed during the proposal activity in a preliminary system definition document (SDD) that will be used as the basis for subsequent lower tier analyses.

1.3.9 Any specifications developed in the proposal effort must be formally reviewed and approved by the proposal manager.

1.4 Work Subsequent to Contract Award

Specification related work to be accomplished subsequent to contract award is illustrated in Figure 4. This work is repetitious in nature progressively working down through the expanding architecture. Top-level teams may shred out during program work yielding sub-teams but in all cases, the top-level teams are responsible as managers for all lower tier team activity. This telescoping management responsibility is applicable throughout the team structures. During program performance, lower tier requirements analysis responsibility may be passed down

through the team structure with immediately superior team reviewing and approving of all immediately lower tier team specifications or the responsibility may be retained by the higher-level team but these decisions must be coordinated with the team budgets and staffing considerations arrived at during proposal work.

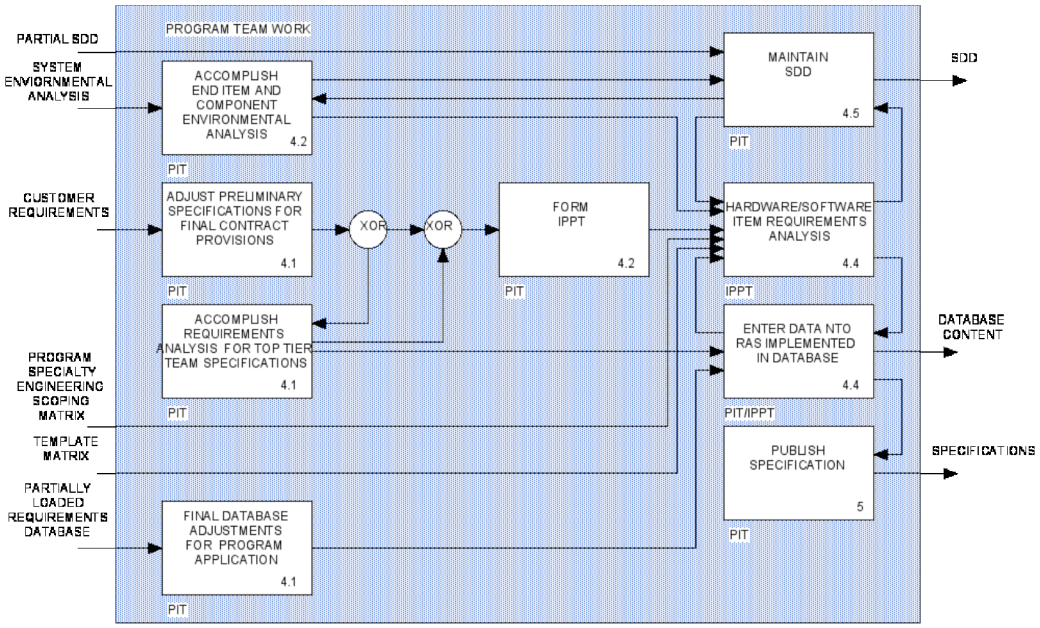


Figure 4 Program Work

1.5 The Preferred Templates

Ideally, the development organization would build a set of templates (paragraphing structure with no content) and data item descriptions (DID) that tell how to build a specification following the related template using a particular set of models. These should be maintained by the functional department in system engineering that has overall requirements and specification work responsibility. These should be available for reference or download by any new program.

Figure 5 offers a view of the preferred template for a system or hardware specification using traditional structured analysis as the modeling choice. The template is annotated with the preferred modeling artifact that will be used to identify the corresponding requirements and the functional department from which the program will obtain personnel to accomplish the related modeling work using the Figure 3 organizational structure. The data item description (DID) acronym in the model column means that the content is driven by the content of the DID used as the basis for the specification. The department references are cut at a very high level in this case and should be identified at one or two layers below this level but Figure 3 goes no deeper. The APP column gives the Appendix in the System Definition Document where the work products can be found.

The structure in Figure 5 can also be used for computer software requirements specifications (SRS) with paragraph 3.1 rewritten for UML and the model column updated to reflect UML artifacts.

PARA	TITLE	MODEL	DEPT	APP
1.	SCOPE	DID	2100	
2.	APPLICABLE DOCUMENTS	DID	2100	
3.	REQUIREMENTS	DID	2100	
3.1	Functional and performance requirements	DID	2100	
3.1.1	Missions	Mission Analysis	2100	
3.1.2	Threat	Threat Analysis	2100	
3.1.3	Required states and modes	DID	2100	
3.1.3.1	Functional analysis	DID	2100	A
3.1.3.2	Subordinate entities	DID	2100	C
3.1.3.3	Interface relationships	DID	2100	D
3.1.3.4	Specialty engineering requirements	DID	2100	E
3.1.3.5	Environmental model	DID	2100	B
3.2	Entity capability requirements	Functional Analysis	2100	A
3.2.m	Capability m	Functional Analysis	2100	A
3.2.m.n	Capability m, requirement n	Functional Analysis	2100	A
3.3	Interface requirements	N-Square Diagram	2100	D
3.3.1	External interface requirements	N-Square Diagram	2100	D
3.3.1.m	External interface m	N-Square Diagram	2100	D
3.3.1.m.n	External interface m, requirement n	N-Square Diagram	2100	D
3.3.2	Internal interface requirements	N-Square Diagram	2100	D
3.3.2.m	Internal interface m	N-Square Diagram	2100	D
3.3.2.m.n	Internal Interface m, requirement n	N-Square Diagram	2100	D
3.4	Specialty engineering requirements	DID	2100	E
3.4.m	Specialty Engineering Discipline m	Specialty Scoping		E
3.4.m.n	Specialty Engineering Discipline m, Requirement n	Specialty Scoping		E
3.5	Environmental conditions	3-Layered Env Model	2100	B
3.6	Precedence and criticality of requirements		2100	
4.	VERIFICATION	DID	2100	
5	PACKAGING	DID		
6.	NOTES	DID	2100	
6.1	Requirements traceability	DID	2100	
6.1.1	Inter-specification specification traceability	DID	2100	
6.1.2	Verification traceability	DID	2100	
6.1.3	Modeling traceability	DID	2100	
6.1.4	Section 2 traceability	DID	2100	
6.1.5	Programmatic traceability	DID	2100	
6.2	Glossary	DID	2100	
6.3	Specification maturity tracking	DID	2100	
A.	APPENDIX A	DID	2100	

Figure 5 System and Hardware Specification Template

A similar mapping should be provided for the software requirements specification (SRS) based on, in the author's view, the use of UML. Customers often require conformance to a DID supplied by them but may permit tailoring. The outline included in Figure 5 is a significantly tailored version of MIL-STD-961E to group all requirements so as to correspond with the modeling components contained in traditional structured analysis described in paragraph 2.2.1. k of the standard. The author believes this format will work with software as well but a different DID is required coordinated with the modeling approach selected (UML encouraged).

1.6 Modeling Work Product Capture Document

Programs should also be provided with a template for a System Definition Document (by whatever name) within which they can easily capture the results of all modeling work so that it can be preserved beyond the period of time when that work is actively being pursued. A later section describes this document. The appendices of the SDD are referred to in Figure 5 in the APP column and explained in Figure 20 and related text.

2 Structured Analysis

There are many models that can be used to accomplish an organized requirements identification effort that is preferred to an ad hoc method because it will most often hit the target noted earlier - identification of all of the essential characteristics and identification of no unessential characteristics. This process description encourages the use of traditional structured analysis in the near term to develop and identify the requirements for systems, hardware, facilities, real property improvements, and personnel actions captured in procedures as discussed in paragraph 2.1. One of the most difficult tasks in system development revolves around the relations between the system entities, the interfaces. This discussion of traditional structured analysis also contains a complete algorithm for identifying all system relations. Where the product is going to be implemented in computer software, unified modeling language (UML) is encouraged within a process context offered in paragraph 2.2. It is intended that a development organization should develop a process transformation roadmap needed to transition from this mixed method for the modeling work to a universal modeling approach, still evolving, that can be applied to all requirements work at the earliest possible. See paragraph 2.3 for a discussion of integrated modeling.

2.1 Traditional Structured Analysis

Figure 6 illustrates an overview of the traditional structured analysis process (TSA). The eleven numbered steps are briefly introduced followed by additional details in subordinate paragraphs.

1. Understand the User Requirements - Through conversation with the user and/or reading a user requirements statement or specification, the developer tries to understand the user's need. This is not ever easy because the user is able to explain only what their mission interests are and the developer needs hard engineering data.
2. Decompose - Users commonly have problems that are too grand to be easily understood in a single small document or simple diagram. These problems commonly have to be decomposed or partitioned into a collection of smaller related problems.
3. Functional Flow Diagram - The TSA approach employs some form of functional analysis as the decomposition medium.
4. Performance Requirements Analysis - The functions are translated into performance requirements.
5. Requirements Analysis Sheet (RAS) - The strings of functions, performance requirements, and product entities to which they are allocated appear as rows in the RAS. The RAS also is used to capture all system requirements linked to the model from which they were derived.

6. Requirements Allocation - Performance requirements are allocated to product entities in the RAS.
7. Product Entity Structure - The physical and logical entities that comprise the system are arranged in a hierarchical structure that is used as the basis for WBS, specification identification, team assignments, and many other program applications.
8. N-Square Diagram - The interfaces that must exist between the product entities are identified through a pair-wise analysis of all possible interface relationships.
9. Environmental requirements for the expanding product entities are determined through application of a three-layer model.
10. Specialty engineering requirements are identified by a group of specialist linked to the product entity structure by a specialty engineering matrix.
11. This process is applied iteratively as lower tier entities are identified through functional analysis.

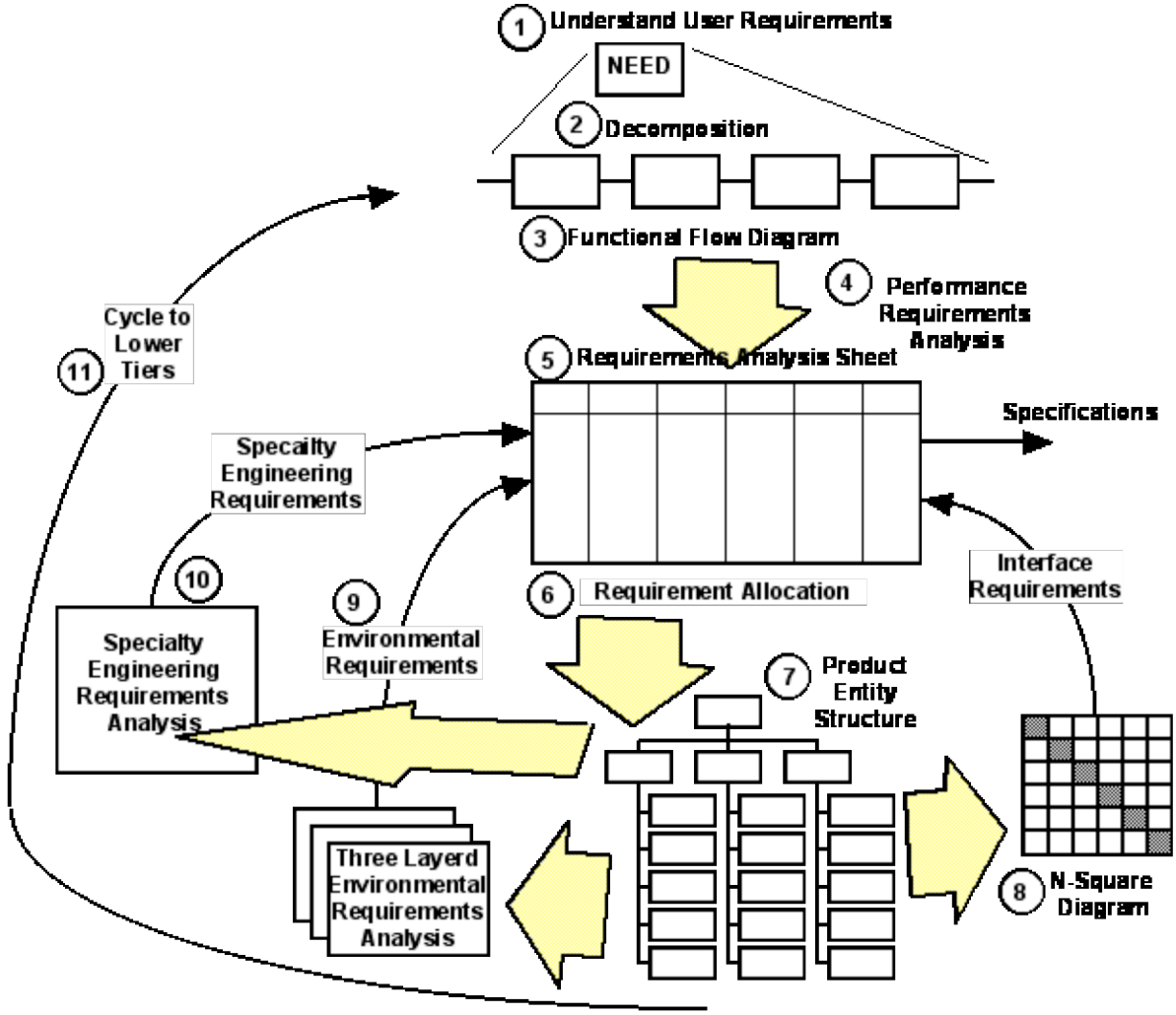


Figure 6 Overview of the Traditional Structured Analysis Process

2.1.1 A System Defined

A man-made system is a collection of entities that are meant to interact in predictable ways with an environment and with each other via relations between them to achieve a useful function identified and articulated by a customer as a mission need statement. Therefore, systems are composed of entities and relations between the entities. The system is intended to satisfy the mission need statement, the system's ultimate function, depicted on system diagrams as a rectangular block titled System Need and identified with a functional identifier F. The need is allocated to the system depicted on system entity model by a rectangular block named "system" (or a particular system) and identified with a product entity identifier A.

A system interacts within an environment as shown in Figure 7. The environment for every system is everything in the Universe (U) less the entities that are part of the system product entity structure (Q = U - A). One can reduce the scope of the environment to those elements that will have some influence on the system. The line joining the system and environment in Figure 7 (I2) indicates the relations between the two (external interfaces). The line joining the system on both terminals (I1) indicates the internal relations between system entities (internal interfaces) yet to be defined within the system.

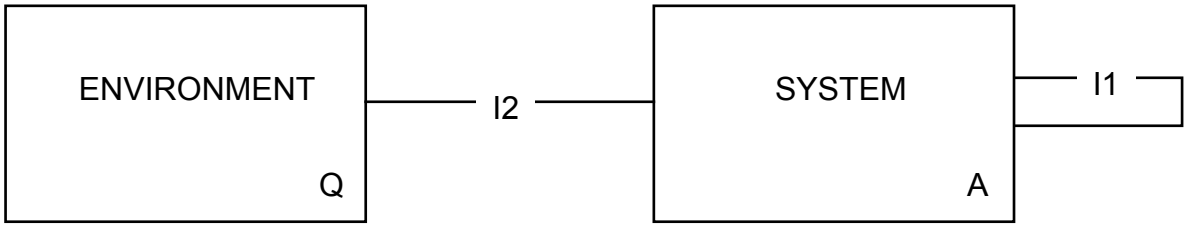


Figure 7 Ultimate System Diagram

2.1.2 The System Environment

The environment for any system is composed of the subsets illustrated in Figure 8. While all environmental effects on the system are relations, they may be partitioned between those that are commonly considered environmental stresses and the cooperative environmental elements that are treated as external interfaces commonly developed by a pair of teams or contractors responsible for the terminal product entities.

A context diagram, such as that shown in Figure 9, even though similar in nature to Figure 7, offers a useful simple model for focusing attention on identifying all external relations. Some of these terminators will be natural, non-cooperative, or induced environmental stresses. Others include hostile stresses determined through a system threat analysis as well as both stresses and useful relations with cooperative systems. Though this diagram was conceived as the beginning of the modern structured analysis model, it has a useful purpose in TSA as a means of viewing all of the inside-outside relationships and in UML as an organizer of use cases. It can be used to make a first impression in the line I2 in Figure 7.

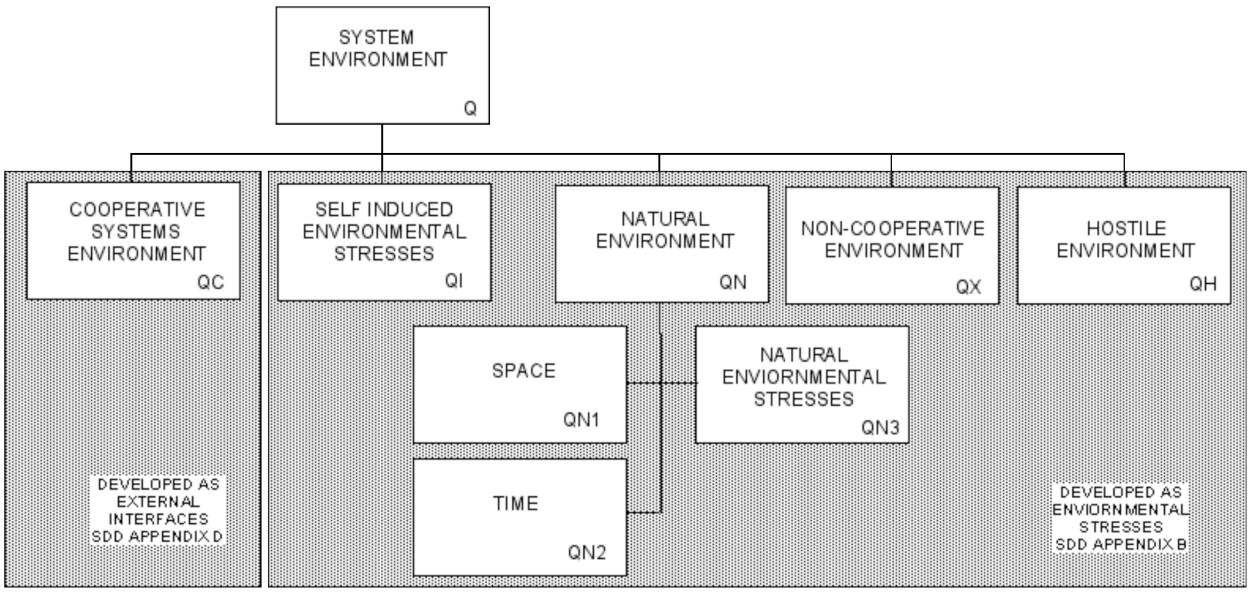


Figure 8 System Environment

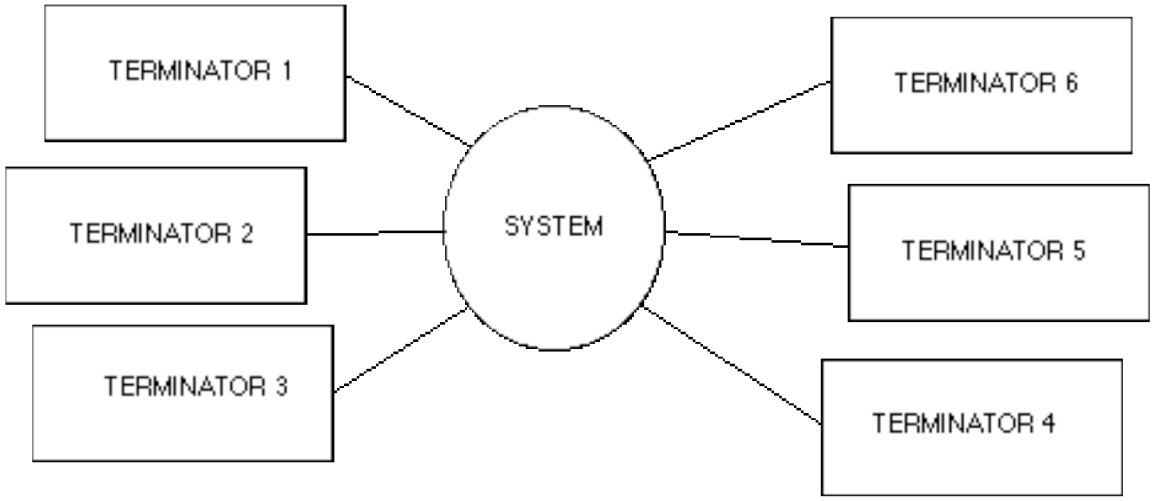


Figure 9 System Context Diagram

The system natural environment is determined by defining all of the spaces within which the system will be employed based on an analysis of the intended mission and basing concept. The spaces are coordinated with a set of environmental standards. Each standard is studied for necessary content and the remainder tailored out. Each selected parameter is then studied for an appropriate range. The system natural environment is then the union of the selected parameters from the selected standards.

The non-cooperative environment is defined by determining what stresses will be applied to the system from man made systems which are neither hostile nor cooperative. An example of non-cooperative stress is electromagnetic energy. Self-induced environmental stresses are not easily

determined at the system level because one needs to understand energy sources and other stressors within the system determined as part of the design of end items.

System cooperative environmental relations are defined by determining how the system to be developed will associate with other friendly systems already in existence or being developed. These associations may be coupled into or out of the system in terms of information, physical association, materials, or energy.

2.1.3 System Functionality

A function is a necessary activity for a system to perform. It may be static, dynamic, or both. It should be named using an action verb followed by a noun. A function is depicted in modeling the system as a rectangular block identified by an action verb name centered in the box and a function identifier (ID) in the lower right corner. The ultimate function for any system is the customer need the name of which is the need statement possibly paraphrased to fit into the space provided coupled with a function identifier F.

Two or more functions can be linked together using directed line segments to show a sequence of functions. In Figure 10 the understanding is that function F1 must be accomplished before function F2. Combinatorial symbols may be added to permit more complex sequential relationships. The combinatorial symbols encouraged are AND, inclusive OR (IOR), and exclusive OR (XOR) with the common logical meanings. Enhanced functional flow block diagramming adds loop (repeat a function until a specific outcome has been attained) and iterate (repeat a function a specific number of times) combinatorial symbols that can be useful. Diagrams so constructed are called functional flow diagrams. These diagrams may be oriented on the page with their primary flow axis arranged horizontally or vertically with the flow in either direction.

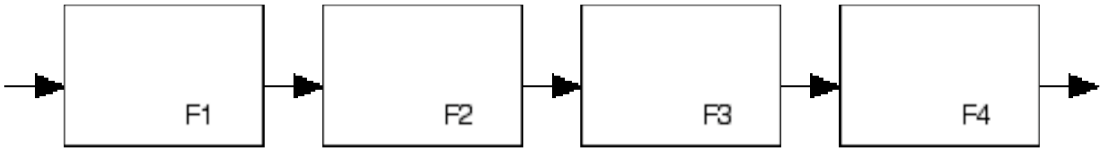


Figure 10 Function Sequence

For any function with identifier F@ (where @ is a string of length n (including n=0) composed of characters from the set {A through Z less O)U(a through z less l)U(0 through 9)}) there may exist one or more subordinate functions F@# (where # is a single character from the same set identified above which differentiates other functions at that level from one another). This is illustrated in Figure 11. Every function need not have an expansion. There is no need to assign function identifiers in alpha numeric sequence on a page of the diagram but it helps the human to use the diagram if they are assigned initially coordinated as much as possible with the function sequence. If a function is deleted subsequent to a release of the diagram, that identifier should not be used again. If the number of functions on any one diagram exceeds the maximum number of symbols available, 60, change the diagram to reduce the number to less than 60.

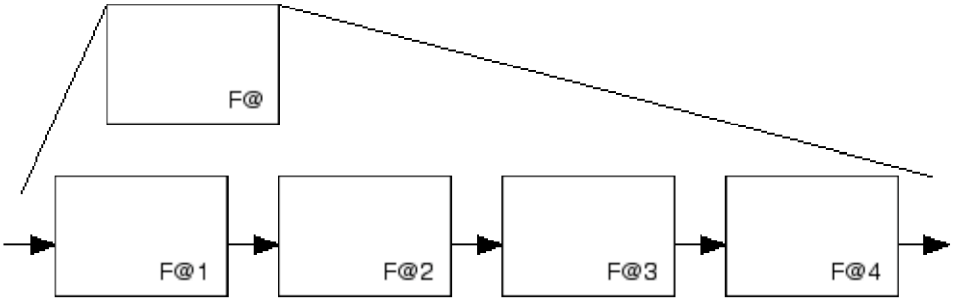


Figure 11 Function Decomposition

Ideally, who ever accomplishes the initial analysis of the need, would do so using the functional analysis process described here where the first decomposition is the system life cycle as shown in Figure 12 and the second is an expansion of the life cycle function "Use System" (F47) to expose the top level operational intent and initial content of the user requirements documentation or preliminary system specification. If the customer or other initial analysis agent applies an unstructured or ad hoc approach, then the development organization may have to accomplish a functional analysis and try to map the requirements identified by the customer (user and acquisition agent) to the functionality exposed when they do accomplish this work.

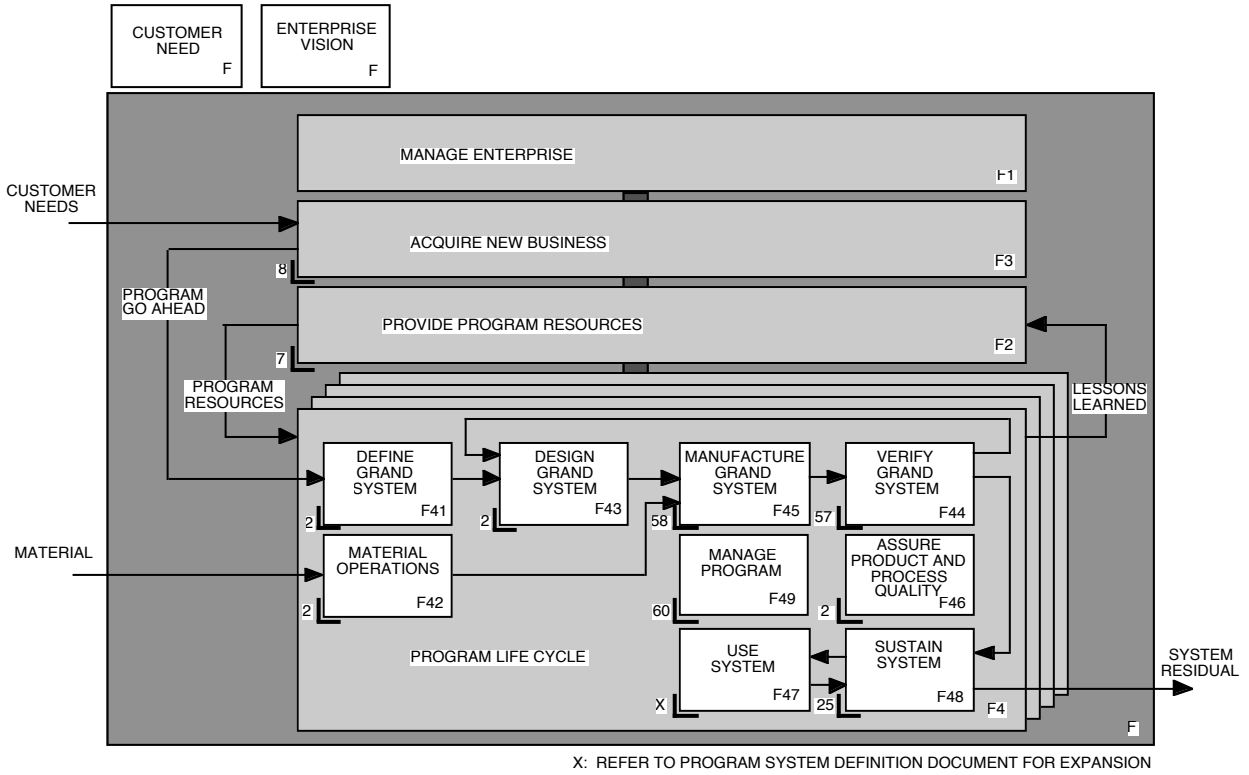


Figure 12 System Life Cycle

Ideally, the development organization would extend the functional analysis into the remainder of the system life cycle functions as well as the Use System function determining appropriate resources for the process steps of the development program and the product system being developed such that the physical product delivered will be jointly optimized relative to its product and process. Commonly, process functions do, or should, influence product functions and the corresponding product entities needed as well as the opposite case.

2.1.4 Performance Requirements Derivation and Allocation

The functions identified in the functional flow diagram must be translated into performance requirements that tell what the system and its parts must do and how well it must do them as shown in Figure 13. These statements can be first developed as primitive statements for example phrased as "Velocity $\geq$ 600 knots" without complete sentence structures and subsequently transformed into complete sentences in the chosen language. Traditionally, a requirements analysis sheet (RAS) has been used to capture the function identification, the primitive performance requirements statements, and the allocation of these performance requirements to product entities. One could allocate the function names directly to architecture but often one finds a one-to-many allocation result this way whereas allocation of performance requirements tends to follow a one-to-one pattern. To fully characterize a function it may require identification of multiple performance requirements and these several performance requirements may be allocated to different product entities.

The RAS as traditionally used is incomplete, unfortunately, and this discussion will use a RAS complete. We will show how the three kinds of constraints can also be captured in the context of Figure 13 shortly. The intent is to be able to capture all requirements in a RAS linked to a modeling artifact implemented in a computer application.

FUNCTION		PRODUCT ENTITY		REQUIREMENTS	
MID	NAME	PID	NAME	RID	STATEMENT
F123	Provide Thrust	A12	Engine	WE89FS	Thrust > 10,000 pounds at sea level

Figure 13 Traditional Requirements Analysis Sheet

2.1.5 Product Entity Structure

System functionality is accomplished by physical entities that form part of the physical system. These entities in the system, in the aggregate, comprise the product entity structure that the author formerly referred to as the system architecture. The word architecture has taken on a significantly broader meaning in recent years convincing the author that what he formally referred to as architecture should now be referred to by the more isolating term product entity

structure. The function F is accomplished by product entity structure element A, the system, by definition as shown in Figure 7. Lower tier entities should be exposed following Sullivan's encouragement of form follows function. Trade studies may be appropriate to make hard decisions on the best implementation of a particular function in early program phases. The physical entities that accomplish functionality can be partitioned into five classes: (1) hardware, (2) computer software, (3) facilities and improvements to real property, (4) procedural definition, and (5) humans following procedures or acting autonomously. We could merge the last two into one. The relationship between functions, the physical entities in the product entity structure, and the corresponding performance requirements is depicted on a simple requirements analysis sheet as shown in Figure 13.

The aggregate product entity structure for a system is illustrated in a hierarchical model connecting the product entities arranged into a breakdown block diagram as illustrated in Figure 14. The entity identifiers follow the same pattern as the function identifiers explained earlier beginning with a letter A because of the author's previous use of the word architecture for this view. The entities stream out of the RAS available for structuring into the product entity block diagram. Ideally, this work would be accomplished by a team of people representing hardware and software engineering, manufacturing, procurement, and verification with strong system/program leadership by the PIT. Initial functional analysis and allocation must concentrate on understanding user mission needs. This will generally require intense interaction with the user, ideally using system models to encourage mutual understanding. Alternative ways of implementing functionality with different product entities should be considered and where the decision is very difficult they should be subjected to a trade study.

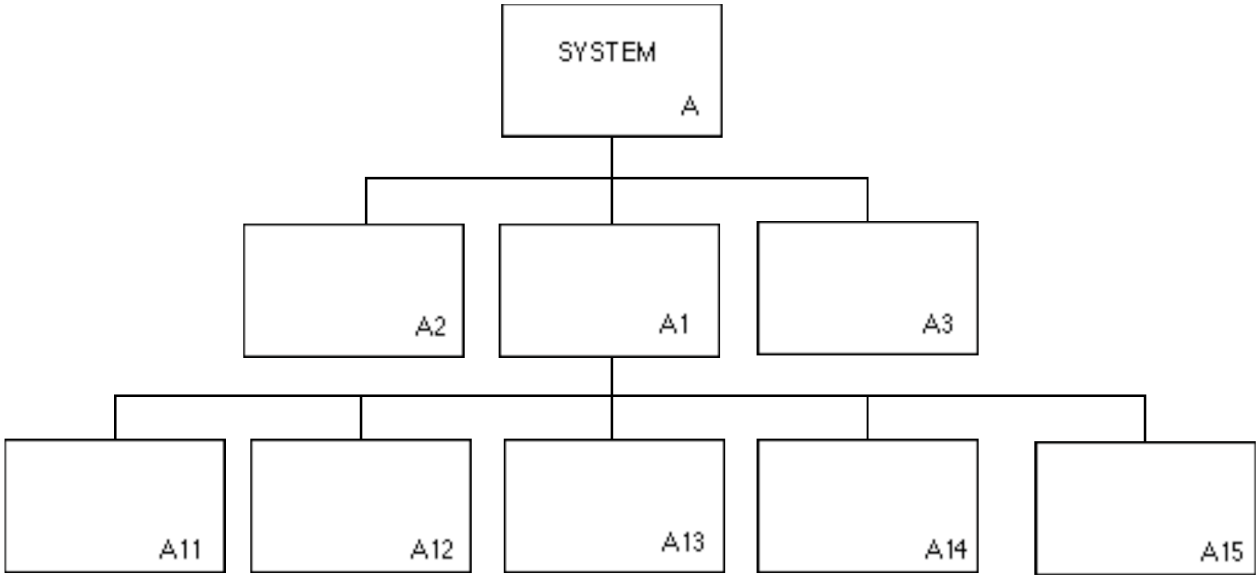


Figure 14 Product Entity Structure

2.1.6 Allocation Pacing Alternatives

The conduct of the functional analysis and allocation work can be paced in one of four different ways:

(1) Instant – As soon as functions and/or their corresponding performance requirements are identified, they must be immediately allocated to the expanding product entity structure.

(2) Terminal - All of the functional analysis must be complete before any functions and/or their corresponding performance requirements can be allocated to product entities.

(3) Layered - The analyst completes one layer of the expansion of a functions and must allocate all of the exposed functionality and/or the corresponding performance requirements to product entities before further expanding that function. This works best if all product entities related to a layer are defined in terms of their requirements and design concept before pursuing the next functional layer and its allocation but there generally is not sufficient time available and one must accomplish a lot of this work in parallel leading to some risk and interaction. The layered approach has been popularized by Mr. Bernard Morais and the late Dr. Brian Mar under the title FRAT.

(4) Progressive – The analyst completes several layers of the functional analysis without allocating any of it to product entities. At a layer guided by experience the analyst begins allocating performance requirements derived from higher tier functionality to product entities. Design concepts are defined for the product entities at the higher levels and these act to both guide and constrain lower tier functionality progressively. Allocation is delayed throughout the analysis such that higher tier design concepts help to steer lower tier functional analysis tending to isolate iteration to one structure (functionality, architecture, or allocation) at a time. The two extreme cases (1 and 2) are flawed due to a need to iterate F, A, and allocations excessively in the case of 1 and a common need to significantly change lower tier functionality after the higher tier allocations begin in the case of 2. The progressive approaches either by layer or guided by experience will generally produce better results with less modeling hysteresis.

There exists a downward limit in decomposition or expansion of the functional portrayal. This limit for any one branch in the expansion is best determined on the product entity plane based on the analyst's understanding of the problems related to the lowest tier product entities. Where all of the lowest tier entities are fully characterized supporting procurement or in-house design, the functional analysis work can be reduced to maintenance of the models and related data. There is one more layer of requirements related to parts, materials, and processes but that is driven by design decisions during synthesis and commonly has already been completed by those responsible for the sources of these entities. The logistics analysis process may require a switch to a process diagram and if the progressive allocation approach described has been followed the functional flow will have been migrated toward a process diagram at the lower tiers.

Figure 15 illustrates a geometric view of the process of deriving performance requirements from functions and the allocation of those performance requirements to product entities. For example, a performance requirement has been derived from F4713 and allocated to product entity A11. This plane represents the normal RAS used only for function allocations. As suggested by the other structures we will use the geometric structure to expand the RAS to cover all requirements.

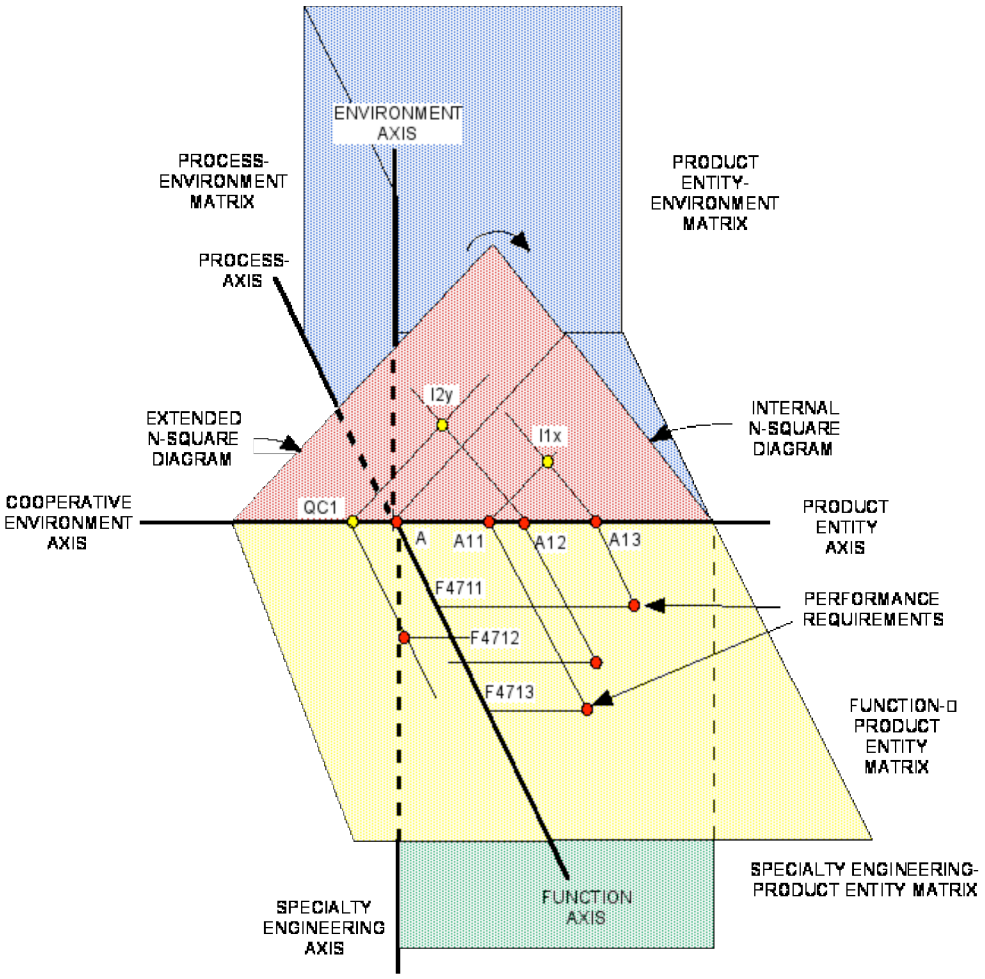


Figure 15 Juxtaposition of RAS and N-square Diagrams

2.1.7 System Relations

As the product entity structure is formed, the analyst can begin to identify the needed internal relations between the system entities by using an n-square diagram where the product entities are identified down the diagonal at some level of indenture. For a given analysis where the number of entities being studied for interface relations in an n-square diagram is N the largest possible number of relations is N(N-1) counting each direction as one possibility between each pair of entities. Interfaces between these entities is pre-determined by the way that functionality is allocated to the entities. Therefore, one may explore the list of function-product entity pairs

associated with each product entity in the n-square diagram. This is referred to as a pair-wise analysis of the n-square diagram intersections.

Figure 15 includes an n-square diagram with only half of the square showing (the remainder hiding behind the other structures in the diagram). The diagonal (containing the product entities we are interested in accomplishing the pair-wise analysis on) has been aligned with the product entity axis of the function-product entity matrix (which corresponds to the simple RAS of Figure 13).

The process for marking the intersections of the function driven relations matrix (n-square plane) entails a pair-wise analysis of the function-product entity pairing relationships marked on this matrix. Interface Ix is encouraged by the conclusion that if F4711 maps to A13 and F4712 maps to A11 then there is a possible demand for an interface between A11 and A13 to implement that relationship. So we map functions to product entities but we map F-A pairs to interfaces and those interfaces are pre-determined by the way we allocate functions to the product entities.

If the system is a modern war ship or the system that will take man to Mars, a pair-wise analysis of the function driven relations matrix would be beyond human comprehension if attempted all at one time, We can, however, partition the analysis work to one interface expansion at a time and it is not so overwhelming. At any one level of product entity granularity, we can explore one layer of product entity structure expansion for internal interfaces within the parent item. If there are five subordinate entities then the number of possible interfaces to be examined in a pair-wise fashion would be 5x4 or 20. Similarly, external interfaces can be analyzed individually. Of course, it will still be necessary to accomplish considerable interface integration work because of the partitioned process. This process will go very fast if the analyst is very familiar with the problem space and the evolving solution concepts. The complete algorithm is extended in subsequent paragraphs.

The requirements analysis sheet (RAS) identifies every possible pairing of functions and architecture entities. We may sort this listing so that all of the functions (or performance requirements derived from those functions) allocated to each entity are grouped by entity. Then we can pile up the allocations onto the product entity squares on the diagonal of a physical n-square diagram as suggested in Figure 15.

The discussion so far has dealt only with internal interface identification all defined on the function driven relations matrix of Figure 15. To cover external interfaces we add the larger n-square diagram on Figure 15. The diagonal in this case includes all of the product entities plus all of the external entities in the cooperative environment. We can identify relations between these external and internal entities in the same way we did the internal ones. The association of function F4712 to cooperative environment entity QC1 and allocation of a function to A12 may define a need for an interface I2y.

2.1.8 Environmental Relations Algorithm

2.1.8.1 System Environmental Relations

The system environment consists of all entities in the Universe less those that are in the system. That is, Q=(U-A) where Q is the environment, U is the Universe, and A is the product entity structure of the system being developed. It is convenient to partition all system level environmental relations into the sets illustrated in Figure 8. The system cooperative environment (QC) can actually be treated as an external interface and can be developed using the algorithm covered in Paragraph 2.1.7 very effectively. External cooperative systems are simply located on an extension of the product entity axis forming the cooperative environment axis. The hostile environment (QH) can best be understood through analysis of threats posed by hostile forces. The non-cooperative environment will yield to the same thought process applied in the threat analysis except that the stresses applied to the system are not applied for hostile purposes, rather simply because the system being developed is sharing a common operating space with other systems. Electromagnetic interference is an example of the stresses applied in this set.

System time (QN2) is studied using time lines oriented about the functions that the system must satisfy. When we allocate those function (or their corresponding performance requirements) to architectural entities the timing requirements corresponding to the functions are applied to the entity as timing requirements.

System space (QN1) is defined through mission analysis such that it is determined in what volumetric spaces of the Earth (surface, subsurface, and aerodynamic) and/or surrounding space and/or distant bodies the system shall function within, on, or around. For each space, that space is teamed up with one or more natural environmental standards that define that space. Each of these standards is then studied to determine which natural environmental (QN3) parameters included in the standard shall apply to the system being developed. Those that do not apply are tailored out of the standard. The selected parameters are then studied to ensure that the range of values is appropriate for the system. If the range in the standard is too broad it is tailored to narrow the range of values to that for which the system shall be designed. This process is repeated for each standard linked to a system operating space.

Figure 15 also extends the RAS concept to include environmental requirements analysis. The system environment as depicted in Figure 7 is illustrated at the diagram “origin” on the environmental axis that happens to coincide with the architecture “origin” that corresponds to the whole physical system A.

2.1.8.2 End Item Service Use Profile

An end item is a major element of a system that generally retains its physical configuration throughout its mission performance and has an end use function. The end items may remain fixed in place during use or move over great distances and maneuver within the system spaces as a function of the system mission and the end item’s application in the system. Each end item should be designed to endure only those natural environmental stresses anticipated so it is necessary to determine what subset of the system natural environment each end item shall be

exposed to. To accomplish this, one must create a physical process flow diagram. This is not the same as the functional flow diagram used to identify system architecture and performance requirements. The blocks on a functional flow diagram represent things that have to happen whereas the block of a process diagram represent a real world analogy. You cannot profitably consider system entities flowing through the functional flow diagram, indeed we are using the diagram to determine what those entities should be. However, we can imagine the system product entities flowing in the process diagram where each process acts as a transformation on the system entities.

The first step in defining end item environmental relations is to map the system environmental parameters to the process steps at some level of indenture, generally at a level where the environmental map does not change significantly below that process level. This work defines an environment for each process. The next step is to map the architectural entities to the process blocks. If an entity only maps to a single process step, it simply inherits the process environmental set. If, as is so often the case, the architectural entity maps to two or more process blocks, then it will be necessary to apply some kind of integrating process to any differences in environmental stresses and their values observed between the two or more process blocks. The rule most often selected initially is to pick the worst-case range for each parameter across the process values being evaluated. If this rule does not adversely influence system cost, then it is an adequate solution. If this approach either results in an adversely narrowed system solution space, then an alternative to "worst case" must be derived. Often time lines will show that the problem environmental stress will be applied over such a short time as to be insignificant. In other cases, one may find that the problem can be narrowed to some particular combinations of values that are very unlikely to occur. If the problem is intractable, it may be necessary to restrict one or more system environmental parameters more severely than is currently being done. Generally, this will have an adverse effect on system performance.

The self induced environment (QI) can best be studied and defined at the end item level since it is end items which contain the sources of energy and other stresses of interest which will reflect commonly through a natural environmental parameter right back into the system. Since the self-induced stresses are commonly greater in magnitude than the corresponding natural stresses for that same parameter, these induced relation values have the effect of extending the range defined through the application of the end item service use profile algorithm discussed above.

2.1.8.3 Component Environmental Relations

The environmental relations appropriate for components installed within end items are simply the end item environmental stresses if the end item has no altering effect on those stresses and all spaces within the end item offer the same environmental stresses to components installed within them. Where an end item does have a modifying effect on end item stresses but all spaces within and upon the end item offer the same stresses, it is necessary to determine the end item design effect on end item environmental stresses and the result is the set of installed component environmental stresses. The most complex case occurs when the end item must be partitioned into two or more spaces more often called zones of common environmental stresses. The value of each end item parameter must be determined for each zone thus defining the environment for each zone. Then one maps the components to the zones and they inherit the zone environments.

Commonly, the job is not complete at this point because it is found that there is no zone within which one or more components can be installed in a particular end item that will cause their environmental stress limits to be satisfied. When this happens, it is necessary to either change the component environmental specification values or include an environmental control system as an added entity into which the components with the environmental range shortfall problem are placed.

2.1.8.4 Environmental Impact

The environmental effects discussed in the three previous cases deal with environmental stresses the environment will apply to the system but there may be cases of the opposite direction that will cause damage to the environment. Once identified by someone skilled in environmental impact, these can be treated like safety hazards to be mitigated through re-design, procedural controls, or compensating environmental actions. In the case of military systems it is very difficult to mitigate the damaging effects of warfare but these systems can also have damaging effects on the environment in peaceful use such as training and maintenance. Often these materials are just naturally dangerous to be around as illustrated in the difficulties observed in the base closure efforts where many adverse environmental effects have had to be identified and mitigated.

2.1.9 Specialty Engineering and RAS Complete

The system engineering agent for the system must build a list of all of the specialty engineering disciplines that will be applied in the development of the system. A specialty engineering scoping matrix should be prepared between specialty engineering disciplines that may be required in development of the system and the product entities. This will help to determine team staffing needs in that area and connect people in those disciplines with a need to do specialty engineering requirements analysis for the indicated items. Figure 16 adds one more plane, the specialty engineering scoping matrix, to the construct previously illustrated in Figure 15 providing for allocation of specialty engineering disciplines listed to architecture.

Specialty discipline H7 is shown mapped to architecture item A11. This must be followed by analyst definition of one or more discipline H7 requirements that will flow into the specification corresponding to the product entity. The structure exposed in Figure 16 is a complete RAS showing all of the important requirements related relationships supporting the requirements analysis process leading to the identification of every kind of requirement appropriate to the system and hardware specifications and all of them linked to and derived from a model.

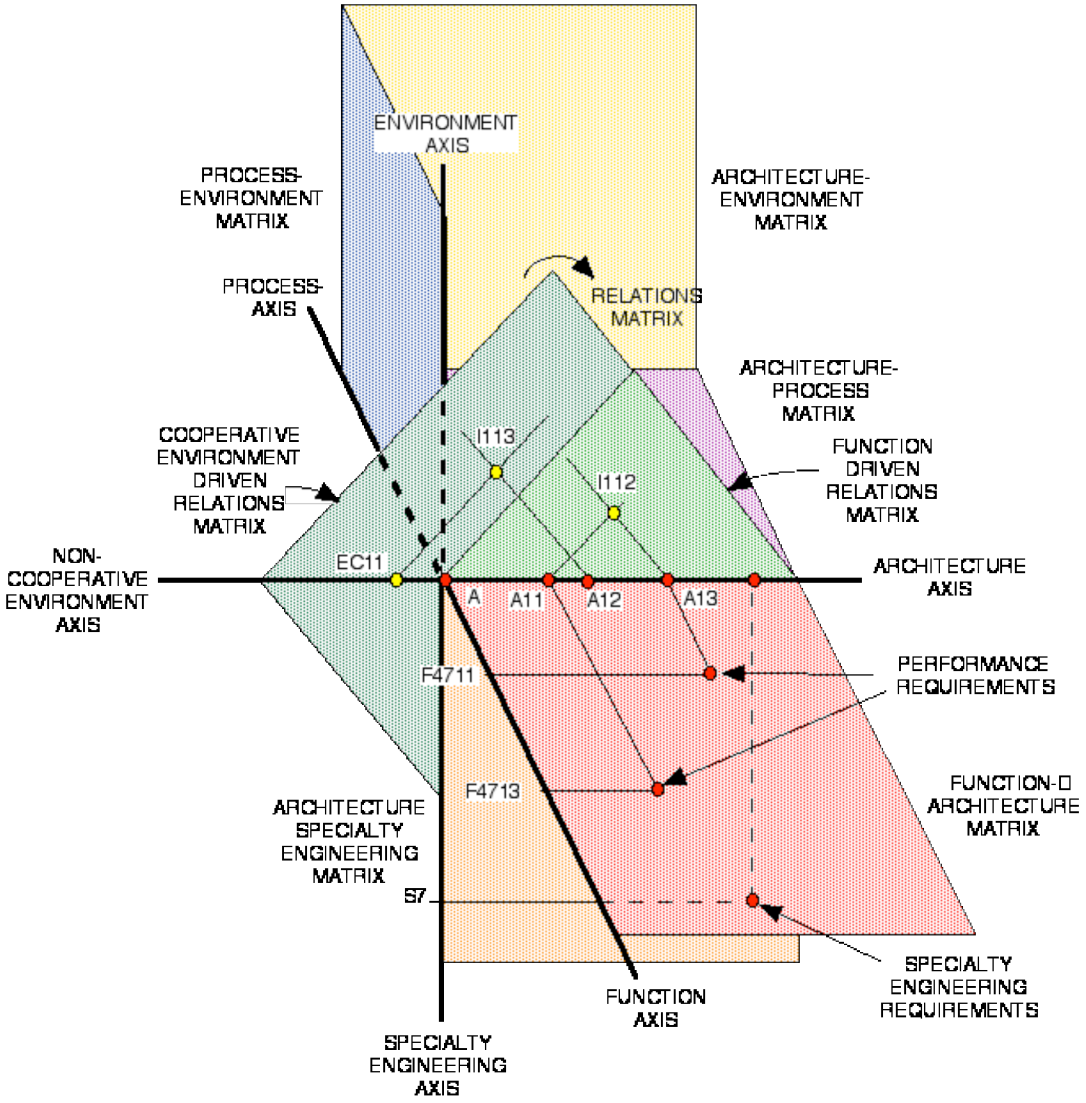


Figure 16 A Geometric View of the RAS Complete

The fact that an aircraft airframe will have to be checked for alignment during manufacture and after hard use (hard landing or pulling excessive g's in flight, for example) identifies a need for a relation between the airframe and the equipment which will be used to accomplish the alignment. Today this will generally call for some form of laser optical application so the airframe would have to either include targets, detectors, or mirrors or provisions for these to be applied. The manufacturing and maintenance engineers would have to consider all of the ways there may have to be relations between support equipment and the operational entity. There may be other cases where these disciplines have to call for internal relations within the system entities. For example, if the system must include built in test (BIT), there will have to be relations between most of the on board equipment and some entity that concentrates the BIT effects for display to operations and/or maintenance personnel.

The needs of operations personnel, such as aircraft pilots, locomotive engineers, ship captains, and automobile drivers provide a tremendously rich class of entity relation possibilities. The physical relations are always fairly simple in that the human operator has only his/her senses, mental acuity, and physical strength through which to interact with the system. This problem is made much more complex, however, because not all humans will react in precisely the same way to a particular stimulus. It will be necessary to determine the complete data set that the human will require under all operating conditions and in what way the human shall influence system behavior in terms of controls. Operator sequence diagrams, built like a UML activity diagram with swim lanes, can be useful in doing this work.

Every one of the specialty engineering disciplines selected for the program must be evaluated for entity relation driving potential and those persons doing that work alerted to their responsibilities in identifying relations for further consideration by the whole team.

2.1.10 RAS-Complete in Table Form

The results from the analyses noted in prior text must be captured on its way into program specifications. Certainly, the most advantageous way to do that is in a computer database systems such as DOORS, CORE, or SLATE. Therefore we would expect that some form of tabular structure would suffice. Figure 17 is offered as the candidate view of this table for use during development in capturing the relationship between model, requirements, product, and document entities. The model ID (MID identifies the model from which the requirement was derived. The requirement columns identify the requirement ID (RID) assigned by the computer system for use in traceability. The RID is a made-up computer-assigned unique field using a base 60 numbering system in this example but a hexa-decimal implementation is probably more common. Ideally the requirement statement should be captured in primitive form (attribute, relation, value, and units) wherever possible with different fields for each component of the string. The primitive form is shown concatenated in Figure 17. The final column pair offers specification paragraph number and title.

The sample data included is ordered by model ID alphanumerically separating the data into the four kinds of requirements found in a system or hardware specification. The lists the MIDs respected by the author is still in a state of change as different RAS-Complete possibilities are explored. This may explain the apparent unthinking selection of H and Q for specialty engineering and environmental requirements, respectively. The intent is to identify all possible modeling artifacts with a letter as a source from which every conceivable requirement may be derived. Model letters for UML artifacts have been included in the set and will be introduced later.

This view provides clear traceability between the models from which the requirements were derived and the product entities to which they were allocated for all of the requirements, not just the performance requirements. What the author calls lateral traceability is captured in the database implementing the RAS-Complete. It is also a simple matter to link the rows in the matrix in a database to the corresponding verification requirements as well as the tasks to which they are allocated and their corresponding plans, procedures, and reports. Vertical traceability is, of course, simply a matter of relating the unique RIDs from pairs of requirements in specification parent-child relationships identified by their PID.

MODEL ENTITY		REQUIREMENT ENTITY		PRODUCT ENTITY		DOCUMENT ENTITY	
MID	MODEL ENTITY NAME	RID	REQUIREMENT	PID	ITEM NAME	PARA	TITLE
F47	Use System			A	Product System		
F471	Deployment Ship Operations			A	Product System		
F4711	Store Array Operationally	XR67	Storage Volume < 10 ISO Vans	A1	Sensor Subsystem		
H	Specialty Engineering Disciplines			A	Product System		
H11	Reliability	EW34	Failure Rate < 10 x 10-6	A1	Sensor Subsystem	3.1.5	Reliability
H11	Reliability	RG31	Failure Rate < 3 x 10-6	A11	Cable	3.1.5	Reliability
H11	Reliability	FYH4	Failure Rate < 5 x 10-6	A12	Sensor Element	3.1.5	Reliability
H11	Reliability	G8R4	Failure Rate < 2 x 10-6	A13	Pressure Vessel	3.1.5	Reliability
H12	Maintainability	6GHU	Mean Time to Repair < 0.2 Hours	A1	Sensor Subsystem	3.1.6	Maintainability
H12	Maintainability	U9R4	Mean Time to Repair < 0.4 Hours	A11	Cable	3.1.6	Maintainability
H12	Maintainability	J897	Mean Time to Repair < 0.2 Hours	A12	Sensor Element	3.1.6	Maintainability
H12	Maintainability	9D7H	Mean Time to Repair < 0.1 Hours	A13	Pressure Vessel	3.1.6	Maintainability
I	System Interface			A	Product System		
I1	Internal Interface			A	Product System		
I11	Sensor Subsystem Innerface			A1			
I181	Aggregate Signal Feed Source Impedance	E37H	Aggregate Signal Feed Source Impedance= 52 ohms ± 2 ohms	A1	Sensor Subsystem		
I181	Aggregate Signal Feed Load Impedance	E37I	Aggregate Signal Feed Load Impedance= 52 ohms ± 2 ohms	A4	Analysis and Reporting Subsystem		
I2	System External Interface			A	Product System		
Q	System Environment			A	Product System		
QH	Hostile Environment			A	Product System		
QI	Self-Induced Environmental Stresses			A	Product System		
QN	Natural Environment			A	Product System		
QN1	Temperature	6D74	-40 degrees F< Temperature < +140 degrees F	A	Product System		
QX	Non-Cooperative Environmental Stresses			A	Product System		

Figure 17 RAS-Complete in Tabular Form

2.1.11 Traditional Structured Analysis Summary

In summary, a system is defined by identifying its functionality starting with the need (F), allocating that functionality to entities that become part of the system architecture (A). These entities that form the system architecture are selected by determining the performance requirements that the system must satisfy to meet the top-level customer's need. The pairs of functions and product entity allocations pre-determine how the entities will have to relate to each other through interfaces (I) between the product entities. The environmental elements (E) are defined at the system level in terms of the spaces within which the system must function and the corresponding characteristics of those spaces drawn from appropriate environmental standards covering those spaces. As depicted in Figure 18, the traditional structured analysis effort attempts to define the most cost effective solution such that in N cycles of the process axis of the physical system (generally cyclical in the interest of reuse of system elements) the relation P (process) maps the cross product of the power sets of architecture (A*), interface (I*), and environment (Q*) to the function set F such that F is covered.

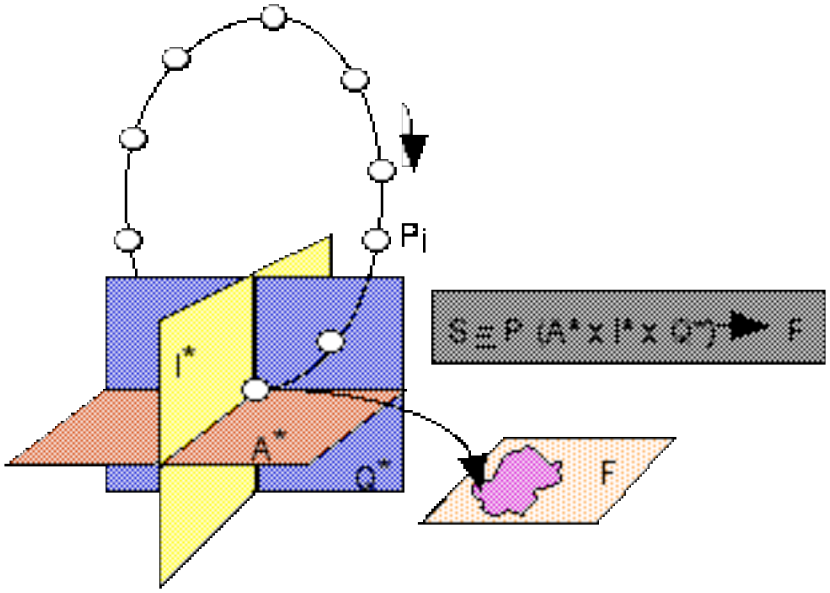


Figure 18 The System Relationship

For every process Pi, there exists a combination of architecture, interface relations, and environmental stresses such that some subset of the function set is covered or accomplished. The power sets of these entities include all of the possible subsets of these entities within their own set thus the power set of A includes every useful combination of product entities relative to every process step. Useless subsets are also included in the power set as well, of course. It is important that the functions be covered in the correct order determined by the sequence of the processes linked together in the process axis. If all of the functions are satisfied in N revolutions of the process axis as planned, then we may say that the system is consistent relative to the use of its product entities, interfaces, and environmental stresses. If there are product entities that are not used in the process or some that are needed but not available we may not have the optimum product entity structure.

This whole process happens in practice somewhat backwards in that for an unprecedented system, one begins the development process only knowing the ultimate function, the need, and must expand everything from that one perspective.

2.1.12 SDD Content and Format

When used to support the application of traditional structured analysis on a program, the preferred SDD format consists of a main body and seven appendices, each providing a capture point for the work products of one of the several fundamental analytical system requirements analysis process areas. Figure 19 shows how the document is structured. A series of seven interactive system analysis activities feed the development of the appended data explained in subordinate paragraphs. The appended data then becomes the basis for lower tier analysis that produces content for the lower tier specifications and adds to the appended data.

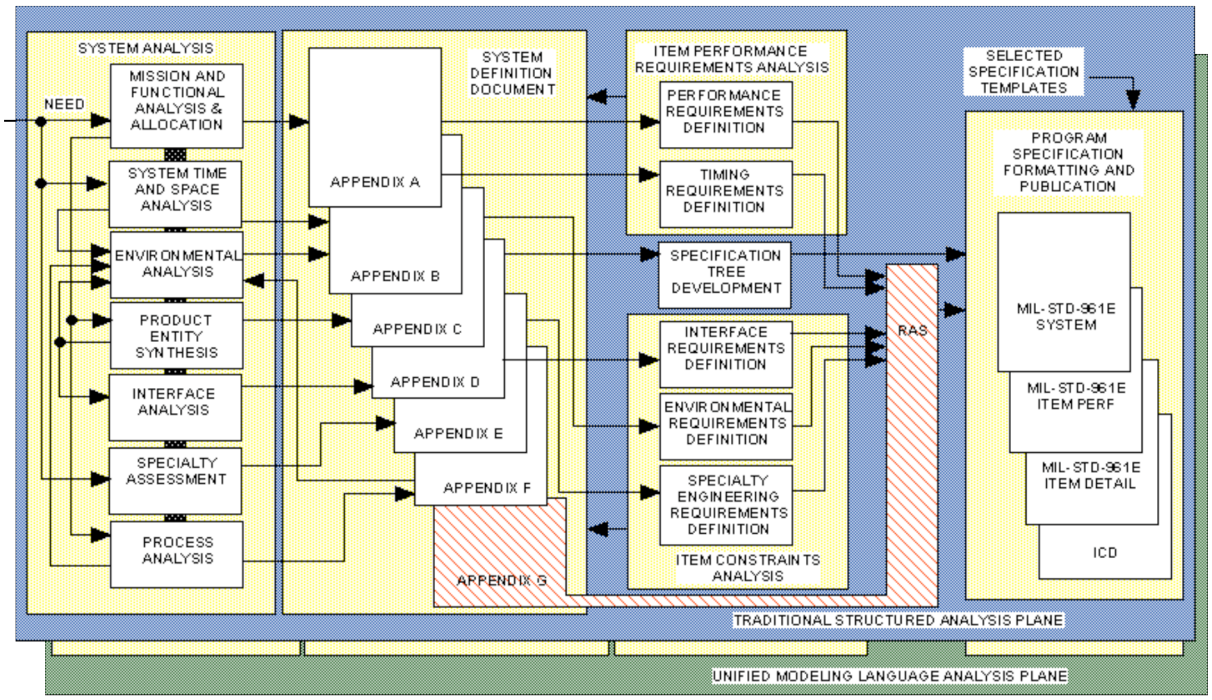


Figure 19 SDD Structure

2.1.12.1 Document Main Body

The main body simply contains a table of contents, list of illustrations, and list of tables for the document plus it should provide text explaining the capture of work products in the seven work areas during system and lower tier analyses. The body should also explain that the SDD couples the structured analysis work and its work products to specification content as guided by the selected specification standard templates.

2.1.12.2 Appendix A, Functional Analysis

This appendix captures the functional flow diagram starting with the identification of the system need and the life cycle flow diagram. The Use System Function is initially decomposed progressively to expose more details about the user need. For each block in the functional flow diagram, there should be one line in the function dictionary also contained in Appendix A.

2.1.12.3 Appendix B, System Environment Analysis

The environment consists of several subsets of stresses that are applied to the system. This appendix identifies and characterizes them. Timelines capture critical timing requirements. The spaces within which elements of the system must function are identified and the corresponding environmental stresses defined in terms of standards that describe those spaces. Service use profile analysis is applied to uncover end item environmental requirements. Finally, zoning of end items exposes component environmental requirements.

2.1.12.4 Appendix C, System Product Entity Analysis

The system product entities result from the allocation of functionality to things. As these pairs are defined on the function-product entity matrix, they must be entered into the product entity structure diagram, This work should be accomplished by a team of people knowledgeable in system, hardware, and software engineering, manufacturing engineering, verification engineering, logistics, material and procurement, and logistics in order to evolve an optimum product entity structure which will be universally respected on the program. This product entity structure is also the basis for the specification tree. Each item on the tree must have a responsible agent identified, a template selected, and a release date established. This structure should also be the basis for any IPPT established on the program. It is also the basis for the WBS so the SOW and IMP align perfectly with the teaming structure applied on the program.

2.1.12.5 Appendix D, System Interface Analysis

Interfaces are identified by pair-wise evaluation of function allocations to product entities using an n-square diagram. This appendix identifies all interface needs internal to the system as well as externally to the cooperative systems identified in Appendix B.

2.1.12.6 Appendix E, Specialty Engineering Definition Analysis

Appendix E provides a space in which system engineers can capture their work directed at identifying the specialty engineering disciplines that will have to accomplish work on the various entities in the system product entity structure to define the appropriate requirements and subsequently the needed analyses to confirm that those requirements are being satisfied. A specialty engineering scoping matrix is used to report the results of that analysis.

2.1.12.7 Appendix F, System Process Analysis

Appendix F captures the results of a physical process analysis in the form of a process flow diagram. This is used by logistics engineers to drive out requirements related to training, support equipment, maintenance procedures (tech data content), and spares consumption. It is also needed to complete the environmental use profile study reported upon in Appendix B that drives environmental requirements for end items.

2.1.12.8 Appendix G, Requirements Analysis Sheet

The exposed functions are listed in the Requirements Analysis Sheet (RAS) contained in this appendix. Related performance requirements are defined and allocated to a product entity. These performance requirements have to have a paragraph number assigned, title identified, and they can be outputted into a specification following a particular template. That part of the work can be done inside a requirements database system. Ideally, all of this work would take place within a requirements database tool but some organizations may find it preferable for their purposes to do the traditional structured analysis work using pencil and paper followed by capture of the resulting requirements in a word processor or a computer database tool from which specifications can be generated.

In keeping with the Integrated RAS idea advanced in Paragraph 2.1.10, the RAS is not restricted to performance requirements. Specialty engineering, interface, and environmental requirements can also be included so that every requirement appearing in every specification on a program will transition from the analytical model from which it was derived into a specification via the requirements analysis sheet.

2.1.13 Team Activity During Requirements Work

The PIT is responsible for accomplishing all requirements analysis work focused on developing the system specification and the immediately subordinate specifications that will be the top-level specifications for the top level IPPT. In general, this analysis work will be accomplished using traditional structured analysis following the pattern described in this section. PIT initiates the analysis capturing the work products in the SDD thus initiating that document. Requirements derived from the modeling work are entered into the database application initiating the requirements capture. Any customer requirements documentation is also entered into the database and traceability established between these requirements and those developed from modeling efforts that appear in the system specification. Traceability is continued down to the subordinate specifications to be handed over to the top-level teams when formed.

The PIT identifies all system external interfaces and defines them in the system specification or the beginning of an interface control document. Internal interfaces are also identified and defined for the first tier entities below the system level and the next lower tier in order to complete the internal interface definition for the first tier. All requirements are entered into the requirements database application and traceability entered.

The PIT develops a specialty engineering scoping matrix and maps the needed disciplines to the product entities identified and coordinates the indicated domain experts to derive requirements for the system and top-level end item specifications. The system level environmental requirements are derived from system spaces identified and mapped to corresponding tailored standards.

With the top-level specifications completed, the PIT can bring the top-level IPPT aboard. As those teams assemble and become familiar with their specification and the program planning data prepared by the proposal team, they continue the requirements analysis process relative to the functionality of their product entity. The primary role of the PIT switches to integration and optimization across the IPPT. The teams enter requirements data into the requirements database and maintain traceability. As this process continues, it may become apparent that lower tier teams are required in which case the parent team takes over system responsibilities for them. The parent team in all cases must develop the top-level specification for any new team. Also, at some point, a team will identify an allocation of functionality to computer software and the continued analysis of that entity should switch to UML.

2.2 UML

2.2.1 Entry Analysis and Overview

While it is encouraged that the enterprise apply TSA today as the entry modeling technique, it is entirely possible to initially enter the problem space analysis for a system that will be implemented primarily in software with UML rather than TSA or SysML. The suggested process starts at the top with the problem expressed in the system need and illustrated in a context diagram, borrowed from modern structured analysis, like that shown in Figure 20. The context diagram expresses relationships between the system, represented by the bubble, and terminators, representing outside entities deriving benefits from the system and supplying things needed by the system to function, generally information in a computer software system but more general in a system that will be implemented using a collection of technologies. Figure 20 is an alternative to the traditional depiction of the general system as a block labeled System interacting with a block named Environment.

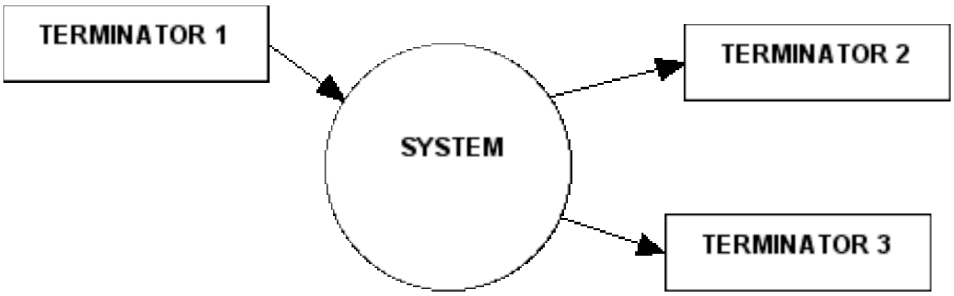


Figure 20 Context Diagram

While UML could be the entry modeling approach at the system level following the approach discussed in this section, the discussion to follow is based on the assumption that the problem space will be entered using TSA with a recognition at some point of a need to switch to UML as entities are identified that must be developed in software. Figure 21 illustrates a process for applying UML starting at whatever level in the system the program chooses to start applying it. Generally, this will be some level below the system level based on allocation of higher tier functionality to a software entity.

In a given system, the initial analysis may have identified one or more entities that will be implemented in computer software so for each of these separate entities one should build a context diagram. It should be noted that the context diagram was popularized in modern structured analysis and was not adopted by UML but it is a useful artifact with which to identify the inside-outside relationship between the software entity and the entities external to that software with which it must interact. The context diagram is offered as an intermediary view that will lend some discipline and order to the identification of use cases. We will identify one or more use cases for each terminator and perform a dynamic analysis on each of those use cases.

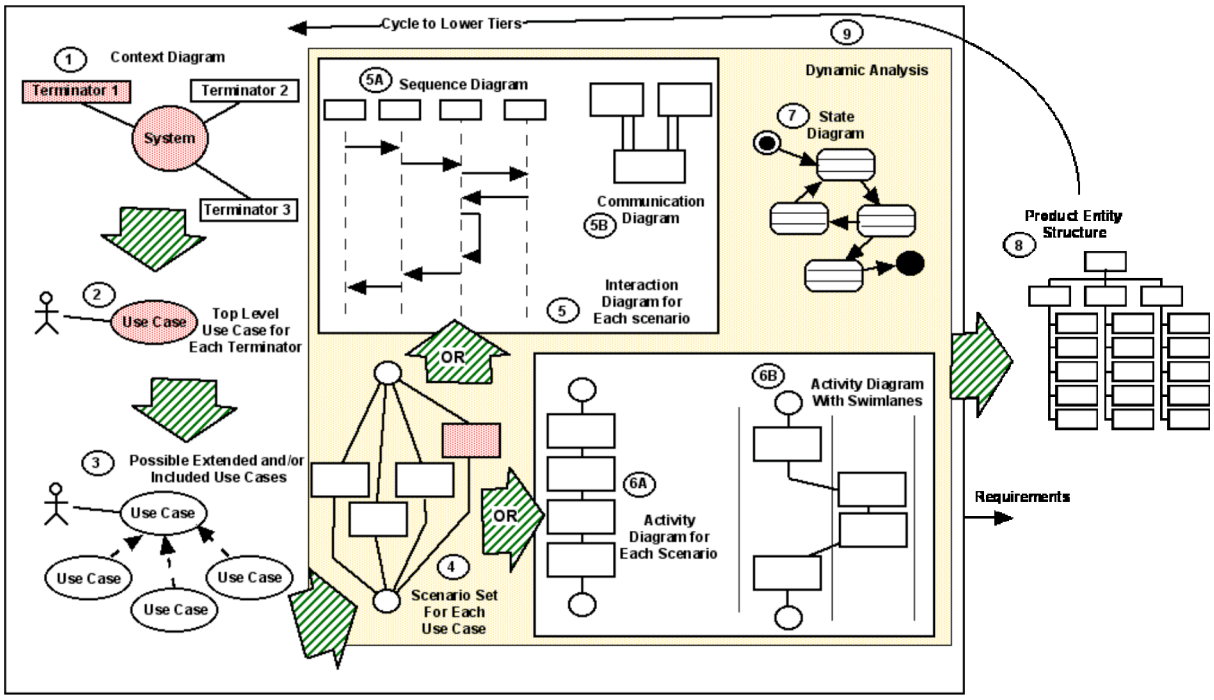


Figure 21 Unified Modeling Language Overview

For each terminator in Figure 20, identify one or more use cases through which the intended functionality will be accomplished identifying the actors deriving benefits from the system to be created. One use case may expand into several extended or included use cases to cover a more complex situation. You will note that the opening gambit has been arranged to provide structure in the identification of needed use cases. Next, for each use case, build one or more scenarios dealing with how that use case interacts with the system. These scenarios can be in text form, a list of events, or some kind of diagrammatic treatment.

Then, express each scenario in the form of an activity and/or sequence diagram cast at the UML entry level initially. The activity diagram can be thought of a functional flow diagram similar to that used by software programmers many years ago or by system engineer in traditional structured analysis. It may be drawn vertically or horizontally as far as the author is concerned. Swim lanes may be overlaid upon the activity diagram each of which corresponds with the lower tier entities (classifiers in UML) that will be responsible for implementing activities within their swim lane. This is a key point in the analysis where the analyst must make lower tier product entity structure decisions that should lead to adding software entities to the product entity structure. Some analysts prefer to think of the two-dimensional artifacts in the diagram as states rather than activities or functions. Simultaneity is not permitted in normal state diagramming but UML permits it in activity diagrams to cover decisions and branching in a way similar to that applied in flow charting.

Alternatively, the analyst can use a sequence diagram to open the exposure of the details of a use case scenario. The entities (classifiers) through which the functionality and behavior are explored are identified on what are called the life-lines drawn as dashed lines below selected physical or

logical classifiers. These lifelines correspond with the swim lanes on the matching activity diagram. Directed line segments connect these life-lines to show passage of messages and relationships between the classifiers. As in the activity diagramming, these life-line decisions may identify entities that already have been identified or they may involve entities not previously identified. In the latter case, the PIT must concur in the model extension and add the new entities to the product entity structure.

Each of these diagrams (activity and sequence) identify a lower tier (white box) view of what entities will be needed to accomplish the exposed functionality and behavior, what will have to happen in the system in order to achieve the intended goals of the use case signified in its name, and offer a detailed view of the order in which these things will occur.

The analyst can allocate top-level functions (activities) to specific entities and arrange the blocks of the activity diagram in corresponding swim lanes linked to these entities. These swim lanes will correspond to nodes, or even higher-level entities, at the system level but in any case we can refer to them in general as physical or logical classifiers within UML. If appropriate, analyze each of these classifiers from a dynamic perspective using some combination of sequence, communication, activity, and/or state diagram. The communication diagram shows the same information as a sequence diagram with an emphasis on the entities rather than the relationships between those entities. A state diagram is useful where there exists some finite number of possible conditions within which the entity can exist and there appear to be understandable rules for the entity to change from one condition (state) to another.

Identify requirements derived from these artifacts and capture them in a program-wide RAS linked to the modeling artifact (MID) that encouraged their identification and the physical or logical classifier, referred to more generally by a product entity ID (PID) in the RAS, that will be responsible for responding to that requirement and in the specification for which it should reside.

2.2.2 The Connection Between Modeling Artifacts, Specification Content, and Product Entities

Figure 22 suggests a hierarchical relationship between the elements of the UML analysis and offers a way of assigning MID. The capabilities in the specification format (template Paragraph 3.2) coordinate with terminators, use cases, extended use cases, and/or scenarios. Figure 22 only shows one terminator expanded but the intent is that for any top level software classifier AX (highest tier SW entities) entered into the product entity structure, one or more terminators would be identified and expanded as shown in the one case shown in Figure 22. The software top-level software classifier, AX, is, of course, a member of the product entity structure (PID) where X is a string of base sixty or decimal delimited base ten numerals. The suggested UML MID stream is identified first with a unique UXh MID (with h = e{1, 2, 3} in the example shown in Figure 22) for each terminator.

The terminator MID can be further decomposed using the MID UXhijk pattern composed from a set of use cases, possible extended use cases, and scenarios for each terminator h. These MID are the entries to place in the RAS for software requirements derived from these artifacts. In general, capabilities will be derived from these artifacts and the requirements subordinate to them will be derived form the dynamic modeling artifacts UXhijk1 through UXhijk4.

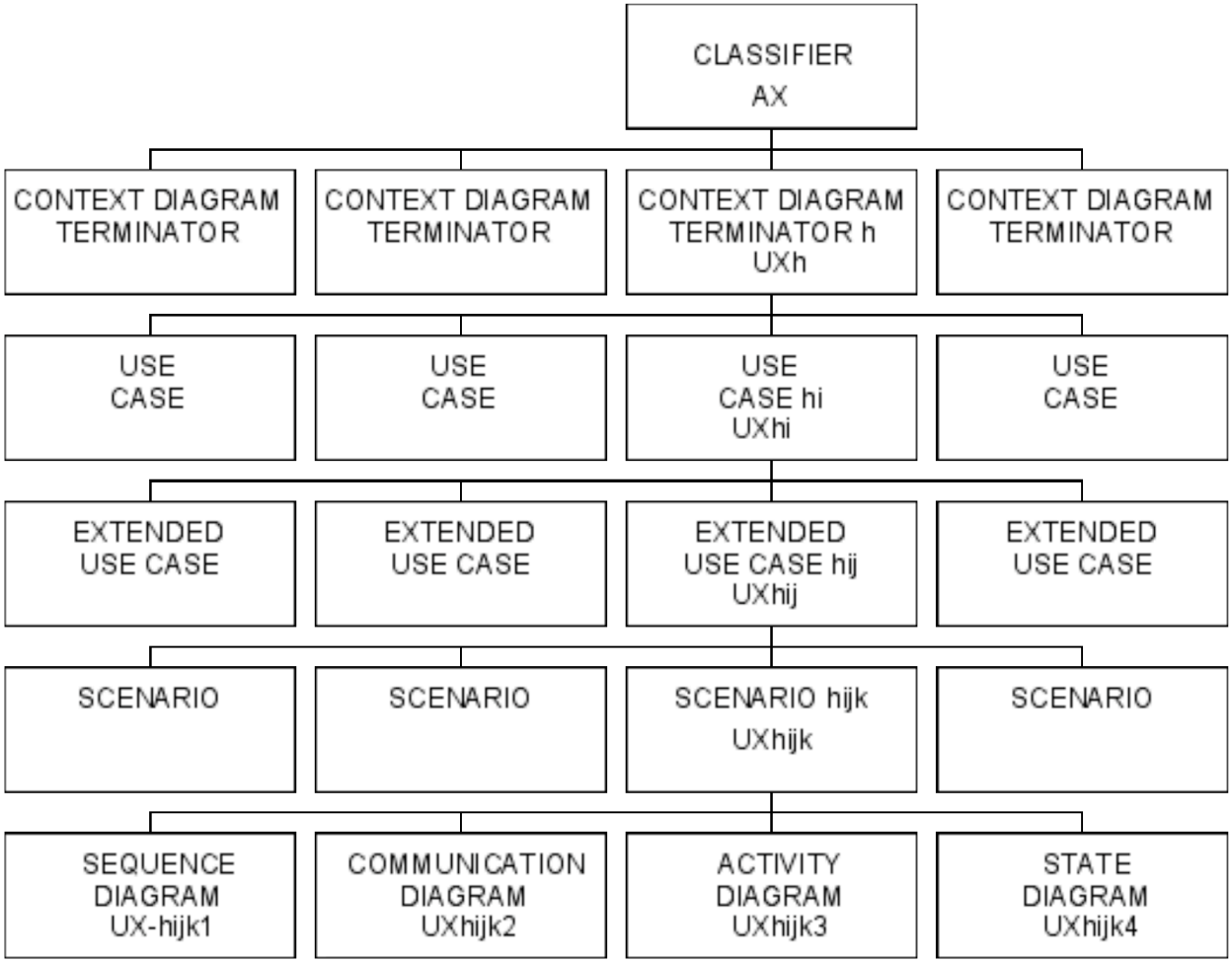


Figure 22 Hierarchical Relationship Between UML Dynamic Modeling Artifacts

So finally, the requirements for each capability flow out of applying the subordinate UML dynamic modeling artifacts. As in TSA with lower tier product entities identified from lower tier function allocation, the lower tier software product entities flow from the sequence (life-lines) and/or activity (swim lanes) diagramming. This is a particularly satisfying coordination between lower tier entities being exposed through functional analysis in TSA and activity analysis in UML where both are using essentially the same model to identify lower tier entities.

We can continue to apply UML in the lower tiers as covered in Figure 21 treating each classifier as a system in accordance with the steps discussed above progressively identifying nodes, components (possibly in more than one layer), and classes. If the system level problem space was entered using UML as this process moves forward and downward, it will be decided that some classifiers will be developed as software and hardware entities, with the latter possibly splitting in lower tiers into hardware and software entities. The continuing analysis of hardware entities can be accomplished using traditional structured analysis or SysML model artifacts while the modeling of software entities continues primarily using UML model artifacts. In that it is

difficult for software to contain hardware, the use of TSA as the entry analysis probably makes more sense.

At the lower tier UML analyses where the physical and logical classifiers are classes and objects, the lines that flow between the classifiers on the corresponding sequence and communication diagrams coordinate with messages and influences applied to/from those classes in relation to external physical and logical classifiers (other classes generally at this level). This same effect is in operation whether the classifiers depicted are classes, components, or nodes. Each classifier must have identified for it one or more operations (services or functions) that it performs relative to an outside set of actors and one or more data entities it deals with internally to accomplish those operations. The data elements will flow into the classifier via the lines on the sequence and communications diagrams and data may be created or altered internally. Initially, the analyst may choose to first identify node, component, class responsibilities and subsequently as the analysis matures translate these responsibilities into operations and attributes.

In lower tier analyses, the assigned IPPT are responsible for identifying and defining needed interfaces below the level initially identified by the PIT. Where the two terminals of an interface touch only entities that are the responsibility of a single team, that team is clearly responsible for interface identification, definition, and integration. Where an IPPT is responsible for only one terminal of an interface, that team must cooperate with the team responsible for the other terminal to develop that interface. The integration agent in this case is the lowest common team. If there is only a single layer of teams under the PIT, the PIT is always the lowest common team. In general, it should be the team on the receiving end of an interface that first identifies the need for a new interface since interfaces should not be defined based on what is available but what is needed. If it is not obvious what team shall be responsible for the new interface, the PIT shall act as the integration agent until such time as the source is identified and then the lowest common team will take over that responsibility.

As IPPT identify lower tier entities during use case analysis, the PIT shall approve those additions and assemble them into the formal product entity structure. These entities may be physical or logical entities but eventually all of them must be identified as real product entities that will be developed. These final entities can be logical entities as in the case of computer software that will run on a particular hardware computer entity. Where it appears that lower tier entities will entail significant complexity, new IPPT may be formed by the PIT that will be subordinate to the appropriate existing IPPT. Those lower tier IPPT will take over the continuing analysis of use cases appropriate at that level and the immediately superior team will take on the role of the system engineer for the new team just as the PIT does for all top tier IPPT.

The use case analysis process employed by an IPPT or a collection of teams will necessarily be a collaborative process involving people from several different specialty disciplines. Each such team will have a leader whose responsibility it is to bring the analysis to a conclusion as scheduled. These teams will come into being, exist for a brief period of time, come to a conclusion, and pass from the scene with others replacing them. Once a use case analysis has been completed by a team, the work products will have been captured in modeling applications and integrated relative the existing work products. The modeling front will move down through the advancing product entity structure till the system is fully characterized.

Where personnel from other teams are required to accomplish work on another team's use case analysis, the owning team will cover the manhours of all personnel working the use case. Where all team members are physically collocated in the same facility, they may be depended upon to interact well through a traditional meeting in the common facility. Where at least some of the people required on a use case analysis effort are not collocated, it will be necessary to extend the meeting place geographically through the use of a product such as webex where people from several different geographical locations can cooperate in the development of a set of information.

Leaders of use case analysis teams are responsible for the prompt completion of team activity with good results but in so doing they will be well served to identify the most effective collaborative engineer on the team to lead collaborative team activity in the form of meetings especially where those meetings entail the use of distance integration aids like webex. The collaborative discussions in meetings need not necessarily be led by the team leader.

As the team completes its modeling and requirements work, the results should be reviewed by the parent team and, if approved, the team should be empowered to proceed with design at a level appropriate to the team responsibilities. The team must continue any responsibilities it may have for integration and optimization and leadership of subordinate teams that may still be involved in modeling work. This design work will entail some combination of hardware and software design development.

2.2.3 Dynamic Modeling Artifacts Explained

The use case diagram is considered a dynamic modeling artifact also but it is treated here as a transition medium between classifiers and the dynamic model set. The remaining dynamic models are implemented in four diagrams from which we may select any subset including all of them, any one of them, any pair, or any trio for a particular scenario analysis. It is not necessary to use them all for any particular analysis. Use those that make it possible to understand the problem space and properly characterize it in requirements. Some very large programs have done much of the analysis with only use case and sequence diagrams. The more complex the problem space and the more intimately the parts of the evolving system interact, the more different views of problem space that will be useful.

The first two kinds of diagrams covered, sequence and communications are both modeling the same relationships and collectively referred to as interaction diagrams. The sequence diagram emphasizes the time ordering of messages and the communication diagram emphasizes the organization of the objects that participate in the interaction. The second two are forms of state diagrams in the mind of many analysts.

2.2.3.1 Sequence Diagram

Some programs apply only the sequence diagram to explore the dynamic behavior of use cases and this may be adequate on relatively simple problems. In this tutorial we are assuming that the analyst employs the sequence diagram as the initial dynamic model, though an alternate approach would be to use the activity diagram for that purpose, but uses at least some of the

other three models to explore the use case more thoroughly. The sequence diagram example in Figure 23 illustrates the fundamentals showing two classifiers that comprise the classifier AX that has previously been analyzed. Here we conclude that AX must consist of AX1 and AX2 and that in order to accomplish the behavior defined for AX these lower tier classifiers must interact with an outside entity called an actor which will derive some kind of benefits from the relationship. The two subordinate classifiers will provide certain operations that are not clearly defined on this diagram. In the process of doing so, they will exchange messages in a certain order with time running down the page.

Each classifier including the actor has a lifeline in the form of a dashed line running down the page. A block is overlaid on this dashed line to indicate the active period(s) of classifier. Between these classifier lifelines we see messages being passed from one classifier to another. The names of these classifiers are formed of one or more words concatenated together without spaces and all but the first word capitalized. After the message name one can insert an argument list parenthetically.

The model permits the creation of classifiers and when they have performed their activity they can be killed. These features are not illustrated in Figure 23. The kinds of messages identified are: (1) a call invokes an operation on a classifier on the arrow end of the message, (2) a return message returns a value to the caller (dashed line used), (3) a send message sends a signal to a classifier, (4) a create message creates a classifier, and (5) a destroy message destroys a classifier. Message two and four could be an example of messages types 1 and 2. Message three is an example of message type 1 where the classifier is making the call upon itself.

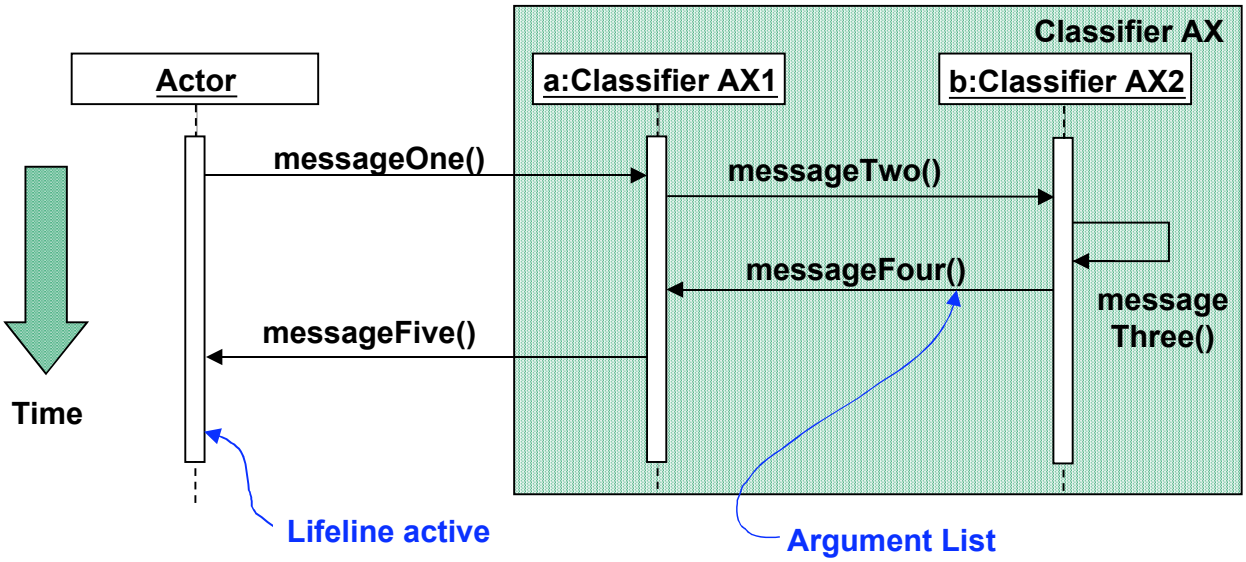


Figure 23 Sequence Diagram Example

After a classifier sends a signal to another classifier the sending classifier returns continues its own execution. The target classifier independently decides what to do about it. A common reaction would be to trigger a state machine causing the target classifier to execute actions and change state.

2.2.3.2 Communication Diagram

In some cases we are primarily interested in the time ordered sequence of messages between classifiers but in other situations we may be more interested in the organization of the classifiers and a communication diagram can offer better results even thought the sequence and communication diagrams are semantically equivalent. Figure 24 illustrates a communication diagram reflecting the same situation as Figure 23. The classifiers are joined by lines corresponding to the relationships between them and messages are related to these line each in terms of message name, direction, and the relative timing of the message.

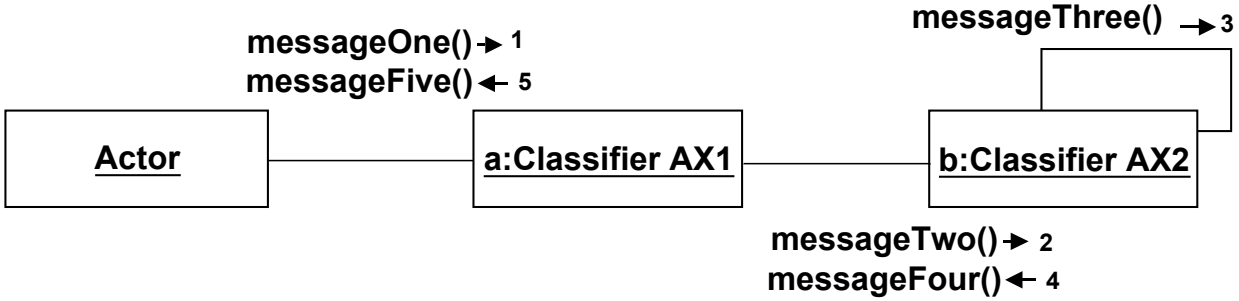


Figure 24 Communication Diagram Example

2.2.3.3 Activity Diagram

An activity diagram can be used to express the things that one or more classifiers must accomplish. It is weak in terms of absolute timing of accomplishing those things but strong in expressing the relative order of those things. As shown in Figure 25, activities are illustrated by rounded corner boxes and they are connected into a sequence by directed line segments. In addition to the activities, we also will wish to show simultaneity through the use of forking and joining structures and alternative paths using branch and merge structures. The guard expressions give information about the conditions that correspond to movement through one branch or another.

This is the functional flow diagram of UML though many analysts prefer to think of the blocks as states rather than functions. The author prefers not to consider them states because it is in conflict with the notion that an entity must be in only one state at a time.

The diagram can be built with swim lanes, or not, that relate to classifiers, the same classifiers identified on sequence diagrams using the lifelines. It is through these two devices that we can allocate software functionality to classifiers. As we do so we determine the next lower tier product entity structure and should offer up newly identified entities to the PIT for inclusion in the product entity structure.

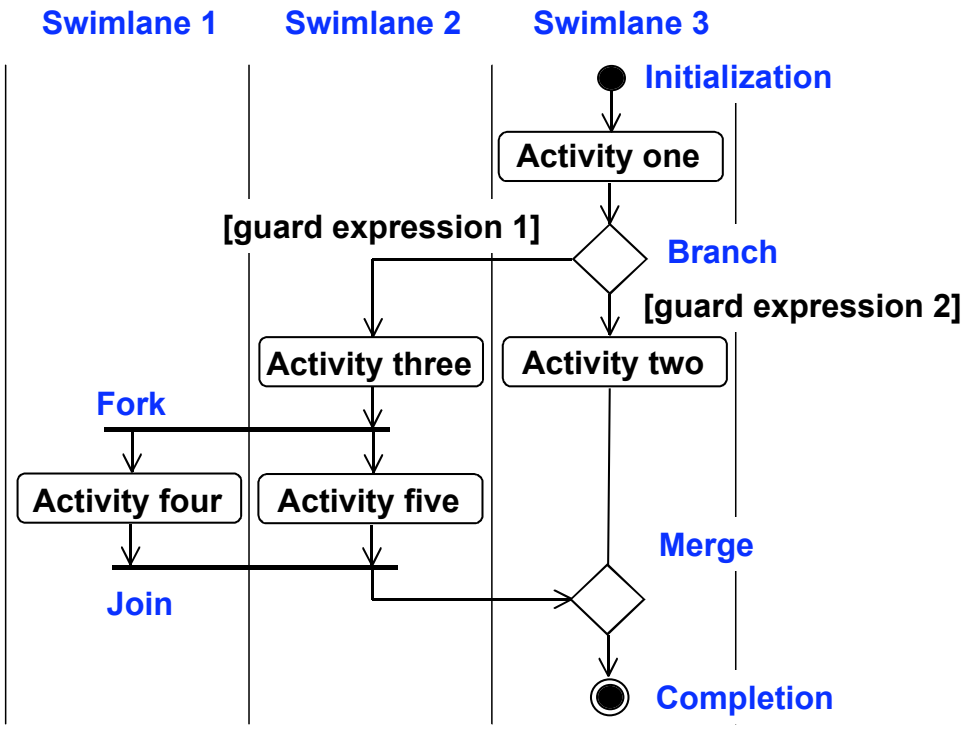


Figure 25 Activity Diagram Example

2.2.3.4 State Diagram

An interaction diagram (sequence or communication) models the relationships between a collection of classifiers while a state machine models the behavior of a single classifier. The classifier in this case can be the whole system or a classifier at any level of abstraction. The state machine models the possible condition that a classifier can exist in and the transition of that classifier from one condition or state to another based on a stimulus that might be a signal from another classifier, the passage of time, receipt of a call message invoking an operation, or a change in some condition.

A sate is drawn on a state diagram, as shown in Figure 26 showing a state machine for temperature control, as a rounded corner box with the name of the state written inside. Generally the diagram must have an entry and final state symbols though it is possible that once entered the state machine may continue forever. In some cases the intent may be for the machine to continue forever, as in a traffic light system, the reality is that such a system may have to be shut down for maintenance and does need a final state. Transitions between states triggered by events in the life of the classifier being modeled are shown diagrammatically as directed line segments between a pair of state and named in a way to convey how that transition is triggered. It is understood that the machine must be in only one state at a time and that only a single transition is possible at one time. Transitions are generally thought of as taking place instantaneously.

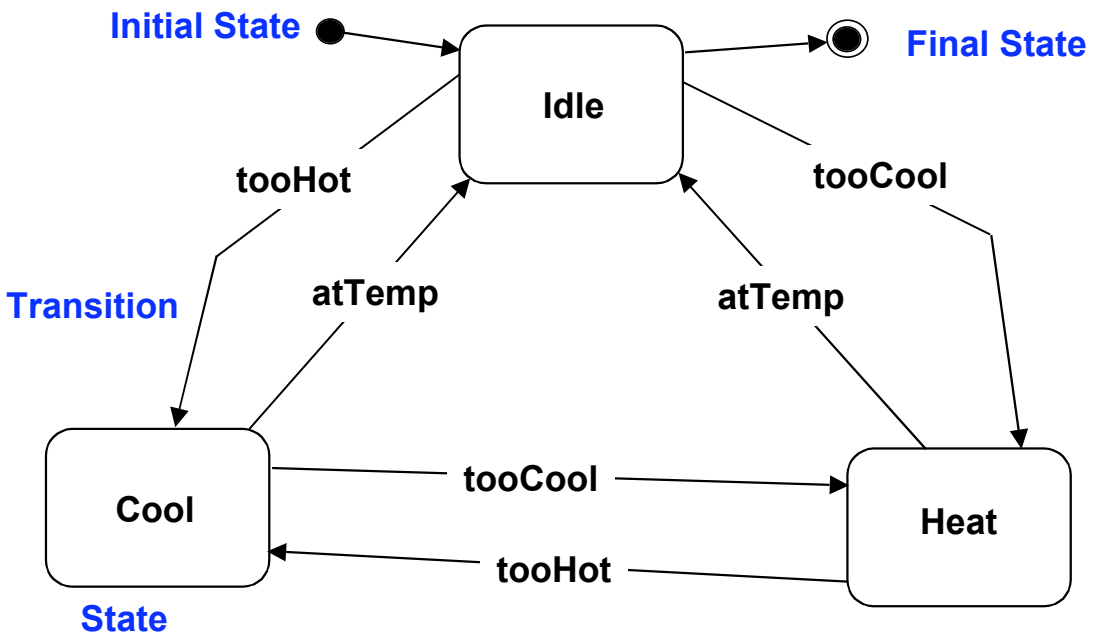


Figure 26 State Diagram Example

While UML does not prescribe it, a pair of dictionaries can be helpful in clearly stating the intended operation of a state diagram. One dictionary lists the states and defines them with precision while the other lists and defines the transitions. You will note that the transitions in Figure 26 have not been named uniquely but should be in the general case so that each can one can differentiate between them in such a listing.

2.2.4 Static Entity Analysis

In early object oriented analysis (OOA) as prescribed by Grady Booch, Peter Coad teamed with Edward Yourdon, James Rumbaugh, et. al., and many other practitioners, the proper way to enter problem space was using the static view with objects. Then they encouraged the analysis of the objects from a dynamic perspective with data flow diagrams for functionality and state diagrams for behavior. This approach is possible with UML using the foursome of dynamic modeling artifacts discussed in the prior section and it can be effective when developing a largely precedented system that can be observed in the real World like a new payroll system. The author believes that largely unprecedented problems are best attacked using Sullivan's encouragement of form follows function leading with the dynamics views and identifying the static entities that populate the product entity structure from a software perspective.

UML identifies three levels of static entities but they are all product entities and while drawn on modeling diagrams using different images, they are essentially the same at different levels of abstraction. All are simply illustrated on the system product entity diagram illustrated in Figure 14 as blocks. Figure 27 illustrates the three static entities collectively referred to as classifiers in this tutorial.

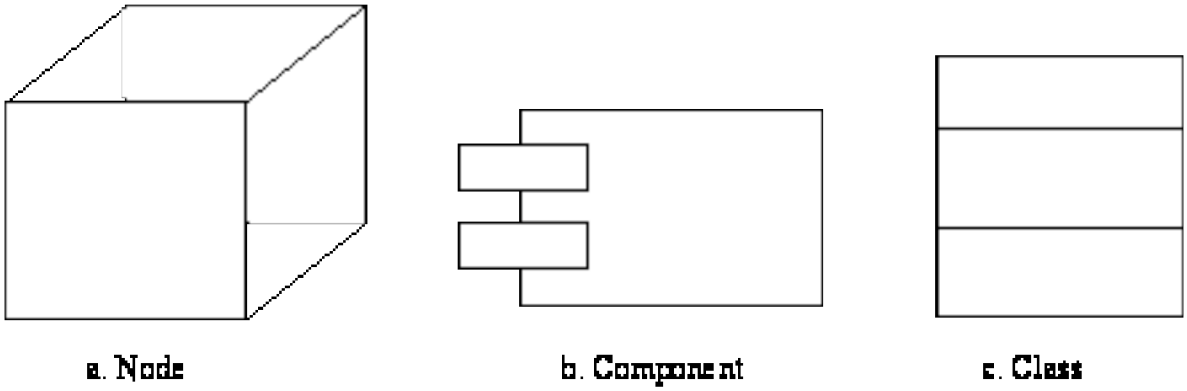


Figure 27 The UML Static Classifiers

In this tutorial the case is made for first identifying the nodes which are entities that will be associated with run time software. They are higher-tier entities. Like classes and components, they have associated attributes and operations. They interface with each other and possess lower tier interfaces between components and classes that comprise them. The nodes are identified in the dynamic analysis of the top-level software entity by identifying sequence diagram lifelines and activity diagram swim lanes. We then analyze these nodes from a dynamic perspective and identify components in the same fashion.

The alternative approach first identifies classes corresponding to observable entities in the problem space which are then dynamically analyzed leading to an understanding about how best to package these entities based on collecting the classes with the most intense interface relationships together as components. Just as in the use of interface analysis in TSA to validate the product entity decisions in functional analysis by observing possible unintended interface intensities, we can in UML re-consider the particular swim lanes and lifelines we selected in the dynamic analysis.

In this section, the intent is to explain what the UML static entities are and how they are used on diagrams. We will use classes in order to do so with the understanding that nodes and components are but higher tier classes. First we will describe a general class then explore structural relationships between these classes and finally we will cover the messages that are passed between them.

2.2.4.1 The Class

A class is illustrated as a box as shown in Figure 27c. The name of the class is placed in the top portion of the box, attributes are listed in the middle portion of the box, and operations are listed in the lower portion of the box. A forth box can be included below the operations in which we inscribe class responsibilities in free-form text. A responsibility is a contract or an obligation of a class. You may find it useful to begin the analysis of classes this way translating these into attributes and operations that best fulfill the class's responsibility as the model is refined.

2.2.4.2 Class Relationships

Figure 28 illustrates the structural relationships recognized between classes. A class is said to be dependent on another if it depends on that other class for information or services. A class can be linked hierarchically to another through a generalization. Class associations can have the three adornments illustrated in Figure 29.

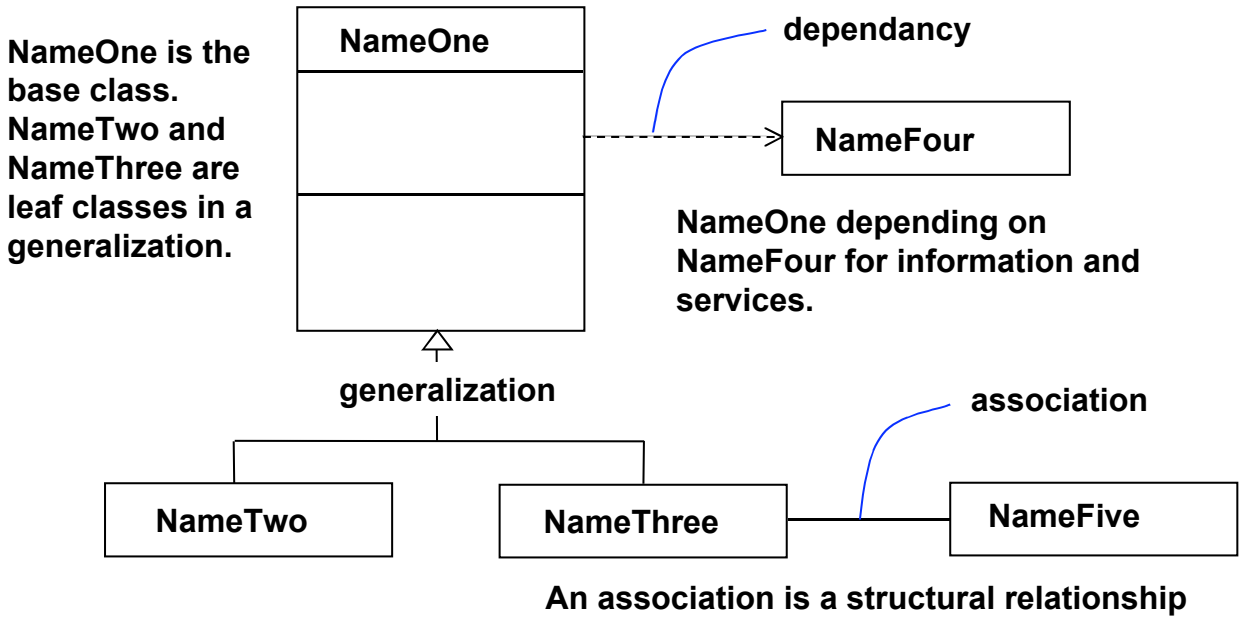


Figure 28 Structural Relationships

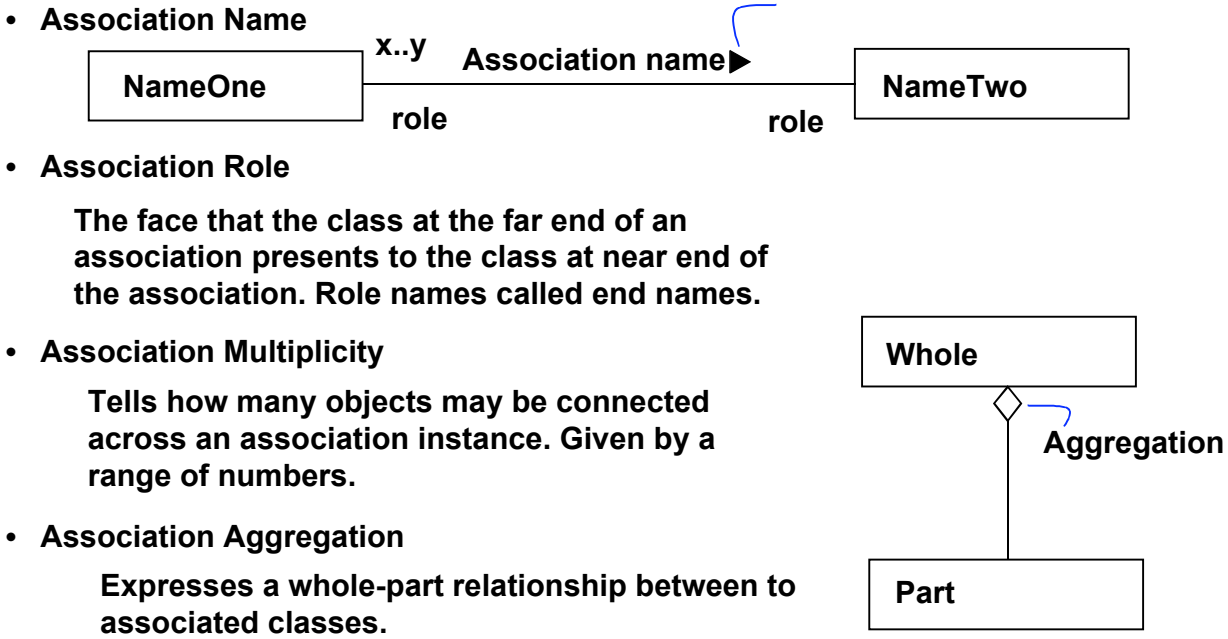


Figure 29 Association Adornments

2.2.4.3 Messages

Classes can also be related through the five kinds of messages discussed in Paragraph 2.2.3.1. Such a message can convey to a class a variable argument (value) that is needed in a class operation or it can convey a signal that causes the class state machine to transition to a new state for example. The messages that must be passed between classes are understood in the context of the sequence diagram under the assumption that the dynamic analysis is accomplished prior to the static analysis.

2.2.5 Related Analyses

2.2.5.1 Specialty Engineering

The specialty engineering matrix discussed in paragraph 2.1.9 can be used in software as well as hardware to identify all product entities for which specialty engineering requirements must be developed. This includes, for example, safety, reliability, and security. The software interface requirements fall out of the sequence and communication diagram analyses and flow into the specification template offered in Figure 27.

2.2.5.2 Environmental Requirements

Software environmental requirements are somewhat different from the hardware and system environmental requirements that tend to be dominated by the natural environmental factors. The software being an intellectual entity rather than a physical one, is shielded from the natural environmental stresses. True, the software operating within a machine that can suffer adverse environmental stresses can as a result fail, but this is a secondary effect. Reasonable software environmental relationships include any language restrictions and the computer architecture upon which it must run, for example.

2.2.6 Specification Structure

The specification outline offered in Figure 5 can also be applied to software entities where the capabilities are linked to either the terminators, use cases, extended use cases, or scenarios and the subordinate requirements listed under each capability are drawn from the corresponding dynamic diagramming (activity, sequence, communication, and state diagramming) work. Thus the requirements can be clearly shown to trace to modeling artifacts just as the performance requirements in hardware can be shown to flow so clearly from functions. Figure 30 offers an outline for a software requirements specification (SRS) using an edited template from EIA J STD 016 to integrate the supporting modeling work into the specification. Note the similarity to the outline in Figure 5 for a system or hardware item performance specification.

PARA	TITLE	MODEL	DEPT	APP
1.	SCOPE	DID	2100	
2.	APPLICABLE DOCUMENTS	DID	2100	
3.	REQUIREMENTS	DID	2100	
3.1	Required states and modes	DID	2100	
3.1.1	Classifier context diagram	DID	2100	
3.1.2	Use case analysis	DID	2100	
3.1.2.h	Terminator h	DID	2100	
3.1.2.h.i	Terminator h, use case i	DID	2100	
3.1.2.h.i.j	Terminator h, use case i, extended use case j	DID	2100	
3.1.2.h.i.j.k	Terminator h, use case i, extended use case j, scenario k	DID	2100	
3.1.3	Dynamic Analysis	DID	2100	D
3.1.3.h	Terminator h dynamic analysis	DID	2100	
3.1.3.h.i	Use case hi dynamic analysis	UML	2100	
3.1.3.h.i.j	Extended use case hij dynamic analysis	UML	2100	
3.1.3.h.i.j.k	Scenario hijk dynamic analysis	UML	2100	
3.1.3.h.i.j.k.1	Sequence diagram hijk1 analysis	UML	2100	
3.1.3.h.i.j.k.2	Communication diagram hijk2 analysis	UML	2100	
3.1.3.h.i.j.k.3	Activity diagram hijk3 analysis	UML	2100	
3.1.3.h.i.j.k.4	State diagram hijk4 analysis	UML	2100	
3.1.4	Lower tier classifier identification	UML	2100	
3.2	Entity capability requirements	UML	2100	A
3.2.m	Capability m	UML	2100	A
3.2.m.n	Capability m, requirement n	UML	2100	A
3.3	Interface requirements	UML	2100	D
3.3.1	External interface requirements	UML	2100	D
3.3.1.m	External interface m	UML	2100	D
3.3.1.m.n	External interface m, requirement n	UML	2100	D
3.3.2	Internal interface requirements	UML	2100	D
3.3.2.m	Internal interface m	UML	2100	D
3.3.2.m.n	Internal Interface m, requirement n	UML	2100	D
3.4	Specialty engineering requirements	DID	2100	E
3.4.m	Specialty Engineering Discipline m	Specialty Scoping		E
3.4.m.n	Specialty Engineering Discipline m, Requirement n	Specialty Scoping		E
3.5	Environmental conditions	3-Layered Env Model	2100	B
3.6	Precedence and criticality of requirements	DID	2100	
4.	VERIFICATION	DID	2100	
5	PACKAGING	DID		
6.	NOTES	DID	2100	
6.1	Requirements traceability	DID	2100	
6.1.1	Inter-specification traceability	DID	2100	
6.1.2	Verification traceability	DID	2100	
6.1.3	Modeling traceability	DID	2100	
6.1.4	Section 2 traceability	DID	2100	
6.1.5	Programmatic traceability	DID	2100	
6.2	Glossary	DID	2100	
6.3	Specification maturity tracking	DID	2100	

Figure 30 Software Requirements Specification (SRS) Template

Refer to Exhibit B of this student manual for a JOG System Engineering data item description (DID) that shows how the subparagraphs of paragraph 3.1 relate to the subparagraphs of paragraph 3.2. Paragraph 3.1 essentially provides an opportunity to capture the complete UML analysis for the classifier covered in the specification. This can be done by actually including the diagrams discussed in prior paragraphs in this text or by referencing them in a computer application within which the modeling is done and derived requirements captured or the diagrams can be completed manually (or with a computer graphics application like Microsoft Visio or Powerpoint) and contained in the appendices of the system definition document (SDD).

The reader will note that in Figure 19 there is a second plane labeled UML for the case where the program chooses to capture the UML analysis work products in the SDD. On a program that is dealing only with a UML analysis for a software product the Appendix structure shown in Table 1 column A could be used. If the program must deal with both TSA and UML, the software appendices could be simply added to those required for TSA and lettered G through N with the TSA Appendix G (the RAS that should also contain the software requirements derived from UML artifacts using the MID pattern illustrated in Figure 22) becoming a common RAS in Appendix O. The latter pattern is captured in column B. In this combined case, the classifier diagramming can remain in Appendix N but Appendix C should capture the aggregate product entity structure including hardware and software entities.

Table 1 Independent and Combined SDD Appendices

A	B	MODEL ARTIFACT
A	G	Context Diagrams
B	H	Use Case Diagrams
C	I	Scenarios Text
D	J	Sequence Diagrams
E	K	Communication Diagrams
F	L	Activity Diagrams
G	M	State Diagrams
H	N	Classifier (Object, Class, Component, and Deployment) Diagrams
I	O	RAS

The reader should note that in software using UML it is possible to relate the modeling machinery very clearly with the classifiers about which specifications are to be written. It is not quite so easy in hardware using TSA because all of the performance requirements derived from a particular function may be allocated to the same entity. It might be possible to force fit the functions into a closer alignment with the product entity structure by restarting the functional analysis for each entity identified but the author does not think this would be particularly helpful.

2.2.7 Software Requirements Close-Out

When a set of classes that collectively form a component have been thoroughly characterized including the sets of requirements associated with those classes and dynamic relationships and the related requirements have been captured, reviewed, and approved, the responsible team can

start to develop the computer code that will implement the component. This process can begin across the lower tier of the components and continue to next assemble into computer code for the nodes. This process may take place in a single string or simultaneously following the same pattern in multiple paths where the software is distributed to some extent. There is, of course, a continuing need for integration and optimization across the development being accomplished by different teams that focus primarily on interfaces just as in hardware development.

The PIT and program manager will reach a conclusion for the program with possible local differences whether to pursue software design and coding in a top-down or bottom-up fashion. In either case, any code developed must be tested at each level of build. This may require the development of special stub (top-down) and/or drivers (bottom-up) to permit testing of completed portions before higher or lower tier software has been completed.

The classifiers identified in the UML analysis should be entered into the product entity structure by the PIT so that a consolidated view of the overall system matures. Figure 31 shows the product entity structure evolving based on feed from all modeling activities being applied on the program under the watchful integrating eye of the PIT.

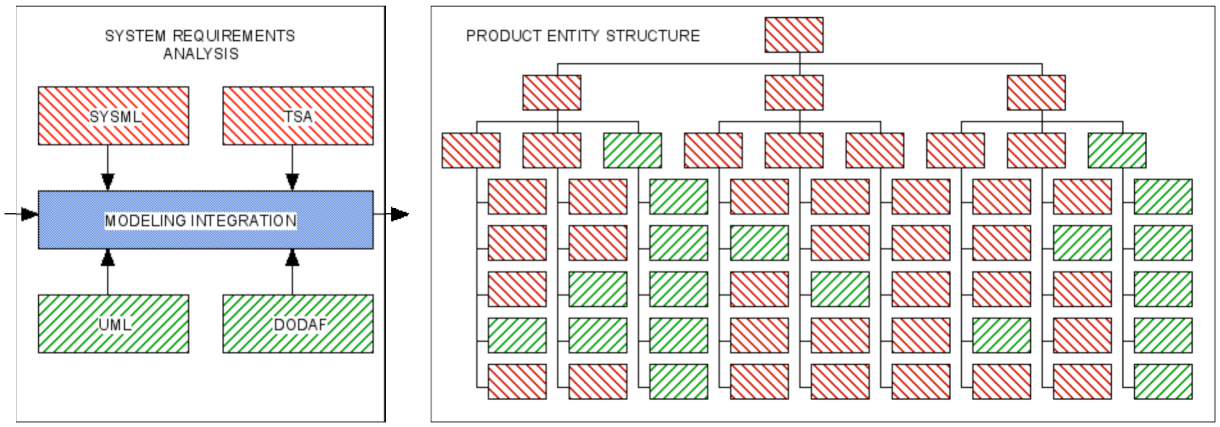


Figure 31 Evolving Product Entity Structure

The UML analysis process encouraged does involve a partitioning of the whole problem space into parts hierarchically arranged and assigned to teams. These teams, in hardware and software, can generally be depended upon to integrate and optimize at the level for which they are responsible rather than the next higher level. Therefore the PIT (and superior IPPT) must integrate and optimize across the best efforts of the teams. The PIT should also consider special integration studies across the grain of the system based on overarching functions such as shutdown and activation as well as others that may be suggested through system level states and modes analysis. These efforts should be led by the PIT and participated in by people from a wide range of teams.

2.3 Opening the Analysis With DoDAF

DoDAF was developed to support modeling of the complex information systems that DoD has been assembling in recent years to interconnect their many sensor and weapons systems to form

an effective military capability often referred to as sensor-to-shooter connectivity. It does not provide a complete modeling set primarily because of the absence of a model artifact for the physical product entities. The product entity view can be provided by augmenting the DoDAF model set with the TSA product entity diagram, however, from the aggregate model set this description assembles. Physical products identified in the DoDAF process can be inserted into the TSA product entity diagram just as they can be using any of the model sets and as suggested in Figure 31.

The DoDAF problem space entry is similar to the UML entry and in fact can use some UML artifacts to do so. DoD does not prescribe any particular set of modeling artifacts for the several views. In this paper we will use a combination of UML, IDEF, and TSA artifacts. DoDAF employs four modeling views: (1) two all view products that offer a textual overview of the system (as in what was once called a mission need statement) and an overall glossary explaining all terms used; (2) two technical view products that define standards that must be respected on the program and the evolution of those standards over time; (3) nine operational view products that capture how the user views their needs; and (4) thirteen system view products that offer the engineering perspective on the needed system. Figure 32 illustrates a recommended DoDAF development sequence. The simulation work could better be shown overarching both the analysis and design work interacting with and serving both. Note that the two all and two technical views would be a good addition to any modeling apparatus.

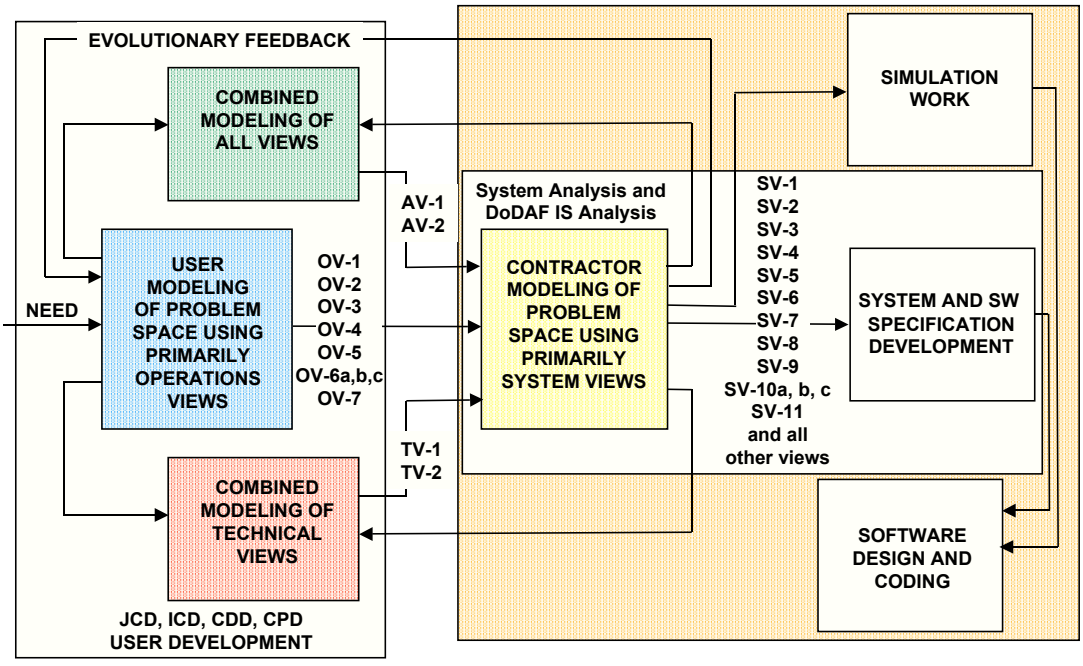


Figure 32 DoDAF Development Sequence

The technical and all views work products should be developed initially by the user and acquisition agents and evolved by the contractors selected to develop the system. The user should have developed the operational views in the process of evolving a set of documents used to refine their need. These documents include a Joint Capabilities Document (JCD that takes the

place of the old need statement cut very broadly. In the study process prescribed by DoD for a new system, the capability identified in the JCD may be cut up into n systems each of which should have developed for it an Initial Capabilities Document (ICD) that is the equivalent of the old mission needs statement for a given system to be procured. The ICD is then matured into a Capabilities Development Document (CDD) and eventually a Capabilities Production Document (CPD). In between these last two a contract would commonly be let resulting in the development of a system specification attached to the contract. DoDAF can be used to gain a systematic insight into the content needed in all of these documents.

2.4 Integrated Modeling

A program must make a decision about the modeling techniques it will apply as it builds the proposal. This may include some mix of TSA, UML, SysML, DoDAF, and IDEF-0. A program that is going to be primarily computer software or networked assets could enter the program with UML or DoDAF. If the product is a database, it could enter with IDEF-0 or UML. But, generally, modeling entry should involve the use of TSA or SysML at the system level because software, an intellectual entity, must run on hardware entities that provide the product with real substance. As noted earlier, at the time this was written SysML was not yet fully ready for use so TSA would be the author's preference now. But, an enterprise should continue to follow the development of SysML and work toward replacing TSA with SysML. In any case, there is a need to recognize that for some period of time there will be a need for a model that works well for systems and hardware and another model that works well for SW. For now, also, there is not one great computer application within which one can model HW and SW requirements work and permit easy cross model traceability and provide specification publication capability so it will be necessary to use two or three applications to cover the needed tool set.

Work can be accomplished well for systems and HW as covered under paragraph 2.1 and for SW using the approach covered under paragraph 2.2. When applying both, problems will tend to arise when transitions have to be made between these two approaches. It is not possible for SW to include HW but the opposite case is perfectly normal. So, the transitions will only be a problem as the analysis shifts from HW to SW moving from the use of TSA or SysML to UML. There are two concerns at this point: one in the models applied and the other in the computer applications employed.

The transition point will occur when the highest tier software entities are identified. There may, of course, be several of these transitions distributed about the expanding product entity structure. The program has the option of pooling all of the software into an integrated entity or permitting it to be distributed within multiple processors that may still all be under the responsibility of one team or distributed among teams with both hardware and software responsibilities. If we can solve one of these hardware-software handoffs we will have solved the general problem of requirements traceability across these gaps.

It should be clear that requirements traceability to models is assured in the approach covered in this tutorial because all of the requirements are to be derived from a model. Vertical or hierarchical requirements traceability is very simple in specialty engineering areas in hardware, software, and across the gap as described in this tutorial. The environmental requirements are

vitally different between hardware and software and one can make a case that lower tier software environmental requirements should not have to respect traceability across the gap to higher tier hardware or system environmental requirements that are largely environmentally related. Precisely the same method of identifying hardware interface requirements can be used to identify software interfaces as well as hardware-software interfaces because we identify them between entities that appear in the joint product entity structure. So, if interface requirements traceability involves lower tier interface expansion requirements to higher tier interface requirements, traceability is assured. This leaves only the performance requirements a remaining problem from a vertical or hierarchical traceability perspective.

Given that the system entry analysis was accomplished in TSA using some form of functional analysis and the lower tier software analysis is going to be done using UML, there is a temptation to employ activity diagrams in UML to analyze software entities from a dynamic perspective because it is very similar to functional flow diagramming and might give us some interesting opportunities to link up hierarchical traceability. However, for a given software entity there may have been 10 performance requirements derived from 8 functions allocated to the software entity in question. There is no clear way to link up the activity analysis and requirements derived from it with the several functional analysis strings and the performance requirements derived from them that can easily be automated.

So, let us pursue another tack in an attempt to coordinate the traceability relative to the sequence oriented dynamic analysis approach described previously. If requirement $R\%_1$ is one of a set of requirements $R\%_1$ through $R\%_{10}$ where $R\%_1$ is derived from function $F\#_1$ of a set of functions $F\#_1$ through $F\#_8$ and requirement $R\%_1$ is allocated to product entity AX2 and it is decided that AX is going to be developed as a software entity, then one of the scenarios to be analyzed will be UXhijk. Assume that we accomplish the dynamic analysis using sequence diagram UXhijk1 from which we derive requirement $R@_1$. What we are looking for is a way to establish hierarchical traceability between requirement $R@_1$ and some requirement in the set $R\%_1$ through $R\%_{10}$. The X, %, and # characters are being used to designate base 60 strings in this discussion. We know that requirement $R@_1$ must be traceable to one of the 10 performance requirements allocated to classifier AX2 and we can look at that list of requirements and select the one most closely related.

To make this selection more organized, we can form an x by y matrix, in this case a 10 by 12 matrix, and pair-wise compare the sets R@ and R%. In Figure 33 you can see this whole process taking place The 10 functionally derived requirements are captured in the RAS mapped to the set of functions $F\#_1$ through $F\#_8$ and allocated to product entity A&. Based on these requirements we build a context diagram for entity A& and analyze A& from the perspective of each of the three terminators shown. As an example Use Case U&3 is extended to three use cases and we build three scenarios one of which, U&3111 is analyzed from a dynamic perspective with some combination of sequence, communication, activity, and state diagrams. Requirements $R@_1$ through $R@_{12}$ are derived from these analyses and captured in the RAS (possibly linked to the RAS database from a UML modeling application).

There are 10 requirements ($R\%_1$ through $R\%_{10}$) to which the requirements in the set $R@_1$ through $R@_{12}$ will have to hierarchically trace. We can build a 10 x 12 matrix and pair-wise analyze the

relationships between the two sets of requirements, perhaps concluding that one of the matches is $R@_1$ traces to $R\%_1$. All of the matches are marked in the requirements management database table for traceability relationships. There are no known databases that provide the traceability evaluation matrix so it may have to be accomplished as a pencil and paper aid. However we should be able to set the requirements management database filter for the two sets of requirements of interest aiding in the identification of the sets of interest for a particular case. In this example, there might be ten or more sets of requirements like R@ derived from ten or more dynamic analyses. In each case an x by y matrix would be needed to pair-wise analyze the traceability relationships. In any case, it should be clear that we can have good requirements to modeling traceability and even good hierarchical traceability across the gap between performance requirements.

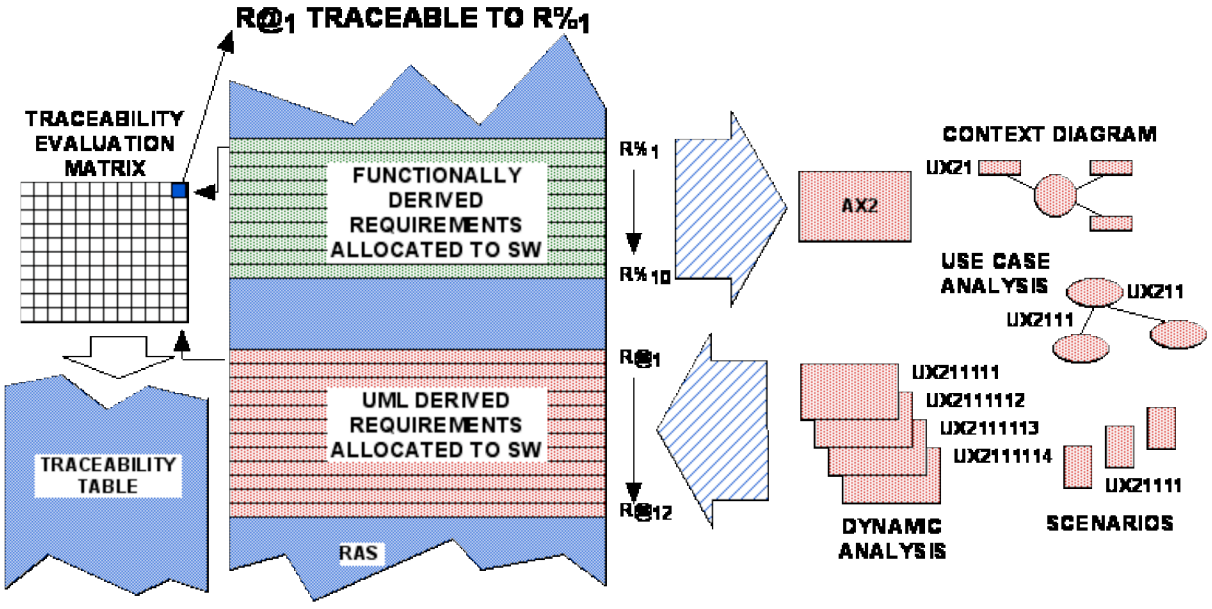


Figure 33 Requirements Traceability Across the Gap

In both TSA and UML we have discussed a decomposition process that partitions the problem space into parts in which the analysis is accomplished. Whenever we partition any whole we have an obligation to integrate and optimize across the boundary conditions thus created. The PIT must accomplish this integration work relative to the top level IPPT and each IPPT with lower tier teams must accomplish this work relative to it's own immediately subordinate teams. Much of this integration work will take place at the interfaces ensuring that requirements on one end of an interface are compatible with those for the other terminal. Each team with subordinate teams, however, should also integrate across its immediately subordinate teams relative to the requirements derived at the subordinate team level relative to those at the parent team level. Part of this work can be accomplished by simply establishing the traceability between the requirements at the two levels. Another approach of value is to accomplish higher tier function effects across the lower tier team responsibilities. For example, one can inquire collaboratively into lower tier performance of higher tier functions like turning the system or entity on or off,

moving from one major mode to another, accomplishing some kind of transfer function, or physical separation or joining of two entities.

Another kind of traceability can also be used to stimulate integrating results. This was pointed out to the author by an engineer at Puget Sound Naval Base in Bremerton, WA. Given a requirement at level m, we can inquire if the intent of the requirement was fully implemented in the requirements for the n entities at level m+1 (downward). This kind of traceability inspection must await the development of the subordinate specifications, of course, as does all hierarchical traceability.

At one time, in the 1950s when software was a very young discipline, it happened that hardware and software analysis, to the extent that it was done, was accomplished using exactly the same model, flow charting as shown in Figure 34. Over time, probably encouraged by the ease with which flow charts could be outputted onto line printers using ASCI symbols, computer software people got into the habit of building flow charts in the vertical rather than the horizontal axis still used by system engineering in their functional flow diagrams. The activity diagrams of UML still reflect the vertical orientation but it is really of little significance which orientation is used. The absolutely fascinating approaching reality is that system and software people will rejoin the same house in the near future. As UML and SysML become more fully integrated as suggested in Figure 34, we will achieve a tremendous milestone of universal unified modeling capability.

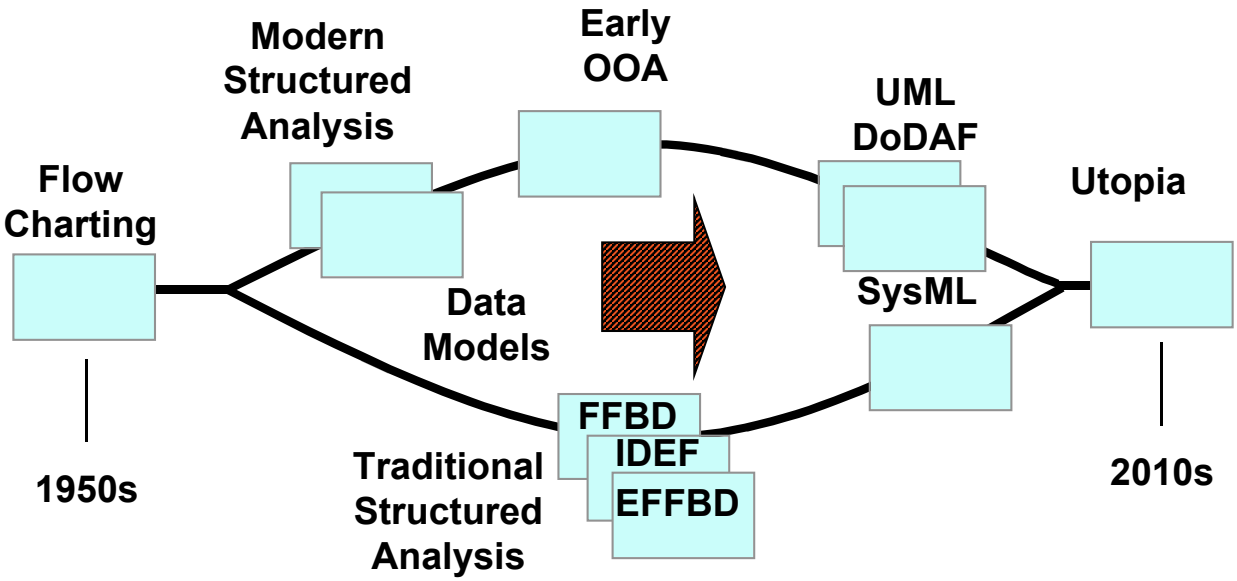


Figure 34 Modeling Over the Years

As we pass through this door into a world of integrated modeling and supporting computer applications, we will find it a more reasonably affordable task to integrate across the hardware-software boundary than has been the case for many years. But then as now, integration takes place in the minds of the system engineers working on the program. These engineers must be

every vigilant for inconsistencies between information sets that signal that two different domains are not working from a common understanding of the problem and solution spaces.

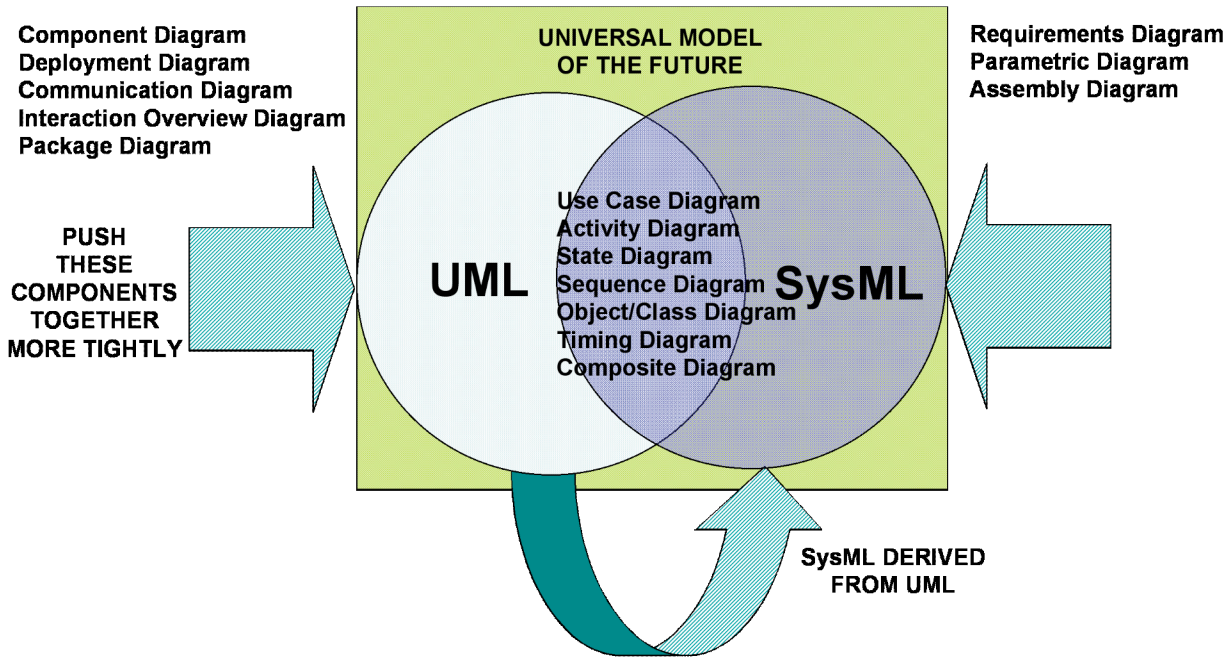


Figure 35 The Approaching Merge

3 Requirements Management

3.1 Summary of Team Activity During Requirements Work

During initial product development work, the PIT will model the problem space using a predefined set of modeling methods selected from the list of enterprise-approved modeling methods and apply those methods to identify top-level system entities and interface relationships and their requirements. This level of system definition shall be completed before program level IPPT are initiated on the program and staffed. These top-level product entities are the basis for assignment of these teams. When a team is established, the leader shall be presented with the specification for the top-level entity for which the team shall be responsible, a design concept (in particular if it is HW or SW), a clear definition of all external interfaces, and the corresponding components of the WBS, SOW, IMP, and IMS. The team will be charged with the development of that item and all subordinate entities and interface relationships. In the process of so doing, the team may conclude that lower tier teams are required and must request that action of the PIT.

The work products from the IPPT will be loaded into the computer applications used on the program by PIT and checked for consistency. The PIT shall perform integration and optimization work on modeling work contributed by the IPPT fitting the work products into a coherent system analysis of the problem space. As part of this work where the product is to be implemented in SW, the responsible team will seek to identify all needed use case analyses and assign them for completion by people on the team if possible. Where this work must involve people from other

teams, the responsible team must request help from the PIT and a cross team analysis effort will be established. The responsible lead team must ensure that each analysis is complete with all needed supporting modeling work.

The PIT shall maintain the product entity structure, the interface relationships, and all requirements modeling and management assets. The requirements shall be retained in a relational database from which specifications may be printed to paper or computer screen and within which traceability may be maintained. This database shall be linked to one or more modeling applications used on the program. The modeling applications shall be used for the purpose of identifying the system entities, interfaces, and appropriate requirements in each case. All content of these applications shall be under the control of the PIT until such time that it is formally approved at which time it shall fall under configuration management control and shall not be changed without a formal approval as well. Responsibility for data entry can be distributed to IPPT or retained by PIT. Entry may be aided by special Microsoft Office applications making it unnecessary for personnel to develop and maintain computer application skills.

The PIT will seek to establish IPPT overlaid upon the product entity structure so as to minimize the interface relationships between the entities for which the teams are responsible. The purpose of this arrangement is to minimize the need for cross team coordination and staffing for use cases analyses.

The preferred lower tier HW development approach entails a continued application of TSA using the same modeling database application applied at the system level. The preferred SW product development approach entails PIT and IPPT application of unified modeling language (UML). In the near term traditional structured analysis (TSA) will be applied in combination with UML maturing to a combination of UML and SysML as the latter matures into a formally released model by the Object Modeling Group. TSA or SysML are to be used initially to identify system and top level product entities that will more often be hardware end items. As the analysis proceeds downward and identifies a need for computer software, the analysis should switch to UML.

3.2 Requirements Tools Base

Figure 36 illustrates the preferred tools environment for programs. A requirements management database is used to capture all of the requirements that will be published in specifications and those specifications may be published from this database. In Figure 36 this is referred to as a big dumb database with no slur intended for the makers of tools that do not include integrated modeling capabilities. This is a relatively simple relational database application that can contain text information in a tabular structure. Each table row corresponds to a unique requirement with data captured in table columns for the several fields needed. Additional relational tables may be required for vertical traceability, verification traceability, and lateral (to models from which the requirements are derived) traceability. The program may use available modeling applications for UML (such as Rational Rose) and TSA (such as CORE) and arrange for traceability between these applications and the management application in an application like DOORS.

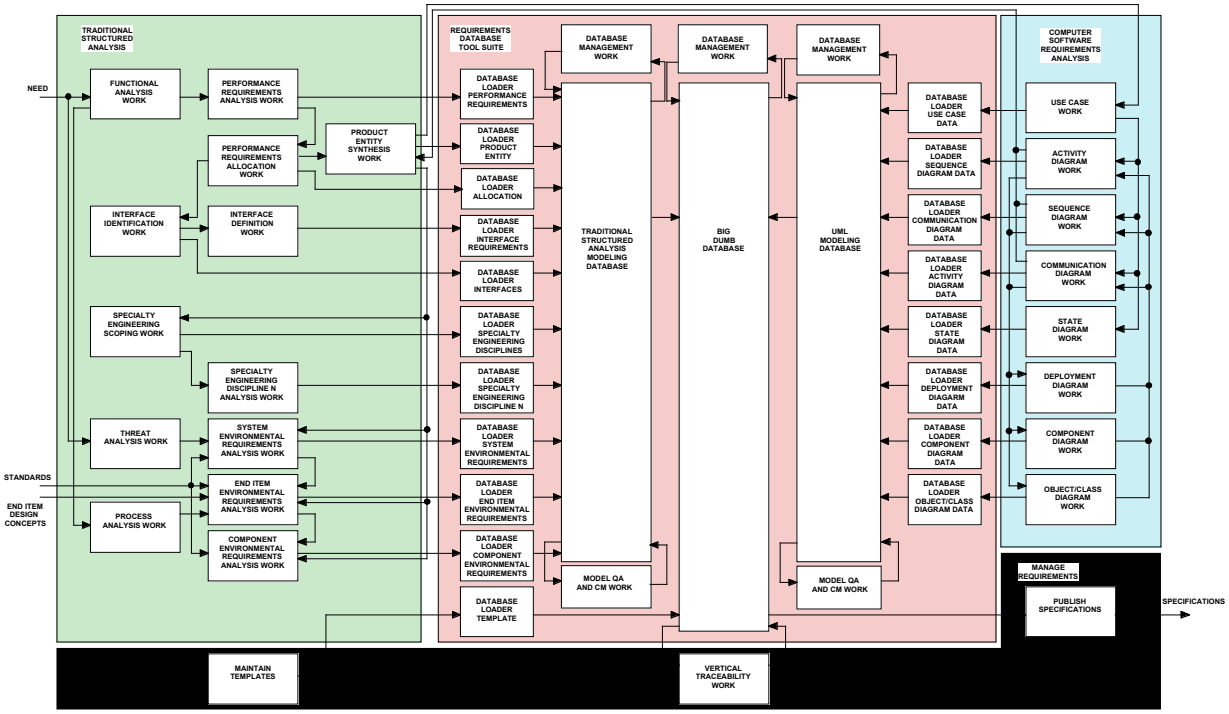


Figure 36 Tools Environment

Many enterprises find it difficult for engineers to maintain currency with a set of requirements database applications as well as other applications more directly related to their work. All or most of these engineers will, as a function of accomplishing their normal work, maintain proficiency with the three fundamental Microsoft Office applications (Word, Powerpoint, and Excel). Loader applications crafted from Microsoft Office applications may be used to permit all engineers to enter data into the requirements database suite without a need for the engineers to become intimately familiar with these applications.

The PIT must exercise integration and optimization control over the requirements application suite and will require some members who really understand the applications, how they work, and how their content is inter-related.. The suit must be set up to permit passing control of approved content to configuration management while retaining control of all in-work data.

3.3 Recommended Specification Responsibility Pattern

In the author's view, a program should staff a program integration team (PIT) that should begin the requirements analysis process at the system level and develop the top level diagrams in the SDD. This work should continue as necessary to develop the content of the system specification and the specifications corresponding to the top-level teams. The structured analysis for each of these teams should be taken over by the corresponding IPPTs in each case until they have completed the content of the specifications that define the problem for any subordinate teams. If no subordinate teams have been identified then they would have to complete the analysis needed to develop all of the specifications subordinate to their top-level specification. This same pattern

carries down to the lowest level. Each team should act as the system agent for all of its lower tier teams and principal engineers. This starts at the PIT for the system and works its way down through the lower tier teams. The Program Manager and Chief Engineer/PIT Manager should review and approve the system specification and all top-level specifications. PIT should establish rules for review and approval of lower tier specifications created by the teams.

With different parties doing the structured analysis, it is necessary to apply process integration and the PIT should do that accepting data into the several appendices of the SDD, numbering the figures, and cross-checking the data submitted. At least one team will be involved in software development and if traditional structured analysis has been applied for the system level, then that or those teams responsible for software will want to switch to some form of software modeling such as UML. Regardless of the modeling methods applied, all of the requirements modeling artifacts should be captured in the SDD either in the paper appendices noted earlier or referenced in the database systems used. The integrated RAS should be implemented in the big dumb database of Figure 36 as simply the basic relational database table used to capture requirements.

3.4 Requirements Risk Management

The principal risks that appear during the requirements development work involve a sensed inability to satisfy the requirements. The risk may be motivated by the conclusion that insufficient financial, or schedule resources have been made available. The concern may be that the requirement simply cannot be satisfied with available technology. Finally, the concern may be motivated by the conclusion that the value is simply too hard to achieve with available skill and knowledge. It is not uncommon to partition all into the categories of cost, schedule, technology, and performance as a result.

3.4.1 Requirements Validation

EIA 632 identifies an activity called requirements validation where we make an effort to determine to what extent we can satisfy particular requirements. The simplest way of reaching this conclusion is to simply ask the person(s) responsible for accomplishing the related design work if they can satisfy the requirements. If there is a lack of confidence then we need to proceed further in our efforts to identify potential performance risks. As requirements are identified and written we should validate them at that time. In most cases, the conclusion will be that there is no problem. Should we conclude that either there is a problem or we are not certain that can satisfy the requirement the first alternative investigated should be to ask if the requirement can be changed making it more certain that it can be satisfied. If that is not possible, then we should choose a means to mitigate the risk through an appropriate analysis, development evaluation test, simulation, or demonstration. If we believe that the requirement is very important in the development effort and that it will require some time to reach a final conclusion, we may select the requirement for more intense management as a technical performance measurement (TPM).

Parameters are managed through TPM by placing them on a list and assigning each TPM to a specific engineer who is charged with closing the gap between the required value and the currently demonstrated capability. The parameter principal engineer must maintain two charts: (1) a parameter chart that tracks these two values over time annotated with notes citing important

events coinciding with significant changes in the relationship between the values and (2) an action plan stating what is going to be done, when, and to what anticipated effect.

3.4.2 Margins and Budgets

Every program manager will experience difficult problems each requiring a tough decision. These problems can be made less severe by ensuring that the program manager has the resources available. This outcome is encouraged where the values for the most difficult requirements are combined with margins such that there is an opportunity to award engineers with a very difficult design problem some slack. These margins come in three varieties: cost, schedule, and performance. Cost margin is commonly applied as a management reserve such that the program manager can award a team more cost to solve a problem. Similarly, if a team has a design problem that can be solved through award of schedule slack time, the design may be possible. The third kind of slack is requirements margins. In all of these cases, the margin is derived by invariably making the job more difficult. Cost margin is often made available by skimming the task estimates of 10-15%. Schedule slack is obtained by subtracting available time for tasks on the critical path. Risky requirements are made more difficult to achieve. For example a weight margin may be realized by subtracting 5% from all weight figures. So the engineers will be challenged to accomplish their design with a weight value of required value - 5%. This is more difficult, clearly. The good news is that engineers will most often make these more demanding requirements preserving the margin values for the most difficult problems. When a very difficult weight problem appears, the manager can allocate some available margin. The margins invariably will be consumed before the design process is complete but there are ways of using available margins from requirements not of the same kind. For example, if an engineer has difficulty reaching his/her reliability figure after all of the reliability margin is gone, the manager can award some cost margin to permit the use of better parts or some mass margin permitting a heat sink that will reduce junction temperatures.

Requirements budgeting also has a risk reduction effect because it partitions available requirement values to the several designs at any one level of indenture. For example, given that it has been decided that 1500 watts of electrical power will be available from a source and there are 10 loads to be attached to this source, an engineer must partition this available power in a rational way between these several loads and integrate the results.

3.4.3 Risk Tracking

Risk is often measured using a dual axis criteria dealing with the probability that the concern will be realized and the degree of difficulty it will present if it does come to pass. This makes it a little difficult to track a single risk parameter over time and the way many people apply the dual axis system makes it difficult to accumulate a program metric that can be tracked over time. A variation on the safety hazard index described in MIL-STD-882 offers a way to measure risk with a single parameter that responds properly to characterize instantaneous values and a historical record for the program.

Figure 37 shows the risk matrix. Tables 2 and 3 corresponding provide the dictionaries explaining the values entered on the matrix axes. The intersections contain an index number that is simply the product of the axis numbers.

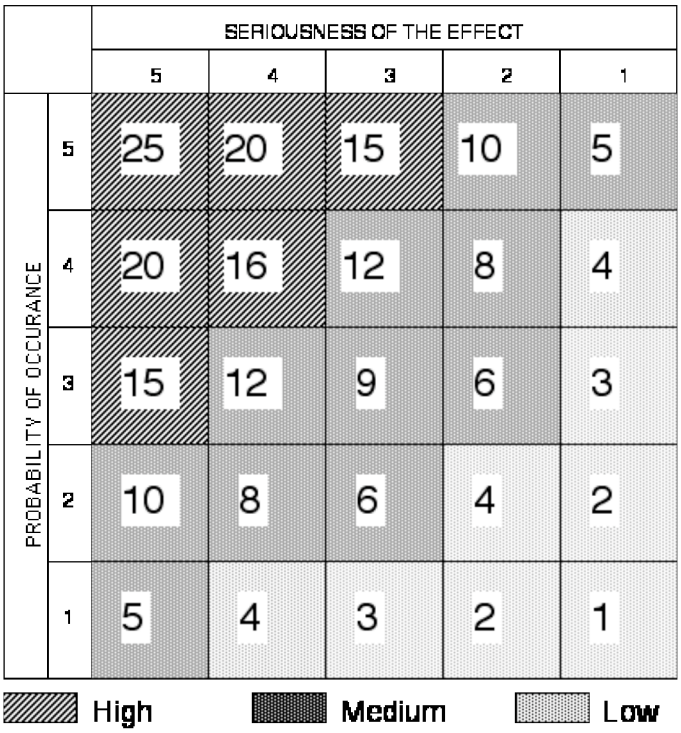


Figure 37 Risk Matrix

Table 2 Risk Probability of Occurrence Criteria

CAT	TITLE	P(O)	DESCRIPTION
5	Nearly Certain	0.95-1.00	Will occur at least once during program
4	Probable	0.75-0.95	Will probably occur once during program
3	Possible	0.50-0.75	May occur during program
2	Unlikely	0.25-0.49	Will probably not occur during program
1	Nearly Impossible	0.00-0.24	Will not occur during program

Table 3 Risk Effects Criteria

CAT	TITLE	DESCRIPTION
5	Catastrophic	Program in jeopardy of termination
4	Serious	Serious damage to program
3	Moderate	Problems cause program focus difficulties
2	Minor	Problems that can be easily be overcome
1	Null	No problem

If you were to compare this information with the MIL-STD-882 safety hazard matrix, you would find that the safety hazard matrix offered in the military standard uses letters for one of the two axes and that the highest hazards (risks) have the lowest indices. Our matrix in Figure 37 uses numbers on both axes and the index values are higher for more serious risks. Therefore, it is possible to apply the index values in a mathematical sense as a program metric. Given the 6 program risks listed in the program risk list shown in Table 4 with the indicated risk index values, the instantaneous program risk index is 97.

Table 4 Program Risk List

RISK NBR	RISK TITLE	PROB	EFF	INDEX	TM	PRINCIPAL	SUSP
2	Life Cycle Cost	4	4	16	1	Burns	02-10-05
5	Payload Capacity	4	5	20	1	Adams	03-08-05
7	Stoddard Supplier Risk	5	4	20	3	Thornton	04-20-05
12	Program Funding	4	4	16	0	Connolly	03-10-05
15	Computer Software Schedule	5	5	25	4	Sampson	05-23-05

	CURRENT PROGRAM INDEX			97			

Thus, we have arrived at a program risk index or metric. If we maintain a chart of this metric over time we see that it characteristically will rise early in a program as risks are identified but there is a delay in mitigating them causing a rising metric as shown in Figure 38. As a program progresses, this value will rise to a peak at some point in the program and subsequently start a long decline. Risks continue to be identified so we see the accumulated total number of risks continue to increase but the program is being successful in mitigating risks and later risks are commonly lower in index than those identified earlier in the program.

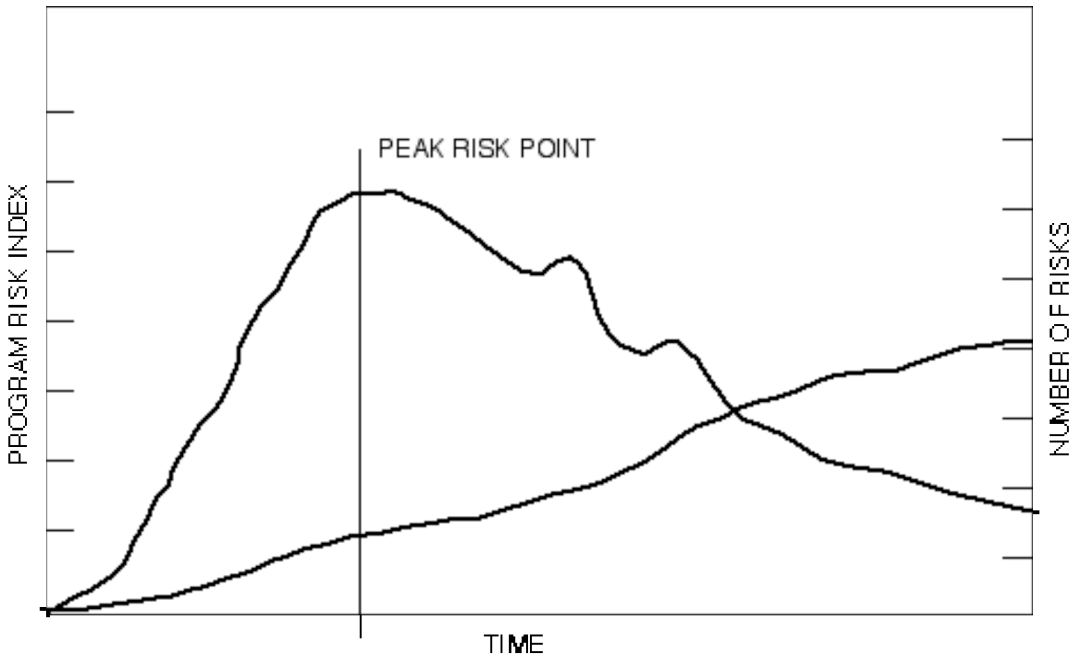


Figure 38 Program Risk Tracking Chart

3.5 Verification Requirements

The author respects the six-section specification structure of the DoD specification standard where Section 3 contains the product requirements and Section 4 the verification requirements. For every product requirement in Section 3 there should be one or more verification process requirements in Section 4. Whoever writes the Section 3 requirement should also write the verification requirement and with little time between the two actions. The rationale here is that the author of an unverifiable requirement will have some much difficulty writing a verification requirement and this difficulty may stimulate them to look for better ways of writing the requirement leading to a verifiable requirement.

We commonly respect four methods of verification: test, analysis, demonstration, and examination. A verification traceability matrix should be included in every specification that correlates every requirement in Section 3 with a method of verification and one or more verification requirements in Section 4. Each one of the rows in the verification traceability matrix forms a verification string. All of the verification traceability matrices are pooled into a single program-wide verification compliance matrix shown in Figure 39 that lists every verification string.

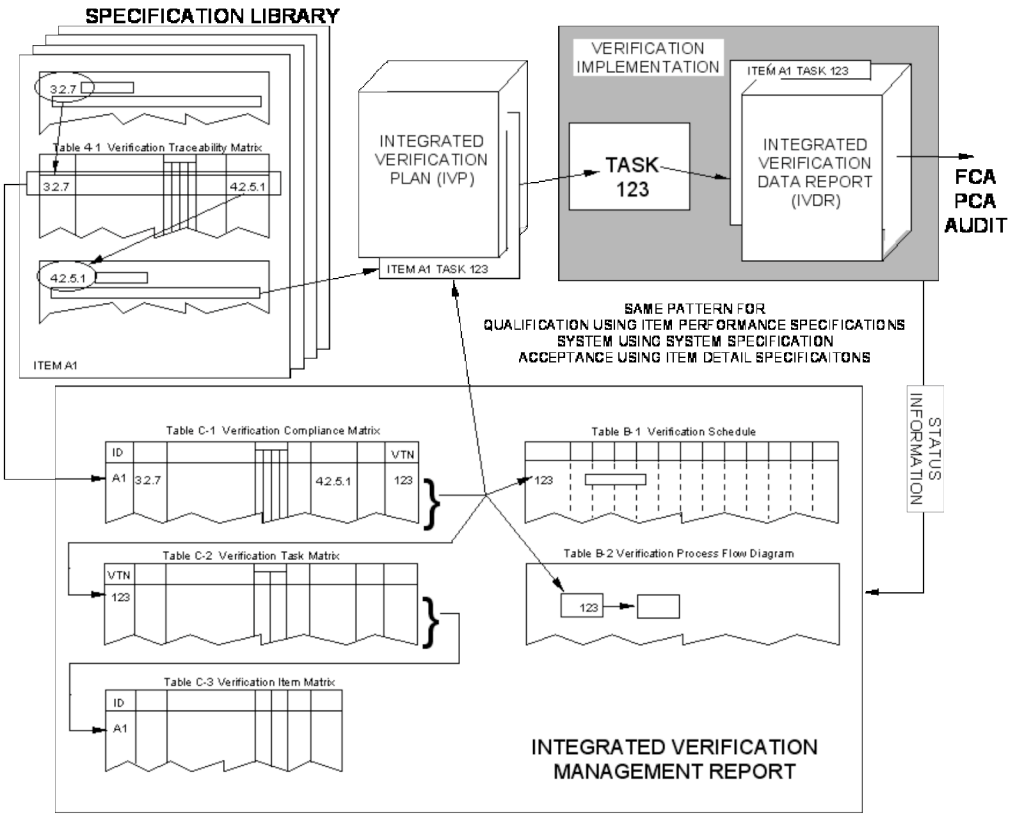


Figure 39 Verification Traceability

A verification engineer or team must now assign verification task numbers to all of the strings in the compliance matrix. Each task is identified in a task matrix and coordinated with the name of a principal engineer who must plan the task, make arrangements for the needed resources in a timely way relative to the task, accomplish the task on schedule, and produce a verification report. The reports from all of the tasks for a given item may be subjected to a configuration audit by the customer to ensure that the contractor did meet all of the requirements in the specification.

3.6 Specification Review and Approval

No matter the path the specification has taken through the requirements analysis process relative to modeling, the program should pass it through a review and approval process before it becomes a part of the formal configuration baseline definition. The review and approval process, shown in Figure 40, offers a formal or informal peer review way of comparing the content of a specification with a set of standards that all specifications should meet. Following approval, the specification must be formally released, published, and made available to program personnel either in paper or on-line form. The released specification must thereafter be formally protected through configuration management. Any changes must pass back through this same process to gain approval.

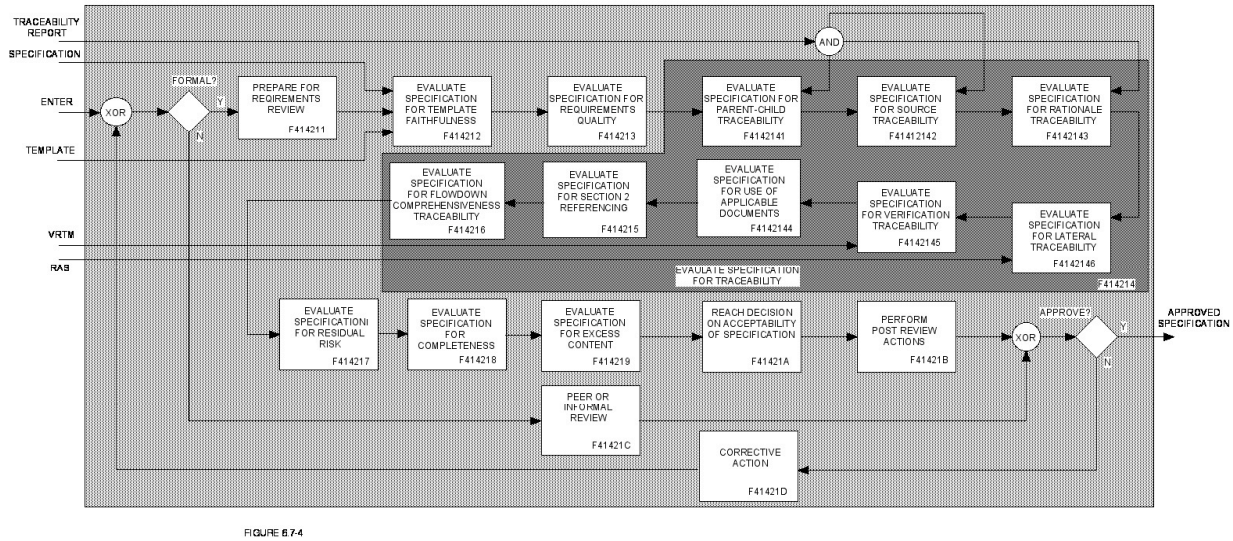


Figure 40 Specification Review and Approval

The formal review process should include a conscious evaluation of specification template faithfulness and overall quality measured in accordance with a specification checklist. Next, the specification should be checked for adherence with good traceability standards. The program may choose to fully implement traceability standards shown in the figure or some subset thereof. The final string of checks shown in Figure 40 assesses the specification for residual risk, completeness and excess content. A decision is reached by the reviewers followed by the review chairman calling either for corrective action or approval of the specification, if needed.

Specifications prepared on small or advanced programs may not have sufficient budget to support a formal review process. In this case, while not as supportive of a low risk approach, a

specification can be reviewed by experienced people in a less organized fashion, called a peer review. The team is assembled and asked to review the document either together or on-line at their desk followed by a group meeting to discuss the content.

The master copy of each specification must be retained and protected by an assigned authority in order to protect the integrity of the document. Once approved and released, this master must be accurately identified and protected against change. In one organization the author recalls, the master was changed during work on an engineering change proposal (ECP) but the ECP was subsequently canceled. The organization no longer had a master for the specification in effect because it had become corrupted by the change work that did not materialize. It helps to consider each specification build or change a separate campaign that results in the release of a document that will exist forever. If subsequently that document is changed, the change is built anew on the preserved baseline past.

Specifications must be readily available to personnel working a program. As they are released they must be distributed to those who need them. As they are revised the same is true of the revisions. Years ago specifications were crafted with typewriters and type setting. These were published in paper form and distributed using shoe leather and mail systems. If most of your specifications are in paper media, you may have no near term option but to place them in a paper document library from which program personnel may check them out physically. But, even if this is the current case, you should be making plans to move to an on-line specification library for cost, efficiency, and document configuration control purposes.

In a paper media, after a specification is formally released, the master must go to reproduction where sufficient copies are made to cover the needs for distribution and the library. The master should be returned to what the author has accurately heard referred to as the vault, a physically secure facility (not in the classified data sense unless this is also a valid concern) where all of the engineering masters are retained. The copies must then be physically distributed. If the specification in question is also a customer-approved specification, another loop will be required to gain their approval prior to distribution. A networked library will avoid a great deal of this busy work.

Adios paper and good riddance. Even if you are currently using a paper media for distribution you probably already have the resources in place to convert to networked computer media. It requires specifications captured in computer file media, a network with adequate storage capacity and speed, and easy access from work stations on the part of the people. These features are present in most everyone’s shop today or are not beyond the pale to achieve. It is unimaginable that anyone would use a typewriter today to prepare a specification so they will always be created in some form of word processor or computer database. The results of this work can be passed on to the document release function on a disk or via the network connection and thereafter transferred to an on-line library from which anyone may open it but not change it.

JOG SYSTEM ENGINEERING, INC
GRAND SYSTEMS DEVELOPMENT
TRAINING PROGRAM
TUTORIAL

VOLUME 2R
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM
STUDENT MANUAL

EXHINIT A
PRESENTATION MATERIALS

JOG SYSTEM ENGINEERING, INC.
GRAND SYSTEMS DEVELOPMENT
TUTORIAL PROGRAM

AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

SPECIFICATION READINESS

VERSION 10.1

Who Is Jeff Grady?

CURRENT POSITION

President, JOG System Engineering, Inc.
System Engineering Assessment, Consulting, and Education Firm

PRIOR EXPERIENCE

1954 - 1964 U.S. Marines
1964 - 1965 General Precision, Librascope Div
Customer Training Instructor, SUBROC and ASROC ASW Systems
1965 - 1982 Teledyne Ryan Aeronautical
Field Engineer, AQM-34 Series Special Purpose Aircraft
Project Engineer, System Engineer, Unmanned Aircraft Systems
1982 - 1984 General Dynamics Convair Division
System Engineer, Cruise Missile, Advanced Cruise Missile
1984 - 1993 General Dynamics Space Systems Division
Functional Engineering Manager, Systems Development

FORMAL EDUCATION

BA Math, SDSU
MS Systems Management, USC

INCOSE First Elected Secretary, Founder, Fellow

AUTHOR System Requirements Analysis (1993, 2006), System Integration, System Validation and Verification, System Engineering Planning and Enterprise Identity, System Engineering Deployment

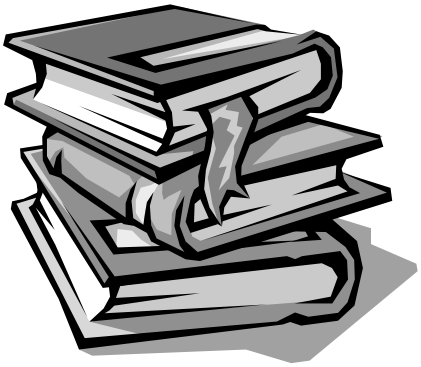
An Effective Specification Development Algorithm Tutorial Outline

2.1	Specification Preparation Readiness
2.2	Specification Preparation Readiness
2.3	Traditional Structured Analysis
2.4	Traditional Structured Analysis
2.5	Unified Modeling Language
2.6	Unified Modeling Language
2.7	Requirements Management
2.8	Requirements Management

Success Is Possible

- **The Goal**
  - **Good specifications**
  - **On time**
  - **Affordable**
- **The Plan**
  - **A sound beginning - be prepared**
  - **A clear path to a successful state - always clear what must be done**
  - **An effective closing - a specification review and approval process**

What is a Specification?



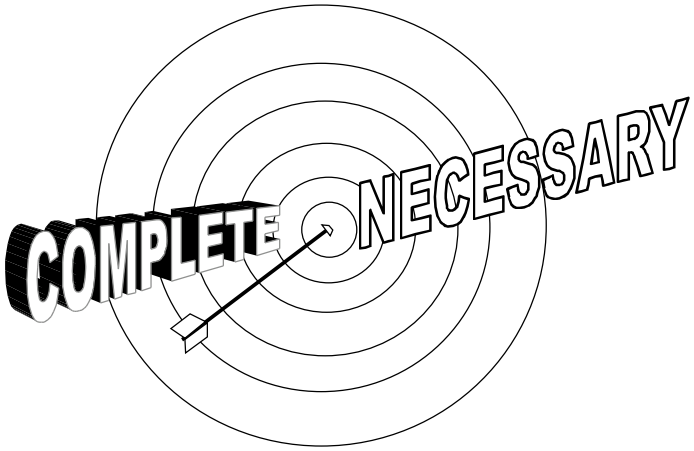
A specification contains all of the require-ments for a given item.

The Word Requirement, From The Dictionary

- **Something wanted or necessary.**
- **Something essential to the existence or occurrence of something else.**
- **A necessary character-istic or attribute of some thing (or item).**

ITEM

In Writing a Specification, What Is the Target?



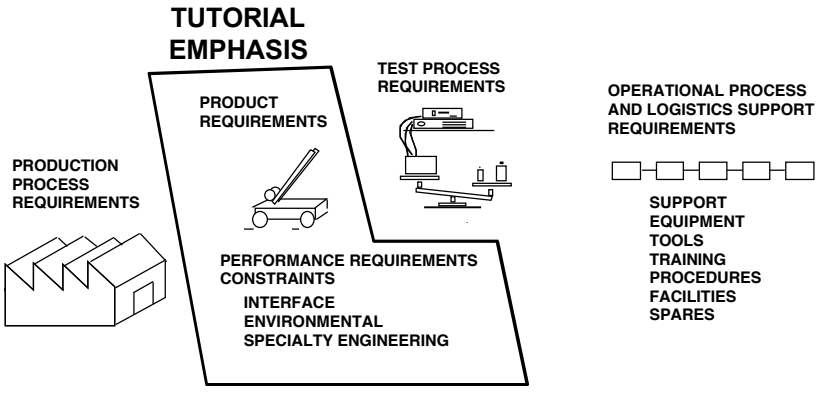
How to Hit the Target of Minimized Completeness

- **Every performance requirement traceable to a model from which it was derived**
- **Every external interface for the item identified and defined in interface requirements in the specification (unless ICD applied)**
- **Every specialty engineering discipline that has been mapped to the item is included in the specification**
- **Every environmental influence defined in the appropriate model (system, end item, component) mapped to appropriate specification content.**
- **Every requirement in the specification traceable to a parent item specification requirement (ideally applies to the system specification relative to user requirements as well).**
- **Requirements are quantified as appropriate to the statement.**
- **Requirements are validated (risks understood and mitigated).**

Product Requirements Types

- **Hardware**
  - **Performance**
  - **Constraints**
    - **Interface**
    - **Specialty Engineering**
    - **Environmental**
- **One view of software requirements**
  - **Functional**
  - **Non-Functional**
  - **Quality**
- **My view of software types**
  - **Same as systems and hardware**

Requirements Types



All of these requirements must be identified before product and process detailed design work is started and they must be mutually consistent.

Attributes of a Good Requirement

- Achievable (validated)
- Quantified
- Achievable (validated)
- Verifiable/testable
- Unambiguous
- Complete (covers all cases)
- Performance specification
  - Design independent
  - What, not how
- Detail specification
  - Design dependent

Some Good Examples

Frequency coverage. Item frequency coverage shall be 225.0 to 399.9 Megahertz inclusive in tenth Megahertz steps.

Weight. Item weight shall be less than or equal to 240 pounds.

Range. Maximum achievable range shall be greater than or equal to 5,000 nautical miles while recognizing a fuel loading safety margin of 10% or more.

Range. Maximum range shall be greater than or equal to 2,500 nautical miles.

Reliability. Item MTBF shall be greater than or equal to 10,000 hours.

Some Bad Examples

- The screens used in the system shall be designed in a user friendly manner.
- Item weight shall not be greater than 153 pounds.
- Aircraft shall identify their position within 1000 feet of actual along and across track position using Loran C.
- Brakes shall function smoothly and stop the train in a safe distance.
- There shall be no hailstorms in the path of the aircraft.
- On most days, transmitter power output should be 100 watts.
- Go fast.
- Item shall work well and last a long time.
- Any favorites from your past?

Specifications Are Full of Sentences

- THESE SENTENCES SHOULD BE WRITTEN IN THE SIMPLEST POSSIBLE WAY
- THE SUBJECT IS THE ITEM CHARACTERISTIC ABOUT WHICH THE REQUIREMENT IS WRITTEN
- VERB SHALL CLEARLY CALLS FOR COMPLIANCE
-

SUBJECT	VERB	OBJECT	
Item memory margin	shall be	greater than	100%.
		RELATION	VALUE

The Verb

- **SHALL** Mandatory
- **WILL** Contractor intends to perform
- **SHOULD** Recommended, Desirable

The Subject

- **Writing requirements is easy**
- **The difficult job is knowing <u>what to write them about</u> - the subject of the sentence**
- **That is why we model the problem space**

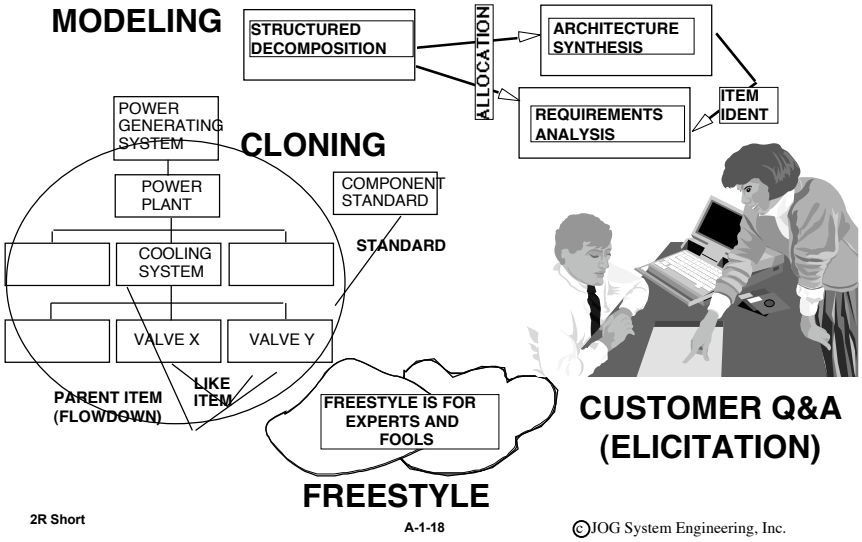
Good Examples In Primitive Form

weight ≤ 240 pounds

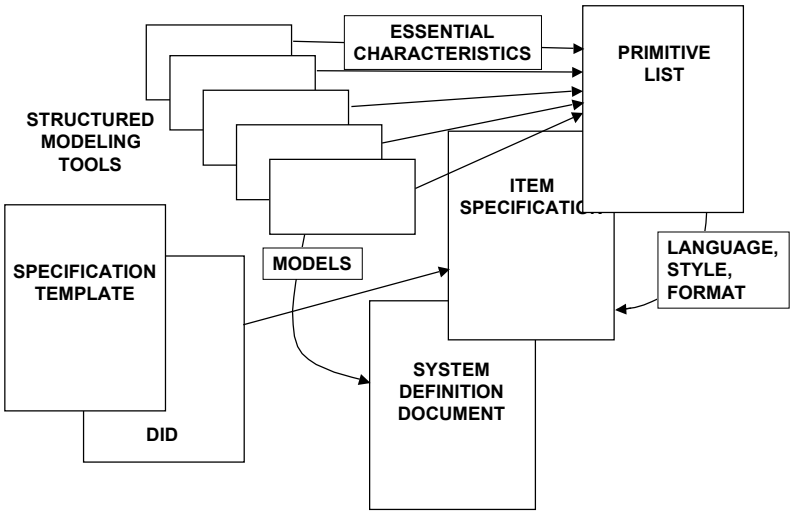
range ≥ 5,000 nautical miles

MTBF ≥ 10,000 hours

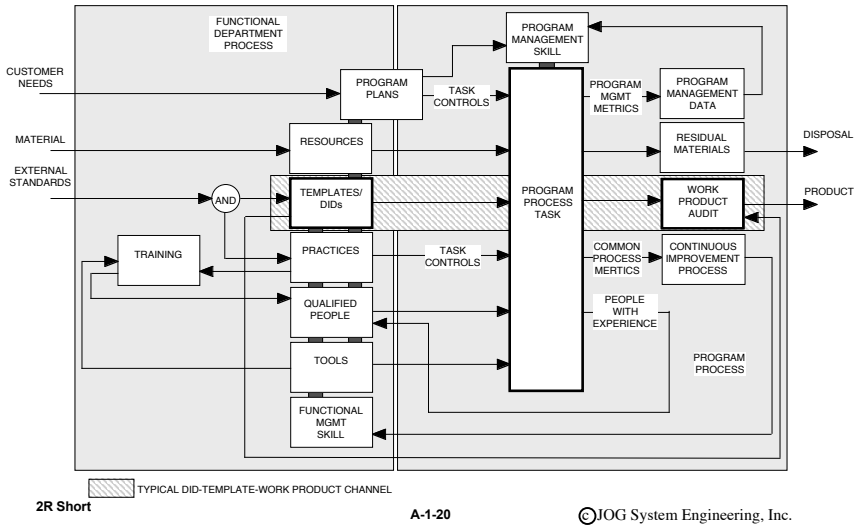
Requirements Analysis Strategies



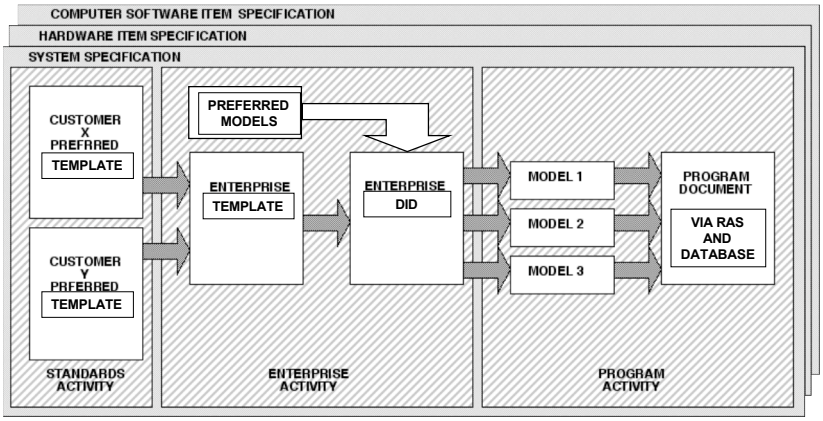
A Foolproof Search For Subjects



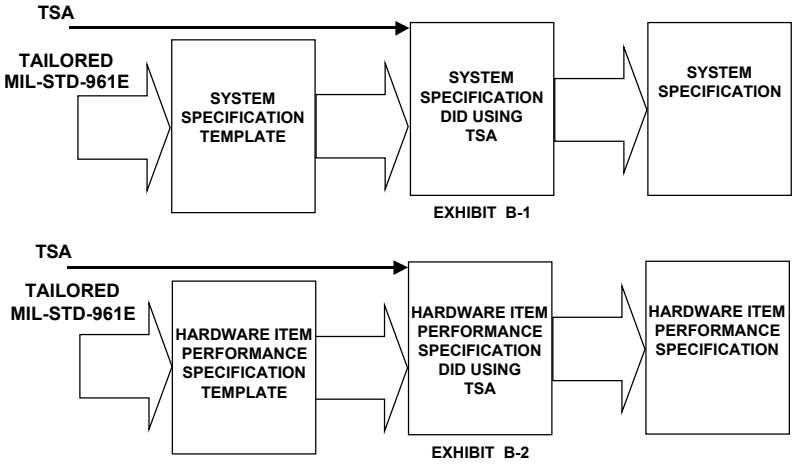
General Program Task-Resource Relationships



Work Product Development Suite
Specification Case



Document Progressions



Document Progressions

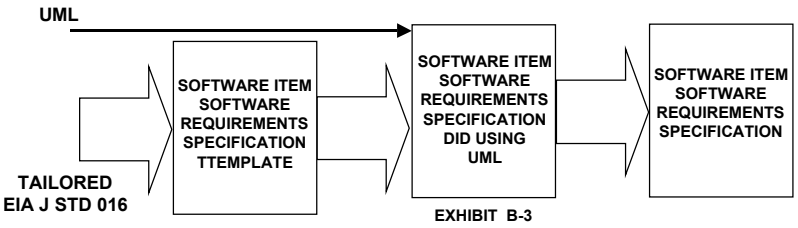
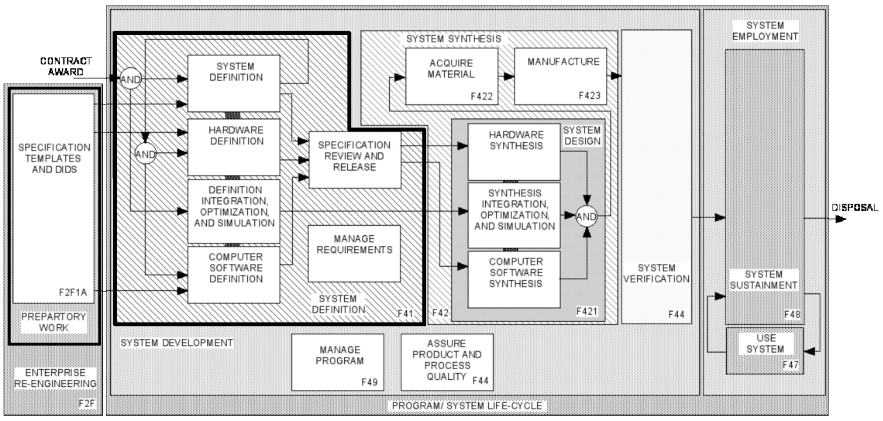


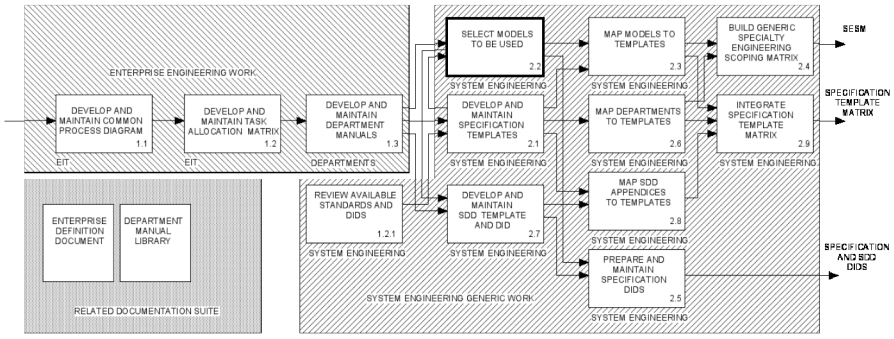
EXHIBIT B-3

System Development Process Overview

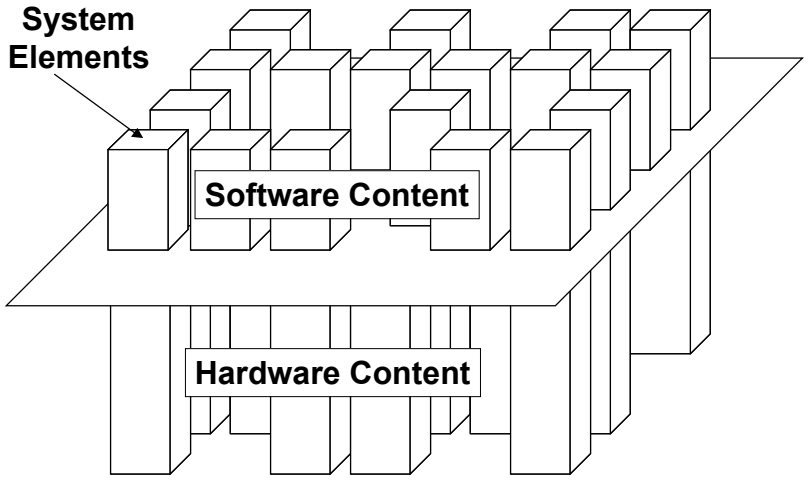


Covered in this tutorial

Preparatory Steps



A Single Model Will Not Work



Hardware and Systems Analysis Models

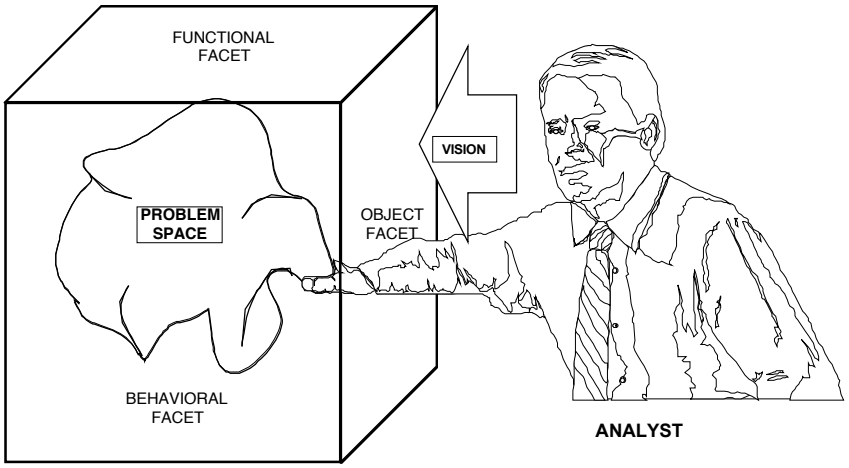
- Traditional structured analysis
  - Functional analysis
    - Functional flow diagramming
    - Enhanced functional flow diagramming (CORE)
    - Behavioral diagramming (RDD/IPO)
    - IDEF 0 (SADT)
    - Process flow analysis
    - Hierarchical functional analysis
  - Constraints analysis
- State diagramming
- SysML

Computer Software Structured Analysis Models

- Process-oriented analysis
  - Flow charting
  - Modern Structured Analysis (Yourdon-Demarco)
  - Modern Structured Analysis (Hatley-Pirbhai)
- Data-oriented analysis
  - Table normalizing
  - IDEF-1X
- Object-oriented analysis
  - Early models
  - Unified Modeling Language (UML)
- DoD architecture framework (DoDAF)

NDIA

Structured View of a Problem Space



2R Short

A-1-29

NDIA

Structured Analysis Methods Comparison

MULTI-FACETED APPROACHES	PRODUCT ENTITY FACET	FUNCTIONAL FACET	BEHAVIOR FACET
TRADITIONAL STRUCTURED ANALYSIS	PRODUCT ENTITY BLOCK DIAGRAM	FUNCTIONAL ○ FLOW DIAGRAM	SCHEMATIC BLOCK DIAGRAM
MODERN STRUCTURED ANALYSIS	HIERARCHICAL DIAGRAM	DATA FLOW ○ DIAGRAM	P SPEC, STATE DIAGRAM
EARLY OBJECT-ORIENTED ANALYSIS	CLASS AND ○ OBJECT DIAGRAM	DATA FLOW DIAGRAM	STATE DIAGRAM
UML	CLASS/OBJECT , COMPONENT, & DEPLOYMENT DIAGRAMS	USE CASES AND ACTIVITY DIAGRAMS ○	STATE, SEQUENCE, AND COMMUNICATION DIAGRAMS

○ UNPRECEDENTED ANALYTICAL ENTRY FACET

2R Short

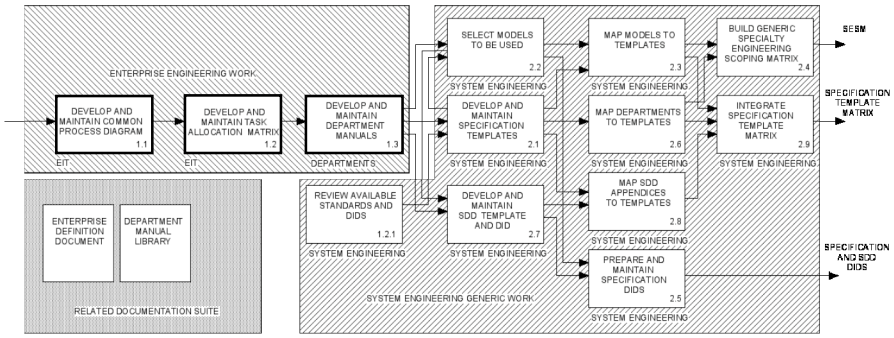
A-1-30

Model Suggestions for Today

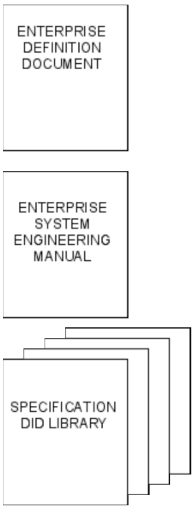
SPECIFICATION TYPE	MODEL SUGGESTED
System Specification	Traditional Structured Analysis
Hardware Performance Specification	Traditional Structured Analysis
Software Performance Specification, General	Unified Modeling Language (UML)
Software Performance Specification, Database	IDEF-1X
Software Performance Specification, DoD IS	DoDAF

But, be prepared to move to the use of SysML coupled with UML and the eventual merge of the two into a more fully integrated common modeling method.

Preparatory Steps



Enabling Documentation

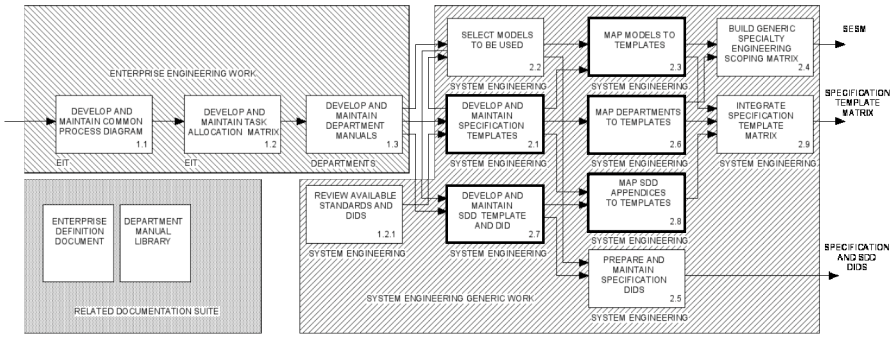


EDD defines the enterprise common process to be applied on all programs, the functional departments, and their process responsibilities which include preparing a department manual covering all common process work allocated to the department.

The system engineering department department manual that defines the way system engineering will be applied on all programs including requirements analysis and specification management.

All functional departments should define templates and data item descriptions for all work products that are prepared as documents. One example of these artifacts is a set of specification data item descriptions that are coordinated with a particular template and a method of acquiring the content through modeling.

Preparatory Steps



MIL-STD-961E Template

1	Scope
2	Applicable Documents
3	Requirements
3.1	Functional and Performance Rqmts.
3.1.1	Missions
3.1.2	Threat
3.1.3	Required States and Modes
3.1.4	Entity Capability Requirements
3.1.5	Reliability
3.1.6	Maintainability
3.1.7	Deployability
3.1.8	Availability
3.1.9	Environmental Conditions
3.1.10	Transportability
3.1.11	Materials and Processes
3.1.12	Electromagnetic Radiation
3.1.13	Nameplates and Product Markings
3.1.14	Producibility
3.1.15	Interchangeability
3.1.16	Safety
3.1.17	Human Factors Engineering
3.1.18	Security and Privacy
3.1.19	Computer Resource Requirements
3.1.20	Logistics
3.1.21	Personnel and Training
3.1.22	Requirements Traceability
3.2	Interface Requirements
3.2.1	GFP Interfaces
3.2.2	External Interface Requirements
3.3	Design and Construction
3.3.1	Production Drawings
3.3.2	Software Design
3.3.3	Workmanship
3.3.4	Standards of Manufacture
3.3.5	Process Definition
3.3.6	Material Definition
3.4	Precedence and Criticality of Rqmts.
4	Verification
4.1	Methods of Verification
4.2	Classes of Verification
4.3	Inspections
5	Packaging
6	Notes

Recommended Template With Map for Traditional Structured Analysis

PARA	TITLE	MODEL	DEPT	APP
1.	SCOPE	DID	2100	
2.	APPLICABLE DOCUMENTS	DID	2100	
3.	REQUIREMENTS	DID	2100	
3.1	Functional and performance requirements	DID	2100	
3.1.1	Missions	Mission Analysis	2100	
3.1.2	Threat	Threat Analysis	2100	
3.1.3	Required states and modes	DID	2100	
3.1.3.1	Functional analysis	DID	2100	A
3.1.3.2	Subordinate entities	DID	2100	C
3.1.3.3	Interface relationships	DID	2100	D
3.1.3.4	Specialty engineering requirements	DID	2100	E
3.1.3.5	Environmental model	DID	2100	B
3.2	Entity capability requirements	Functional Analysis	2100	A
3.2.m	Capability m	Functional Analysis	2100	A
3.2.m.n	Capability m, requirement n	Functional Analysis	2100	A
3.3	Interface requirements	N-Square Diagram	2100	D
3.3.1	External interface requirements	N-Square Diagram	2100	D
3.3.1.m	External interface m	N-Square Diagram	2100	D
3.3.1.m.n	External interface m, requirement n	N-Square Diagram	2100	D
3.3.2	Internal interface requirements	N-Square Diagram	2100	D
3.3.2.m	Internal interface m	N-Square Diagram	2100	D
3.3.2.m.n	Internal Interface m, requirement n	N-Square Diagram	2100	D

Modeling Methods Map

Recommended Template With Map for Traditional Structured Analysis

PARA	TITLE	MODEL	DEPT	APP
1.	SCOPE	DID	2100	
2.	APPLICABLE DOCUMENTS	DID	2100	
3.	REQUIREMENTS	DID	2100	
3.1	Functional and performance requirements	DID	2100	
3.1.1	Missions	Mission Analysis	2100	
3.1.2	Threat	Threat Analysis	2100	
3.1.3	Required states and modes	DID	2100	
3.1.3.1	Functional analysis	DID	2100	A
3.1.3.2	Subordinate entities	DID	2100	C
3.1.3.3	Interface relationships	DID	2100	D
3.1.3.4	Specialty engineering requirements	DID	2100	E
3.1.3.5	Environmental model	DID	2100	B
3.2	Entity capability requirements	Functional Analysis	2100	A
3.2.m	Capability m	Functional Analysis	2100	A
3.2.m.n	Capability m, requirement n	Functional Analysis	2100	A
3.3	Interface requirements	N-Square Diagram	2100	D
3.3.1	External interface requirements	N-Square Diagram	2100	D
3.3.1.m	External interface m	N-Square Diagram	2100	D
3.3.1.m.n	External interface m, requirement n	N-Square Diagram	2100	D
3.3.2	Internal interface requirements	N-Square Diagram	2100	D
3.3.2.m	Internal interface m	N-Square Diagram	2100	D
3.3.2.m.n	Internal Interface m, requirement n	N-Square Diagram	2100	D

Modeling Methods Map

Functional Department Map

Recommended Template With Map for Traditional Structured Analysis

PARA	TITLE	MODEL	DEPT	APP
1.	SCOPE	DID	2100	
2.	APPLICABLE DOCUMENTS	DID	2100	
3.	REQUIREMENTS	DID	2100	
3.1	Functional and performance requirements	DID	2100	
3.1.1	Missions	Mission Analysis	2100	
3.1.2	Threat	Threat Analysis	2100	
3.1.3	Required states and modes	DID	2100	
3.1.3.1	Functional analysis	DID	2100	A
3.1.3.2	Subordinate entities	DID	2100	C
3.1.3.3	Interface relationships	DID	2100	D
3.1.3.4	Specialty engineering requirements	DID	2100	E
3.1.3.5	Environmental model	DID	2100	B
3.2	Entity capability requirements	Functional Analysis	2100	A
3.2.m	Capability m	Functional Analysis	2100	A
3.2.m.n	Capability m, requirement n	Functional Analysis	2100	A
3.3	Interface requirements	N-Square Diagram	2100	D
3.3.1	External interface requirements	N-Square Diagram	2100	D
3.3.1.m	External interface m	N-Square Diagram	2100	D
3.3.1.m.n	External interface m, requirement n	N-Square Diagram	2100	D
3.3.2	Internal interface requirements	N-Square Diagram	2100	D
3.3.2.m	Internal interface m	N-Square Diagram	2100	D
3.3.2.m.n	Internal Interface m, requirement n	N-Square Diagram	2100	D

Modeling Methods Map

Functional Department Map

Model Artifact Capture Map

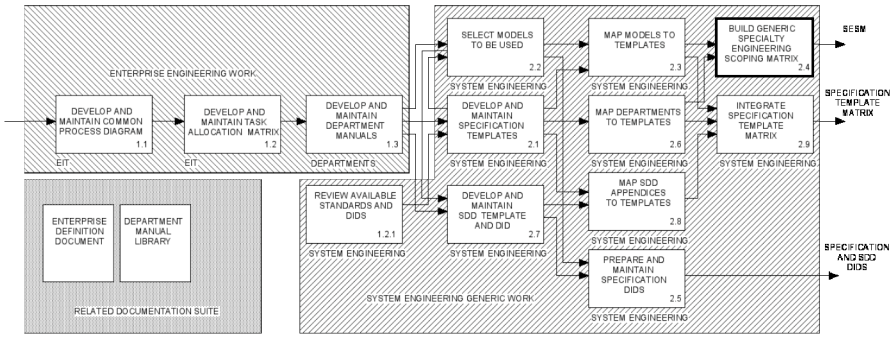
Prepare and Maintain DIDs

- **The DID follows the template format**
- **The recommended DID is focused on a particular modeling approach**
- **The DID tells how to create a specification using that template with a particular modeling approach**

Here is a sample DID for a system specification using TSA.

Document

Preparatory Steps



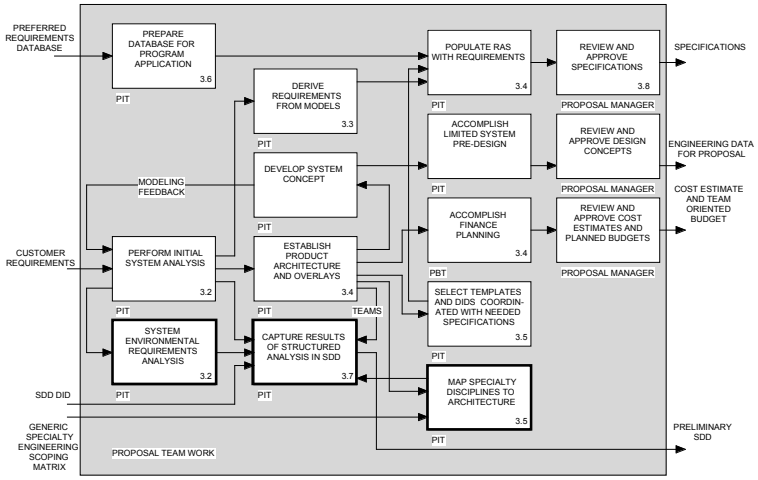
Generic Specialty Engineering Scoping Matrix

	SPECIALTY DISCIPLINE	DEPT	PARA	A	A1	A2	A3	A4
H1	Reliability	D2164	3.4.1					
H2	Maintainability	D2164	3.4.2					
H3	Availability	D2164	3.4.3					
H4	Deployability and transportability	D2164	3.4.4					
H5	Logistics	D2311	3.4.5					
H6	Maintenance	D2311	3.4.5.1					
H7	Interchangeability	D2113	3.4.5.2					
H8	Supply	D2311	3.4.5.3					
H9	Facilities and facility equipment	D2311	3.4.5.4					
HA	Personnel	D2313	3.4.5.5					
HB	Training	D2313	3.4.5.6					
HC	Safety	D2165	3.4.6					
HD	Human factors engineering	D2165	3.4.7					
HE	Security and privacy		3.4.8					
HF	Electromagnetic compatibility	D2136	3.4.9					
HG	Lightning protection	D2136	3.4.10					
HH	Producibility		3.4.11					
HI	Affordability	D2168	3.4.12					
HJ	Computer resource requirements		3.4.13					
-	Design and construction		3.4.14					
HK	Quality Engineering	D5100	3.4.14.1					
HL	Parts, materials, and processes	D2167	3.4.14.2					

Generic Specialty Engineering Scoping Matrix

	SPECIALTY DISCIPLINE	DEPT	PARA	A	A1	A2	A3	A4
HM	Workmanship	D5100	3.4.14.3					
HN	Nameplates and product markings	D2113	3.4.14.4					
HO	Serialization	D2113	3.4.14.5					
HP	Mass properties	D2124	3.4.14.6					
HQ	Structural properties	D2124	3.4.14.7					
HR	Shock and vibration		3.4.14.8					
HS	Earthquake survivability	D2123	3.4.14.9					
HT	Aerodynamics	D2144	3.4.14.10					
HU	Thermodynamics	D2143	3.4.14.11					
HV	Chemical, electrical, and mechanical properties		3.4.14.12					
HW	Stability		3.4.14.13					
HX	Coatings and Corrosion Control	D2167	3.4.14.14					

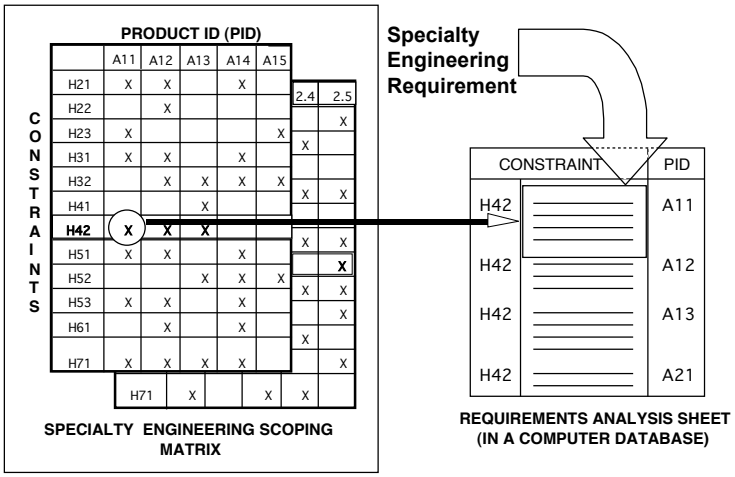
Proposal Team Specification Actions



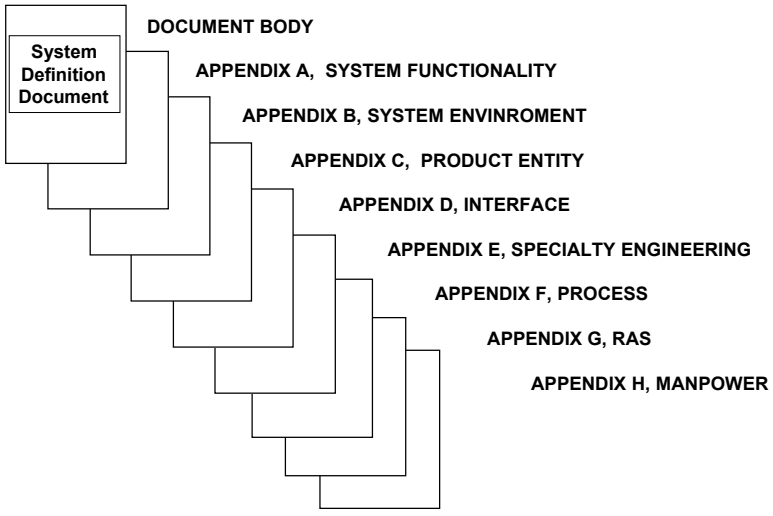
Environmental Requirements

- **System**
  - **Identify spaces within which the system will have to function**
  - **Select standards covering those spaces**
  - **For each standard, select parameters that apply**
  - **Tailor the range of selected parameters**
- **End item**
  - **Build three dimensional model of end items, physical processes, and process environments**
  - **Extract item environments**
- **Component**
  - **Zone end item into spaces of common environmental characteristics**
  - **Map components to zones**
  - **Components inherit zone environmental requirements**

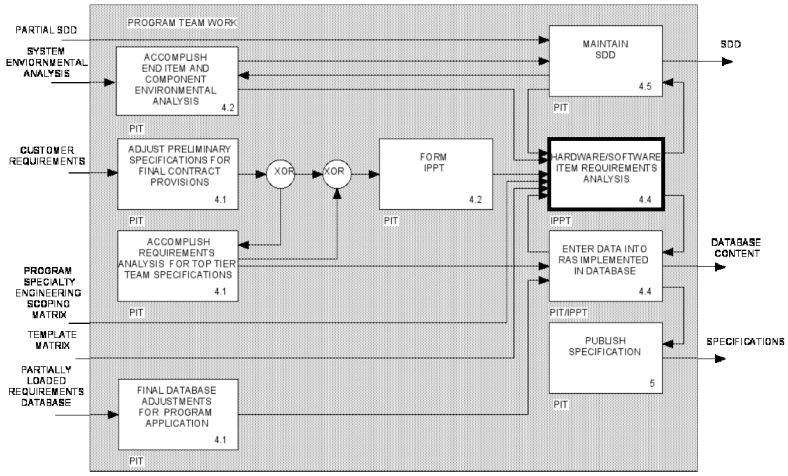
Specialty Engineering Scoping Matrix Applied to Program



Configuration Control the Models



On To Program Work



2R Short

NDIA
NATIONAL DEFENSE INDUSTRIAL ASSOCIATION

JOG SYSTEM ENGINEERING, INC.
GRAND SYSTEMS DEVELOPMENT
TUTORIAL PROGRAM

AN EFFECIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

TRADITIONAL STRUCTURED ANALYSIS

VERSION 10.1

NDIA
NATIONAL DEFENSE INDUSTRIAL ASSOCIATION

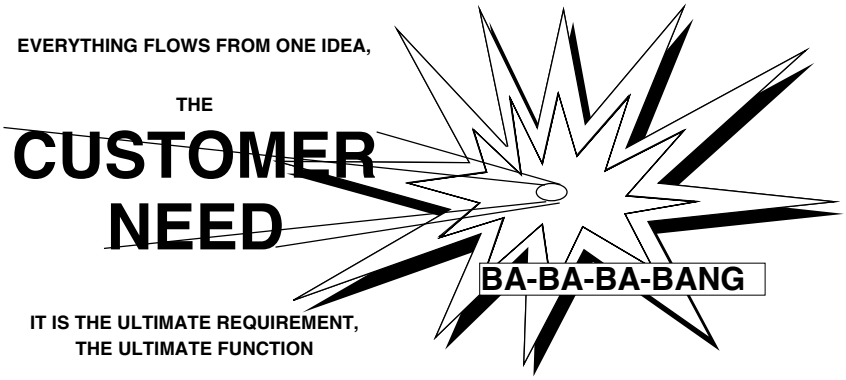
The Big Bang Theory Of System Development

THE TRADITIONAL APPROACH

EVERYTHING FLOWS FROM ONE IDEA,

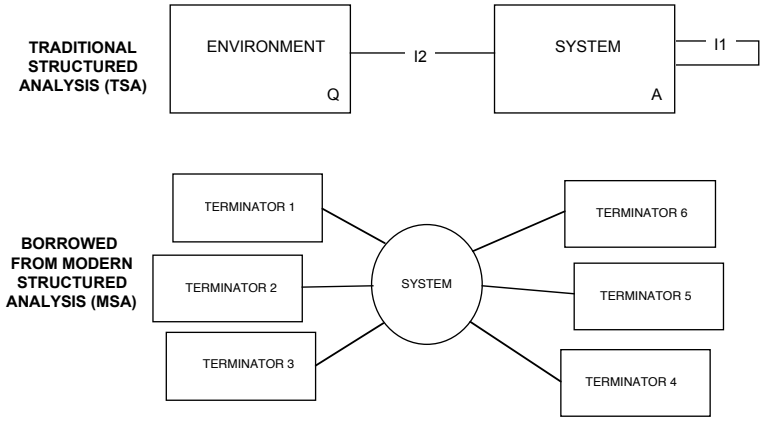
THE

CUSTOMER NEED

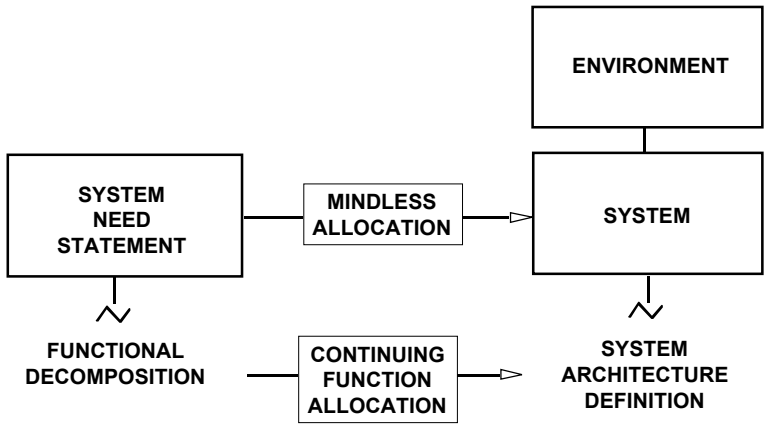


IT IS THE ULTIMATE REQUIREMENT,
THE ULTIMATE FUNCTION

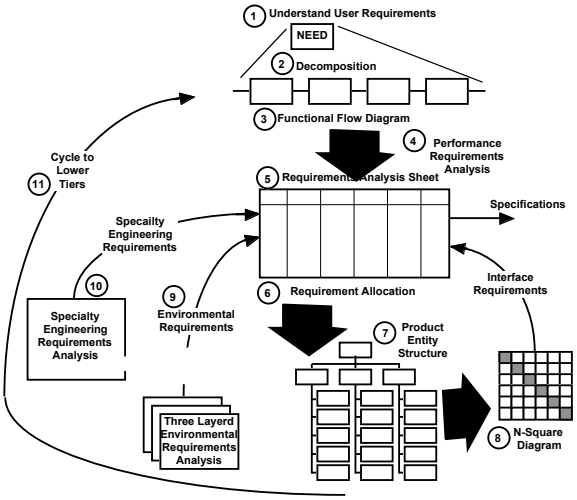
Two Top-Level Views of a System



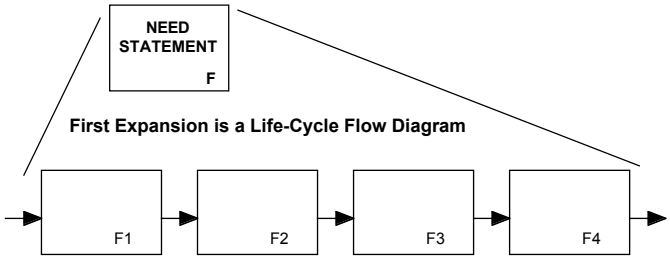
The Beginning Of Functional Decomposition



Traditional Structured Analysis Model Overview



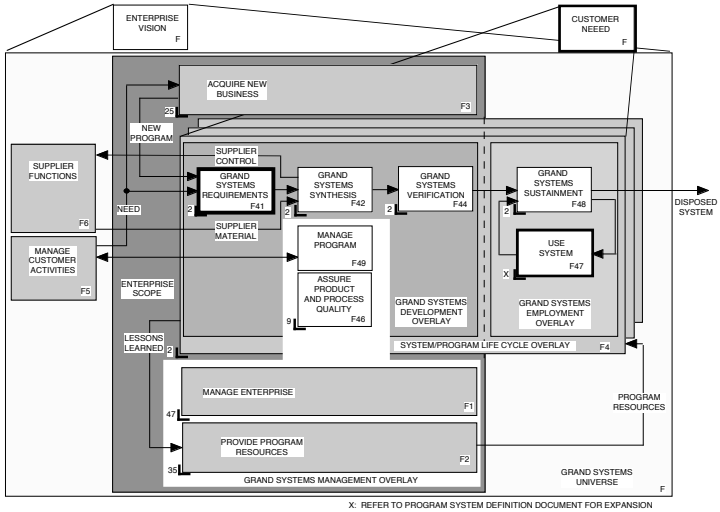
The Ultimate Function and Its First Expansion



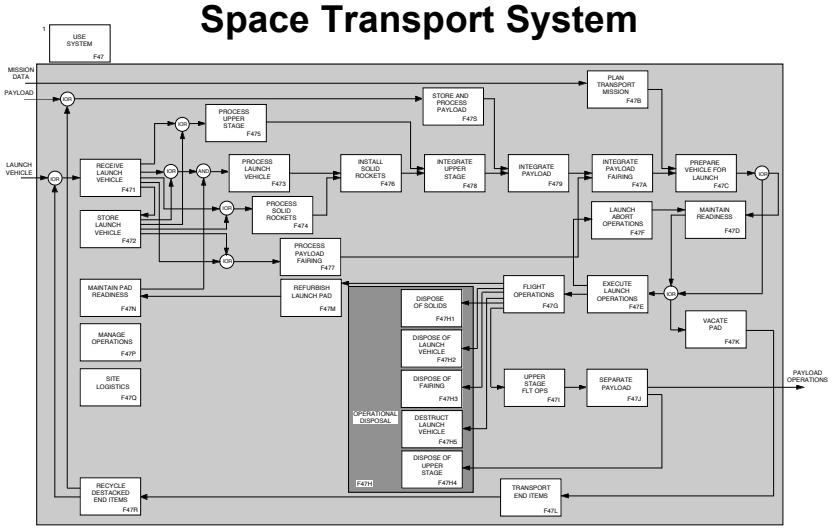
Alternative functional analysis techniques

Enhanced functional flow block diagramming (CORE)
Behavioral diagramming (RDD)
IDEF-0

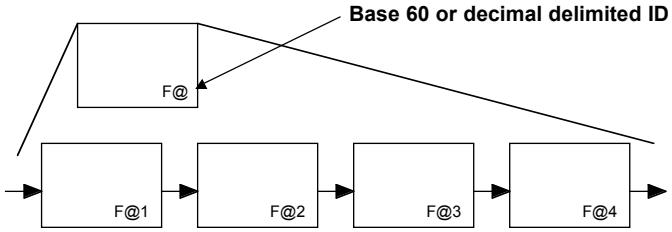
Example of a Life Cycle Model



Use System Expansion Example
Space Transport System



Continued Function Decomposition



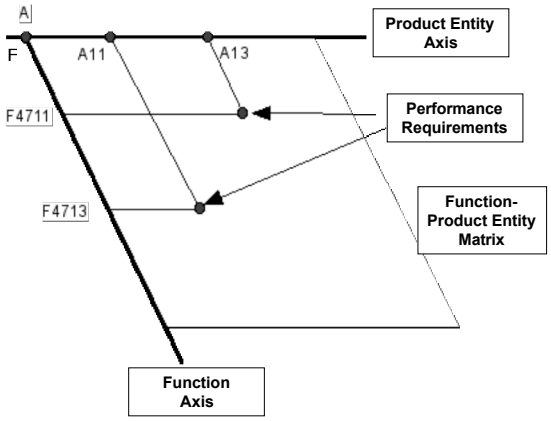
An orderly exposure of needed functionality moving from the known to the unknown, from simple to the complex, from the top to the bottom.

Performance Requirements Analysis and the RAS

FUNCTION		PRODUCT ENTITY		REQUIREMENTS	
MID	NAME	PID	NAME	RID	STATEMENT
F123	Provide Thrust	A12	Engine	WE89FS	Thrust > 10,000 pounds at sea level

Exposing what the system must do and how well it must do it encouraging identification of all essential characteristics and avoidance of unnecessary characteristics.

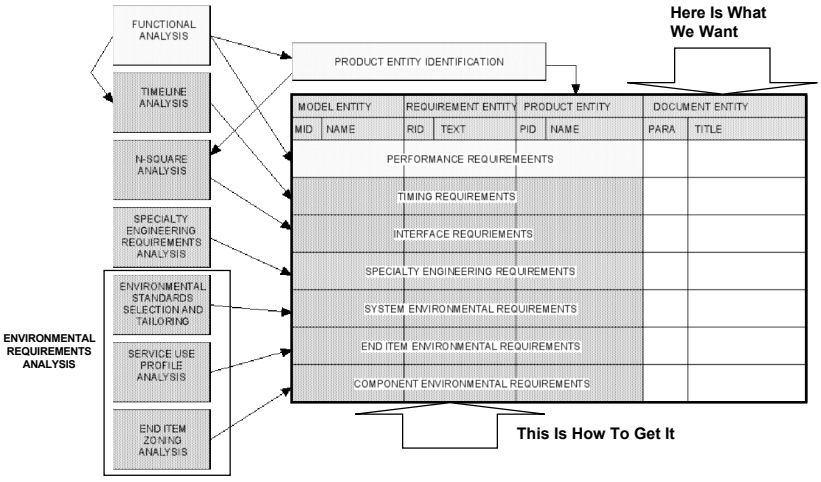
The Function-Product Entity Plane



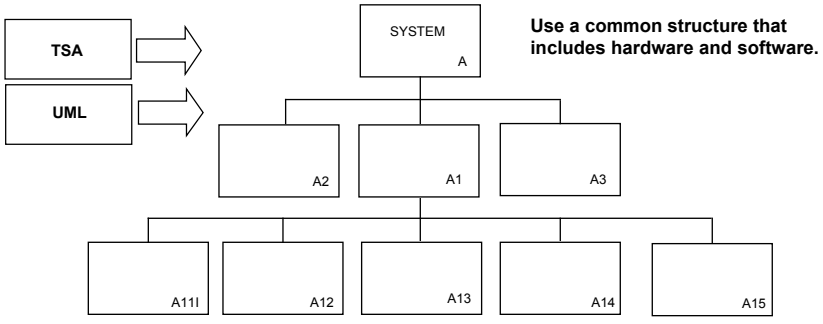
Functional Analysis Alternatives

- **IDEF 0**
  - **A variation on SADT**
- **Behavioral Diagramming**
  - **From Ascent Logic's RDD**
  - **Based on IPO**
- **Enhanced Functional Flow Block Diagramming**
  - **Employed in Vitech's CORE**
- **Hierarchical Functional Block Diagramming**

Requirements Capture Using the RAS-Complete Format

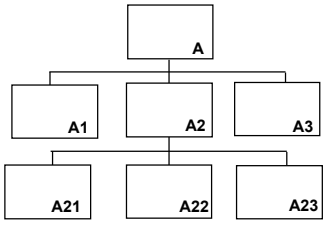


Product Entity Structure

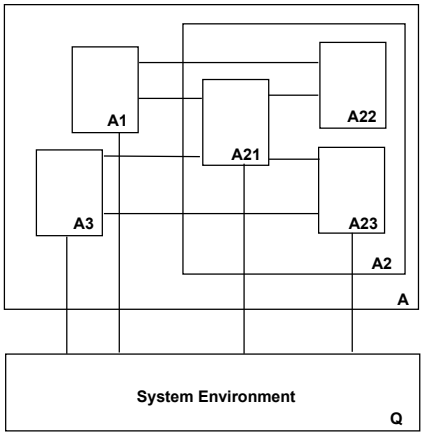


Systems Consist of Things and Relationships

Product Entity Structure Diagram



System Relationships Block Diagram

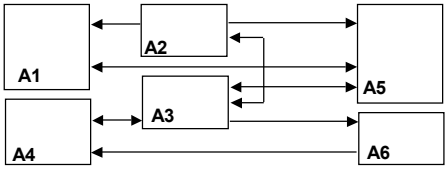


Organizing For Interface Development

- **Decompose needed functionality and allocate to product entities**
- **Map product entities to responsible development organizations**
  - **Create cross-functional integrated product and process teams (IPPT)**
  - **Assign principal engineers for lowest tier responsibilities on teams (everything has someone responsible)**
- **Establish clear rules for interface development responsibility**
  - **Identify needed interfaces as a function of how functionality was allocated to entities**
  - **Analyze product entity pair relationships using n-square diagrams**
  - **Partition interface into subsets as a function of product entity principal views**
  - **Assign interface responsibility to product entity principal engineers as a function of a receiving terminal rule (if you need an interface you must come forward)**
  - **System engineering manage the aggregate external and inter-team interface sets applying a lowest common team integration concept**
- **Minimize external (cross-organizational) interface at all levels, iterating product entity structure and/or development organization responsibilities to do so, if necessary, then apply system engineering integration resources to that which remains**

Two Interface Definition Models

SCHEMATIC BLOCK DIAGRAMMING



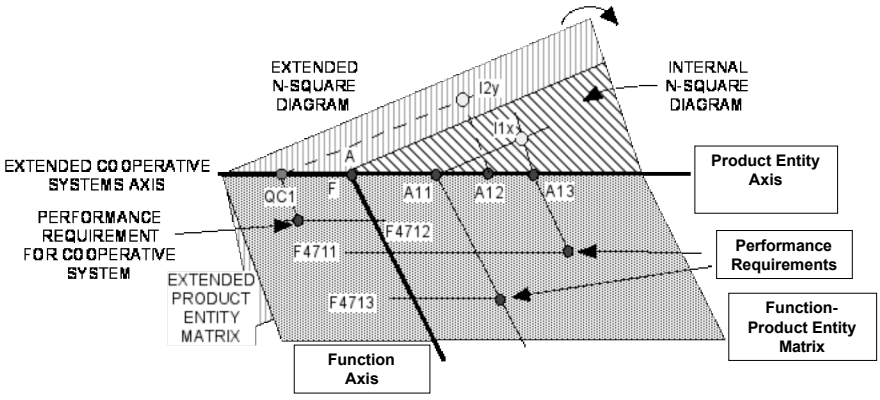
- Lines define interfaces
- Blocks are objects only from the product entity structure

N-SQUARE DIAGRAMMING

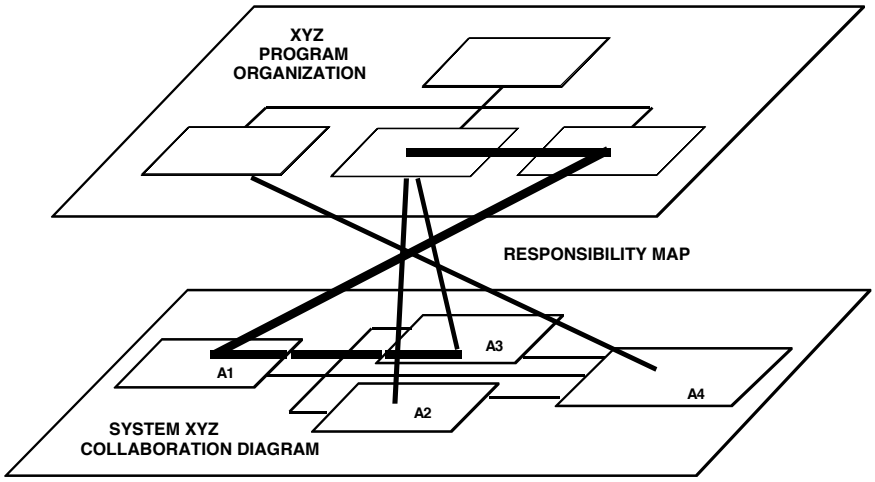
A1				X	
X	A2	X		X	
	X	A3	X	X	X
		X	A4		
X		X		A5	
			X		A6

- Marked intersections define interfaces
- Diagonal blocks are objects only from product entity block diagram
- Apparent ambiguity reflects directionality

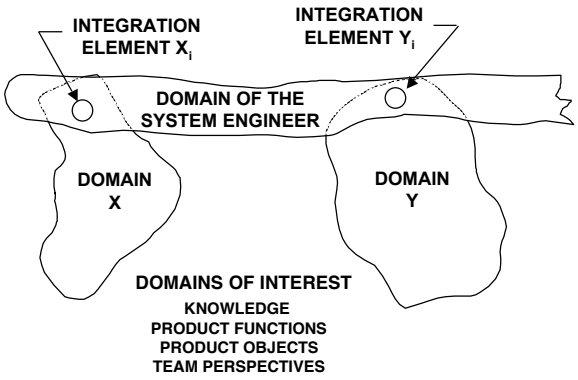
Interface Requirements Derivation Geometrical View



Development Often Fails at the Cross-organizational Interfaces

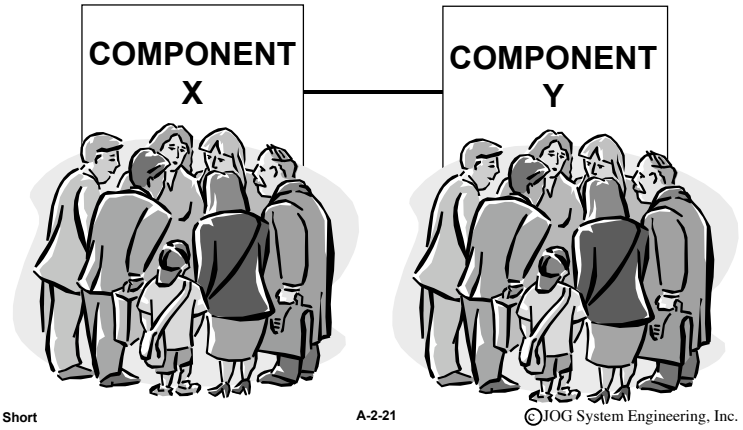


Interface Integration Focus



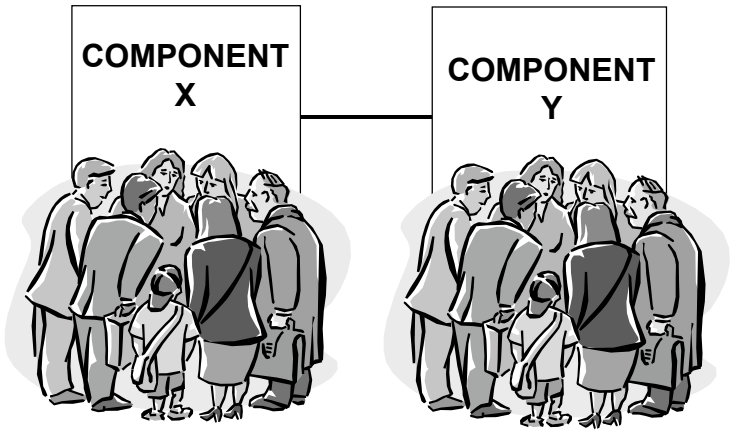
The Fundamental Problems in Interface Work

There is a one-to-one correspondence between teams and components. There is a one- to-two correspondence between teams and interfaces.



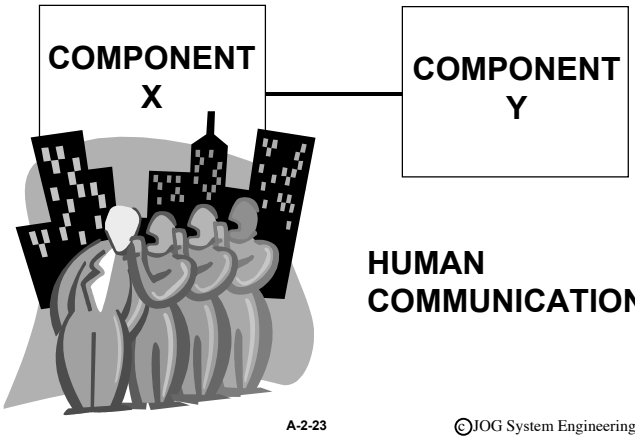
The Fundamental Problems in Interface Work

We tend to focus inwardly

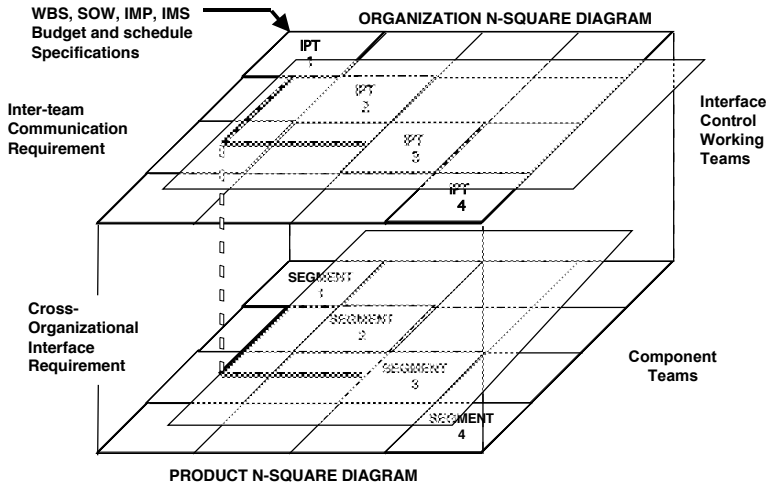


The Fundamental Problems in Interface Work

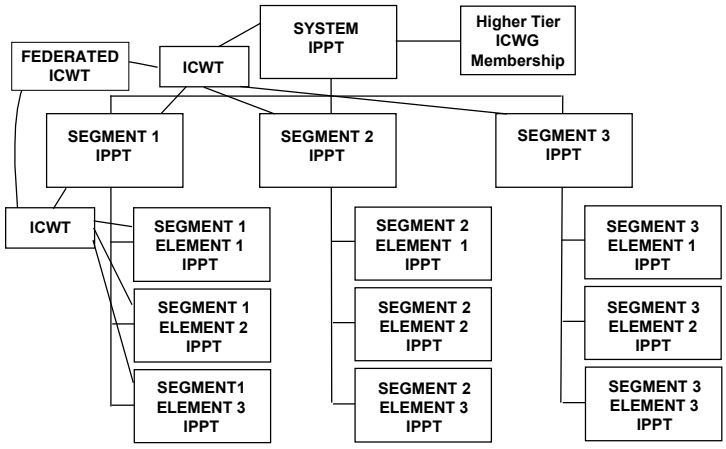
We are dependent on the worst interface on planet Earth in the development of interfaces.



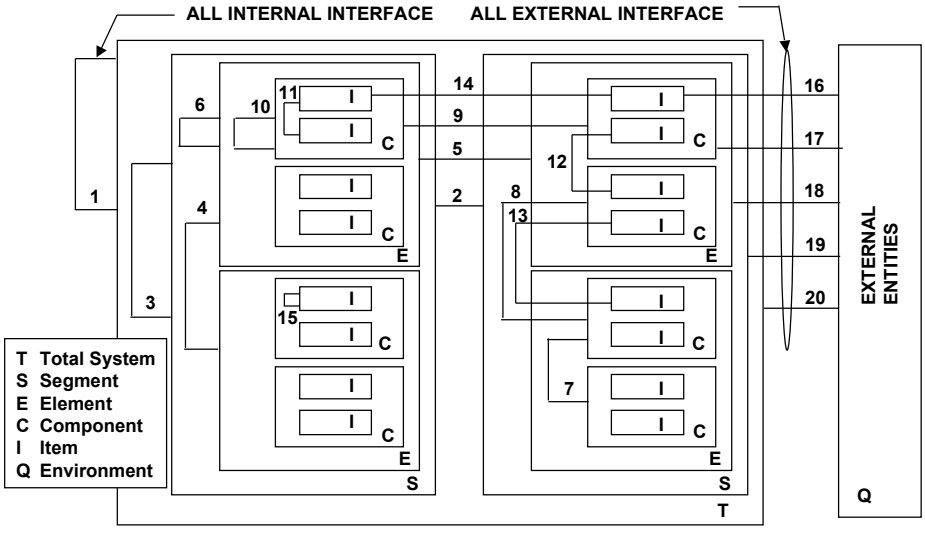
Benefits Of Product Team Organization



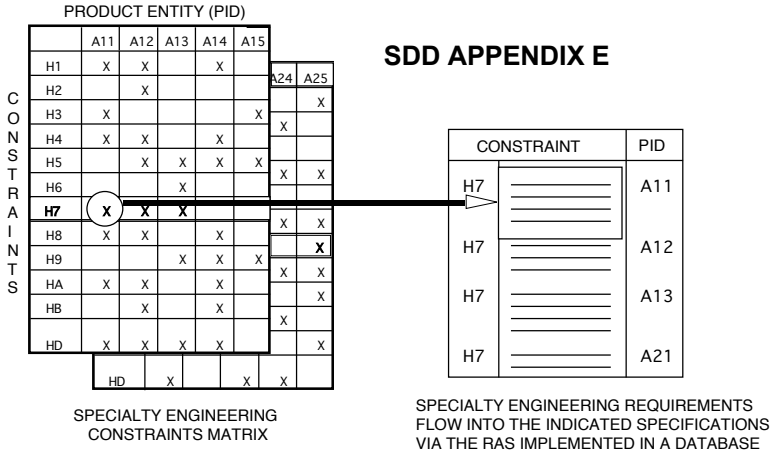
Product Entity and Interface Responsibility



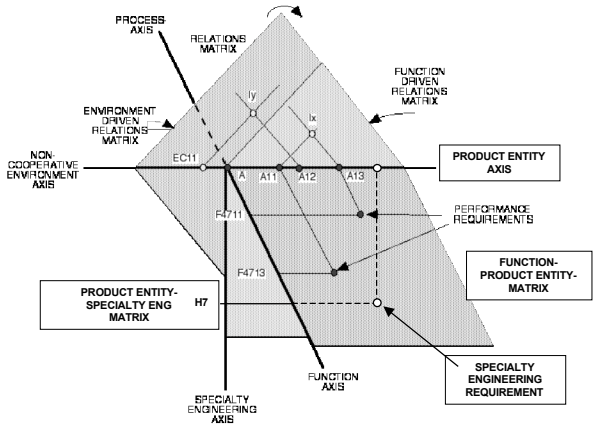
Lowest Common Team Concept



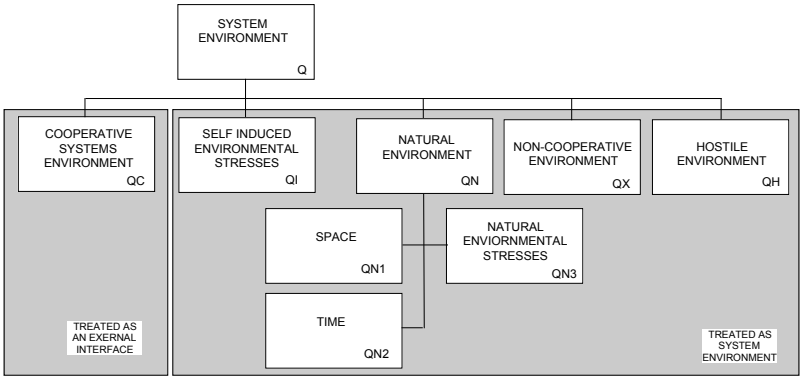
Specialty Engineering Identification of Constraints



Specialty Engineering Plane Added



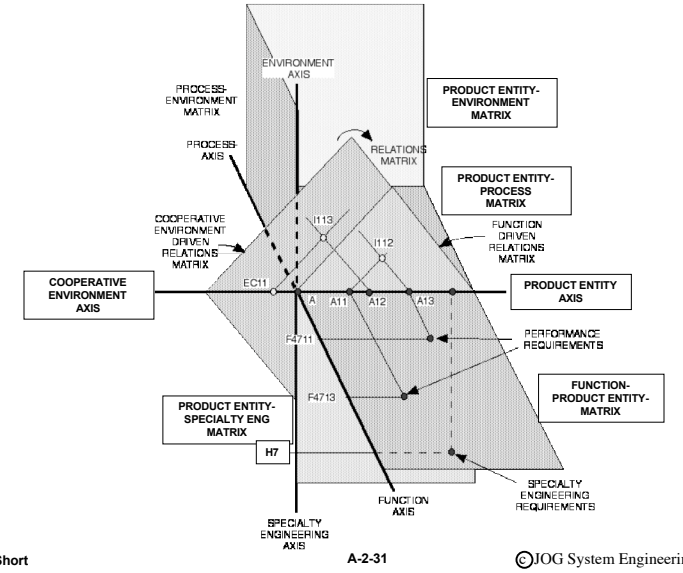
Environment Subsets



Environmental Requirements

- **System**
  - **Identify spaces within which the system will have to function**
  - **Select standards covering those spaces**
  - **For each standard, select parameters that apply**
  - **Tailor the range of selected parameters**
- **End item**
  - **Build three dimensional model of end items, physical processes, and process environments**
  - **Extract item environmental requirements**
- **Component**
  - **Zone end item into spaces of common environmental characteristics**
  - **Map components to zones**
  - **Components inherit zone environmental requirements**

Environmental Planes Added

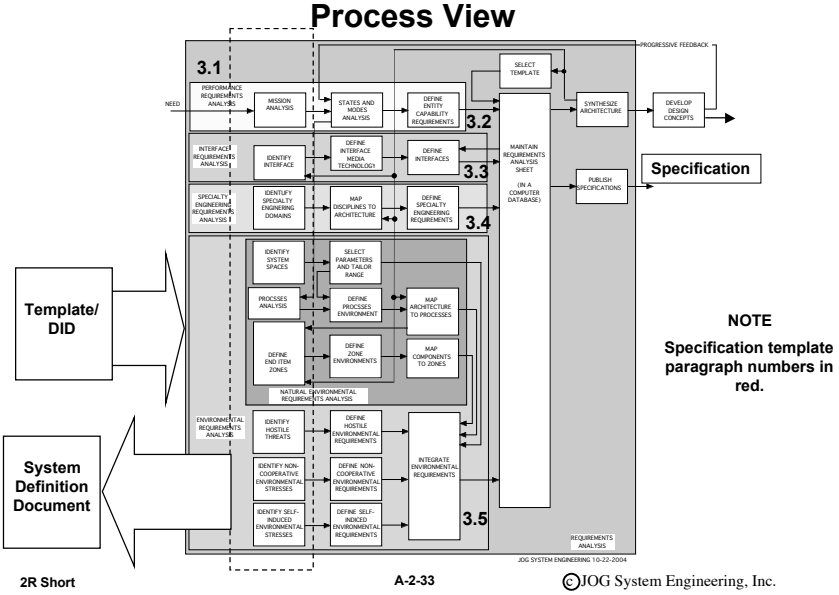


RAS Complete In Tabular Form

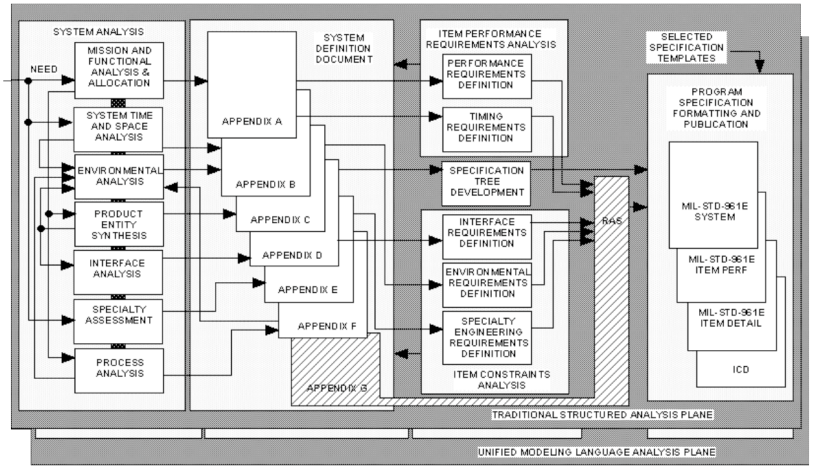
	MODEL ENTITY MID	MODEL ENTITY NAME	REQUIREMENT ENTITY RID	REQUIREMENT	PRODUCT ENTITY PID	ITEM NAME	DOCUMENT ENTITY PARA	TITLE
Performance Requirements	F47	Use System			A	Product System		
	F471	Deployment Ship Operations			A	Product System		
	F4711	Store Array Operationally	XR67	Storage Volume < 10 ISO Vans	A1	Sensor Subsystem		
Specialty Engineering	H	Specialty Engineering Disciplines			A	Product System		
	H11	Reliability	EW34	Failure Rate < 10 x 10-6	A1	Sensor Subsystem	3.1.5	Reliability
	H11	Reliability	RG31	Failure Rate < 3 x 10-6	A11	Cable	3.1.5	Reliability
	H11	Reliability	FYH4	Failure Rate < 5 x 10-6	A12	Sensor Element	3.1.5	Reliability
	H11	Reliability	G8R4	Failure Rate < 2 x 10-6	A13	Pressure Vessel	3.1.5	Reliability
	H12	Maintainability	6GHU	Mean Time to Repair < 0.2 Hours	A1	Sensor Subsystem	3.1.6	Maintainability
	H12	Maintainability	U9R4	Mean Time to Repair < 0.4 Hours	A11	Cable	3.1.6	Maintainability
	H12	Maintainability	J897	Mean Time to Repair < 0.2 Hours	A12	Sensor Element	3.1.6	Maintainability
	H12	Maintainability	9D7H	Mean Time to Repair < 0.1 Hours	A13	Pressure Vessel	3.1.6	Maintainability
Interface	I	System Interface			A	Product System		
	I1	Internal Interface			A	Product System		
	I11	Sensor Subsystem Innerface			A1			
	I181	Aggregate Signal Feed Source Impedance	E37H	Aggregate Signal Feed Source Impedance= 52 ohms ± 2 ohms	A1	Sensor Subsystem		
	I181	Aggregate Signal Feed Load Impedance	E37I	Aggregate Signal Feed Load Impedance= 52 ohms ± 2 ohms	A4	Analysis and Reporting Subsystem		
	I2	System External Interface			A	Product System		
Environmental	Q	System Environment			A	Product System		
	QH	Hostile Environment			A	Product System		
	QI	Self-Induced Environmental Stresses			A	Product System		
	QN	Natural Environment			A	Product System		
	QN1	Temperature	6D74	-40 degrees F< Temperature < +140 degrees F	A	Product System		
	QX	Non-Cooperative Environmental Stresses			A	Product System		

Traditional Structured Analysis

Process View



Save the Models!



JOG SYSTEM ENGINEERING, INC.
GRAND SYSTEMS DEVELOPMENT
TUTORIAL PROGRAM

AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

UNIFIED MODELING LANGUAGE

VERSION 10.1

Agenda

- The Dead Sea scrolls of software development
- UML modeling artifacts
- UML modeling approach
- Integration
- Requirements and modeling documentation

NDIA

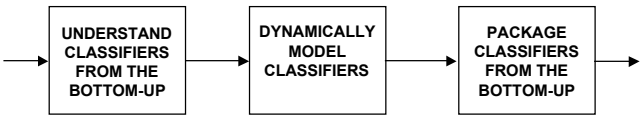
Software Dead Sea Scrolls

- **Flow chartings**
  - **Functionality examined**
  - **Data in the back seat**
- **SADT and IPO**
  - **Two axis models**
- **Modern structured analysis and HP**
  - **Functionality and data examined**
- **Early OOA**
  - **Search for objects and their behavior**
  - **Anti Sullivan**

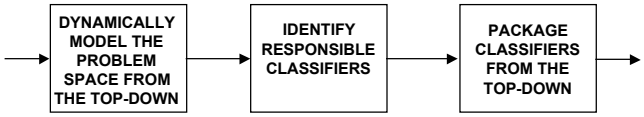
NDIA

A Preferred Modeling Order

Early object oriented analysis encouraged this pattern.



We will follow Sullivan's encouragement in this tutorial - form follows function.



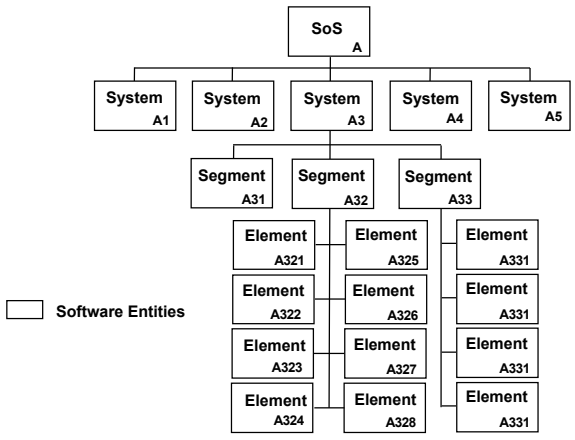
UML can support either direction.

Note: A classifier is a general term for a software product entity represented by a node, component, or class in UML.

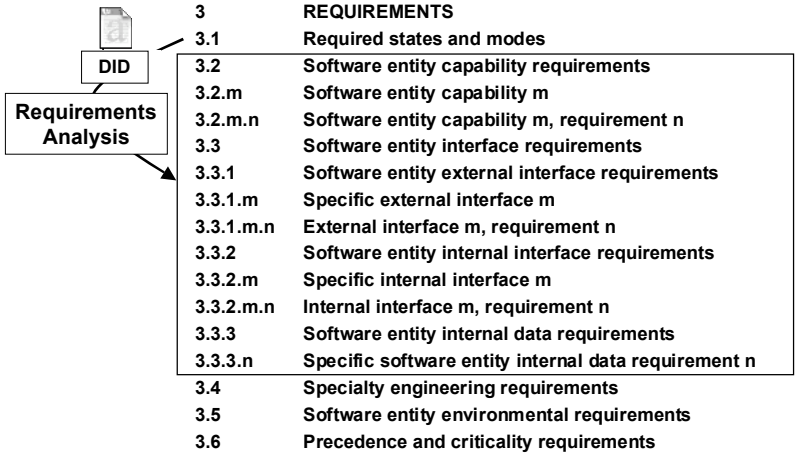
The Software Development Process

- **Identify a product entity that will be developed as computer software.**
- **Dynamically analyze the entity.**
  - **Use cases**
  - **Sequence diagram**
  - **Communication diagram**
  - **activity diagram**
  - **State diagram**
- **In the sequence, communication, and activity diagramming analysis you will have to identify lower tier product entities.**
- **And the process continues to expand and move deeper translating problem space into solution space.**
- **At the bottom are classes about which code can be written based on requirements derived from the dynamic modeling work.**

Consolidated System Product Entity Structure



Suggested SRS Structure



3	REQUIREMENTS
3.1	Required states and modes
3.2	Software entity capability requirements
3.2.m	Software entity capability m
3.2.m.n	Software entity capability m, requirement n
3.3	Software entity interface requirements
3.3.1	Software entity external interface requirements
3.3.1.m	Specific external interface m
3.3.1.m.n	External interface m, requirement n
3.3.2	Software entity internal interface requirements
3.3.2.m	Specific internal interface m
3.3.2.m.n	Internal interface m, requirement n
3.3.3	Software entity internal data requirements
3.3.3.n	Specific software entity internal data requirement n
3.4	Specialty engineering requirements
3.5	Software entity environmental requirements
3.6	Precedence and criticality requirements

The Diagrams of UML

- **For modeling dynamic aspects of the system**
  - Use case diagram
  - Sequence diagram
  - Timing diagram
  - Communication diagram (renamed in 2)
  - State diagram
  - Activity diagram
  - Interaction overview diagram (2)
- **For modeling static aspects of the system**
  - Object and class diagrams
  - Component diagram
  - Deployment diagram
  - Composite structure diagram (2)
  - Package diagram (2)

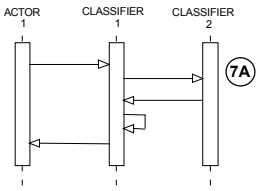
Classifiers

Covered in this tutorial

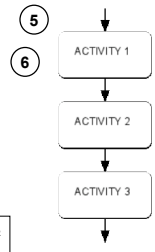
(2) = added in UML 2.0

The Dynamic Models

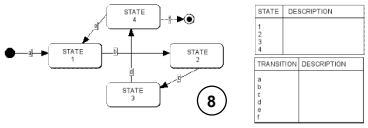
Sequence Diagram UX11321



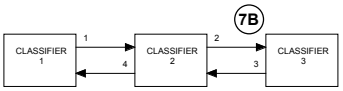
Activity Diagram UX11323



State Diagram UX11324



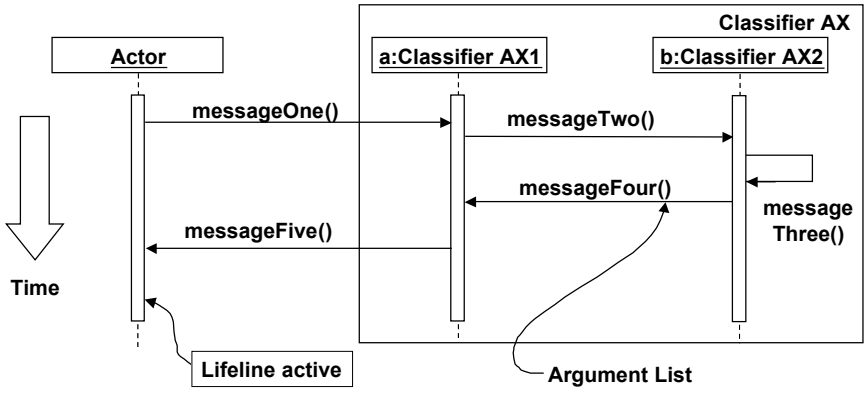
Interaction Diagrams



Communication Diagram UX11322

Sequence Diagram UX-11321

Emphasizes the time ordering of messages



It is understood that the classifiers are performing operations, possibly modeled in activity or state diagrams, relative to the message content.

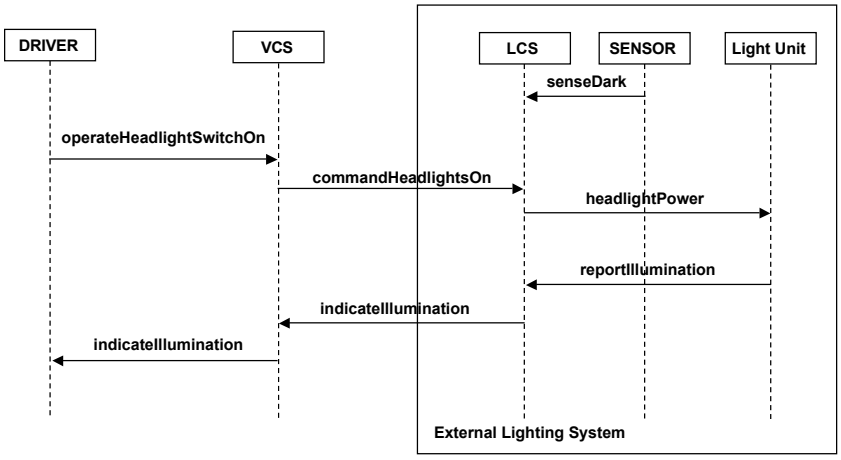
Messages Between Lifelines

- A message is the specification of a communication among objects on a class or object diagram or between the objects represented by life lines on the sequence diagram or blocks of a communication diagram.
- When a message is passed from one object to another some action usually results on its receipt
- The action may result in a change of state in the object on the arrow head.
- State related requirements in terms related to the target object.

Sequence Diagram Message Types

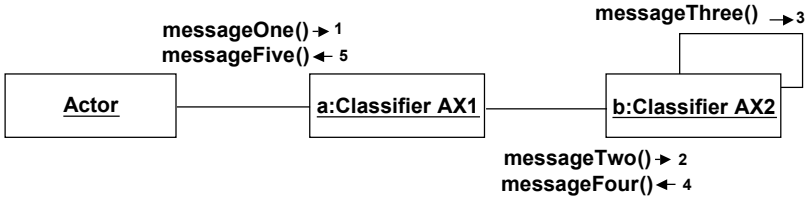
- **Call**
  - Invokes an operation on an object represented by the lifeline
  - An object can send a call to itself resulting in a local invocation
- **Return**
  - Returns a value to the caller
- **Send**
  - Sends a signal to an object
- **Create**
  - Creates an object
- **Destroy**
  - Destroys an object

A Simple Example



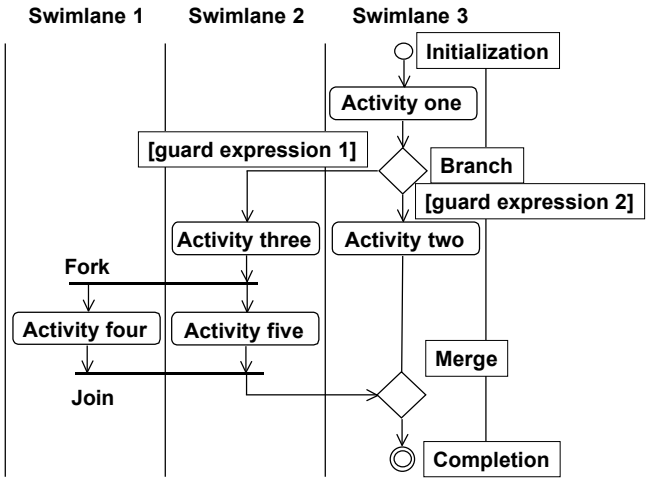
Communication Diagram UX11322

Emphasizes structural relationships

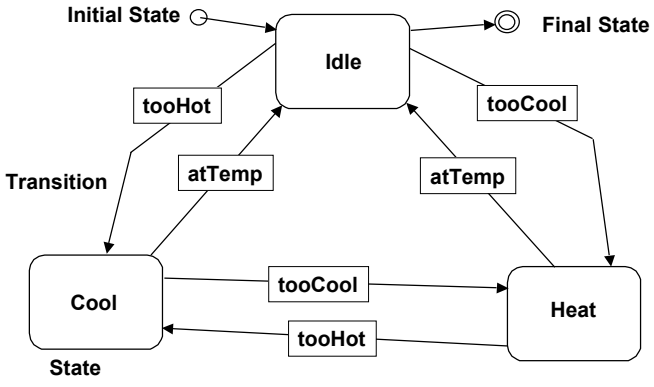


Semantically identical to the sequence diagram.

Activity Diagram UX11323



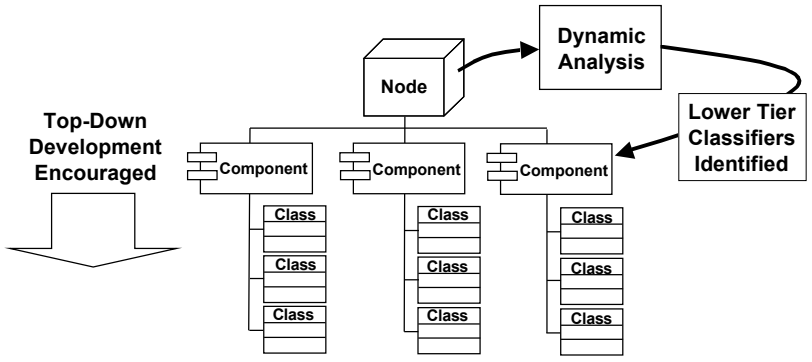
State Diagram UX11324



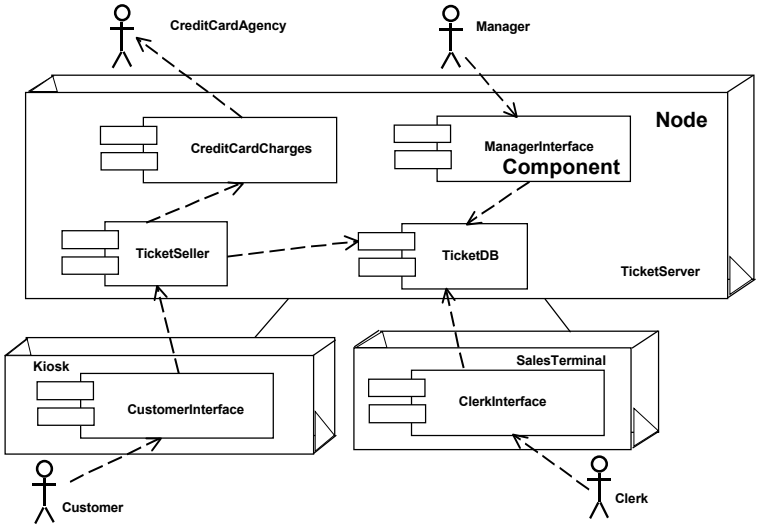
The Static Entities in UML

- **System/Subsystem**
  - **The highest level software entity. There can be many of these entities in a real system composed of hardware and distributed software. A node or collection of collection of nodes.**
- **Node**
  - **Appears on a deployment diagram that exists at run time and a computational resource, generally having at least some memory and often processing capability. A collection of components.**
- **Component**
  - **A modular part of the system consisting of classes.**
- **Class**
  - **A description of a set of objects that share the same attributes, operations, relationships, and semantics.**
- **Object**
  - **An instance of a class.**

UML Structural Artifacts in a Product Entity Structure



Deployment and Component Diagrams



Classes and Objects

A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. An object is an instance of a class. Graphically a class is rendered as a rectangle.

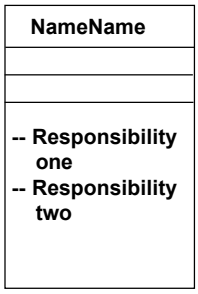
NameName
attributeOne attributeTwo attributeThree attributeFour
operationOne() operationTwo() operationThree()

The name is a noun or noun phrase

An attribute is a named property of a class that describes a range of values that instances of the property may hold. An attribute represents some property of the thing you are modeling that is shared by all objects of that class.

An operation is the implementation of a service that can be requested from any object of a class to effect behavior. An operation is an abstraction of something you can do to an object that is shared by all objects of that class

Class Responsibilities

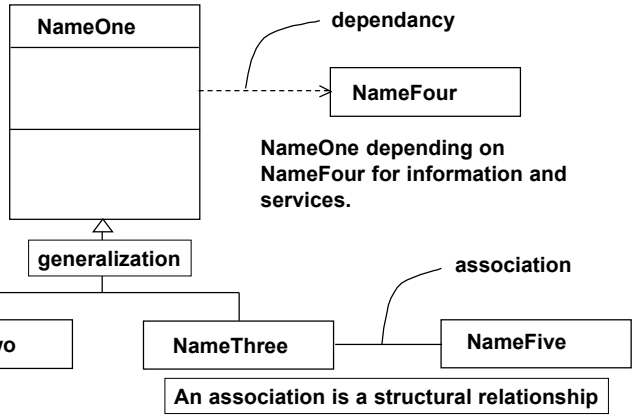


A responsibility is a contract or an obligation of a class. You may find it useful to begin the analysis of classes this way translating these into attributes and operations that best fulfill the class's responsibility as the model is refined. The responsibility is noted in an added compartment in which descriptive free form-text is entered.

Structural Relationships

These Have Nothing To Do With Messages

NameOne is the base class. NameTwo and NameThree are leaf classes in a generalization.



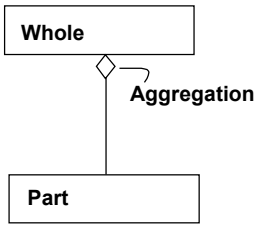
Structural Relationships
Association Adornments

- **Association Name**
- **Association Role**

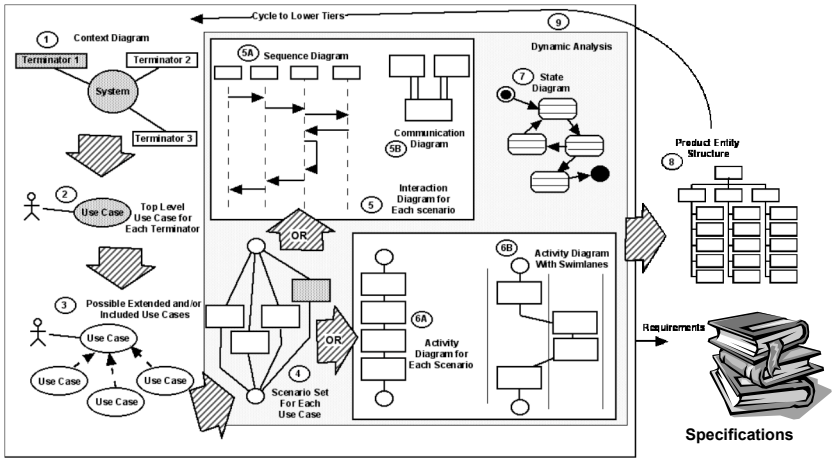
  The face that the class at the far end of an association presents to the class at near end of the association. Role names called end names.
- **Association Multiplicity**

  Tells how many objects may be connected across an association instance. Given by a range of numbers.
- **Association Aggregation**

  Expresses a whole-part relationship between to associated classes.



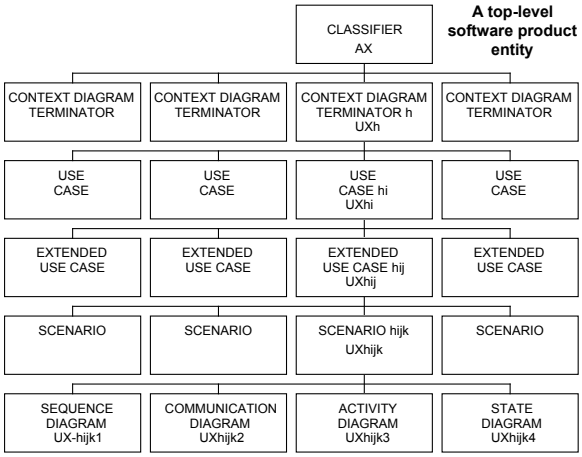
A Flexible Dynamic Modeling Overview



Organizing the Dynamic Modeling

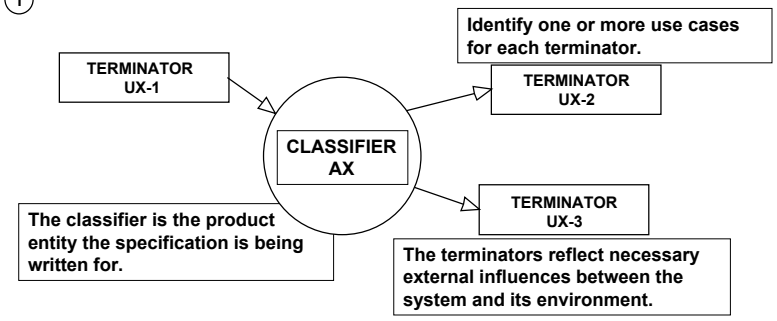
- **Use a context diagram to organize the use cases.**
- **Recognize a family of use cases if necessary.**
- **If use cases complex, recognize two or more scenarios for each use case.**
- **For each scenario build a sequence diagram and in the process identify next lower tier classifiers and messages between the actors and lower tier classifiers.**
- **Apply communication, activity, and state diagrams as needed.**
- **Derive requirements from dynamic modeling artifacts.**

Hierarchical Modeling Relationships

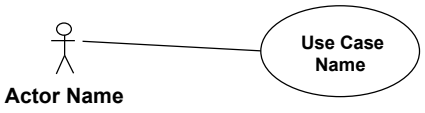


Context Diagram

Borrowed from Modern Structured Analysis to provide an organized approach to use case identification.

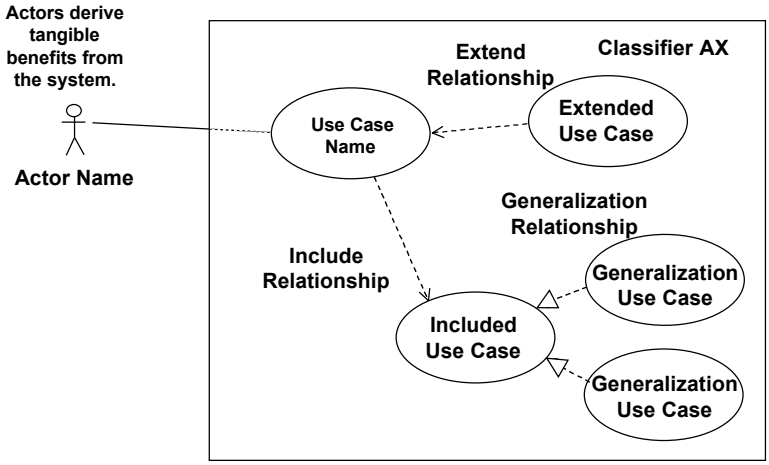


Use Case Fundamentals



- A use case is a more expressive context diagram common in modern structured analysis.
- A use case bubble represents some aspect of the system being developed.
- An actor represents some external agent gaining benefit from the system.

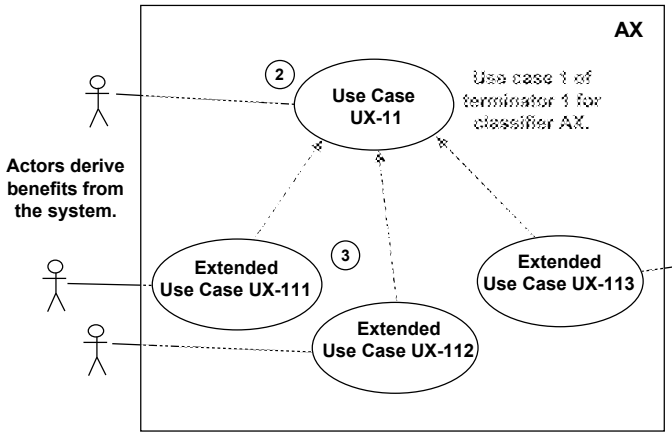
Use Case Relationships



Use Case Relationships

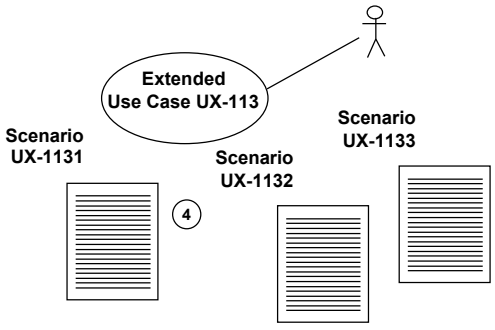
- **Extend**
  - **Pushes common behavior into other use cases that extent a base use case**
- **Include**
  - **Pulls common behavior from other use cases that a base use case includes**
- **Generalization**
  - **A child use case inherits behavior and meaning of the base use case**
  - **The child use case may add or override the behavior of the base use case**

Use Case UX-11



The word extend is used here in a generic way here to embrace extend, include, and generalization relationships.

Possible Multiple Scenarios



Textual scenario descriptions

The word extend is used in a generic way here to embrace extend, include, and generalization relationships.

Scenario

- A sequence of actions that illustrates behavior.
- A scenario may be used to illustrate an interaction or execution of a use case instance.
- Text description that can be captured in paragraph 3.1.2.h.i.j.k of the classifier specification.

Examine Each Scenario Dynamically

- Activity, sequence, and communication diagrams require identification of lower tier entities leading to additional of entities on the consolidated product entity diagram
- State diagrams may also be useful in identifying essential characteristics appropriate for the entity being analyzed
- Requirements flow out of the dynamic analysis and into the specification for the entity being analyzed

JOG SYSTEM ENGINEERING, INC.
GRAND SYSTEMS DEVELOPMENT
TUTORIAL PROGRAM

AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

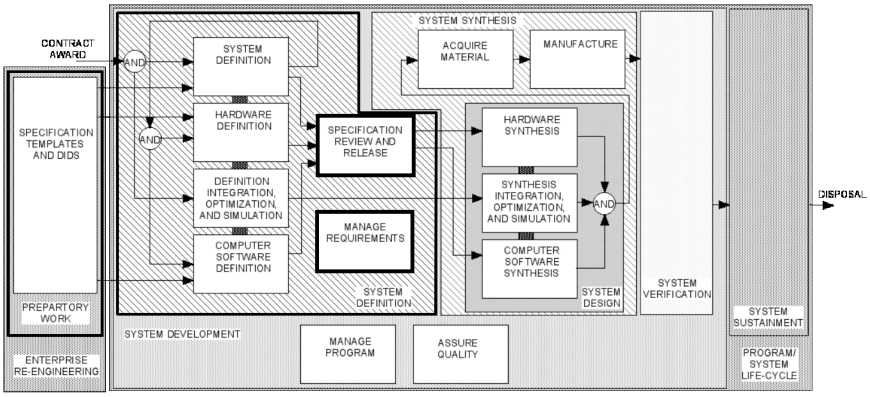
REQUIREMENTS MANAGEMENT

VERSION 10.1

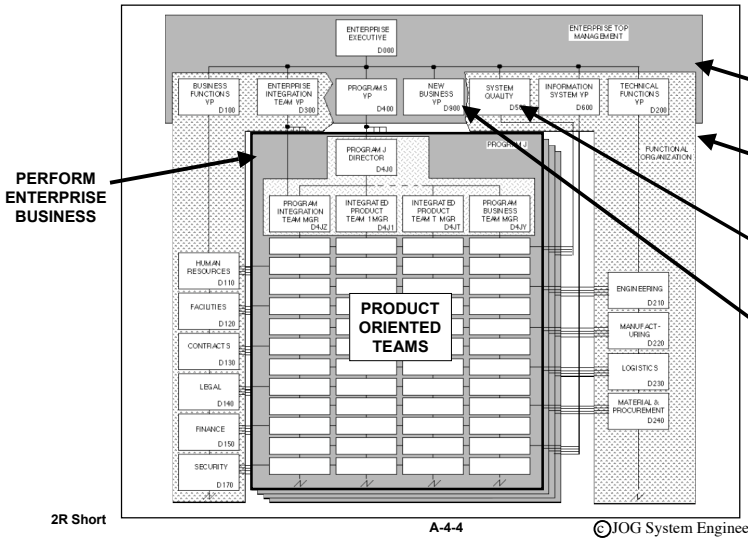
Agenda

- Process summary
- Organizing and specification responsibility
- Requirements risk management
- Traceability
- Tools
- Review, approval, release, and distribution
- A peek into the future

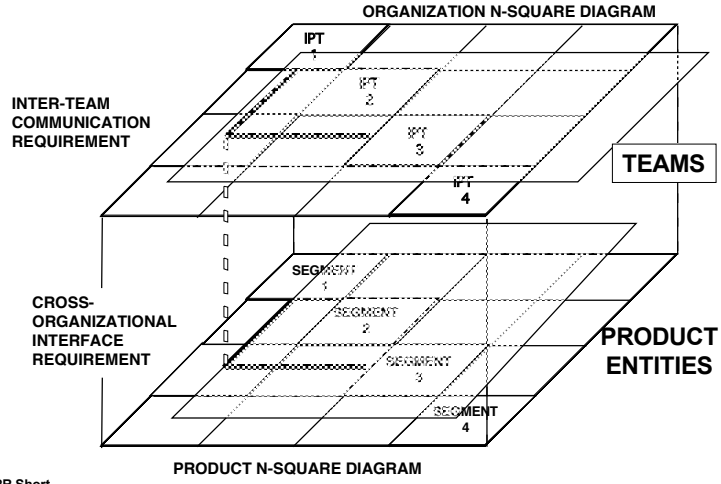
Life Cycle Model



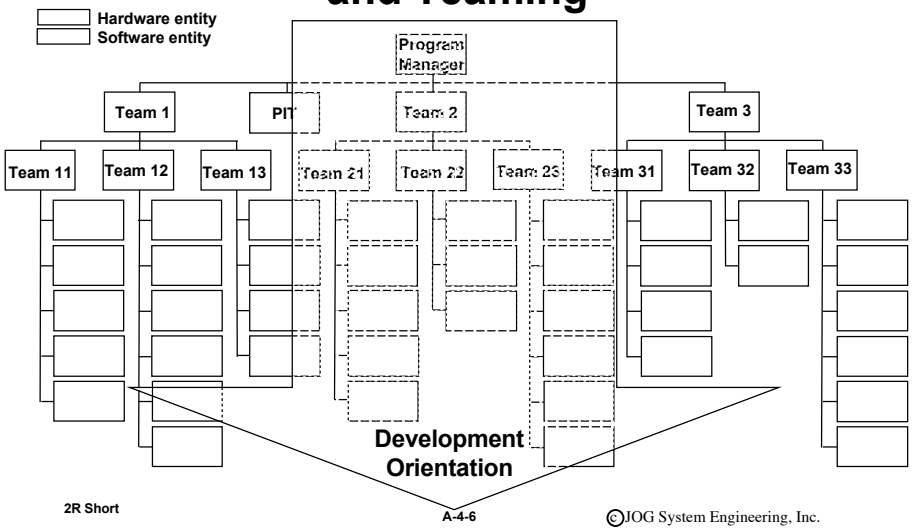
Suggested Enterprise Structure



Benefits of Product-Oriented Team Structure



The System Product Entity Structure and Teaming



NDIA

Risk Defined

- The danger that injury, damage, or loss will occur
- Somebody or something likely to cause injury, damage, or loss
- The probability, amount, or type of possible loss incurred by an insurer
- The possibility of loss in an investment or speculation
- The statistical chance of danger from something, especially from the failure of an engineered system

NDIA

Risk Measurement Parameters

Probability of Occurrence

CAT	TITLE	P(O)	DESCRIPTION
5	Nearly Certain	0.95-1.00	Will occur at least once during program
4	Probable	0.75-0.95	Will probably occur once during program
3	Possible	0.50-0.75	May occur during program
2	Unlikely	0.25-0.49	Will probably not occur during program
1	Nearly Impossible	0.00-0.24	Will not occur during program

Seriousness of Effect

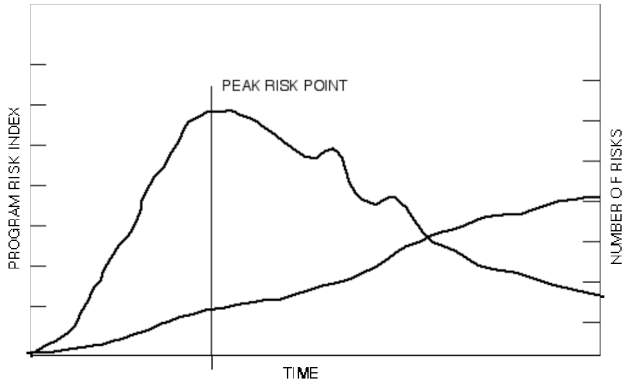
CAT	TITLE	DESCRIPTION
5	Catastrophic	Program in jeopardy of termination
4	Serious	Serious damage to program
3	Moderate	Problems cause program focus difficulties
2	Minor	Problems that can be easily be overcome
1	Null	No problem

Risk Metric Values

PROBABILITY OF OCCURANCE	SERIOUSNESS OF THE EFFECT 5	4	3	2	1
5	25	20	15	10	5
4	20	16	12	8	4
3	15	12	9	6	3
2	10	8	6	4	2
1	5	4	3	2	1

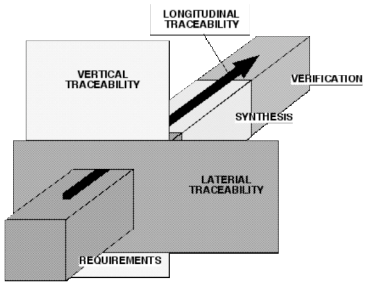
High Risk
Medium Risk
Low Risk

Aggregate Program Risk Index



Traceability Forms

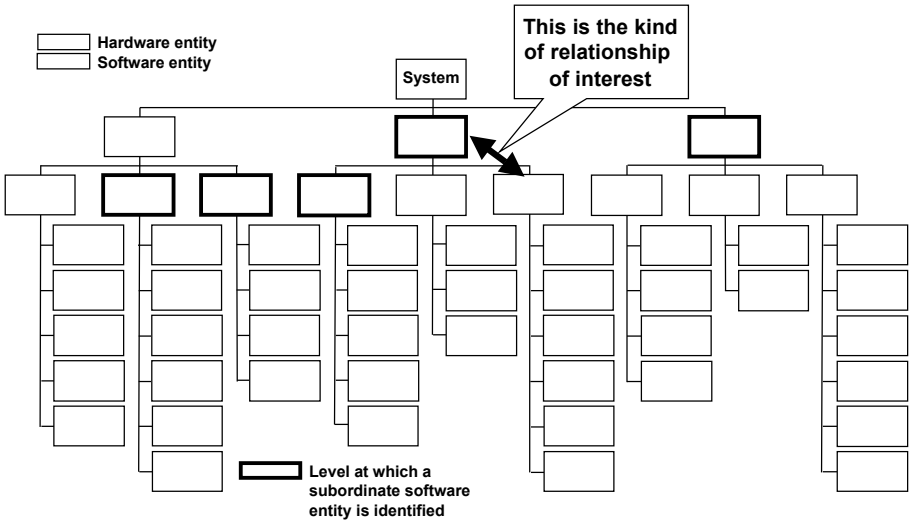
- **Vertical requirements traceability**
  - **Hierarchical or parent-child**
  - **Requirements source traceability**
  - **Requirements rationale traceability**
- **Longitudinal traceability**
  - **Requirements to design and verification**
- **Lateral traceability**
  - **Traceability to method**
- **Applicable document**
  - **Internal integrity**



Traceability Integration

- **Lateral traceability within TSA and UML independently should be no problem with all requirements derived from models**
- **Vertical interface traceability is decomposition driven within TSA and UML as well as across the HW-SW gap**
- **Environmental requirements are significantly different between HW and SW so traceability is not a significant issue between them**
- **Performance requirements across the HW-SW gap offer a vertical traceability challenge**

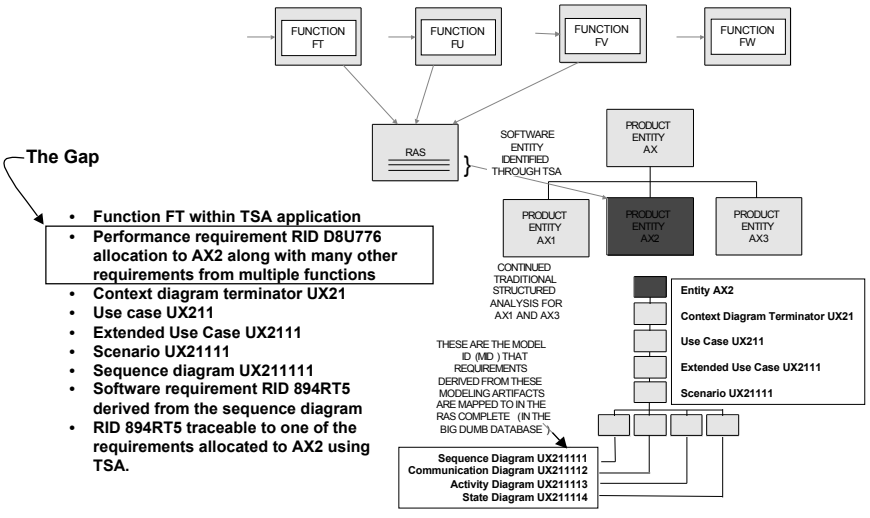
The System Product Entity Structure



2R Short

A-4-13

Traceability Across the Gap



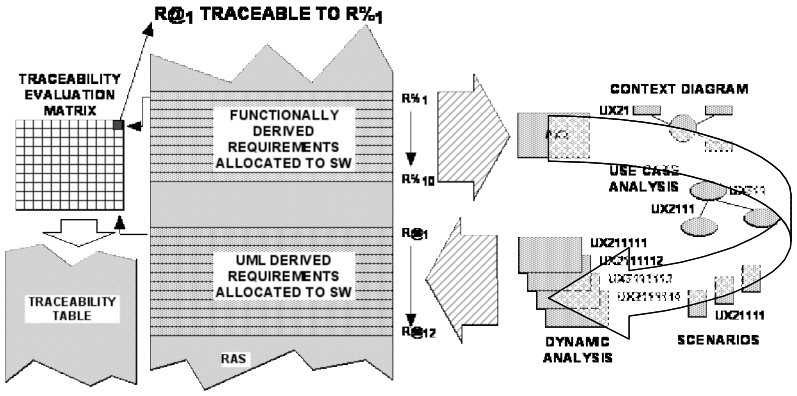
The Gap

- **Function FT within TSA application**
- **Performance requirement RID D8U776 allocation to AX2 along with many other requirements from multiple functions**
- **Context diagram terminator UX21**
- **Use case UX211**
- **Extended Use Case UX2111**
- **Scenario UX21111**
- **Sequence diagram UX211111**
- **Software requirement RID 894RT5 derived from the sequence diagram**
- **RID 894RT5 traceable to one of the requirements allocated to AX2 using TSA.**

2R Short

A-4-14

Traceability Evaluation Matrix



Traceability Evaluation Matrix

Requirements Derived From UML Modeling

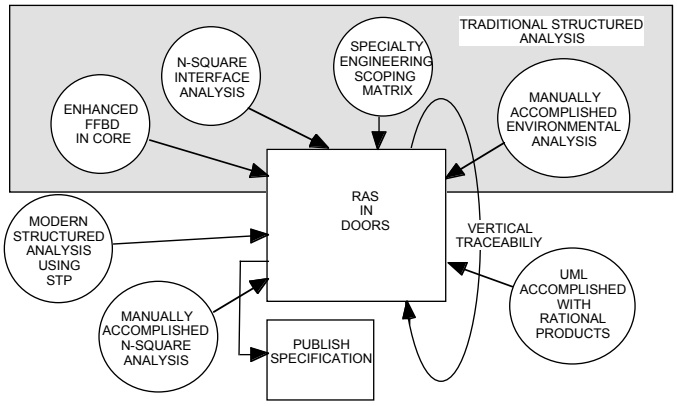
Requirements Derived From TSA Modeling	R@1	R@2	R@3	R@4	R@5	R@6	R@7	R@8	R@9	R@10	R@11	R@12
R%1												
R%2												
R%3												
R%4												
R%5												
R%6												
R%7												
R%8												
R%9												
R%10												

ACTUAL RID EXAMPLE

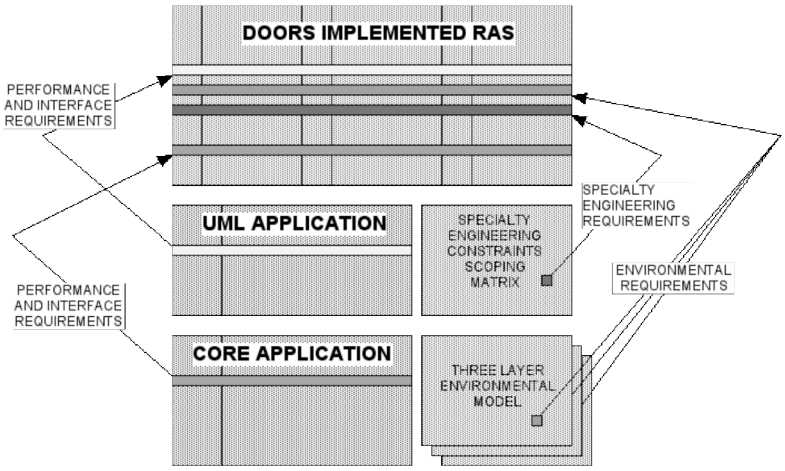
R%1	RU7Z7H
R%2	R9IER6
R%3	R937YF
R%4	RJ8E6G
R%5	RJYT6T
R%6	RHGT5T
R%7	RID87W
R%8	RBJ8S7
R%9	RL34DF
R%10	R456HD

Alternatively, one could rely upon experienced inspection without the organizing influence of the matrix.

Today's Tools



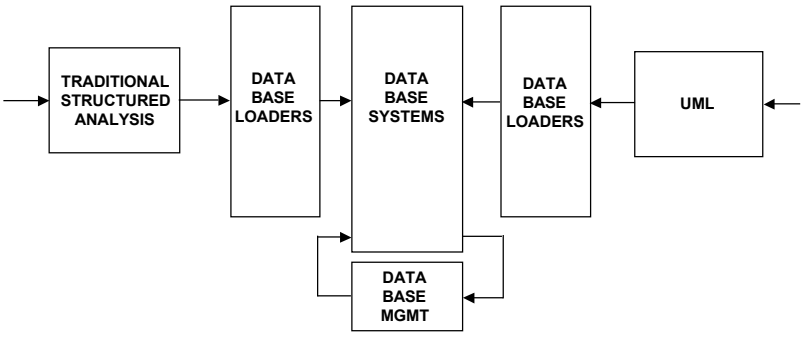
Tools Integration



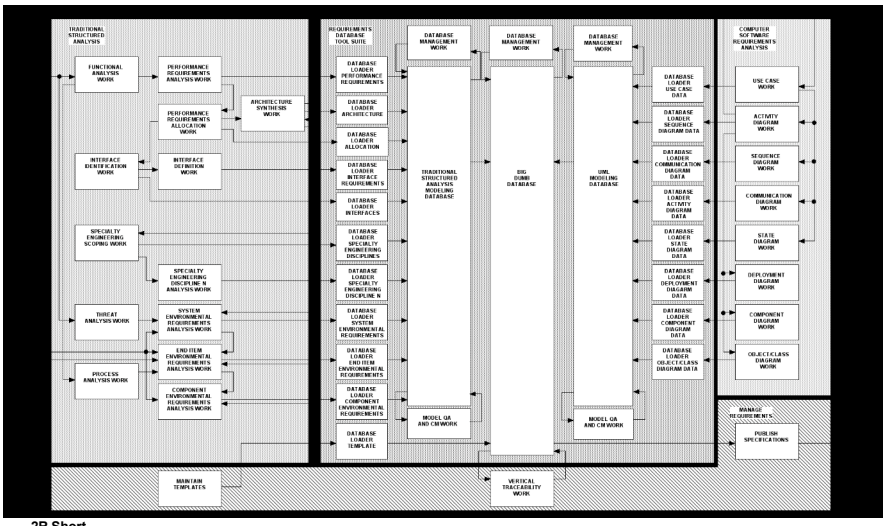
Tomorrow's Tools

- **Front end modeling tools**
  - **Use case modeling**
  - **Function/activity modeling**
  - **State modeling (behavioral modeling)**
  - **Sequence/timeline modeling**
  - **Product entity and interface modeling**
  - **Specialty engineering database linkage**
  - **Environmental coverage**
- **Connection of modeling to management database**
- **Big dumb database**
  - **Requirements capture**
  - **Traceability**
  - **Value management**
  - **Specification publishing**

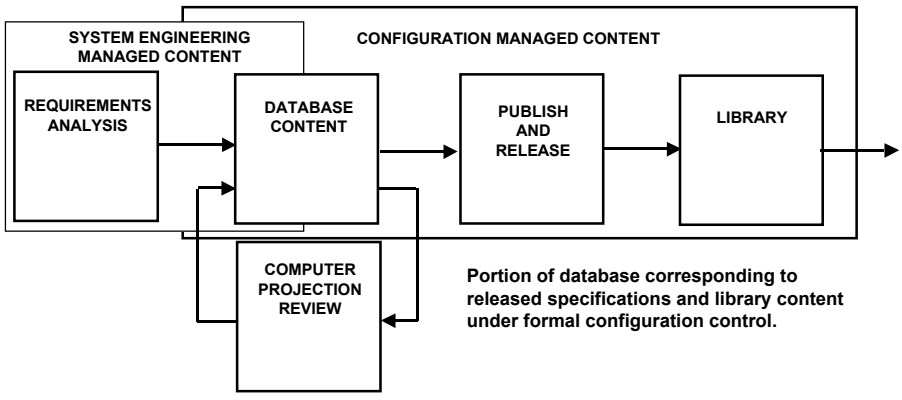
Tools Integration



Integrated Tool Set



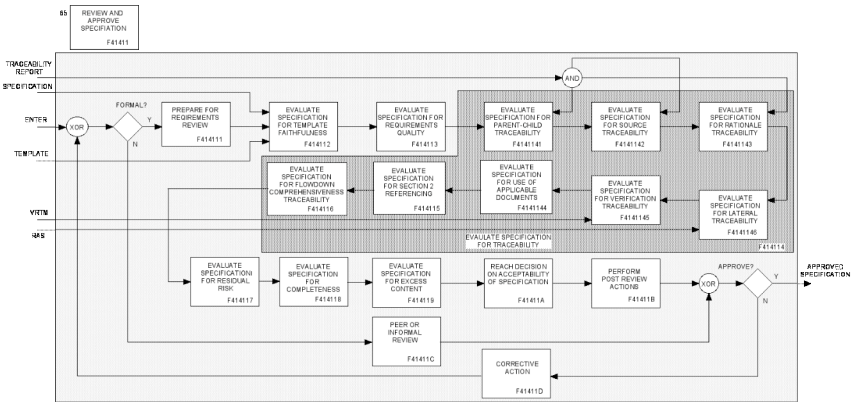
Configuration Management of Requirements Documentation



Utility Of Computer Projection

- ON-LINE NETWORK CAPABILITY
- PUT THE PROJECTION CAPABILITY IN THE WORK AREA
- APPLY REAL-TIME CONCURRENT DEVELOPMENT (IPD)
- FORM AND REFORM BETWEEN MEETING AND INDIVIDUAL WORK QUICKLY

Specification Review and Approval Process



Evaluate for Template Faithfulness

- Compare specification cover data with template (standard)
- Compare specification paragraphing structure with template
- Compare specification style with template style guide
-
-

Individual Requirement Quality

- Spot check specification requirements for requirements quality checklist compliance
- Spot check specification for requirements quantification where appropriate
-
-
-

Section 2 Traceability

- All documents listed in Section 2 called somewhere in the specification
- All documents tailored, if necessary, to limit coverage to the application
- All documents called in the requirements listed in Section 2
- Spot check for excessively tailored standards which could be quoted instead of being called
- Ensure documents called are current and accepted authorities for the application

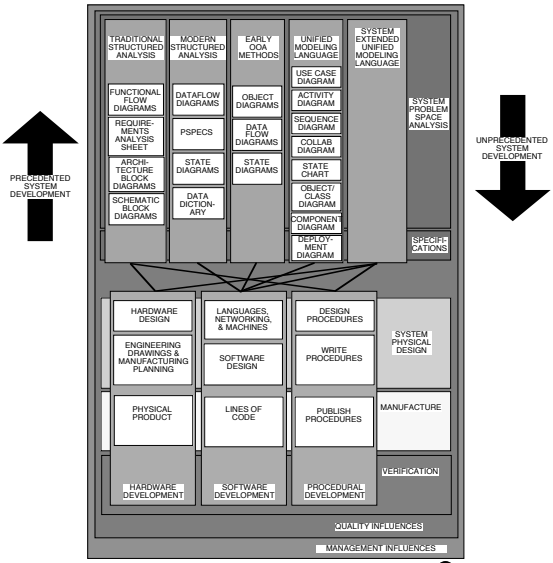
Completeness and Avoidance of Unnecessary Content

- Ask principal engineer how content was derived
  - If ad hoc, there should be concern
  - If through structured analysis, spot check how a few requirements were derived (ask to see the supporting modeling data)
- Ensure all requirements traceable to parent requirements
-

Residual Risk Evaluation

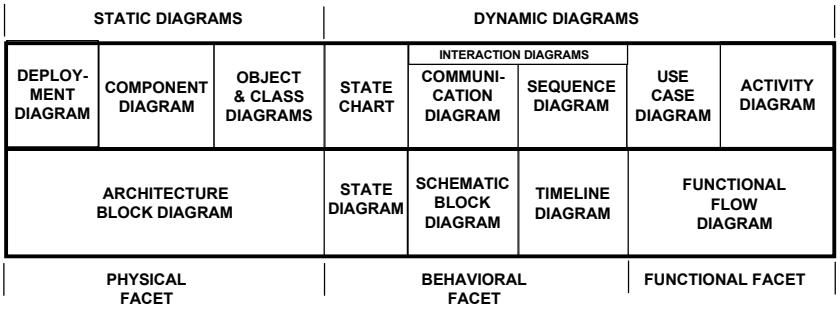
- **All TBD/TBR are closed or, if not, are being carried as program or team risks**
- **An approved concept exists**
-
-
-

Movement To Universal Method



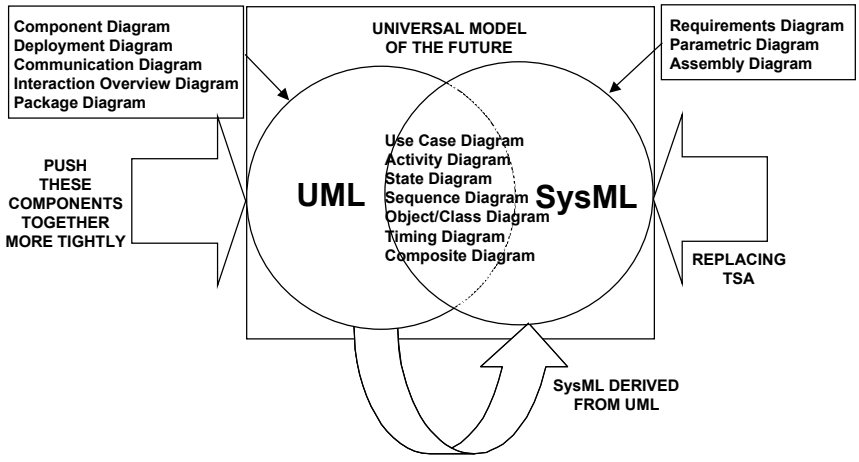
UML and Functional Analysis

Unified Modeling Language (UML)

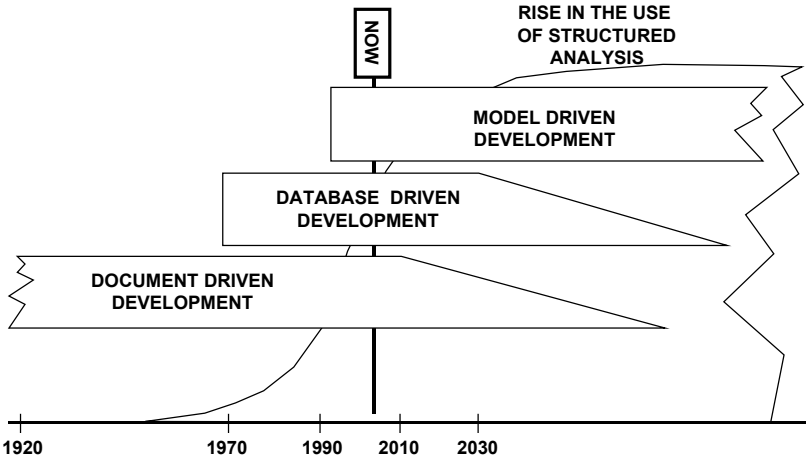


Traditional Structured Analysis
A Subset of UML?

Modeling Changes In the Near Term



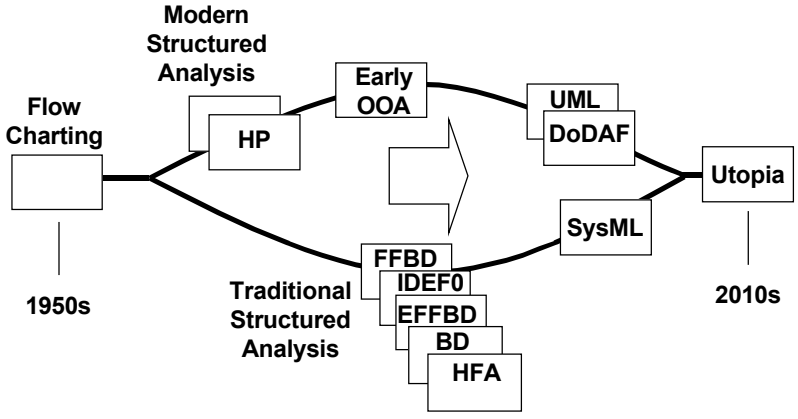
System Modeling Evolution Timeline



05-15-2002 DATA UNSUBSTANTIATED
DATES ARE APPROXIMATE

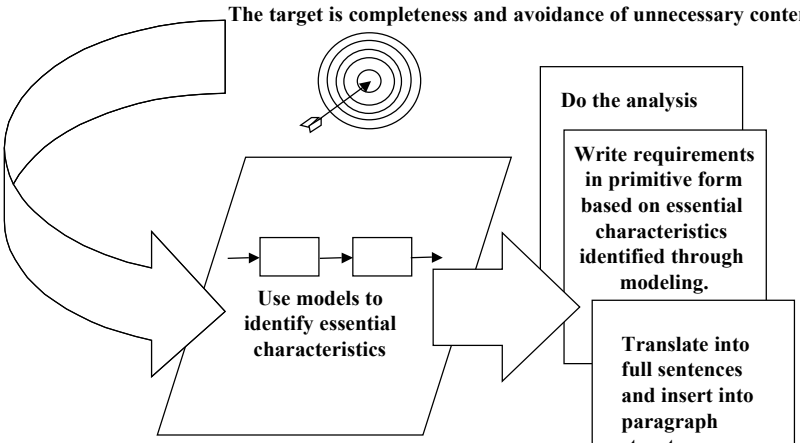
2R Short

Over the Hill and Through the Woods to Utopia



2R Short

Review and Summary



2R Short

A-4-35