Documentation and Validation of the Requirements Specifications – An XML Approach

Kenza Meridji

A Major Report

In

Department

Of

Computer Science

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science

Concordia University

Montreal, Quebec, Canada

August 2003

Your file Votre référence
ISBN: 0-612-83917-6
Our file Notre référence
ISBN: 0-612-83917-6

Abstract

Documentation and Validation of the Requirements Specifications – An XML Approach

Kenza Meridji

In this major report, we have stressed the importance of documentation in software engineering field and have proposed a flexible requirements specifications template, easily adaptable to object-oriented development for specific organizations. XML was used to formally represent the requirements specification documents, so that the structured SRS documents could be used over the web. The approach is illustrated on a case study. We have defined theoretically valid measures that validate the consistency achieved in the functional requirements. The measurement mechanism is based on the XML representation.

Acknowledgement

I would like to express my deepest gratitude to my supervisor, Dr. Olga Ormandjieva. She gave me helpful guidance and advices all along the way.

My sincere thanks to Dr. Joey Paquet, the examiner of this major report, for his helpful advises and comments.

I would like to thank Mr. Pankaj Kamthan who has played a significant role in shaping my major report.

On a personal level, I thank my family for the support during the years of my graduate studies.

Table of Contents

List of Figures

Chapter 1: Introduction

The main purpose of this report is to contribute to the simplification and standardization of the software engineering procedures for writing requirements specification documents.

1.1 The Importance of Documentation in Software Engineering

Software Engineering is a discipline for the systematic construction and support of software products so they can safely fill the uses to which they may be subjected. Software impacts almost every aspects of modern society. Developing software is not a trivial task, and involves different teams of developers. The communication between the users and software developers is based on documentation. Documentation is the lifeblood of software engineering, and is critical for the development of correct software. Concrete, realistic, and measurable procedures for writing documentation make task much more of a professional challenge.

1.2 Purpose and Problem Statement

The purpose of this report is to contribute to the simplification and standardization of the procedures for writing requirements and the validation of the domain against use case models. Our main goal is to give a flexible requirements specifications template, easily adaptable to object-oriented development for specific organizations. The template is mapped to XML database. XML is the Extensible Markup Language, which was created so that richly structured documents could be used over the web. XML tags are not

predefined it can be used to represent any form of data. The only viable alternative, HTML is not practical for this purpose because the tags used to mark up HTML documents and the structure of the HTML documents is predefined.

The XML database would allow for:

- Creating a set of validation criteria for checking consistency between the domain and use case models for requirements specifications.
- Adding links between the requirements that would simplify the readability and traceability of the requirements, in order to manage evolution over time;
- Individuals to accomplish the following goals:

  1) Develop a standard software requirements specification (SRS) outline for their own organizations;

  2) Define the format and content of their specific software requirements specifications;

  3) Develop additional local supporting items such as an SRS quality checklist, or an SRS writer's handbook.

1.3 Report Outline

The present major Report is organized as follows. Chapter 2 is an introduction to Software Engineering discipline. The Requirements Engineering goals, activities and documents are described in Chapter 3. The international software engineering standards and their role in the documentation are explained in Chapter 4. Chapter 5 contains the template for the requirements document, as suggested by IEEE Std.830-1998 and the object-oriented development practice. Chapter 6 introduces XML as a markup language

and presents the XML template for the software requirements document. Chapter 7 describes the validation criteria for the analysis modeling, and the verification of its consistency. In Chapter 9 we present the conclusions and the future work directions. Appendix 1 consist of a Case Study "the Air Traffic Management (ATM) System" which is an illustration of our approach. In Appendix 2 we present the schema description of requirement specification document.

Chapter 2. Software Engineering

2.1 The Notion

Software Engineering has come to mean at least two different things in our industry [SEYP]. First of all the term "software engineer" has generally replaced the term "programmer". So, in that sense there is a tendency to extrapolate in people's minds that Software Engineering is merely the act of programming. Secondly, the term "Software Engineering" has been used to describe "building of software systems which are so large or so complex that they are built by a team or teams of engineers", as was used in Fundamentals of Software Engineering by Ghezzi, Jazayeri, and Mandrioli [GJM91]. Yet, there is increasing evidence that many of the processes that have been developing for large groups of engineers also apply to the best practices of even individual engineers. Therefore, Software Engineering is intended to mean the best-practice processes used to create and/or maintain software, whether for groups or individuals, in attempt to rid ourselves of the usual haphazard methods that have plagued the software industry. This would include subjects like Configuration Management, Project Planning, Project Tracking, Software Quality Assurance, Risk Management, Formal Inspections, etc.

According to the NATO Science Committee Conference (quoted by [PRE97]), software engineering is "the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines". Another source states that software engineering is "...promoting the establishment of theoretical foundations and practical disciplines for software, similar to those found in the established branches of engineering" (quoted by [BER92]). The author of a well-known

textbook on software engineering [SCH90] is defining it as "a discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements." In the IEEE Standard Glossary of Software Engineering Terminology, the notion of software engineering is defined as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" [IEEE91].

The "practice of professional engineering" as described in [PPE96] means any act of planning, designing, composing, evaluating, advising, reporting, directing or supervising, or managing any of the foregoing, that requires the application of engineering principles, and that concerns the safeguarding of life, health, property, economic interests, the public welfare or the environment.

2.2 Software Process

Software development process is a coherent set of activities for software modeling and associated artifacts. Modeling a system involves identifying the things that are important to your particular view. These things from the vocabulary of the system you are modeling. For example as described in [BRJ99], if you are building a house, things like walls, doors, windows, cabinets, and lights are some of the things that will be important to a home owner. Each of these things can be distinguished from the other. Each of them has a set of properties. Walls have a height and a width and are solid. Doors also have a height and width and are solid, as well, but have the additional behavior that allows them to open in one direction. Windows are similar to doors have slightly different properties.

Windows are usually (but not always) designed so that you can get out of them instead of pass through them.

Individual walls, doors, and windows rarely exist in isolation, so you must also consider how specific instances of these things must fit together. The things you identify and the relationships you choose to establish among them will be affected by how you expect to use the various room to room, and the general style and feel you want this arrangement to create.

Users will be concerned about different things. For example, the plumbers who help build your house will be interested in things like drains, traps, and vents. You, as a home owner, won't necessarily care about these things except in so far as they interact with the things in your view, where a drain might be placed in a floor or where a vent might intersect with the roof line.

The Unified Modeling Language (UML) was developed on the way to standardize the description of software designs, particularly objects oriented designs. The Object Management Group has accepted the UML as a standard. In UML, all of these things are modeled as classes. A class is an abstraction of the things that are part of the vocabulary. A class is not an individual object, but rather represents a whole set of objects. Thus you may conceptually think of "wall" as a class of objects with certain common properties, such as height, length, thickness, load-bearing or not, and so on. You may also think of individual instances of wall, such as" the wall in the southwest corner of my study".

Software processes (like most business processes) are complex and involve a very large number of activities. The development process usually begins when the client approaches the organization with regard to a software product that, in the opinion of the client, is

either essential to the profitability to his or enterprise or somehow can be justified economically.

2.2.1 Requirements phase

To build something, we must first understand what that "something" is to be. The development process usually begins when a client approaches a development organization with regard to a software product. The requirements phase usually begins with one or more members of the requirements team meeting with one or more members of the client organization to determine what is needed to the target organization. This preliminary investigation of the client's need sometimes is called *concept exploration* [SCH02]

2.2.2 Specification phase

Once the client agrees that the developers understand the requirements, the functionality of the software and constraints on its operation must be defined. A specification team writes the specification document. The specification document (or specifications) explicitly describes the functionality of the product (that is precisely what the product is supposed to do) and lists any constraints that the product must satisfy. The specification document includes the inputs to the product and the required outputs [SCH02].

2.2.3 Design phase

The specification of a product describes WHAT the product is to do. The software to meet the specification must be produced. The aim of the design phase is to determine HOW the product is to do it. Starting with the specification, the design team determines

the internal structure of the product. The designers decompose the product into modules, independent pieces of code with well-defined interfaces to the rest of the product [SCH02]. Engineering design, as stated in [CEA00] integrates mathematics, basic sciences, engineering sciences and complementary studies in developing elements, systems and processes to meet specific needs. It is a creative, iterative and often open-ended project, subject to constraints, which may be governed by standards or legislation to varying degrees depending upon the discipline. These constraints may relate to economic, health, safety, environmental, social or other pertinent factors."

During the design phase, the specifications are analyzed carefully and a modular decomposition of the product is developed. The ***architectural design*** is a higher level description of the organization of class into packages, and how they are to be interconnected. In the ***detailed design***, each class is designed in detail, with its methods and attributes. For each method, a *pre-* condition and a *post-* condition are specified. The pre-condition describes the guard conditions for the method to be executed, and the post-condition states the outcome from method's execution.

2.2.4 Implementation phase

During the implementation phase, the design is "mapped" to a programming language. The resulting code has to be tested thoroughly to insure that it is reliable, and fulfills the user's requirements.

2.2.5 Testing phase

During the testing phase, each unit is tested thoroughly. Integration testing then determine whether the product as a whole function correctly. The way in which the modules are integrated in a specific order can have a critical influence on the quality of the resulting product.

2.2.6 Delivery and Maintenance phase

Once the product has been delivered, it enters maintenance phase, which involves repairs and enhancement. Maintenance consumes as much as 80% of software engineering resources. The maintenance phase ends when the product is not in use any more.

2.3 Software Process Models

A software process model is an abstract representation of a software process. Each process model represents a process from a particular perspective, and only provides partial information about that process.

A number of different general models of software development are listed below.

2.3.1 Waterfall model

The *waterfall model* sometimes called the *classic life cycle* is divided into phases such as analysis, design, coding, testing, and support. Software project is developed stage by stage without feedback loops.

Modules are implemented, documented and integrated to form a complete product. This is a good model to use when requirements are well understood. [PRE97].

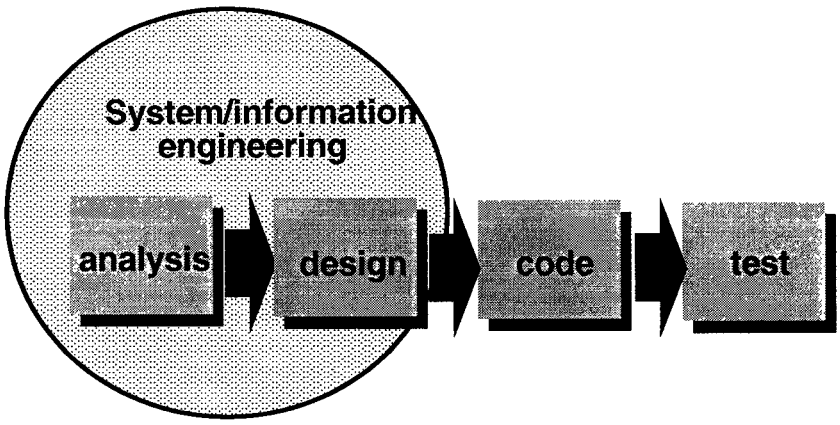

Figure 1: Waterfall model

2.3.2 Prototyping model

Prototyping paradigm begins with requirements gathering. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. Then a quick design is done. The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g. input approaches and output formats). The quick design leads to the construction of a prototype. The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is turned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what need to be done. Both customers and developers like the prototyping paradigm. Users get a feel for the actual system and developers get to build something immediately [PRE97].

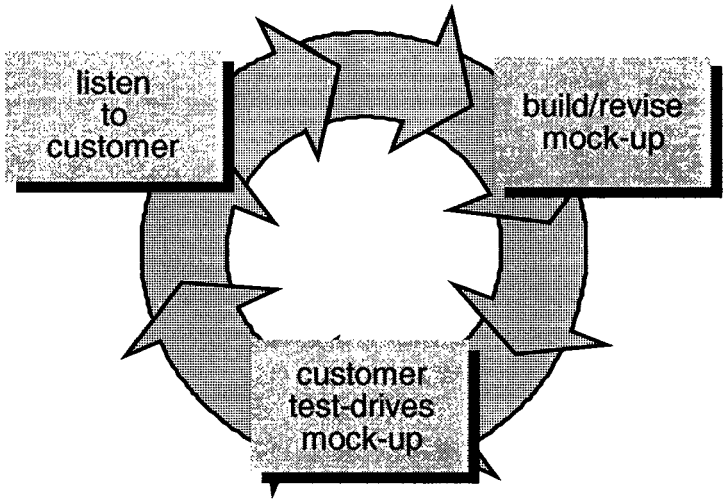

Figure 2: Rapid prototyping

2.3.3 Incremental model

The *incremental model* combines elements of the linear sequential model with the iterative philosophy of prototyping. Each linear sequence produces a deliverable "increment" of the software. When the incremental model is used the first increment is often a core product. That is, basic requirements are addressed, but many supplementary features remain undelivered. The customer uses the core product. As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meets the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.

With each increment the incremental process model delivers an operational product. Incremental developing is useful when staffing is unavailable for the complete implementation. In addition increments can be planned to manage technical risks [PRE97].

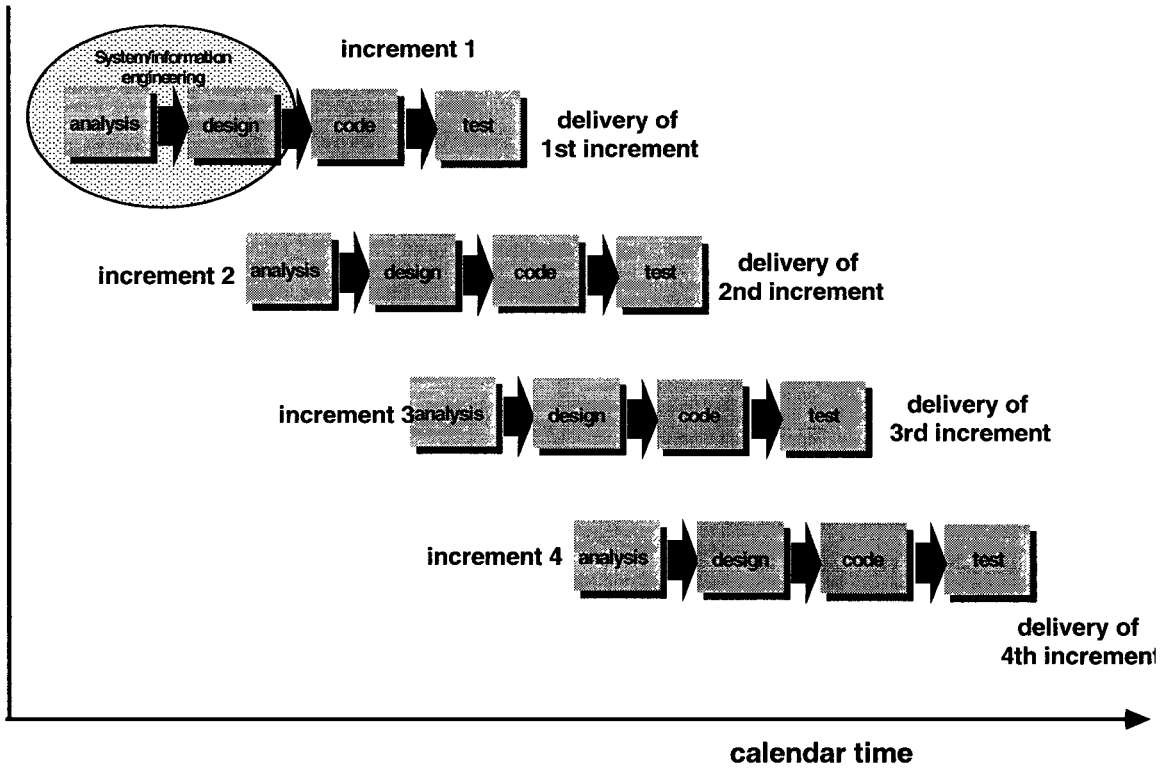

Figure 3: Incremental Model

2.3.4 RUP: Rational Unified Process

The *Unified Process* (UP) is a software development process based on UML. It is use-case driven, architecture-centric, iterative and incremental. In the use-case driven approach, use-cases are used for establishing the desired behavior of the system. It is a best approach for providing solid foundation for incremental software development.

Each cycle in the UP consists of four phases such as inception, elaboration, construction and transition. Each phase can be divided into iterations. Each iteration address a set of related use cases or mitigates some of the risks identified at the beginning of the iteration. The workflows such as Requirements, Analysis, Design, Implementation and Testing participate in each of these iterations. During elaboration phase, the requirements and

analysis activities are allocated most of the resources and also during the construction phase, the resource requirements for requirements and analysis activities diminish, but the design and implementation activities are allocated more resources. A typical iteration goes through all the five workflows as shown in the figure below.

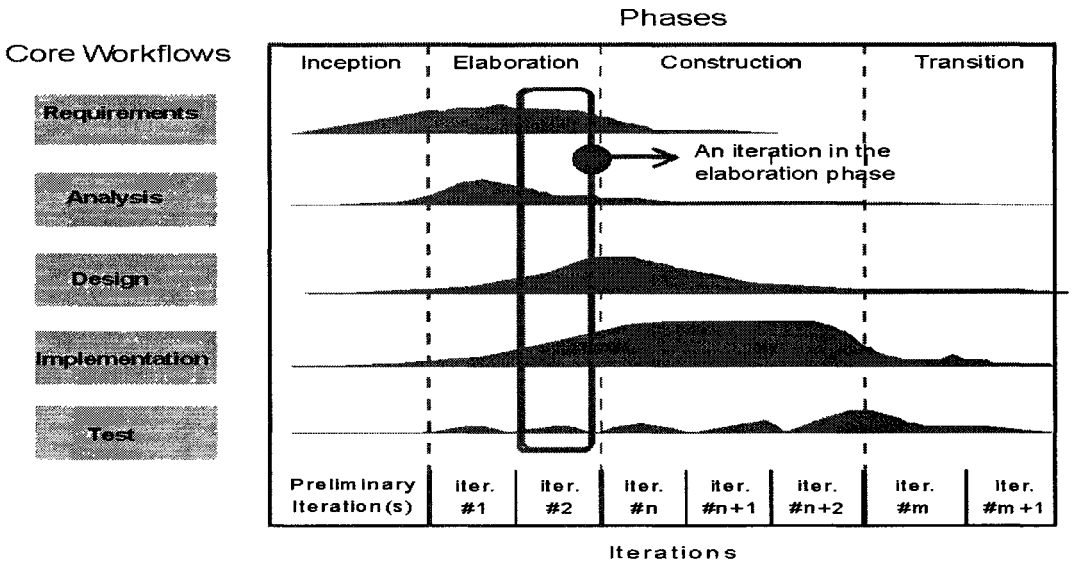

Figure 4: RUP model

2.3.5 Spiral model

The *spiral model* is an evolutionary software process model that couples the iterative nature of the prototyping with the controlled and systematic aspects of the linear sequential model. Using the spiral model, software is developed in a series of incremental releases. During later iterations, increasingly more complete versions of the engineered system are produced.

The spiral model consists of six phases listed below:

Customer communication – in this phase effective communication between developer and customer is established.

Planning – in this phase resources, timelines, and other project-related information are defined.

Risk Analysis – in this phase technical and management risks are assessed.

Engineering – in this phase one or more representations of the application are built.

Construction and release – in this phase construction, test, installation and user support are provided.

Customer Evaluation – in this phase, the customer feedback based on evaluation of the software representation created during the engineering and implemented during the installation stage is obtained.

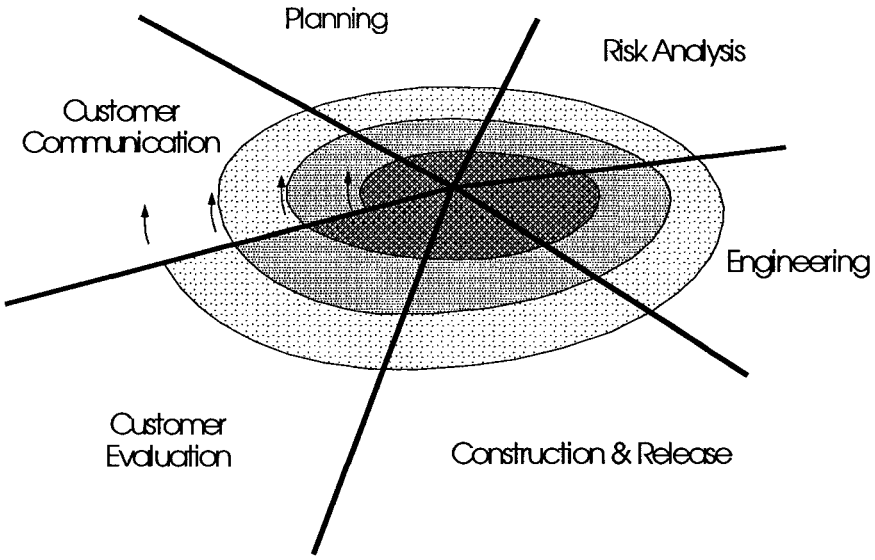

Figure 5: Spiral model

The spiral model is appropriate for large software projects. Because software evolves as the process progresses, the developer and customer understand better and react to risks at each evolutionary level. The spiral model allows the developer to apply the prototyping

approach at any stage in the evolution of the product, and if properly applied should reduce risks before they become problematic [PRE97].

2.3.6 Extreme programming

Extreme Programming [BEC99] is a new approach to software development based on the incremental model. The first step is that the software development determines the various features that the client would like the product to support. For each feature, the team informs the client how long it will take to implement that feature and how long it will cost. The client selects the features to be included in each successive build on the basis of the time and the cost estimate provided by the development team as well as the priority of the feature to his or her business. The proposed build is broken down into pieces, named *tasks*. A programmer first draws up test cases for a task. Then, working with a partner on one screen (*pair programming*) [WKCJ00], the programmer implements the task, ensuring that all the test cases work correctly. The task then is integrated into the current version of the product. The test cases used for the task are retained and utilized in all further integration testing. XP has been used successfully on a number of small-and medium-size projects. However, XP has not yet been used widely enough, the figure below illustrates this approach [SCH02].

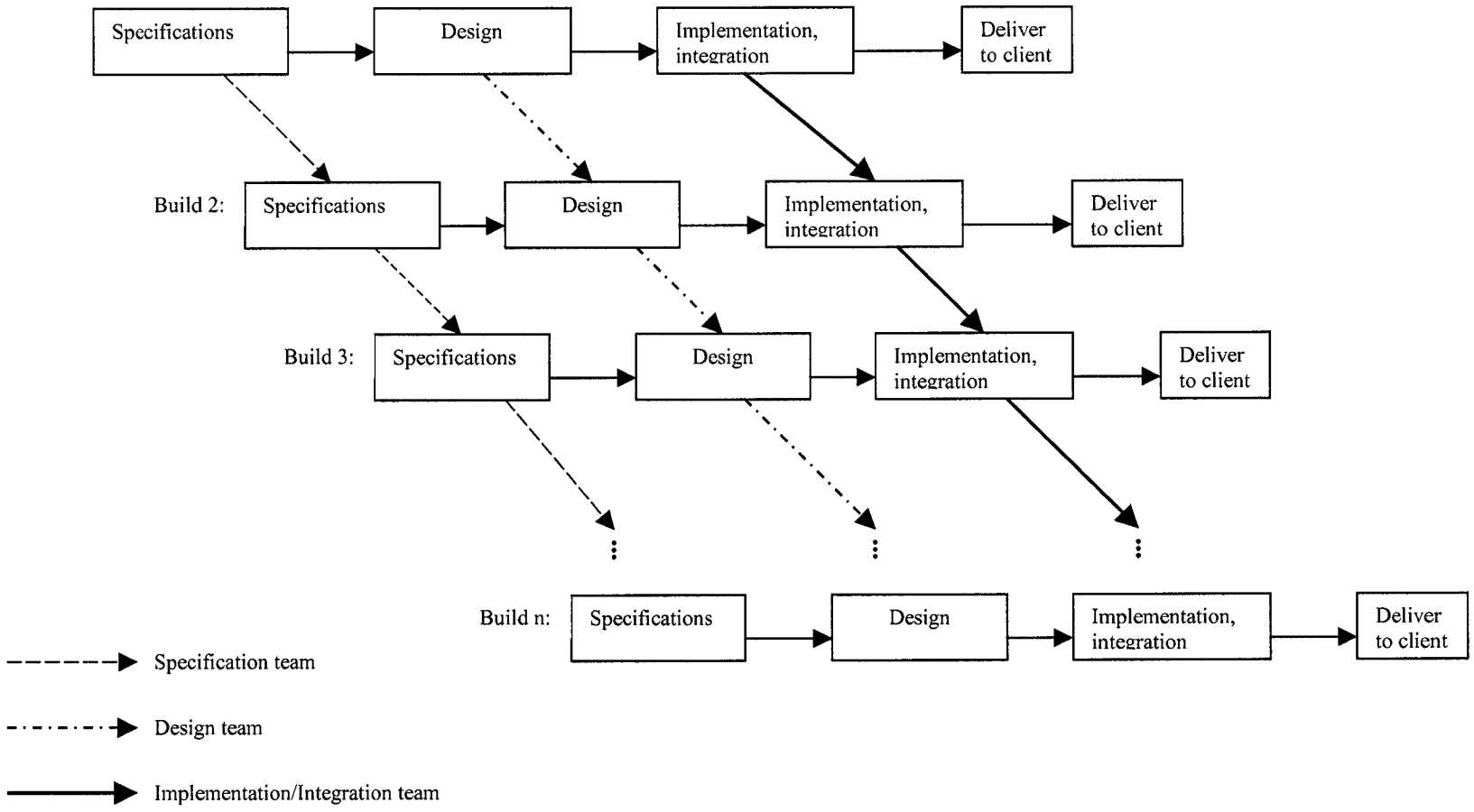

Figure 6: More risky concurrent incremental model

The most critical stage in the software development process is the requirements phase as errors at this stage inevitably lead to later problems in the system design and implementation. The next chapter is an overview of the requirements engineering goals, activities and artifacts.

Chapter 3. Requirements engineering

Software specification is intended to establish what services are required and the constraints on the system's operation and development. This activity is called ***requirements engineering (RE)***.

3.1 Major Objective

The major objective of the requirements engineering is defining the purpose of a proposed system and outlining its external behavior. Requirements generally express what an application is meant to do. They do not attempt to express how to accomplish these functions. The set of requirements for the system should describe the ***functional*** and ***non-functional*** requirements so that they are understandable by system users who don't have detailed technical knowledge. The functional requirements for a system describe the functionality or services that the system is expected to provide. Non-functional requirements, as the name suggests, are those requirements that are not concerned with the specific functions delivered by the system. They may relate to emergent system properties such as reliability, response time.

Requirements engineering traditionally was considered to be restricted to a particular phase of the software development life cycle, which would normally occur before design, implementation, testing and utilization. This view is based primarily on the waterfall model for software development. However, this restrictive view of requirements has evolved in the last two decades. Some of the activities that were traditionally thought of as design, such as feasibility study have become crucial to requirements engineering.

Furthermore, it is now generally accepted that the requirements phase is not confined to the initial stage of the software development, as requirements are continually being refined throughout the life cycle. Indeed work in the maintenance phase involves requirements elicitation in order to assess what the "problems" are and how to fix them. RE revolves around a collection of activities. The activities are explained in the following section.

3.2 Activities

Activities in requirements engineering are diverse in nature and approach. Each of these activities may present those involved in developing and managing requirements with different kinds of problems.

3.2.1 Feasibility study

An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study will decide if the proposed system will be cost-effective from a business point of view. This study should be cheap and quick. The result should inform the decision of whether to go ahead with a more detailed analysis.

3.2.2 Requirements elicitation and analysis

This is the process of deriving the requirements of through observation of existing systems. This may involve the development of one or more different system models and prototypes. These help the analyst understand the system to be specified.

3.2.3 Requirements specification

Requirements specification is the activity of transforming the information gathered during the analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document: user requirements are abstract statements of the system requirements for the customer and end-user of the system; system requirements are a more detailed description of the functionality to be provided.

3.2.4 Requirements validation

Software Requirements Specifications describe the To-Be-Delivered Software System. This activity checks the requirements for realism, consistency and completeness. During this process, errors in the requirements document are inevitable discovered. Errors in requirements specifications have a large effect on the software costs. It is evident that early detection and correction of potential problems during requirement analysis may alleviate much larger problems later on during testing and maintenance. Over a decade ago, [BOE84] claimed that by investing more up-front effort in validating the software requirements, software projects are benefiting of reduced integration and test costs, higher software reliability and maintainability. Requirements analysis and validation continues during elicitation and specification, and new requirements come to light throughout the process. Therefore, the activities of elicitation, analysis, specification and validation are interleaved.

Requirements engineering process leads to the production of requirements document, which is the specification for the system, and is generally referred to as a *requirements specification* or *software requirements specification (SRS)*.

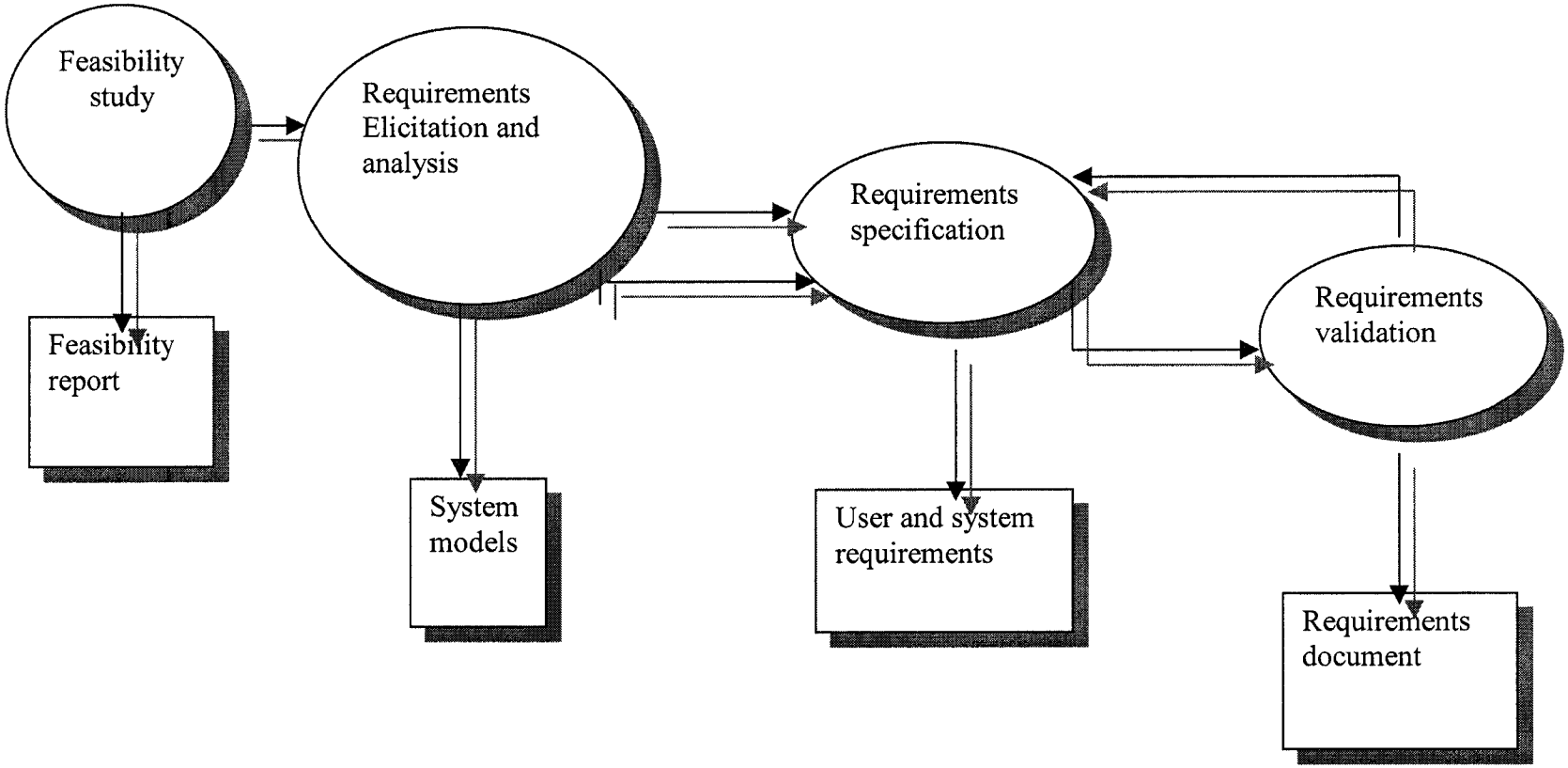

Figure 7: Requirements activities

3.3 Artifact: SRS

This section is describing the final product of requirements engineering, normally referred to as the Software Requirements Specification (SRS). SRS is the official statement of what is required of the system developers. It should include the user requirements for the system and a detailed specification of the system requirements.

This document must be internally consistent with the existing business practice documents, correct and complete in relation to the users' needs, clear to users, customers, designers, and testers and capable of serving as a basis for both design and testing procedures. The SRS may be written by one or more representatives of the supplier, one or more representatives of the customer, or by both.

3.3.1 Roles of SRS

SRS acts as a legal document between the customers and the developers. Project managers utilize it for project planning and management. In many circumstances it is also used as the basis of the end-users manual or generally a document that the customers will use to understand how the delivered system fits together. In addition, SRS must separate essential, desirable, and optional requirements, and identify which items are stable and which might be volatile.

To the customers, suppliers, and other individuals, a good SRS should provide the following:

- ***Establish the basis for agreement between the customers and the suppliers on what the software product is to do.*** The complete description of the functions to be performed by the software specified in the SRS will assist the potential users to

determine if the software specified meets their needs or how the software must be modified to meet their needs.

- ***Reduce the development effort.*** The preparation of the SRS forces the various concerned groups in the customers organization to consider rigorously all of the requirements before design begins and reduces later redesign, recoding, and retesting.
- Careful review of the requirements in the SRS can ***reveal omissions, misunderstandings, and inconsistencies early in the development cycle*** when these problems are easier to correct.
- ***Provide a basis for estimating costs and schedules.***
- ***Provide a baseline for validation and verification.*** Organizations can develop their validation and verification plans much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured.
- ***Facilitate transfer.*** The SRS makes it easier to transfer the software product to new users or new machines.
- ***Serve as a basis for enhancement.*** Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued production evaluation.

Neglecting the roles of requirements document will result ultimately in lack of management control, hostility, under extreme circumstances, lawsuits, and inability to use rigorous software engineering techniques.

3.3.2 Content of SRS

The SRS is a specification for a particular software product, program, or set of programs that performs certain functions in a specific environment. The basic issues that the SRS writer(s) shall address are the following:

a) ***Functionality.*** What is the software supposed to do?

b) ***External interfaces.*** How does the software interact with people, the systems hardware, other hardware, and other software?

c) ***Performance.*** What is the speed, availability, response time, recovery time of various software functions, etc.?

d) ***Attributes.*** What are the portability, correctness, maintainability, security, etc. considerations?

e) ***Design constraints imposed on an implementation.***

Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s), etc.

f) ***Quality attributes***

Considerations of reliability, maintainability, portability, security, etc.

G) ***Others***

Database, operations, site adapting, etc.

3.3.3 Benefits of SRS

The unambiguous and complete specification document should help:

a) Software customers to accurately describe what they wish to obtain;

b) Suppliers to understand exactly what the customer wants.

3.3.4 Characteristics of a good SRS

Since the SRS has a specific role to play in the software development process, the SRS writer(s) should be careful not to go beyond the bounds of that role. This means the SRS:

a) Should correctly define all of the software requirements. A software requirement may exist because of the nature of the task to be solved or because of a special characteristic of the project;

b) Should be complete, i.e., no requirements are overlooked;

c) Should be consistent - no set of individual requirements conflicts with any other set;

d) Should be unambiguous, i.e., there is only one semantic interpretation;

e) Should not describe any design or implementation details. These should be described in the design stage of the project;

f) Should not impose additional constraints on the software. These are properly specified in other documents such as a software quality assurance plan.

The SRS writer(s) should avoid placing either design or project requirements in the SRS. System requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system.

3.3.5 Environment of the SRS

It is important to consider the part that the specified software plays in the total project plan, which is defined in [IEEE93]. The software may contain essentially all the functionality of the project or it may be part of a larger system. In the latter case typically there will be an SRS that will state the interfaces between the system and its software portion, and will place external performance and functionality requirements upon the software portion. Of course the SRS should then agree with and expand upon these system requirements.

3.3.6 Joint preparation of the SRS

The software development process should begin with supplier and customer agreement on what the completed software must do. This agreement, in the form of an SRS, should be jointly prepared. This is important because usually neither the customer nor the supplier is qualified to write a good SRS alone:

a) Customers usually do not understand the software design and development process well enough to write a usable SRS;

b) Suppliers usually do not understand the customer's problem and field of endeavor well enough to specify requirements for a satisfactory system.

Therefore, the customer and the supplier should work together to produce a well-written and completely understood SRS. A special situation exists when a system and its software are both being defined concurrently. Then the functionality, interfaces, performance, and other attributes and constraints of the software are not predefined, but rather are jointly defined and subject to negotiation and change. This makes it more

difficult, but no less important, to meet the characteristics stated in Section 3.3.4. In particular, an SRS that does not comply with the requirements of its parent system specification is incorrect.

3.3.7 SRS Evolution

The SRS may need to evolve as the development of the software product progresses. It may be impossible to specify some details at the time the project is initiated (e.g., it may be impossible to define all of the screen formats for an interactive program during the requirements phase). Additional changes may ensue as deficiencies, shortcomings, and inaccuracies are discovered in the SRS.

There are **two major considerations** in this process:

1. Requirements should be specified as completely and thoroughly as is known at the time, even if evolutionary revisions can be foreseen as inevitable. The fact that they are incomplete should be noted.

2. A formal change process should be initiated to identify, control, track, and report projected changes. Approved changes in requirements should be incorporated in the SRS in such a way as to:

    a) Provide an accurate and complete audit trail of changes;

    b) Permit the review of current and superseded portions of the SRS.

3.3.8 Embedding design in the SRS

A requirement specifies an externally visible function or attribute of a system. A design describes a particular subcomponent of a system and/or its interfaces with other subcomponents. The SRS writer(s) should clearly distinguish between identifying

required design constraints and projecting a specific design. Note that every requirement in the SRS limits design alternatives. This does not mean, though, that every requirement is design.

3.3.9 Embedding project requirements in the SRS

The SRS should address the software product, not the process of producing the software product. Project requirements represent an understanding between the customer and the supplier about contractual matters pertaining to production of software and thus should not be included in the SRS. These normally include items such as:

a) Cost;

b) Delivery schedules;

c) Reporting procedures;

d) Software development methods;

e) Quality assurance;

f) Validation and verification criteria;

g) Acceptance procedures.

Project requirements are specified in other documents, typically in a software development plan, a software quality assurance plan, or a statement of work.

3.4 Requirements Engineering Key Issues

Requirements collection is crucial to the development of successful information systems. The SRS has business and technical considerations added which the customer may or may not be able to provide in the original Requirements List.

Software requirements volatility- There is a strong belief among the researchers, however, that software requirements volatility is one of the major problem areas affecting software productivity. The term requirements volatility is used to describe the evolving nature of software requirements specification. As requirements evolve, it becomes necessary to modify and manage the SRS. Although the field of requirements engineering has attracted a great deal of attention in the recent years, the fact remains that not much has been done to fully identify many important and diverse issues underlying the problem of requirements volatility.

Lack of training in the software engineering community on issues- Most software engineers know very little about requirements engineering notations, techniques, methods and tools. A very small fraction of people involved in requirements elicitation have had formal training to apply these methodologies and tools to realistic problems. There is also a general lack of training in the software engineering community on issues related to elicitation techniques such as interviewing and working effectively with groups. To make matters worse, very few analysts are well-versed in the application domain that they are tackling. Consequently, techniques and tools are often mistreated, giving the impression that requirements engineering practices are not effective. This results in the lack of confidence in the outcome of requirements engineering research.

Representation and notation of requirements- Representation and notation of requirements is yet another major issue that needs to be addressed carefully. One of the most important properties of the notation used to specify requirements is that it must be coherent and comprehensive for both software developers and users. This can be

achieved by setting up a standard for writing the documentation in a software development organization.

To achieve a high level of quality, it is essential that the SRS be developed in a systematic and comprehensive way. International Software Engineering Standards contribute to making a good documentation, and to increase the reliability and effectiveness of the communication between the engineers, and the engineers and users. The overview of international standards is presented in the next Chapter.

Chapter 4. SE Standards

Standards are documented agreements containing technical specifications or other precise criteria to be used consistently as rules, guidelines, or definitions of characteristics, to ensure that materials, products, processes and services are fit for their purpose.

International standardization is well-established for many technologies in such diverse fields as information processing and communications, textiles, packaging, distribution of goods, energy production and utilization, shipbuilding, banking and financial services. It will continue to grow in importance for all sectors of industrial activity for the foreseeable future.

The documents supporting a project vary among organizations, but they correspond roughly to the waterfall phases. ISO 12207 and IEEE software engineering standards are examples of such a document set.

4.1 Why is international standardization needed?

Users have more confidence in products and services that conform to International Standards. When the large majority of products or services conform to the same standards, industry-wide standardization is a condition existing within a particular industrial sector. It results from agreements reached between suppliers, users, and often governments in that industrial sector. They agree on specifications and criteria to be applied consistently in the choice and classification of materials. The aim is to facilitate trade, exchange and technology transfer. This was the origin of the establishment of the International Organization for Standardization (ISO).

4.2 ISO

The International Organization for Standardization (ISO) developed ISO 9000 standard. ISO 9000 as described in [ISO] consisting of a three-step cycle of planning, controlling, and documenting quality in an organization. It provides minimum requirements needed for an organization to meet their quality certification standards.

ISO 9000 is primarily concerned with *quality management*. In plain language, the standardized definition of quality in ISO 9000 refers to all those features of a product (or service), which are required by the customer. Quality management means what the organization does to ensure that its products conform to the customer's requirements.

4.2.1 What is ISO?

The International Organization for Standardization (ISO) is a worldwide federation of national standards bodies from more than 140 countries, one from each country. ISO is a non-governmental organization established in 1947. The mission of ISO is to promote the development of standardization and related activities in the world with a view to facilitating the international exchange of goods and services, and to developing cooperation in the spheres of intellectual, scientific, technological and economic activity.

ISO's work results in international agreements, which are published as International Standards.

4.2.2 ISO's name

Many people will have noticed a seeming lack of correspondence between the official title when used in full, International Organization for Standardization, and the short form,

ISO. Shouldn't the acronym be "IOS"? Yes, if it were an acronym – which it is not. In fact, "ISO" is a word, derived from the Greek *isos*, meaning "equal", which is the root of the prefix "*iso-*" that occurs in a host of terms, such as "isometric" (of equal measure or dimensions) and "isonomy" (equality of laws, or of people before the law).

From "equal" to "standard", the line of thinking that led to the choice of "ISO" as the name of the organization is easy to follow. In addition, the name ISO is used around the world to denote the organization, thus avoiding the plethora of acronyms resulting from the translation of "International Organization for Standardization" into the different national languages of members, e.g. IOS in English, OIN in French (from Organisation internationale de normalisation). Whatever the country, the short form of the Organization's name is always ISO.

4.2.3 ISO's achievements

Below are some examples of ISO standards that have been widely adopted, giving clear benefits to industry, trade and consumers.

- The ISO film speed code, among many other photographic equipment standards, has been adopted worldwide making things simpler for the general user.

- Standardization of the format of telephone and banking cards means the cards can be used worldwide.

- Tens of thousands of businesses are implementing ISO 9000, which provides a framework for quality management and quality assurance. The ISO 14000 series provides a similar framework for environmental management.

- The internationally standardized freight container enables all components of a transport system.
- m, kg, s, A, K, mol, cd are the symbols representing the seven base units of the universal system of measurement known as SI (Système international d'unités).

- Paper sizes.
- Safety of wire ropes
- The ISO international codes for country names, currencies and languages.

4.3.4 How are ISO standards developed?

ISO standards are developed according to the following principles:

- ***Consensus.*** The views of all interests are taken into account: manufacturers, vendors and users, consumer groups, testing laboratories, governments, engineering professions and research organizations.
- ***Industry-wide.*** Global solutions to satisfy industries and customers worldwide.
- ***Voluntary.*** International standardization is market-driven and therefore based on voluntary involvement of all interests in the market-place.

There are three main phases in the ISO standards development process. The need for a standard is usually expressed by an industry sector, which communicates this need to a national member body. The latter proposes the new work item to ISO as a whole. Once the need for an International Standard has been recognized and formally agreed, the first phase involves definition of the technical scope of the future standard. This phase is usually carried out in working groups, which comprise technical experts from countries interested in the subject matter.

Once agreement has been reached on which technical aspects are to be covered in the standard, a second phase is entered during which countries negotiate the detailed specifications within the standard. This is the consensus-building phase.

The final phase comprises the formal approval of the resulting draft International Standard. The acceptance criteria stipulate approval by two-thirds of the ISO members that have participated actively in the standards development process, and approval by 75% of all members that vote, following which the agreed text is published as an ISO International Standard.

Most standards require periodic revision. Several factors combine to render a standard out of date: technological evolution, new methods and materials, new quality and safety requirements. To take account of these factors, ISO has established the general rule that all ISO standards should be reviewed at intervals of not more than five years. On occasion, it is necessary to revise a standard earlier.

To date, ISO's work has resulted in some 12 000 International Standards, representing more than 300 000 pages in English and French (terminology is often provided in other languages as well).

4.3 IEEE Standards

Below is a description of each document in the IEEE set, with references to full descriptions as described in [BRA01]. Other standards bodies are organized in a similar way.

SVVP: IEEE std 1095-1993 the Software Verification and Validation plan. This plan explains the manner in which the project steps are to be checked, and the product is to be checked against its requirements. Verification is the process of checking that an application is built in a correct manner; validation checks that the right product has been built.

SQAP: IEEE std 730-1998 the Software Quality Assurance plan. This plan specifies the manner in which the project is to achieve its quality goals.

SCMP: IEEE std 828-1998 the Software Configuration Management Plan. The SCMP explains how and where the documents and code, and their various versions, are stored, and how they fit together. It is not advisable to get started without such plan because the very first document generated is bound to change, and we must understand how this change will be managed before we begin writing the document. Medium to large companies generally try to work out configuration management details on behalf of all their projects, and engineers need only learn to fellow the designated procedures in the official SCMP and use the designated tools.

SPMP: IEEE std 1058-1998 the Software Project Management plan. This plan explains the manner in which the project is to be conducted. Typically, it cites a known development process, the company standard process.

SRS: IEEE std 830-1998 the Software Requirements Specification plan. This document states the requirements for the application and is a kind of contract and guide for the customer and the developers.

SDD: IEEE std 1016-1998 The Software Design Document. The SDD describes the architecture and design details of the application. Typically, diagrams such as object models and data flow diagrams are used.

STD: IEEE std 829-1998 the Software Test Documentation. This document describes the way in which the application and its parts are to be tested.

Projects sometimes employ documents additional to those described above. The documentation for iterative development organized in at least two ways. Some documents can contain a version for each iteration. Another way is to add appendices that account for progress on the application.

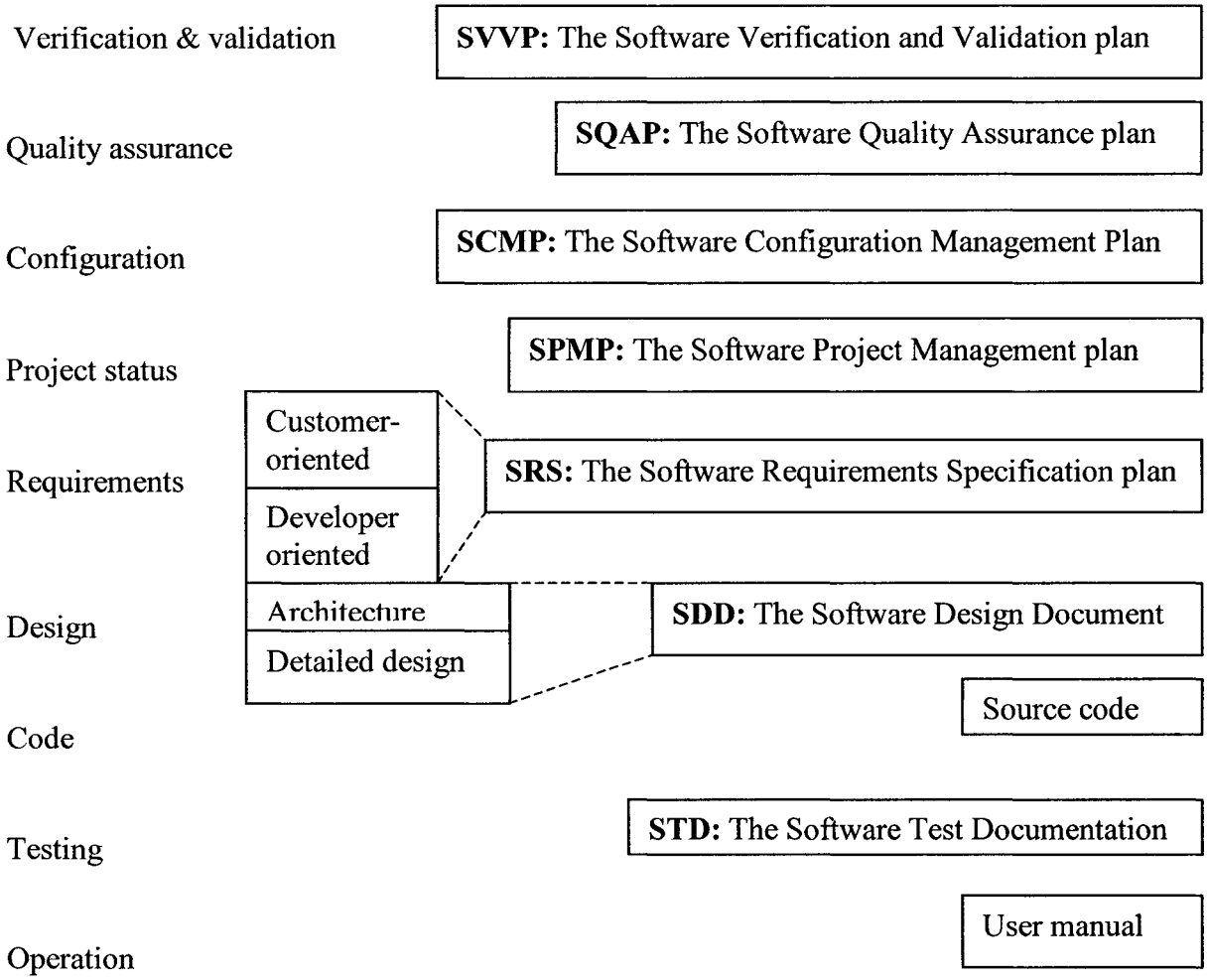

Figure 8: Standards for Documentation

In this major report, we have chosen to follow the IEEE standard for software requirements specification phase. The content of the IEEE SRS document is described in the next Chapter.

Chapter 5. IEEE Std.830-1998 Software Requirements Specification Template

This chapter discusses each of the essential parts of the SRS as described in [IEEE93]. These parts are arranged in Figure 7 in an outline that can serve as an example for writing an SRS. While an SRS does not have to follow this outline or use the names given here for its parts, a good SRS should include all the information discussed here.

1. Introduction
1.1 Purpose
1.2 Scope
1.3 Definitions, acronyms, and abbreviations
1.4 References
1.5 Overview
2. Overall description
2.1 Product perspective
2.2 Product functions
2.3 User characteristics
2.4 Constraints
2.5 Assumptions and dependencies
3. Specific requirements

Figure 9: The software specification template

5.1 Introduction (Section 1 of the SRS)

The introduction of the SRS should provide an overview of the entire SRS. It should contain the following subsections:

5.1.1 Purpose (1.1 of the SRS)

This subsection should delineate the purpose of the SRS, and specify the intended audience for the SRS.

5.1.2 Scope (1.2 of the SRS)

This subsection should:

a) Identify the software product(s) to be produced by name (e.g., Host DBMS, Report Generator, etc.);

b) Explain what the software product(s) will, and, if necessary, will not do;

c) Describe the application of the software being specified, including relevant benefits, objectives, and goals;

d) Be consistent with similar statements in higher-level specifications (e.g., the system requirements specification), if they exist.

5.1.3 Definitions, acronyms, and abbreviations (1.3 of the SRS)

This subsection should provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS.

5.1.4 References (1.4 of the SRS)

This subsection should:

a) Provide a complete list of all documents referenced elsewhere in the SRS;

b) Identify each document by title, report number (if applicable), date, and publishing organization;

c) Specify the sources from which the references can be obtained.

5.1.5 Overview (1.5 of the SRS)

This subsection should describe what the rest of the SRS contains, and explain how the SRS is organized.

5.2 Overall description (Section 2 of the SRS)

This section of the SRS should describe the general factors that affect the product and its requirements. This section consists of six subsections described below:

5.2.1 Product perspective (2.1 of the SRS)

This subsection of the SRS should put the product into perspective with other related products. A block diagram showing the major components of the larger system, interconnections, and external inter-faces can be helpful. This subsection should also describe how the software operates inside various constraints. For example, these constraints could include:

a) ***System interfaces.*** This should list each system interface and identify the functionality of the software to accomplish the system requirement and the interface description to match the system;

b) ***User interfaces.*** This should specify the following:

- The logical characteristics of each interface between the software product and its users.
- All the aspects of optimizing the interface with the person who must use the system.

*c) **Hardware interfaces.*** This should specify the logical characteristics of each interface between the software product and the hardware components of the system.

*d) **Software interfaces.*** This should specify the use of other required software products, and interfaces with other application systems. For each required software product, the following should be provided:

- Name;
- Mnemonic;
- Specification number;
- Version number;
- Source.

*e) **Communications interfaces.*** This should specify the various interfaces to communications such as local network protocols, etc.

*f) **Memory.*** This should specify any applicable characteristics and limits on primary and secondary memory.

g) ***Operations.*** This should specify the normal and special operations required by the user.

h) ***Site adaptation requirements.*** This should:

- Define the requirements for any data or initialization sequences that are specific to a given site.
- Specify the site or mission-related features that should be modified to adapt the software to a particular installation.

5.2.2 Product functions (2.2 of the SRS)

This subsection of the SRS should provide a summary of the major functions that the software will perform:

a) The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time;

b) Textual or graphical methods can be used to show the different functions and their relationships.

5.2.3 User characteristics (2.3 of the SRS)

This subsection of the SRS should describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise.

5.2.4 Constraints (2.4 of the SRS)

This subsection of the SRS should provide a general description of any other items that will limit the developer's options. These include:

a) Regulatory policies;

b) Hardware limitations (e.g., signal timing requirements);

c) Interfaces to other applications;

d) Parallel operation;

e) Audit functions;

f) Control functions;

g) Higher-order language requirements;

h) Signal handshake protocols (e.g., XON-XOFF, ACK-NACK);

i) Reliability requirements;

j) Criticality of the application;

k) Safety and security considerations.

5.2.5 Assumptions and dependencies (2.5 of the SRS)

This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS.

5.2.6 Apportioning of requirements (2.6 of the SRS)

This subsection of the SRS should identify requirements that may be delayed until future versions of the system.

5.3 Specific requirements (Section 3 of the SRS)

This section of the SRS should contain all of the software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system, and all functions performed by the system in response to an input or in support of an output. As this is often the largest and most important part of the SRS, the following principles apply:

a) Specific requirements should be stated in conformance with all the characteristics of a good SRS;

b) Specific requirements should be cross-referenced to earlier documents that relate;

c) All requirements should be uniquely identifiable;

d) Careful attention should be given to organizing the requirements to maximize readability.

IEEE standard suggests five different templates for the SRS specific requirements listed by mode, user, class, object or by feature. The most appropriate template for specifying the object-oriented software is the one listed by feature. A feature is an externally desired service by the system that may require a sequence of inputs to affect the desired result. For example, in a telephone system, features include local call; call forwarding, and conference call. Each feature is generally described in a sequence of stimulus-response pairs. In object oriented approach, and more specifically in the RUP, each system's feature is abstracted as a use of the system called use case.

The IEEE template for Section 3 (Specific requirements) is given in Section 5.3.1. The use-case approach is described in detain in section 5.3.2.

5.3.1 IEEE Template of SRS Section 3 organized by feature

3. Specific requirements

3.1 External interface requirements

3.1.1 User interfaces

3.1.2 Hardware interfaces

3.1.3 Software interfaces

3.1.4 Communications interfaces

3.2 System features

3.2.1 System Feature 1

3.2.1.1 Introduction/Purpose of feature

3.2.1.2 Stimulus/Response sequence

3.2.1.3 Associated functional requirements

3.2.1.3.1 Functional requirement 1

. . .

3.2.1.3.*n* Functional requirement *n*

3.2.2 System feature 2

. . .

3.2.m System feature m

. . .

3.3 Performance requirements

3.4 Design constraints

3.5 Software system attributes

3.6 Logical Database Requirements

3.7 Other requirements

External interfaces. This should be a detailed description of all inputs into and outputs from the software system. It should complement the interface descriptions in 2.1 of the SRS and should not repeat information there. It should include both content and format as follows:

a) Name of item;

b) Description of purpose;

c) Source of input or destination of output;

d) Valid range, accuracy, and/or tolerance;

e) Units of measure;

f) Timing;

g) Relationships to other inputs/outputs;

h) Screen formats/organization;

i) Window formats/organization;

j) Data formats;

k) Command formats;

l) End messages.

Functions. Functional requirements should define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These include:

a) Validity checks on the inputs;

b) Exact sequence of operations;

c) Responses to abnormal situations, including

    1) Overflow;

    2) Communication facilities;

    3) Error handling and recovery.

d) Effect of parameters;

e) Relationship of outputs to inputs, including

    1) Input/output sequences;

    2) Formulas for input to output conversion.

Performance requirements. Specify static and dynamic numerical requirements placed on the software or on human interaction with the software. Static numerical requirements may include the number of terminals to be supported, the number of simultaneous users to be supported, and the amount and type of information to be handled. Dynamic numerical requirements may include the number of transactions and tasks and the amount of data to be processed within certain time period for both normal and peak workload conditions. All of these requirements should be stated in measurable form.

Logical database requirements. This should specify the logical requirements for any information that is to be placed into a database. This may include the following:

a) Types of information used by various functions;

b) Frequency of use;

c) Accessing capabilities;

d) Data entities and their relationships;

e) Integrity constraints;

f) Data retention requirements.

Design constraints. Specify requirements imposed by standards, hardware limitations, etc.

Standards compliance. This subsection should specify the requirements derived from existing standards or regulations. They may include the following:

a) Report format;

b) Data naming;

c) Accounting procedures;

d) Audit tracing.

Software system attributes. The following items provide a partial list of system attributes that can serve as requirements that should be objectively verified:

- ***Reliability*** - Specify the factors needed to establish the software's required reliability
- ***Availability*** - Specify the factors needed to guarantee a defined level of availability
- ***Security*** - Specify the factors that will protect the software from accidental or malicious access, misuse, or modification. These factors may include cryptography, activity logging, restrictions on intermodule communications and data integrity checks
- ***Maintainability*** - Specify attributes of the software that relate to ease of maintenance. These requirements may relate to modularity, complexity, or interface design. Requirements should not be placed here simply because they are thought to be good design practices.
- ***Portability*** - Specify attributes of the software that relate to the ease of porting the software to other host machines and/or operating systems.

Supporting information. The supporting information makes the SRS easier to use. It includes the following:

- ***Table of contents and index*** - The table of contents and index are quite important and should follow general compositional practices
- ***Appendixes*** - The appendixes are not always considered part of the actual SRS and are not always necessary. They may include:

  a) Sample input/output formats, descriptions of cost analysis studies, or results of user surveys;

  b) Supporting or background information that can help the readers of the SRS;

  c) A description of the problems to be solved by the software;

  d) Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements.

  When appendixes are included, the SRS should explicitly state whether or not the appendixes are to be considered part of the requirements.

Disadvantages of IEEE Std.830-1998

The IEEE templates for specific requirements are not easily adaptable to the new object-oriented approach to software development mainly because the standard has been created with structured modeling in mind. In the following section, we are proposing a new template for specific requirements that follows the widely recognized and used in the industry Unified Process approach to the analysis and specification of the functional requirements, based on the *use-case model* and the *domain model.* Our template for specific requirements (Section 3.2 of SRS Document) is explained below.

5.4 New Template for Section 3 by Use-Cases

5.4.1 Our Approach

We are considering the domain model as a ***static view*** on the system's structure, and the use-case model as a ***dynamic view*** on its functionality.

Domain Model

Modeling a system involves identifying the things, or *concepts*, that are important to the environment where the system will function, i.e., system's *domain*. These concepts from the vocabulary of the system's domain are abstracted in the ***domain model*** as conceptual classes. Therefore, we can consider the domain model as a visualization of the concepts in the real-world domain. The relations between the real world things are abstracted as relationships between conceptual classes in the domain model. The dynamics of the relationships between the conceptual classes are described in the ***use case model***.

Use-Case Model

A ***use case*** is a sequence of actions that provide a measurable value to an actor. Another way to look at it is that a use case describes a way in which a real-world actor interacts with the system. An essential use-case is a simplified, abstract, generalized use case that captures the intentions of a user in a technology- and implementation-independent manner. An essential use case is a structured narrative, expressed in the language of the application domain and of users, comprising a simplified, generalized, abstract, technology-free and implementation-independent description of one task or interaction.

An essential use case is complete, meaningful, and well designed from the point of view of users in some role or roles in relation to a system and that embodies the purpose or intentions underlying the interaction. A **use case** is "a specific flow of events through the system, that is, an instance" [JCJO95]. Using the concept of a class as the set of all items, which share a collection of similar characteristics, it is suggested that many similar courses of events be grouped into a "use-case class." (Note that this definition, i.e., a class is a *set* of instances, is not the same definition of class that is used in a Smalltalk, or C++.)

Purpose: Shows the system from the client's or user's view

- It is simple - simplifies communication between the client and the designers
- Allows the designers to focus on the requirements of the system, rather than on its implementation
- Produce a system fit for purpose

Participants/Roles

The participants in a use case are not physical users, they are the roles that a physical user might have with respect to a system. For example, if we're doing a time-sheet authorization system, two roles come to mind immediately: employees (who fill out time sheets) and managers (who authorize them). The fact that the same physical person might take on both roles at some juncture is irrelevant.

Scenario

Scenarios are small narrative descriptions of someone working through the use case. Use a fly-on-the-wall approach: describe what happens as if you're a fly on the wall observing

the events transpire. Each scenario illustrates the message interactions between entities in order to fulfill the required functionality of the system.

What is an Actor? An **actor** is "a role that someone or something in the environment can play in relation to the business" [LAR02]. Alternatively, as defined by [JCJO95] an actor represents "everything that needs to exchange information with the system," and defines actors as "everything that interacts with the system." An "individual actor" (sometimes referred to as a "user") is defined to be an instance of class actor. Further, the same person (or other item) can assume more than one role.

Use-case diagram

Use case diagram shows a set of use cases and actors and their relationship. It illustrates the static use-case view of the system. Also, it is important in organizing and modeling the behaviors of the system.

5.4.2 Modified Template

3. Specific requirements

3.1 External interface requirements

3.1.1 User interfaces

3.1.2 Hardware interfaces

3.1.3 Software interfaces

3.1.4 Communications interfaces

3.2 Domain Model

3.2.1 Objects

3.2.1.1 Object 1

3.2.1.1.1 Name

3.2.1.1.2 Attribute 1: Name, Type

3.2.1.1.3 Attribute 2: Name, Type

...

3.2.1.2 Objet n

3.2.2 Relations

3.2.2.1 Relation 1

3.2.2.1.1 Type relation

3.2.2.1.2 Label relation

3.2.2.1.3 Multiplicity

3.2.2.1.4 From object

3.2.2.1.5 To object

...

3.3 Use Case Model

3.3.1 Actors

3.3.1.1 Actor1: Name, Role, Type

...

3.3.2 Use Cases

3.3.2.1 Use case 1

3.3.2.1.1 Purpose of use-case

3.3.2.1.2 Pre-condition

3.3.2.1.3 Post-condition

3.3.2.1.4 Priority

3.3.2.1.5 Actor description: Name, Role, Type

3.3.2.1.6 Scenarios

3.3.2.1.6.1 Scenario 1 (Main)

...

3.3.2.1.6.*n* Scenario *n*

3.3.2.2 Use Case 2

...

3.3.2.*m* Use Case *m*

3.4 Performance requirements

3.5 Design constraints

3.6 Software system attributes

3.7 Logical Database Requirements

3.8 Other requirements

With UML, the connection between the static and dynamic views is not obvious; therefore the consistency between the use-case and the domain models needs to be checked formally. We propose a definition, validation and measurement of the consistency between the static and dynamic views based on their representation in XML, in Chapter 7. Our choice of XML to formalize the use-case and domain models is justified by the heterogeneous database architecture because XML facilitates the use of such heterogeneous databases. XML is introduced in the next chapter.

Chapter 6. XML

The extraordinary growths of the World Wide Web has been fueled by the ability it gives authors to easily and cheaply distribute electronic documents to an international audience. To address the requirements of commercial Web publishing and enable the further expansion of Web technology into new domains of distributed document processing, the World Wide Web Consortium has developed an Extensible Markup Language (XML) for applications that require functionality beyond the current Hypertext Markup Language (HTML). This chapter is a brief introduction to XML.

6.1 What is XML?

XML is a markup language for structured documentation. Structured documents are documents that contain both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption, etc.) [WAL].

XML offers some real advantages:

1. ***XML Is the Extensible Markup Language***

   Unlike the tags used to mark up HTML documents, XML tags are not predefined so it can be used to represent any form of data. It allows the author to define his own tags and his own document structure.

2. ***XML Is Based on Unicode***

This means that it is fully internationalized out of the box. There are relatively few constraints on the characters that may be used, freeing document authors and processing systems to work in the language that is most convenient for their locale.

3. XML Formats Are Self-Describing

Because each piece of information in XML is identified with a name, it's easy to understand.

4. XML Is an Open Standard

There are no proprietary aspects that open source developers will have to work around or reverse-engineer.

6.2 Basics concepts of XML

Like HTML, XML makes use of *tags* (words bracketed by '<' and '>') and *attributes* (of the form `name="value"`). While HTML specifies what each tag and attribute means, and often how the text between them will look in a browser, XML uses the tags only to delimit pieces of data, and leaves the interpretation of the data completely to the application that reads it.

The basic building blocks of XML documents are elements and attributes. So let's take a look at these.

Elements

Elements are delimited by angle brackets. Elements that have content appear in the form <element-name> (<burns> in the example below) followed by some optional content

followed by </element-name>. The first form is the start tag; the second is the end tag. In XML, every non-empty element must have explicit start and end tags, and they must be properly nested.

Empty elements (<applause/> in this example) have a modified syntax. While most elements in a document are wrappers around some content, empty elements have the form <element-name/>. The trailing slash before the closing angle bracket indicates that the element has no content and consequently no end tag is allowed.

Since XML documents do not require a document type declaration, without this clue it could be impossible for an XML parser to determine which tags were intentionally empty and which had been left empty by mistake.

Another alternate syntax has been introduced for empty elements that allow the end-tag to be present in a very recent modification to the specification. Under this syntax, <applause></applause> would be acceptable as well.

6.3 How is XML defined?

XML is defined by four specifications:

- **XML, the Extensible Markup Language -** defines the syntax of XML. The XML specification is the primary focus of this article.
- **XML Schemas-** XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents [W3C] (see appendices).
- **XSL, the Extensible Style Language -** will define a standard stylesheet language for XML.

- **XLL, the Extensible Linking Language** - defines a standard way to represent links between resources. In addition to simple links, like HTML's tag, XLL has mechanisms for links between multiple resources and links between read-only resources.
- **XUA, the XML User Agent** - will help standardize XML User Agents (browsers, etc.). Work on this specification has not yet started.

6.4 Characteristics of XML

XML is for structuring data. XML is a set of rules for designing text formats that let you structure your data. XML is not a programming language, and you don't have to be a programmer to use it or learn it. XML makes it easy for a computer to generate data, read data, and ensure that the data structure is unambiguous [W3C].

XML is modular. XML allows you to define a new document format by combining and reusing other formats. Since two formats developed independently may have elements or attributes with the same name, care must be taken when combining those formats (does "<p>" mean "paragraph" from this format or "person" from that one?) [W3C].

XML is the basis for RDF and the Semantic Web. W3C's Resource Description Framework [RDF] is an XML text format that supports resource description and metadata applications, such as music playlists, photo collections, and bibliographies.

XML is license-free, platform-independent and well supported. By choosing XML as the basis for a project, we gain access to a large and growing community of tools and engineers experienced in the technology.

The above listed characteristics of XML are justifying our choice of XML to represent formally the content of the SRS document. The formalism allows for validation of consistency between the specific requirements as described in Chapter 7.

6.5 The Structure of Requirement Specification document in XML

Most of the XML elements used are self- explanatory (refer to chapter 5). However below is the description of the one, which does not figure in chapter 5.

<Requirement_specification_template> the root element of the XML file

(Note that Every XML document must have one and only one root element)

<Domain_Model>describe the domain model

<Object_name>describe object name

<Name_Attribute>describe name attribute for object

<Type_Attribute>describe type of the attribute

<Type_Relation> describe the relation type

<Label_Relation>describe label relation

<Multiplicity>give the multiplicity

<System_Uses> is a use case description

<Name_Actor> provide actor name

<Role_of_Actor>provide role of actor

<Type_Actor> provide type of actor

<Reference_Diagram>provide the number of the diagram

<Label_Scenario>describe a scenario

<Type_Scenario>provide type scenario

<Set_objects>describe the objects participating in a specific scenario

<Label_Message>describe the message interchanging between two instances of classes

<Number_of_message>provide a message according to a sequence diagram

<Part1>is an introduction of table of content description

<Part2> describe overall description

<Part3>provide a specific requirements description

Our approach to present the SRS template in XML is illustrated on the Air Traffic Management (ATM) System case study given in the Appendix 1.

Chapter 7. Consistency Validation based on XML

This chapter summarizes the work published in [MO03]. The analysis modeling and the verification of its consistency is a necessity, not a luxury. Validating the domain model versus use case model to check the consistency between them will help to reduce the incoherence in the analysis phase, which will lead later to fewer errors in the development of the software. The measures of consistency proposed in this chapter would provide a quantitative feedback on the actual level of consistency between the static and dynamic views on the functional requirements, and therefore assist the software engineers in controlling the quality of the analysis model.

7.1 Approach

We check the consistency between the domain and use-case models according to the following criteria:

P1. Each class presented in the ***domain model*** should participate in at least ***one scenario.***

P2. The ***associations*** between conceptual classes imply interchange of messages between the corresponding conceptual classes in at least ***one scenario.***

P3. Any ***message*** interchanged between two conceptual classes C1 and C2 in a scenario should imply the presence of an ***association*** between C1 and C2 in the domain model.

In the best scenario, all the above listed criteria ***P1***, ***P2*** and ***P3*** would hold for the analysis model. Any deviation from the listed characteristics would indicate

inconsistency in the description of the system's requirements, and therefore a potential problem in the system's development process.

7.2 Measures of Consistency

For each consistency criteria we are defining a simple measure of the level of consistency as follows:

P1. **Class Consistency Ratio** = (Number of classes for which ***P1*** holds) / Total # of classes

P2. **Associations Consistency Ratio** = (Number of associations for which ***P2*** holds) / Total # of associations

P3. **Messages Consistency Ratio** = (Number of messages for which ***P3*** holds) / Total # of different messages

We have presented three simple, ordinal-scale measures of consistency. The rank of measures' values is [0..1], where higher value indicate higher level of consistency. In general, a value less than one would indicate the presence of inconsistency as defined in the corresponding criteria. The collection of the measurement data requires a tool for its automation. The automated consistency validation and measurement procedure is a 5-step activity described below [FP97].

7.3 Consistency Measurement Procedure

Step 1: Map the Domain Model to XML database.

Object-oriented analysis is concerned with creating a description of the domain from the perspective of classification by objects. A decomposition of the domain involves an

identification of the concepts (objects), attributes, and relations that are considered worthy. The above decomposition justifies the structure of its description in XML

Step 2: Map the Use Case Model to XML database

Each use case consists of external actor(s), and the set of scenarios describing the functionality for a specific use.

Step 3: Merge the two databases

Database #1, which describes the domain model, and *database #2*, which describes the use case model, are merged in one XML file (*Figure 8)*

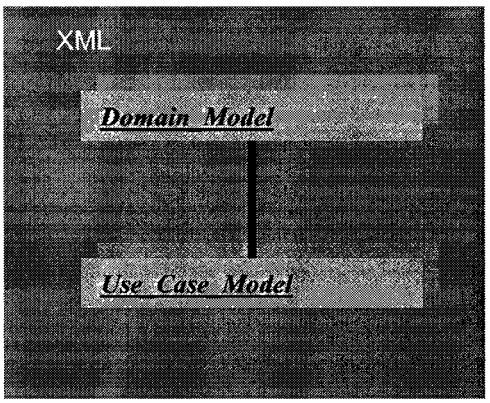

Figure 10: Merge the two databases

Step 4: Define a set of queries to verify the consistency criteria *P1, P2, P3*

Query 1 (*RE: P1*)**:** Given a specific conceptual class in the domain model, find all the scenarios where it is participating

Justification: There are no useless classes in the Domain Model

Query 2 (*RE: P2*)**:** Given a relation in the domain model, verify that the connected classes are interchanging a message in at least one scenario

Justification: There are no useless associations in the domain model

Query 3 (*RE: P3*)**:** Given a message between two instances of classes C1 and C2 in a scenario, verify that there is a relation between the classes C1 and C2 in the domain model

Justification: There are no missing associations in the domain model

Step 5: Collect and report the queries' results (successful or not) applying the consistency measures.

In the following section we present the architecture and the functionality of the consistency measurement tool.

7.4 Tool's Architecture

We have proposed a three-tiers architecture using XML as backend database, HTML and XSL for displaying the data, CGI for communication between database and interface. For validation purposes we have created a heterogeneous XML database containing the descriptions of the domain model and use case model. Bellow the description of the tiers is given:

- **Client-tier:** Responsible for the presentation of data, and receiving user events.
- **The business Logic tier**: Which involves the rules that govern application processing, connects the user in tier 1 (Client tier) with the data in tier 3 (Data storage tier).

- **The Data Storage tier:** Responsible for the data storage. It is the place where it is decided how the data is saved. Our data is saved in an XML file.

The advantages for the architecture described above are:

- ***Easy to change***: you can decide to switch from desktop applications to web based applications by just changing the UI layer (a small part of the application).
- ***Easy to reuse***: if another application is developed for the same domain, it can use a big part of the business layer.
- ***Easy to develop***: each layer can be developed by separate teams, and focus only on theirs specific problems.

Chapter 8. Conclusion and Future Research

In this Major Report, we have described a standard, comprehensive, step-by-step guide for writing a software requirements document, which are identifiable and meaningful to both the customer and the developer.

We have proposed a flexible requirements specifications template, easily adaptable to object-oriented development for specific organizations. XML was used to formally represent the requirement specification document, so that the structured SRS documents could be used over the web. The approach is illustrated on the Air Traffic Management (ATM) System case study.

We have defined theoretically valid measures that report the level of consistency achieved in the requirements between domain and use case models. The measurement mechanism is based on the XML representation. The measurement data is collected for the different criteria of the consistency, as defined in Chapter 7. This early feedback from measurement would allow the discovery of errors that may lead to further problems at the design and implementation.

The directions of our future work include the definition of more consistency validation criteria for the analysis phase, and the automation of requirements documentation capturing using a three-tiers architecture using XML as backend database, HTML and XSL for displaying the data, CGI for communication between database and interface. There is no need for writing an editor to read the software specification template tags; Internet explorer from Microsoft Corporation can be used as web browser.

Bibliography

[BEC99] Beck, Kent. "*eXtreme Programming eXplained: Embrace Change,*" Addison Wesley, Second printing, 1999.

[BER92] Berlack, H.R, *"Software Configuration Management,"* Wiley, 1992.

[BOE84] Boehm, B. W., "*Verifying and Validating Software Requirements and Design Specifications,*" IEEE Software, vol. 1 NO 1, 1984.

[BRA01] BRAUDE Eric, J. "S*oftware engineering: An object-Oriented Perspective*" Wiley, 2001.

[BRJ99] Booch Grady, Rrumbaugh James and Jacobson Ivar, "*The Unified modeling language: User Guide*", Addison-Wesley, 1998.

[CEA00] Canadian Engineering Accreditation Board – Accreditation Criteria and Procedures, 2000: http://www.site.uottawa.ca/~petriu/CEG4392-2002-L1.pdf

[DDNLS00] Deitel M., Deitel J., Nieto T., Lin T., Sadh P., "*XML How to Program*" 1st Edition, Prentice Hall, 2000.

[FP97] Fenton, N and Pleeger, S. "*Software Metrics: A Rigorous & Practical Approach*", Chapman & Hall, 1997.

[GJM91] Ghezzi, Jazayeri and Mandrioli "*Fundamentals of Software Engineering*", Second edition, Prentice Hall, 1991.

[IEEE93] IEEE Std 830-1998 (Revision of IEEE Std 830-1993)

[IEEE91] IEEE Std 610.12-1990, 1991

[ISO] ISO: http://www.iso.ch/iso/en/ISOOnline.frontpage

[JCJO95] Jacobson I., Christerson M., Jonsson P. and Overgaard G., "*Object-Oriented Software Engineering: A Use case Driven Approach*" Addison-Wesley 1995.

[LAR02] LARMAN CRAIG, "Applying UML And Patterns" second edition, PH PTR, 2002.

[MO03] Meridji, K., Ormandjieva, O. "*Measuring Consistency of the Analysis Model: An XML Approach*," Accepted at the 13th International Workshop on Software Measurement IWSM2003.

[PPE96] Practice of Professional Engineering, 1996: http://www.site.uottawa.ca/~petriu/CEG4392-2002-L1.pdf

[PRE97] PRESSMAN, R. S, "*Software Engineering: A Practitioner's approach*" 4th edition, McGraw-Hill, 1997.

[RM01] Ray Erik T., Maden Christopher R., "*Learning XML*" O'Reilly & Associates; 1st edition, 2001.

[RDF] Resource Description Framework: http://www.w3.org/RDF/

[SCH02] Schach, S.R., "*Object-Oriented and Classical: Software Engineering*" Fifth edition, 2002.

[SCH90] Schach, S.R., Vanderbilt University, (in *"Software Engineering*," Aksen Assoc., 1990).

[SOM01] Sommerville Ian., "S*oftware engineering,"* 6th edition, Addison Wesley, 2001.

[SEYP] Software Engineering Yellow Pages:
http://www.practicalprocess.com/seyp/definition.html

[WAL] Walsh Norman:
http://www.linux-mag.com/2001-07/xml_basics_01.html

[W3C] W3C: http://www.w3.org/XML/1999/XML-in-10-points

[WID00] Widrig L., "*Managing Software Requirements: A Unified Approach* ", Addison Wesley, 2000.

[WKCJ00] Wlliams, Kesseler, Cunningham, and Jeffries, "Strengthening the Case for Pair Programming" 2000:
http://collaboration.csc.ncsu.edu/laurie/Papers/ieeeSoftware.PDF

Appendix 1. Case Study

This appendix illustrates our template for SRS documents introduced in Section 5.4.2, on the Air Traffic Management (ATM) System case study. The ATM SRS document states the requirements of a computerized system to support airplanes to land and leave airports safely at scheduled times. This document is intended to be as brief as possible, and focuses only on the important requirements including the timing, safety, and liveness requirements. The System Evolution requirements are added to illustrate how the system may be enhanced in the future.

- <Requirement_specification_template>
  - <Introduction>
    This is a Requirements Specification document for the Air Traffic Management (ATM) System. It states the requirements of a computerized system to support airplanes to land and leave airports safely at scheduled times. This document is intended to describe the main functions of the system that will support airplanes landing and departing from the airports safely and as scheduled.
    - <Purpose>
      The purpose of this document is to define the requirements for the ATM System and its uses. The intended audiences for this document are:
      <Section id="**1**">**The users of the ATM System product, so that they may review and provide input into the requirements**</Section>
      <Section id="**2**">**The procurers of the product, so that they may agree upon the requirements of the ATM System, leading to an approval of this document.**</Section>

<Section id="3">**The system design team, as this document will serve as the basis for the design of the system.**</Section>
<Section id="4">**The test team, as this document will serve as the basis for the verification and validation of the system.**</Section>
</Purpose>
- <Scope>
<Section id="1">**This is the Requirements document for the Air Traffic Management (ATM) System**</Section>
<Section id="2">**ATM system should handle arrival and departure of airplanes at scheduled time, and provide schedule to users of ATM system**</Section>
<Section id="3">**This document is intended to be as brief as possible, and focuses only on the important requirements such as the timing, safety, and liveness requirements.**</Section>
</Scope>
- <Definitions_acronyms_and_abbreviations>
<Section id="1">**Good weather condition are when temperature is between -10C to 45C, speed of wind is less than 20MPH, and no storm and visibility is at least 5 Miles**</Section>
<Section id="2">**Congestion is defined as delay of all flights in scheduling takeoff and landing that exceeds 4 hours.**</Section>
<Section id="3">**ATM system database manages information needed to insure operation of the ATM system**</Section>
<Section id="4">**Flight-scheduling database already exist and provideinformation on the arrival and departure of flights and it is up to date, 24 hours a day.**</Section>
<Section id="5">**Runways are roads used by airplane during landing or takeoff where airplane can reduce /**

increase speed needed for safe landing / takeoff.</Section>
<Section id="**6**">**Sensors are devices used for status detection of some part of Application domain and they transmit information used by Machine domain**</Section>
<Section id="**7**">**Operator is person who operates with ATM system**</Section>
<Section id="**8**">**Traveler is person who arrives of departure by one of the flights handled by ATM system**</Section>
<Section id="**9**">**Pilot is person who handle all operations of airplane (flying, landing, takeoff, crashing :)**</Section>
<Section id="**10**">**ATM system errors can be happening in case of incorrect operation by Operator, Pilot or ATM system.**</Section>
<Section id="**11**">**Alarm will be set if error exist. It can be reset by Operator or by System upon error correction**</Section>
<Section id="**12**">**The system queue is part of ATM system which contains takeoff and lending flights, based on the First in First out algorithm for each runaway that are acknowledged by operator**</Section>
<Section id="**13**">**Normal operations are defined to be the following: no flight congestion has occurred and weather conditions are acceptable**</Section>
<Section id="**14**">**The Terminal Display is the display screen located in the airport terminal**</Section>
<Section id="**15**">**The Control Tower Display is the display screen located in the Control Tower**</Section>
</Definitions_acronyms_and_abbreviations>
- <References>
<Section id="**1**">**IEEE Standard 830-1998**</Section>

</References>

<Overview_of_product>**The computerized system supports airplanes to land and leave airports safely at scheduled times. Safety and timeliness are two important properties to be preserved in the system. Whenever weather conditions are not good or traffic congestion is high, incoming flights are redirected to neighboring airports where normal operation prevails, and departing flights are rescheduled defending on the scheduling database. Landing requests (or departure requests) must be displayed on a console in the air traffic control tower. Every landing request must be acknowledged within 20 seconds, and each departure request must be acknowledged within 30 seconds. The expected time of landing (or departure) for each authorized flight should be displayed in all terminal buildings. Every airline arrival (or departure), whether it is according to schedule or delayed, must be handled without fail. The safe operation of the ATM system depends on the satisfaction of certain time constrains, so that it can provide safe flight operation of reroute airplanes and the efficient control of airplanes at airports. Any faults in application domain should be detected by sensors and handled by system or Operator.**</Overview_of_product>

</Introduction>

- <Overall_description>

This section will illustrate the product perspectives, product functions, identify who the users are, what are the general constraints of the overall system, and what assumptions and dependencies are considered for the system.

- <Product_perspective>

- <System_interfaces>

<Section id="**1**">**The system is designed in the real-time environment**</Section>

</System_interfaces>

- <User_interfaces>
<Section id="1">**The system contains a big screen that shows the airplane position**</Section>
</User_interfaces>

<Hardware_interfaces>**Not_applicable**</Hardware_interfaces>
- <Software_interfaces>
<Section id="1">**The ATM System is a software component that handles the air traffic schedules**</Section>
</Software_interfaces>
- <Communications_interfaces>
- <Memory>
<Section id="1">**The system needs sufficient memory to run the system in real-time**</Section>
</Memory>
- <Operations>
<Section id="1">**The system should be backup and recovery operation everyday**</Section>
</Operations>

<Site_adaptation_requirements>**Not_applicable**</Site_adaptation_requirements>
</Communications_interfaces>
</Product_perspective>
- <Product_functions>
<Section id="1">**Display flight status information in control tower display for the controllers and in terminal displays for the passengers**</Section>
<Section id="2">**Set alarm when safety requirements are not met and transition back to normal operation upon satisfying the safety requirements by pressing the reset button**</Section>

<Section id="3">**When weather conditions are not good or traffic congestion is high at the airport - the system will signal the operator to reroute incoming traffic to other close-by airports listed in ATM system database as non-congested.**</Section>
<Section id="4">**In normal operations, airplanes are ensured to depart and arrive at scheduled times**</Section>
<Section id="5">**Landing and departing requests are handled with respect to the scheduling database to satisfy the timing requirements**</Section>
</Product_functions>
- <User_characteristics>
The Users of the ATM System are Operators, Travelers, and Pilots. The users characteristics are defined below.
<Section id="1">**An operator is employed by the procurer of the ATM, and has access to the internal mechanism of the ATM system. An Operator is able to trouble shoot errors, and Set alarm back to normal operation by pressing the Reset button.**</Section>
<Section id="2">**A Traveler interacts with the system by viewing the flight schedules (arrival and departure). The ATM system sends feedback to the Traveler by updating the status on the Terminal Display.**</Section>
<Section id="3">**A Pilot interacts with the system by sending the landing and departing requests to the ATM controller tower.**</Section>
</User_characteristics>
- <Constraints>
The ATM System has the following constraints:
<Section id="1">**The ATM System should be powered on at all times from the power outlet to protect the more safety**</Section>

<Section id="2">**There should be a backup generator at all times that in case of losing power the ATM system could continue to work**</Section>

</Constraints>

- <Assumptions_and_dependencies>

The ATM System has the following assumptions and dependencies:

<Section id="1">**There are 6 runways total, 3 arrival runways and 3 departure runways**</Section>

<Section id="2">**The 3 arrival runways are broken down into 1 short runway that are useful for small and low speed airplanes and 2 long runways that are useful for heavy and high-speed airplanes. The actual sizes of the runways are predefined**</Section>

<Section id="3">**There exists a fault detector, which works in congestion with alarm system for detecting the existing error like landing in the wrong runway or departure from a wrong direction**</Section>

<Section id="4">**There exists a global clock that will keep track of the system time, and maintain synchronous system processing**</Section>

<Section id="5">**Every plane during landing will execute at a constant speed in order to achieve the desired completion of landing**</Section>

<Section id="6">**Every plane during takeoff will execute at a constant speed in order to achieve the desired altitude**</Section>

<Section id="7">**At any instance, a list of less congested airports is made available to the system from the ATM system database**</Section>

<Section id="8">**The flight-scheduling system is based on the time priority time tag and not on an airline type**</Section>

<Section id="9">**There exist an Airplane sensor (for every airplane) and a Runway sensor (for every**

runway) that both transmits and receives signals for the fault detecting like attending in the wrong runway</Section>

<Section id="**10**">**The gate, taxiing and parking assignments for all airplanes are predefined in ATM system database**</Section>

</Assumptions_and_dependencies>

<Apportioning_of_requirements>**Not_applicable**</Apportioning_of_requirements>

</Overall_description>

- <Specific_requirements>
  - <External_interface_requirements>
    - <User_interfaces>

<Section id="**1**">**Pilot and ATM: used for requests and acknowledgments. It is based on radio frequencies and can be in shape of data or voice message**</Section>

<Section id="**2**">**Operator and ATM: provides communication with all airplanes, their position and status information, information about other airports, status of alarm, sensors, system queue**</Section>

<Section id="**3**">**Traveler and ATM :interact throw Terminal displays where ATM updates flight information**</Section>

</User_interfaces>

- <Hardware_interfaces>

<Section id="**1**">**ATM and Control Tower Display - The ATM system sends information like weather condition and airplane position etc**</Section>

<Section id="**2**">**ATM and Terminal Display - The ATM system send all flight arrival and departure information to the terminal display**</Section>

<Section id="**3**">**ATM and Airplane sensor and Runway sensor system - used to receive signals from the airplane sensors and runway sensors to locate the position of the airplane in the air and on the runway**</Section>
</Hardware_interfaces>
- <Software_interfaces>
<Section id="**1**">**ATM and external Databases interact over network protocols. ATM request of update the list of less congested airports, flights schedules and details**</Section>
<Section id="**2**">**ATM and weather system - The ATM system receive the weather data from the weather system to decide if the weather condition is normal or not**</Section>
<Section id="**3**">**ATM and Radar - ATM system requires an interface to receive the flight information from the radar system**</Section>
</Software_interfaces>

<Communications_interfaces>**Not_applicable**</Communications_interfaces>
</External_interface_requirements>
- <Domain_Model>
- <Object id="**1**">
<Object_name>**Controller**</Object_name>
- <Attribute id="**1**">

<Name_Attribute>**Controller_ID**</Name_Attribute>
<Type_Attribute>**String**</Type_Attribute>
</Attribute>
- <Relation id="**1**">
<Type_Relation>**Association**</Type_Relation>
<Label_Relation>**Controls**</Label_Relation>

<Multiplicity>**1.n**</Multiplicity>
<From_Object>**Controller**</From_Object>
<To_Object>**Airplane**</To_Object>
</Relation>
- <Relation id="**2**">
<Type_Relation>**Association**</Type_Relation>
<Label_Relation>**Has**</Label_Relation>
<Multiplicity>**1.n**</Multiplicity>
<From_Object>**Controller**</From_Object>
<To_Object>**Operator**</To_Object>
</Relation>
</Object>
- <Object id="**2**">
<Object_name>**Airplane**</Object_name>
- <Attribute id="**1**">
<Name_Attribute>**Airplane_ID**</Name_Attribute>
<Type_Attribute>**String**</Type_Attribute>
</Attribute>
- <Relation id="**1**">
<Type_Relation>**Aggregation**</Type_Relation>
<Label_Relation>**Not_applicable**</Label_Relation>
<Multiplicity>**1.n**</Multiplicity>
<From_Object>**Airplane**</From_Object>
<To_Object>**Airplane_Sensor**</To_Object>
</Relation>
</Object>
- <Object id="**3**">
<Object_name>**Runway**</Object_name>
- <Attribute id="**1**">
<Name_Attribute>**Runway_ID**</Name_Attribute>
<Type_Attribute>**String**</Type_Attribute>
</Attribute>
- <Relation id="**1**">
<Type_Relation>**Aggregation**</Type_Relation>

<Label_Relation>**Not_applicable**</Label_Relation>
<Multiplicity>**1.n**</Multiplicity>
<From_Object>**Runway**</From_Object>
<To_Object>**Runway_Sensor**</To_Object>
</Relation>
</Object>
</Domain_Model>
- <System_Uses>
- <Use_Case id="**1**">
<Use_Case_Purpose>**This use case presents the handling of the airplane departure by the ATM system**</Use_Case_Purpose>
- <Actor id="**1**">
<Name_Actor>**Airplane**</Name_Actor>
<Role_of_Actor>**User**</Role_of_Actor>
<Type_Actor>**Primary**</Type_Actor>
</Actor>
- <Actor id="**2**">
<Name_Actor>**Operator**</Name_Actor>
<Role_of_Actor>**User**</Role_of_Actor>
<Type_Actor>**Secondary**</Type_Actor>
</Actor>
<Priority_of_Usecase>**1**</Priority_of_Usecase>
- <Scenario id="**1**">
<Reference_Diagram>**1**</Reference_Diagram>
<Label_Scenario>**Departure scenario between an airplane and ATM system without interruption**</Label_Scenario>
<Type_Scenario>**Main**</Type_Scenario>
- <Set_objects>
- <Object id="**1**">
<Name />
<Type>**Airplane**</Type>
</Object>

<Object id="**2**">
<Name>**a**</Name>
<Type>**Airplane**</Type>
</Object>
<Object id="**3**">
<Name>**o**</Name>
<Type>**Operator**</Type>
</Object>
<Object id="**4**">
<Name>**c**</Name>
<Type>**Controller**</Type>
</Object>
<Object id="**5**">
<Name>**r**</Name>
<Type>**Runway**</Type>
</Object>
<Object id="**6**">
<Name>**dc**</Name>
<Type>**Display_Controller**</Type>
</Object>
</Set_objects>
<Set_Messages>
<Message id="**1**">

<Label_Message>**Send_Request**</Label_Message>
<From_Object>**Airplane**</From_Object>
<To_Object>**Operator**</To_Object>

<Number_of_message>**1**</Number_of_message>
</Message>
<Message id="**2**">

<Label_Message>**Process_Request**</Label_Message>
<From_Object>**Operator**</From_Object>
<To_Object>**Controller**</To_Object>

<Number_of_message>**2**</Number_of_message>
</Message>
- <Message id="**3**">

<Label_Message>**Assign_Runway**</Label_Message>
<From_Object>**Controller**</From_Object>
<To_Object>**Controller**</To_Object>

<Number_of_message>**3**</Number_of_message>
</Message>
- <Message id="**4**">

<Label_Message>**Give_Runway_number**</Label_Message>
<From_Object>**Controller**</From_Object>
<To_Object>**Operator**</To_Object>

<Number_of_message>**4**</Number_of_message>
</Message>
- <Message id="**5**">

<Label_Message>**Grant_Request**</Label_Message>
<From_Object>**Operator**</From_Object>
<To_Object>**Airplane**</To_Object>

<Number_of_message>**5**</Number_of_
message>
</Message>
- <Message id="**6**">

<Label_Message>**Send_Acknowledgm**
ent</Label_Message>
<From_Object>**Airplane**</From_Object>
<To_Object>**Operator**</To_Object>

<Number_of_message>**6**</Number_of_
message>
</Message>
- <Message id="**7**">

<Label_Message>**Update**</Label_Messa
ge>
<From_Object>**Controller**</From_Object>

<To_Object>**Dispaly_Controller**</To_
Object>

<Number_of_message>**7**</Number_of_
message>
</Message>
- <Message id="**8**">

<Label_Message>**Refresh**</Label_Mess
age>

<From_Object>**Dispaly_Controller**</Fr
om_Object>

<To_Object>**Dispaly_Controller**</To_
Object>

```
        <Number_of_message>8</Number_of_
        message>
    </Message>
- <Message id="9">

        <Label_Message>Approach_Runway<
        /Label_Message>
      <From_Object>Airplane</From_Object>
      <To_Object>Controller</To_Object>

        <Number_of_message>9</Number_of_
        message>
    </Message>
- <Message id="10">

        <Label_Message>Lock_Runway</Label
        _Message>
      <From_Object>Controller</From_Object>
      <To_Object>Runway</To_Object>

        <Number_of_message>10</Number_of
        _message>
    </Message>
- <Message id="11">
      <Label_Message>Close</Label_Message>
      <From_Object>Runway</From_Object>
      <To_Object>Runway</To_Object>

        <Number_of_message>11</Number_of
        _message>
    </Message>
- <Message id="12">
     <Label_Message>In_Runway</Label_Messa
        ge>
```

<From_Object>**Airplane**</From_Object>
<To_Object>**Airplane**</To_Object>

<Number_of_message>**12**</Number_of
_message>
</Message>
- <Message id="**13**">

<Label_Message>**Out_Runway**</Label
_Message>
<From_Object>**Airplane**</From_Object>
<To_Object>**Airplane**</To_Object>

<Number_of_message>**13**</Number_of
_message>
</Message>
- <Message id="**14**">
<Label_Message>**Exit**</Label_Message>
<From_Object>**Airplane**</From_Object>
<To_Object>**Controller**</To_Object>

<Number_of_message>**14**</Number_of
_message>
</Message>
- <Message id="**15**">

<Label_Message>**Unlock_Runway**</La
bel_Message>
<From_Object>**Controller**</From_Object>
<To_Object>**Runway**</To_Object>

<Number_of_message>**15**</Number_of
_message>
</Message>
- <Message id="**16**">

<Label_Message>**Open**</Label_Message>
<From_Object>**Runway**</From_Object>
<To_Object>**Controller**</To_Object>

<Number_of_message>**16**</Number_of
_message>
</Message>
</Set_Messages>
</Scenario>
</Use_Case>
- <Use_Case id="**2**">
<Use_Case_Purpose>**This use case presents the handling of the airplane landing by the ATM system.**</Use_Case_Purpose>
- <Actor id="**1**">
<Name_Actor>**Airplane**</Name_Actor>
<Role_of_Actor>**User**</Role_of_Actor>
<Type_Actor>**Primary**</Type_Actor>
</Actor>
- <Actor id="**2**">
<Name_Actor>**Operator**</Name_Actor>
<Role_of_Actor>**User**</Role_of_Actor>
<Type_Actor>**Secondary**</Type_Actor>
</Actor>
<Priority_of_Usecase>**1**</Priority_of_Usecase>
- <Scenario id="**1**">
<Reference_Diagram>**1**</Reference_Diagram>
<Label_Scenario>**Landing scenario between an airplane and ATM system without interruption**</Label_Scenario>
<Type_Scenario>**Main**</Type_Scenario>
- <Set_objects>
- <Object id="**1**">
<Name />

<Type>**Airplane**</Type>
</Object>
- <Object id="**2**">
<Name>**a**</Name>
<Type>**Airplane**</Type>
</Object>
- <Object id="**3**">
<Name>**o**</Name>
<Type>**Operator**</Type>
</Object>
- <Object id="**4**">
<Name>**c**</Name>
<Type>**Controller**</Type>
</Object>
- <Object id="**5**">
<Name>**r**</Name>
<Type>**Runway**</Type>
</Object>
- <Object id="**6**">
<Name>**dc**</Name>
<Type>**Display_Controller**</Type>
</Object>
</Set_objects>
- <Set_Messages>
- <Message id="**1**">

<Label_Message>**Send_Request**</Label_Message>
<From_Object>**Airplane**</From_Object>
<To_Object>**Operator**</To_Object>

<Number_of_message>**1**</Number_of_message>
</Message>

<Message id="**2**">

<Label_Message>**Process_Request**</Label_Message>

<From_Object>**Operator**</From_Object>

<To_Object>**Controller**</To_Object>

<Number_of_message>**2**</Number_of_message>

</Message>

<Message id="**3**">

<Label_Message>**Assign_Runway**</Label_Message>

<From_Object>**Controller**</From_Object>

<To_Object>**Controller**</To_Object>

<Number_of_message>**3**</Number_of_message>

</Message>

<Message id="**4**">

<Label_Message>**Give_Runway_number**</Label_Message>

<From_Object>**Controller**</From_Object>

<To_Object>**Operator**</To_Object>

<Number_of_message>**4**</Number_of_message>

</Message>

<Message id="**5**">

<Label_Message>**Grant_Request**</Label_Message>

<From_Object>**Operator**</From_Object>

<To_Object>**Airplane**</To_Object>

<Number_of_message>**5**</Number_of_message>

</Message>

- <Message id="**6**">

<Label_Message>**Send_Acknowledgment**</Label_Message>

<From_Object>**Airplane**</From_Object>

<To_Object>**Operator**</To_Object>

<Number_of_message>**6**</Number_of_message>

</Message>

- <Message id="**7**">

<Label_Message>**Update**</Label_Message>

<From_Object>**Controller**</From_Object>

<To_Object>**Dispaly_Controller**</To_Object>

<Number_of_message>**7**</Number_of_message>

</Message>

- <Message id="**8**">

<Label_Message>**Refresh**</Label_Message>

<From_Object>**Dispaly_Controller**</From_Object>

<To_Object>**Dispaly_Controller**</To_Obje
ct>

<Number_of_message>**8**</Number_of_
message>
</Message>
- <Message id="**9**">

<Label_Message>**Approach_Runway**<
/Label_Message>
<From_Object>**Airplane**</From_Object>
<To_Object>**Controller**</To_Object>

<Number_of_message>**9**</Number_of_
message>
</Message>
- <Message id="**10**">

<Label_Message>**Lock_Runway**</Label
_Message>
<From_Object>**Controller**</From_Object>
<To_Object>**Runway**</To_Object>

<Number_of_message>**10**</Number_of
_message>
</Message>
- <Message id="**11**">
<Label_Message>**Close**</Label_Message>
<From_Object>**Runway**</From_Object>
<To_Object>**Runway**</To_Object>

<Number_of_message>**11**</Number_of
_message>
</Message>
- <Message id="**12**">

<Label_Message>**In_Runway**</Label_Mess
age>
<From_Object>**Airplane**</From_Object>
<To_Object>**Airplane**</To_Object>

<Number_of_message>**12**</Number_of
_message>
</Message>
- <Message id="**13**">

<Label_Message>**Out_Runway**</Label
_Message>
<From_Object>**Airplane**</From_Object>
<To_Object>**Airplane**</To_Object>

<Number_of_message>**13**</Number_of
_message>
</Message>
- <Message id="**14**">
<Label_Message>**Exit**</Label_Message>
<From_Object>**Airplane**</From_Object>
<To_Object>**Controller**</To_Object>

<Number_of_message>**14**</Number_of
_message>
</Message>
- <Message id="**15**">

<Label_Message>**Unlock_Runway**</La
bel_Message>
<From_Object>**Controller**</From_Object>
<To_Object>**Runway**</To_Object>

<Number_of_message>**15**</Number_of
_message>

</Message>
- <Message id="**16**">
<Label_Message>**Open**</Label_Message>
<From_Object>**Runway**</From_Object>
<To_Object>**Controller**</To_Object>

<Number_of_message>**16**</Number_of
_message>
</Message>
</Set_Messages>
</Scenario>
</Use_Case>
</System_Uses>
</Specific_requirements>

<Performance_requirements>**Not_applicable**</Performance_requir
ements>
<Design_constraints>**Not_applicable**</Design_constraints>
- <Software_system_attributes>
<Reliability>**Not_applicable**</Reliability>
<Availability>**Not_applicable**</Availability>
<Security>**Not_applicable**</Security>
<Maintainability>**Not_applicable**</Maintainability>
<Portability>**Not_applicable**</Portability>
</Software_system_attributes>
- <Other_requirements>
- <Safety_Requirement>
Safety Requirement means, the system has zero faults and system can detected every possible situation.
<Section id="**1**">**If an airplane scheduled for departure is moving towards one of the arrival runways, the system should be able to detect it based on Runway and Airplane sensors, set the alarm within 2 seconds**

of detection, inform the pilot of the situation, and reroute the airplane to the correct runway</Section>

<Section id="**2**">**If an airplane scheduled for arrival is moving towards one of the departure runways, the system should be able to detect it based on Runway and Airplane sensors, set the alarm within 5 seconds of detection, inform the pilot of the situation, and reroute the airplane to the correct runway**</Section>

<Section id="**3**">**If an airplane scheduled for departure on one of the long runways is moving towards the short departure runway, the system should be able to detect it based on Runway and Airplane sensors, set the alarm within 2 seconds of detection, inform the pilot of the situation, and reroute the airplane to the correct runway**</Section>

<Section id="**4**">**If an airplane scheduled for arrival on one of the long runways is moving towards the short arrival runway, the system should be able to detect it based on Runway and Airplane sensors, set the alarm within 5 seconds of detection, inform the pilot of the situation, and reroute the airplane to the correct runway**</Section>

<Section id="**5**">**If two airplanes going in different directions while in the air are within 2 miles from each other, the system should be able to detect it, sound the alarm within 2 seconds of detection, and notify the pilot of rerouting the airplane to the coordinates received from the control tower**</Section>

<Section id="**6**">**System should not allow landing or takeoff acknowledgment from Operator if time difference each runway is less than 10 minutes between lending of 2 airplanes**</Section>

<Section id="**7**">**If Operator or Pilot does not respond according to timing requirements should produce system error**</Section>

</Safety_Requirement>

- <Timing_Requirement>

Timing Requirement means that events/actions will happen as specified.

<Section id="**1**">**For each runway, Operator should acknowledge every landing request from Pilot within 20 seconds of receipt**</Section>

<Section id="**2**">**For each runway, Operator should acknowledge every departing request from Pilot within 30 seconds of receipt**</Section>

<Section id="**3**">**For each runway, each airplane waiting to land should have at least 10 minutes interval from the other airplane during landing**</Section>

<Section id="**4**">**For each runway, each airplane waiting to takeoff should have at least 10 minutes interval from the other airplane during take off**</Section>

<Section id="**5**">**Given an acknowledgement to land or takeoff, every Pilot should acknowledge back to the ATM tower within 15 seconds of receipt of the first acknowledgement**</Section>

<Section id="**6**">**The airport Terminal Display and Control Tower Display screens should be updated with the flight arrival and departure information every 10 minutes under normal operations and every 1-minute when there is a flight delay**</Section>

<Section id="**7**">**After Operator acknowledge request for landing, A Pilot then must send an acknowledgment back to the tower within 15 seconds of receipt of the first acknowledgement**</Section>

<Section id="**8**">**Delay information of flight arrival and departure should be announced within 5 minutes of receiving the information of the delay and updated in**

the airport Terminal Display screens within 1 minute</Section>
</Timing_Requirement>
- <Live_ness_Requirements>
Live ness Requirement means the system will eventually do the fault free work.
<Section id="**1**">**All landing and departure requests should be added in a queue**</Section>
</Live_ness_Requirements>
- <System_Evolution_Requirements>
System Evolution Requirements are future possible enhancements of the system
<Section id="**1**">**If a runway remains free for 60 minutes or more, it can be reallocated**</Section>
<Section id="**2**">**The ATM system should handle more than 6 runways**</Section>
<Section id="**3**">**The Control Tower Display should only display the airplanes that are arriving and departing in 60 minutes**</Section>
<Section id="**4**">**There should be a backup system for an ATM system. Backup System means: the system will have a backup database system**</Section>
<Section id="**5**">**The ATM system should arrange a runway for emergency landing request. Emergency landing means: the flight does not exist in the flight-scheduling database**</Section>
</System_Evolution_Requirements>

<Logical_database_requirements>**Not_applicable**</Logical_database_requirements>

<Standards_compliance>**Not_applicable**</Standards_compliance>
- <Supporting_information>
- <Table_of_contents_and_index>

- <Part1>
Introduction
<Section1>**Purpose**</Section1>
<Section2>**Scope**</Section2>

<Section3>**Definitions_acronyms_and_abbrev**
iations</Section3>
<Section4>**References**</Section4>
<Section5>**Overview_of_product**</Section5>
</Part1>
- <Part2>
Overall_description
- <Section1>
Product_perspective
- <Subsection1>

<Subsection1_1>**System_interfaces**</
Subsection1_1>

<Subsection1_2>**User_interfaces**</Su
bsection1_2>

<Subsection1_3>**Hardware_interfaces**
</Subsection1_3>

<Subsection1_4>**Software_interfaces**
</Subsection1_4>

<Subsection1_5>**Communications_int**
erfaces</Subsection1_5>
</Subsection1>
</Section1>
<Section2>**Product_functions**</Section2>
<Section3>**User_characteristics**</Section3>
<Section4>**Constraints**</Section4>

<Section5>**Assumptions_and_dependencies**</Section5>

<Section6>**Apportioning_of_requirements**</Section6>

</Part2>

- <Part3>

Specific_requirements

<Section1>**External interface requirements**</Section1>

- <Subsection1>

<Subsection1_1>**User_interfaces**</Subsection1_1>

<Subsection1_2>**Hardware_interfaces**</Subsection1_2>

<Subsection1_3>**Software_interfaces**</Subsection1_3>

<Subsection1_4>**Communications_interfaces**</Subsection1_4>

</Subsection1>

- <Section2>

Domain Model

- <Subsection2>

<Subsection2_1>**Object_name**</Subsection2_1>

<Subsection2_2>**Attributes**</Subsection2_2>

<Subsection2_3>**Relations**</Subsection2_3>

</Subsection2>

</Section2>

<Section3>

System Uses

<Subsection3>

<Subsection3_1>**Use_Case1**</Subsection3_1>

<Subsection3_2>**Use_Case2**</Subsection3_2>

</Subsection3>

</Section3>

<Section4>**Performance_requirements**</Section4>

<Section5>**Design_constraints**</Section5>

<Section6>**Software_system_attributes**</Section6>

<Section7>**Other_requirements**</Section7>

</Part3>

</Table_of_contents_and_index>

<Appendixes>**Not_applicable**</Appendixes>

</Supporting_information>

</Other_requirements>

</Requirement_specification_template>

Appendix 2. Schema description of Requirement Specification Document

```
<?xml version="1.0" encoding="utf-16"?>
<xsd:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
version="1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="Requirement_specification_template"
type="Requirement_specification_templateType" />
 <xsd:complexType name="Requirement_specification_templateType">
  <xsd:sequence>
   <xsd:element name="Introduction" type="IntroductionType" />
   <xsd:element name="Overall_description" type="Overall_descriptionType" />
   <xsd:element name="Specific_requirements" type="Specific_requirementsType" />
   <xsd:element name="Performance_requirements" type="xsd:string" />
   <xsd:element name="Design_constraints" type="xsd:string" />
   <xsd:element name="Software_system_attributes"
type="Software_system_attributesType" />
   <xsd:element name="Other_requirements" type="Other_requirementsType" />
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="Other_requirementsType">
  <xsd:sequence>
   <xsd:element name="Safety_Requirement" type="Safety_RequirementType" />
   <xsd:element name="Timing_Requirement" type="Timing_RequirementType" />
   <xsd:element name="Live_ness_Requirements"
type="Live_ness_RequirementsType" />
   <xsd:element name="System_Evolution_Requirements"
type="System_Evolution_RequirementsType" />
   <xsd:element name="Logical_database_requirements" type="xsd:string" />
   <xsd:element name="Standards_compliance" type="xsd:string" />
```

```
    <xsd:element name="Supporting_information" type="Supporting_informationType"
/>
   </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Supporting_informationType">
   <xsd:sequence>
    <xsd:element name="Table_of_contents_and_index"
type="Table_of_contents_and_indexType" />
    <xsd:element name="Appendixes" type="xsd:string" />
   </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Table_of_contents_and_indexType">
   <xsd:sequence>
    <xsd:element name="Part1" type="Part1Type" />
    <xsd:element name="Part2" type="Part2Type" />
    <xsd:element name="Part3" type="Part3Type" />
   </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Part3Type">
   <xsd:sequence>
    <xsd:element name="Section1" type="xsd:string" />
    <xsd:element name="Subsection1" type="Subsection1Type" />
    <xsd:element name="Section2" type="Section2Type" />
    <xsd:element name="Section3" type="Section3Type" />
    <xsd:element name="Section4" type="xsd:string" />
    <xsd:element name="Section5" type="xsd:string" />
    <xsd:element name="Section6" type="xsd:string" />
    <xsd:element name="Section7" type="xsd:string" />
   </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Section3Type">
```

```
  <xsd:sequence>
    <xsd:element name="Subsection3" type="Subsection3Type" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Subsection3Type">
  <xsd:sequence>
    <xsd:element name="Subsection3_1" type="xsd:string" />
    <xsd:element name="Subsection3_2" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Section2Type">
  <xsd:sequence>
    <xsd:element name="Subsection2" type="Subsection2Type" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Subsection2Type">
  <xsd:sequence>
    <xsd:element name="Subsection2_1" type="xsd:string" />
    <xsd:element name="Subsection2_2" type="xsd:string" />
    <xsd:element name="Subsection2_3" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Subsection1Type">
  <xsd:sequence>
    <xsd:element name="Subsection1_1" type="xsd:string" />
    <xsd:element name="Subsection1_2" type="xsd:string" />
    <xsd:element name="Subsection1_3" type="xsd:string" />
    <xsd:element name="Subsection1_4" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Part2Type">
```

```
  <xsd:sequence>
    <xsd:element name="Section1" type="Section1Type" />
    <xsd:element name="Section2" type="xsd:string" />
    <xsd:element name="Section3" type="xsd:string" />
    <xsd:element name="Section4" type="xsd:string" />
    <xsd:element name="Section5" type="xsd:string" />
    <xsd:element name="Section6" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Section1Type">
  <xsd:sequence>
    <xsd:element name="Subsection1" type="Subsection1Type" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Subsection1Type">
  <xsd:sequence>
    <xsd:element name="Subsection1_1" type="xsd:string" />
    <xsd:element name="Subsection1_2" type="xsd:string" />
    <xsd:element name="Subsection1_3" type="xsd:string" />
    <xsd:element name="Subsection1_4" type="xsd:string" />
    <xsd:element name="Subsection1_5" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Part1Type">
  <xsd:sequence>
    <xsd:element name="Section1" type="xsd:string" />
    <xsd:element name="Section2" type="xsd:string" />
    <xsd:element name="Section3" type="xsd:string" />
    <xsd:element name="Section4" type="xsd:string" />
    <xsd:element name="Section5" type="xsd:string" />
  </xsd:sequence>
```

```
</xsd:complexType>
<xsd:complexType name="System_Evolution_RequirementsType">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SectionType">
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="Live_ness_RequirementsType">
 <xsd:sequence>
  <xsd:element name="Section" type="SectionType" />
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SectionType">
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="Timing_RequirementType">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SectionType">
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="Safety_RequirementType">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SectionType">
```

```
  <xsd:attribute name="id" type="xsd:int" />
 </xsd:complexType>
 <xsd:complexType name="Software_system_attributesType">
  <xsd:sequence>
   <xsd:element name="Reliability" type="xsd:string" />
   <xsd:element name="Availability" type="xsd:string" />
   <xsd:element name="Security" type="xsd:string" />
   <xsd:element name="Maintainability" type="xsd:string" />
   <xsd:element name="Portability" type="xsd:string" />
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="Specific_requirementsType">
  <xsd:sequence>
   <xsd:element name="External_interface_requirements"
type="External_interface_requirementsType" />
   <xsd:element name="Domain_Model" type="Domain_ModelType" />
   <xsd:element name="System_Uses" type="System_UsesType" />
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="System_UsesType">
  <xsd:sequence>
   <xsd:element maxOccurs="unbounded" name="Use_Case" type="Use_CaseType" />
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="Use_CaseType">
  <xsd:sequence>
   <xsd:element name="Use_Case_Purpose" type="xsd:string" />
   <xsd:element maxOccurs="unbounded" name="Actor" type="ActorType" />
   <xsd:element name="Priority_of_Usecase" type="xsd:int" />
   <xsd:element name="Scenario" type="ScenarioType" />
  </xsd:sequence>
```

```
  <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="ScenarioType">
  <xsd:sequence>
    <xsd:element name="Reference_Diagram" type="xsd:int" />
    <xsd:element name="Label_Scenario" type="xsd:string" />
    <xsd:element name="Type_Scenario" type="xsd:string" />
    <xsd:element name="Set_objects" type="Set_objectsType" />
    <xsd:element name="Set_Messages" type="Set_MessagesType" />
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="Set_MessagesType">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" name="Message" type="MessageType" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="MessageType">
  <xsd:sequence>
    <xsd:element name="Label_Message" type="xsd:string" />
    <xsd:element name="From_Object" type="xsd:string" />
    <xsd:element name="To_Object" type="xsd:string" />
    <xsd:element name="Number_of_message" type="xsd:int" />
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="Set_objectsType">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" name="Object" type="ObjectType" />
  </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="ObjectType">
 <xsd:sequence>
  <xsd:element name="Name" type="xsd:string" />
  <xsd:element name="Type" type="xsd:string" />
 </xsd:sequence>
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="ActorType">
 <xsd:sequence>
  <xsd:element name="Name_Actor" type="xsd:string" />
  <xsd:element name="Role_of_Actor" type="xsd:string" />
  <xsd:element name="Type_Actor" type="xsd:string" />
 </xsd:sequence>
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="Domain_ModelType">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" name="Object" type="ObjectType" />
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ObjectType">
 <xsd:sequence>
  <xsd:element name="Object_name" type="xsd:string" />
  <xsd:element name="Attribute" type="AttributeType" />
  <xsd:element maxOccurs="unbounded" name="Relation" type="RelationType" />
 </xsd:sequence>
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="RelationType">
 <xsd:sequence>
  <xsd:element name="Type_Relation" type="xsd:string" />
```

```
    <xsd:element name="Label_Relation" type="xsd:string" />
    <xsd:element name="Multiplicity" type="xsd:string" />
    <xsd:element name="From_Object" type="xsd:string" />
    <xsd:element name="To_Object" type="xsd:string" />
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="AttributeType">
  <xsd:sequence>
    <xsd:element name="Name_Attribute" type="xsd:string" />
    <xsd:element name="Type_Attribute" type="xsd:string" />
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="External_interface_requirementsType">
  <xsd:sequence>
    <xsd:element name="User_interfaces" type="User_interfacesType" />
    <xsd:element name="Hardware_interfaces" type="Hardware_interfacesType" />
    <xsd:element name="Software_interfaces" type="Software_interfacesType" />
    <xsd:element name="Communications_interfaces" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Software_interfacesType">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SectionType">
  <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="Hardware_interfacesType">
```

```
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="SectionType">
  <xsd:attribute name="id" type="xsd:int" />
 </xsd:complexType>
 <xsd:complexType name="User_interfacesType">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="SectionType">
  <xsd:attribute name="id" type="xsd:int" />
 </xsd:complexType>
 <xsd:complexType name="Overall_descriptionType">
  <xsd:sequence>
    <xsd:element name="Product_perspective" type="Product_perspectiveType" />
    <xsd:element name="Product_functions" type="Product_functionsType" />
    <xsd:element name="User_characteristics" type="User_characteristicsType" />
    <xsd:element name="Constraints" type="ConstraintsType" />
    <xsd:element name="Assumptions_and_dependencies"
type="Assumptions_and_dependenciesType" />
    <xsd:element name="Apportioning_of_requirements" type="xsd:string" />
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="Assumptions_and_dependenciesType">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
  </xsd:sequence>
 </xsd:complexType>
```

```
<xsd:complexType name="SectionType">
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="ConstraintsType">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SectionType">
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="User_characteristicsType">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SectionType">
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="Product_functionsType">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SectionType">
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="Product_perspectiveType">
 <xsd:sequence>
  <xsd:element name="System_interfaces" type="System_interfacesType" />
  <xsd:element name="User_interfaces" type="User_interfacesType" />
```

```
    <xsd:element name="Hardware_interfaces" type="xsd:string" />
    <xsd:element name="Software_interfaces" type="Software_interfacesType" />
    <xsd:element name="Communications_interfaces"
type="Communications_interfacesType" />
   </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Communications_interfacesType">
   <xsd:sequence>
    <xsd:element name="Memory" type="MemoryType" />
    <xsd:element name="Operations" type="OperationsType" />
    <xsd:element name="Site_adaptation_requirements" type="xsd:string" />
   </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="OperationsType">
   <xsd:sequence>
    <xsd:element name="Section" type="SectionType" />
   </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SectionType">
   <xsd:attribute name="id" type="xsd:int" />
  </xsd:complexType>
  <xsd:complexType name="MemoryType">
   <xsd:sequence>
    <xsd:element name="Section" type="SectionType" />
   </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SectionType">
   <xsd:attribute name="id" type="xsd:int" />
  </xsd:complexType>
  <xsd:complexType name="Software_interfacesType">
   <xsd:sequence>
```

```
    <xsd:element name="Section" type="SectionType" />
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="SectionType">
  <xsd:attribute name="id" type="xsd:int" />
 </xsd:complexType>
 <xsd:complexType name="User_interfacesType">
  <xsd:sequence>
    <xsd:element name="Section" type="SectionType" />
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="SectionType">
  <xsd:attribute name="id" type="xsd:int" />
 </xsd:complexType>
 <xsd:complexType name="System_interfacesType">
  <xsd:sequence>
    <xsd:element name="Section" type="SectionType" />
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="SectionType">
  <xsd:attribute name="id" type="xsd:int" />
 </xsd:complexType>
 <xsd:complexType name="IntroductionType">
  <xsd:sequence>
    <xsd:element name="Purpose" type="PurposeType" />
    <xsd:element name="Scope" type="ScopeType" />
    <xsd:element name="Definitions_acronyms_and_abbreviations"
type="Definitions_acronyms_and_abbreviationsType" />
    <xsd:element name="References" type="ReferencesType" />
    <xsd:element name="Overview_of_product" type="xsd:string" />
  </xsd:sequence>
```

```
</xsd:complexType>
<xsd:complexType name="ReferencesType">
 <xsd:sequence>
  <xsd:element name="Section" type="SectionType" />
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SectionType">
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="Definitions_acronyms_and_abbreviationsType">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SectionType">
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="ScopeType">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SectionType">
 <xsd:attribute name="id" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="PurposeType">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" name="Section" type="SectionType" />
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SectionType">
```

```
    <xsd:attribute name="id" type="xsd:int" />
  </xsd:complexType>
</xsd:schema>
```