Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Notes

Module 1 Introduction

Introduction

In this module, you will:

- Review the workshop objectives and agenda
- Understand where in the project life cycle we will focus
- Discuss logistics

Welcome to ***Advanced Business Analysis***. This workshop is the first of three workshops that make up Systemation's advanced certificate in business analysis program.

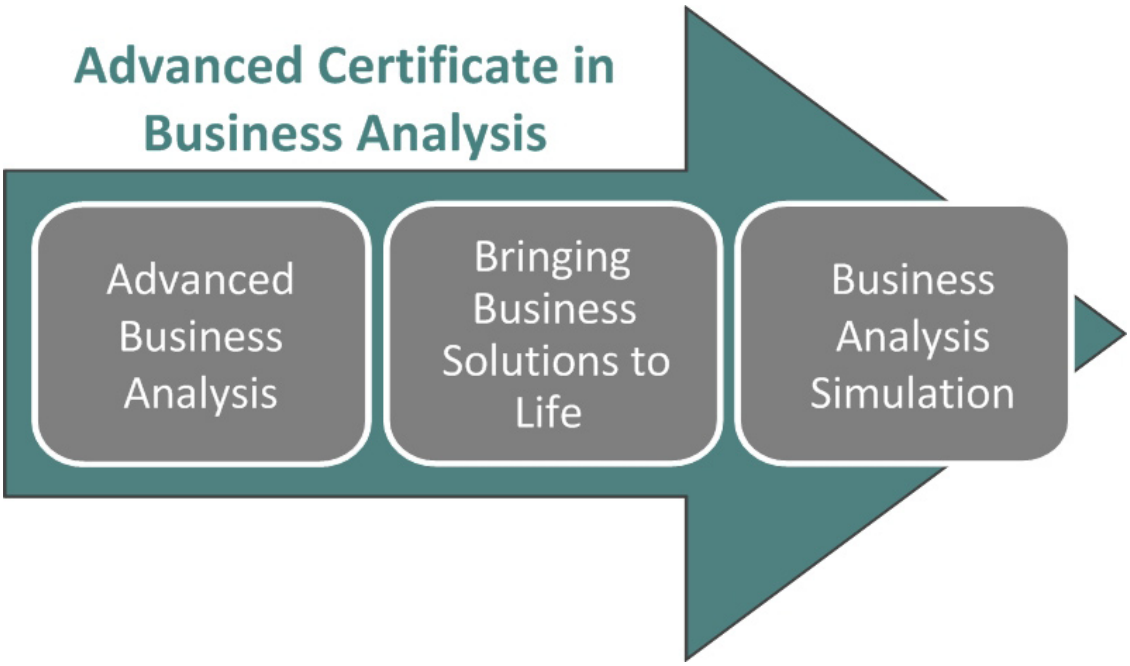


Figure 1.1 Advanced Workshops

If you participated in a workshop in the practitioner certificate in business analysis program, you learned how to write quality requirements by beginning with a current state analysis and problem definition. You were introduced to many techniques to help you produce a requirements document of which you can be proud.

This workshop will focus on additional ways to elicit and write requirements such as use cases and data models. We will introduce object orientation and the Unified Modeling Language (UML) as well as information engineering techniques to model data requirements.

This section of the participant guide will provide information on the workshop objectives and background of the case study, and the logistics for this workshop.

Getting Started

Before we start, let's discuss the following:

1. **What is a model?**

2. **Why model?**

3. **What is a system?**

4. **What is an abstraction?**

Introduction

Workshop Objectives

Advanced Business Analysis has been designed specifically to help you accomplish a great deal in a short period of time. In this workshop, you will:

- Learn about object orientation and apply the Unified Modeling Language (UML) to requirements analysis
- Develop graphical and textual use cases
- Learn about information engineering (IE) and produce data models as an extension of your requirements analysis
- Develop a data model from requirements
- Understand normalization to ensure data models are high quality
- Understand and perform requirements traceability
- Pull it all together to recognize "***value added***" modeling for business analysts

Workshop Agenda

To meet these objectives, ***Advanced Business Analysis*** is organized into the following 11 modules:

General Topics	Object Orientation / UML Approach	Information Engineering Approach
1. Introduction	2. The Building Blocks	8. Conceptual Data Modeling
11. Value Added Modeling	3. Business Modeling	9. Logical Data Modeling
	4. Classes & Objects	10. Normalization
	5. Behavioral Modeling	
	6. Use Cases	
	7. System Views	

Value Added Modeling

This workshop is more than just creating models. It is aimed at ensuring that business analysts know how to **use** models for effectively managing requirements though eliciting, analyzing, documenting, and communicating them to stakeholders for ultimate sign off.

Value added modeling:

> ***Value added modeling refers to the process of producing models, and their subsequent analysis, confirmation and application to improve project quality and success.***

Value added modeling directly supports the critical activities of requirements traceability, requirements confirmation, requirements prioritization, and requirements re-use. All of these will be discussed in the workshop and applied throughout the exercises.

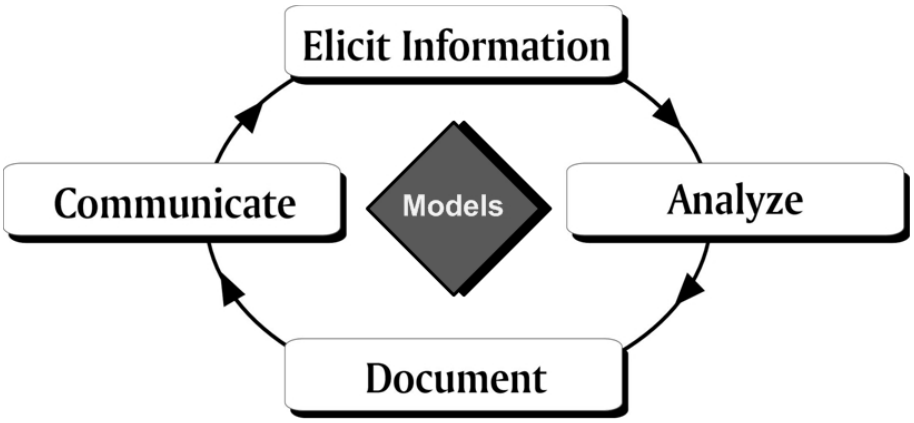


Figure 1.2 Value Added Modeling and the Requirements Process

What is Business Analysis

As the requirements process above illustrates, business analysts focus on eliciting analyzing, documenting and communicating requirements. Business analysis is the set of tasks and techniques used to work as a liaison among stakeholders in order to understand the structure, policies and operation of an organization, and to recommend solutions that enable the organization to achieve its goals.[1] We will introduce value added modeling techniques to enable business analysts to do just this.

[1] IIBA BABOK 2.0

Introduction

The System Development Life Cycle

You learned quite a bit about the initiation, analysis, and conceptual design phases of the system development life cycle (SDLC) in previous Systemation workshops. In this workshop, we will remain in these three phases, but will take a modeling perspective to refine the scope and requirements of a project. The figure below illustrates the SDLC: the top is a typical "waterfall" approach and the bottom represents an iterative approach.

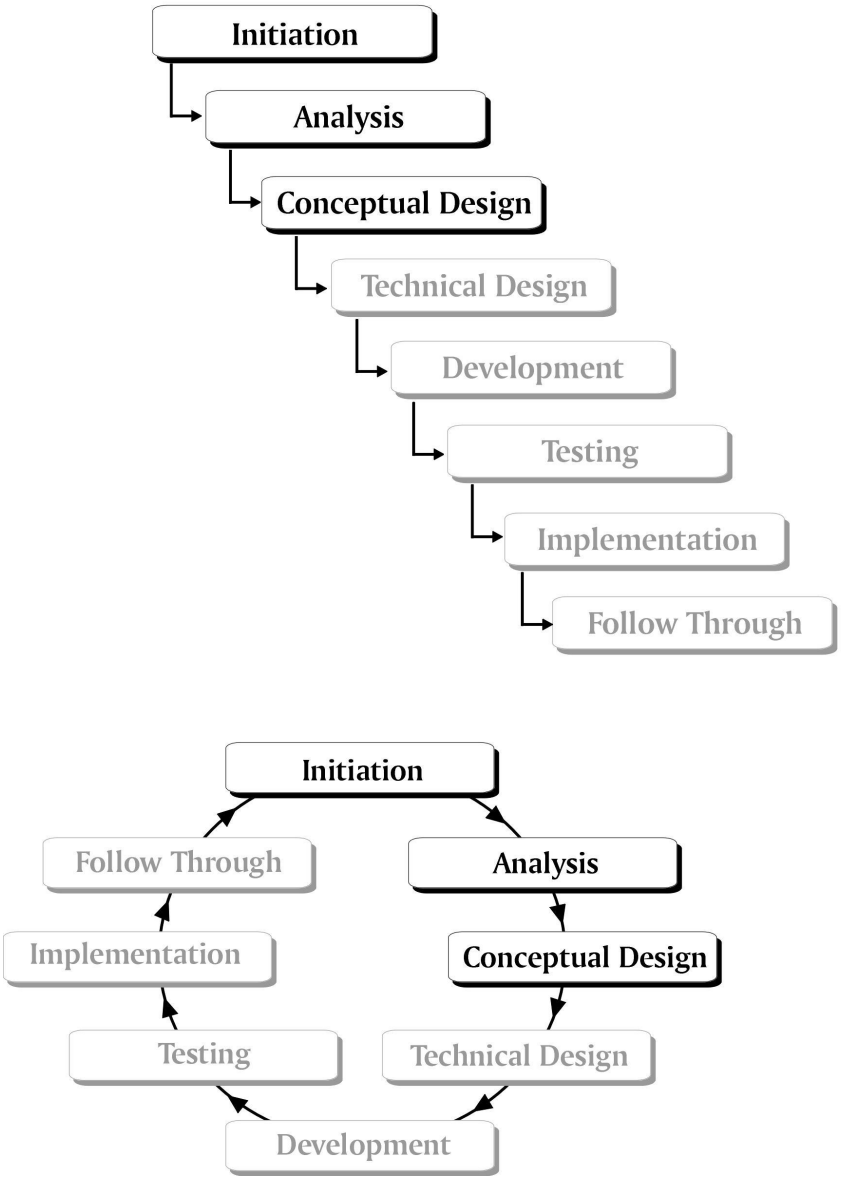


Figure 1.3 SDLC

The Case Study

In the ***Mastering the Requirements Process*** workshop, we focused on a Sales Order System (SOS) for the case study company, Akmee Communications. In parallel, two other project teams were hard at work defining the requirements for processing invoices and payments (Invoice Payment System or IPS) and managing inventory (Inventory Management System or IMS).

This workshop will go back and review some of the requirements for these systems and continue to refine them. We will then define a new project and use the tools and techniques of object orientation and information engineering to work through the new requirements.

Getting the Most from This Workshop

Generally, what you get out of a workshop is related to what you put into it. To help maximize the value you derive from ***Advanced Business Analysis***, please consider doing the following:

- Turn your laptops, pagers and cell phones off
- Respect agreed-upon class start, end, and break times
- Keep an open mind
- Participate at your comfort level
- Think about and apply workshop information
- Ask questions
- Share your relevant stories
- Respect others' views
- Have fun!

Introduction

Workshop Logistics

Use the table below to record the logistics for this workshop.

Facilitator's name	
Facilitator's e-mail	
Start/end times	
Lunch (approximate)	
Breaks	

Workshop Materials

The materials used in this workshop include:

1. **Participant Guide**, including:
   - Class discussion material
   - Case study exercises, worksheets, and templates
2. **Appendices**, including:
   - Recommended reading
   - Glossary of terms
   - Index
3. **Handouts for the case study**
   - Suggested solutions for some of the exercises

Module 2 The Building Blocks

The Building Blocks

In this module, you will:

- *Understand the importance of modeling*
- *Discover how models capture requirements*
- *Review the types of models and requirements*
- *Discuss requirements traceability*

Why Modeling Is Important

People can often describe things better with pictures than with words, especially when thinking about a complex system. Therefore, this workshop begins with visual thinking – with models.

A model is a representation of a concept or a physical thing. Modeling has always been an important activity in science and engineering. With the advent of the computer, modeling became important because system software is intangible and can be very complex; that is, you cannot touch it and manipulate it in the physical world. The software requirements need to address what the system should do, how a user will interact with the system, what data should be captured and stored, and what interfaces are needed to ensure completeness. It turns out that a model of the system can be very useful and effective to unambiguously analyze, document, and communicate requirements.

Business analysts can produce the models of requirements. Within the context of a project, good models can facilitate effective communication amongst the project stakeholders.

Types of Models

Two types of models will be introduced in this workshop:

1. ***Behavioral models*** describe the operation of a system. These are sometimes called process models, and focus on defining the functionality of the system. In the ***Fast Start in Business Analysis©*** workshop, the swim lanes that were produced represent behavioral models. This workshop will introduce object oriented models used to define system functionality.
2. ***Structural models*** describe the organization of a system by focusing on the data and the relationships between the data. We will introduce object-oriented and information engineering concepts and techniques used to model data requirements.

The quality of a model and the level of skill in using a model are directly proportional to delivering project quality, particularly for software systems.

What is a System?

If a model is a representation of a system, then what is a *system*? Think about this question and work with your team to arrive at a consensus answer.

The Building Blocks

Consider the following definitions of ***system.*** How do these compare yours?

- *A system is a set of objects together with relationships between the objects and between their attributes.*[2]
- *A regularly interacting or interdependent group of items forming a unified whole; a group of devices or artificial objects or an organization forming a network especially for distributing something or serving a common purpose.*[3]
- *A set of elements organized to accomplish a specific purpose and described by a set of models, possibly from different viewpoints.*[4]
- *A system is a way of looking at the world.*[5]

Now for the hard question: is a system reality, or is it a description of reality?

Before answering, you must understand the ***source of order*** for a system, or how the system needs to be organized into meaningful parts. However, the source of order can be "out there" in the "real world" or it can be in the observer of that "real world."

Observation leads us to the principal of *abstraction.* Abstractions are the fundamental principle of systems theory.

Abstraction:

An abstraction is a concept or general idea not associated with a specific instance.

For example: You may speak of an abstract "dog" without referring to any specific instance of a dog.

You may speak of an abstract "order entry system" without referring to any specific instance of such a system.

[2] Arthur D. Hall and R. E. Fagen, "Definition of System." In Modern Systems Research for the Behavioral Scientist, Walter Buckley, Ed. Chicago:Aldine, 1968.

[3] "System" in WEBSTER'S Ninth New Collegiate Dictionary, Merriam-Webster, Springfield, MA., 1985

[4] Grady Booch et al., The Unified Modeling Language User Guide, Addison-Wesley, 1999

[5] Gerald M. Weinberg, An Introduction to General Systems Thinking, Silver Anniversary Edition, Dorset House, 2001

The Building Blocks

Why do observers abstract things to describe the "real world?" Because there is no other way! Information about how a system works comes from the observers of that system. Each observer has a *perspective* of the system, or a "way of looking at the world." Of course, there is no guarantee that all perspectives will agree. As a matter of fact, we will introduce five systems perspectives in this workshop. To get anything useful, all of the observers of a system must communicate with each other in a useful way.

Business analysts can abstract the "real world" of systems into models. These models become the communication mechanism between the observers of the system.

One ***system abstraction*** is the framework for documenting and communicating requirements. In past workshops, the framework was predominately a document containing words. In this workshop, we will introduce models that can be used for documenting requirements either instead of words, or to supplement the words communicated to stakeholders.

Key Abstractions

In practice, there are three key abstractions which form the building blocks of modeling requirements:

1. ***Things*****,** which are the fundamental elements of a system model.
2. ***Relationships*****,** which tie things together.
3. ***Diagrams***, which group things and relationships.

Now, can you answer the hard question posed earlier –

Is a system reality?

The answer is: not really. As a business analyst, you use models to document and communicate about systems which, themselves, are abstractions of reality. After the system is implemented and deployed, you collect metrics about the "real world" to see how a system actually is working, and then you return to the models of that system to manage improvements. All along, you never quite get your hands on the real thing, and you must juggle many observer perspectives continuously.

The Building Blocks

Requirements

A requirement is defined as:

> ***A statement of system functionality (a capability) that can be validated, that must be met or possessed by a system to solve a customer problem or to achieve a customer objective, and is qualified by measurable conditions and bounded by constraints.***[6]

IIBA[7] defines four types of requirements:

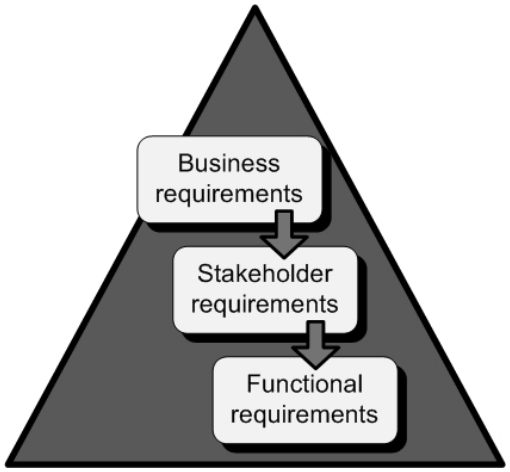


Figure 2.1 Types of Requirements

Business Requirements – High level statements of the goals, objectives or needs of the enterprise. They typically define the reason for initiating the project and ways to measure success. They are most often found as project objectives in the business case, charter or scope.

Stakeholder Requirements – Statements of the needs of a particular stakeholder or class of stakeholders.

Functional Requirements – The behavior and information that the solution will manage, and capabilities the system will be able to perform.

Non-Functional Requirements – Describe environmental conditions under which the solution must remain effective or qualities that the system must have.

[6] IEEE Std 1233, 1998 Edition (R2002), IEEE Guide for Developing System Requirements Specifications, IEEE, Inc., New York, reaffirmed 11 September 2002.

[7] IIBA BABOK 2.0

SMART Requirements

Modeling provides an excellent framework for capturing, documenting, and communicating requirements because it channels the business analyst's thinking toward being SMART. In turn, the business analyst can ensure that the thinking of project stakeholders is in the same direction. Modeling facilitates the work of moving the system abstraction from the general, contextual level to a functional and SMART level.

To simplify things, the acronym SMART helps us remember the key characteristics of a high-quality requirement.

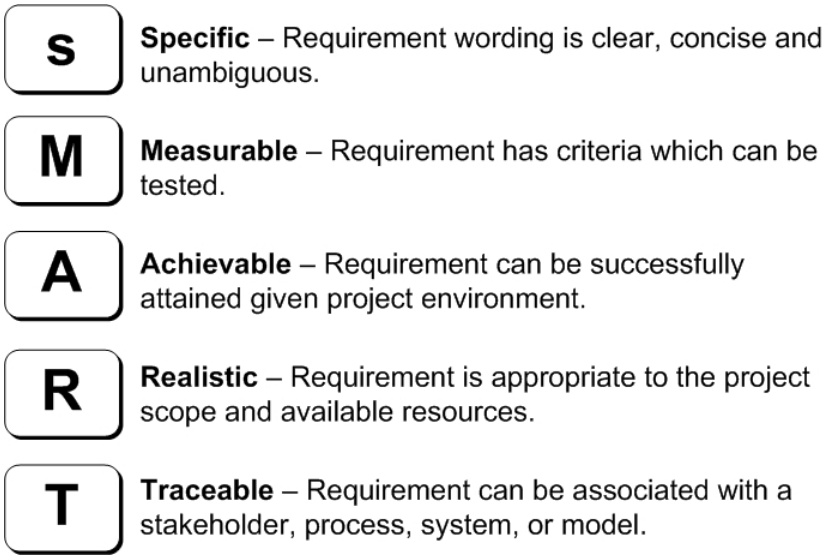


Figure 2.2 SMART Requirements Criteria

Traceable – or requirements traceability – will be a key part of this workshop.

The Building Blocks

Requirements Traceability

The models analysts produce can grow in complexity as understanding of the system deepens. In addition, the models inherently contain dependencies between requirements. As the complexity and dependency increases, requirements and their associated models can become quite unmanageable.

Traceability is a technique that helps manage this complexity. Traceability can help ensure that requirements are efficiently tracked from their initial discovery during the initiation and analysis phases to their ultimate implementation.

Traceability:

> ***Traceability is defined as identifying the relationship between two products (project deliverables or artifacts) in the project life cycle.***

Traceability is illustrated by the following diagram. As you can see, phases and project artifacts are conceptually linked.

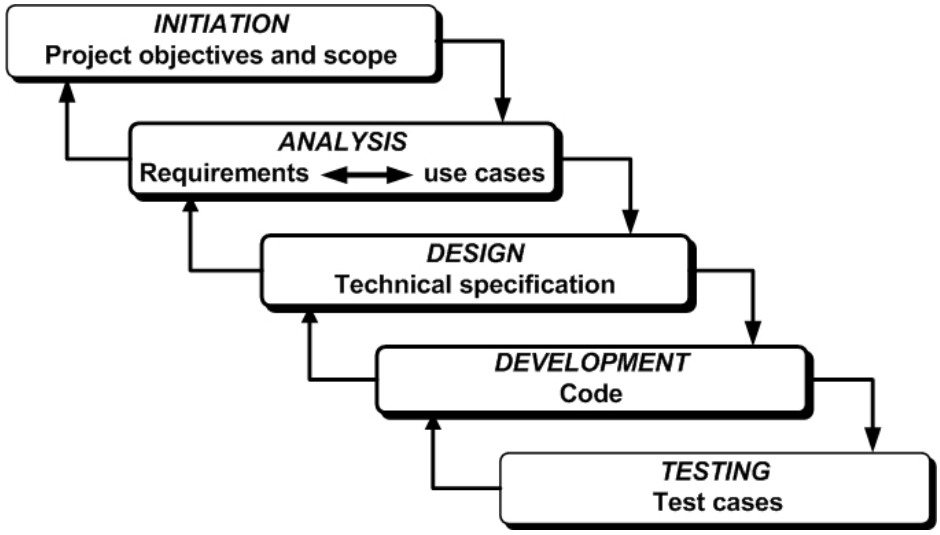


Figure 2.3 Project Life Cycle Traceability

In recognition of the importance of traceability, it has been added as a key activity to project scope management in the latest PMBOK from the Project Management Institute.[8]

How can we perform traceability so that it is not just a concept but a real asset to any project?

[8] A Guide to the Project Management Body of Knowledge, Project Management Institute, Inc. 2008

Traceability matrix:

> ***A requirements traceability matrix is a two-dimensional model relating two project artifacts or deliverables.***

So, what does this definition mean? It means that the relationship between, for example, project objectives and requirements can be identified and documented. It also means that the relationship between requirements and use cases, and use cases and test cases can be identified and documented in the same way. If we start with project objectives and "trace" them throughout the life cycle, we can help ensure that the requirements that are eventually deployed are complete and reflect a common source of information.

A traceability matrix can make the collection and tracking of requirements more manageable. When viewed from the perspective of conceptual modeling, a requirements traceability matrix reaches **backwards** to the initial discovery of either a business or stakeholder requirement, and then forward to its implementation. We will focus on this "***forward traceability***" in this workshop.

An example of a traceability matrix is below. It can be easily created using a table or spreadsheet:

Requirements / Test cases	Test case 1	Test case 2	Test case 3
Search catalog	✓		
Select product	✓		
Check out			✓
Credit cards			✓
Create account		✓	
Update account		✓	

Figure 2.4 Traceability Matrix

The Building Blocks

Benefits

There are many benefits to traceability. Traceability can help **improve the quality** of the implementation by "connecting the dots" between the deliverables or artifacts to ensure that no critical information is lost.

It helps ensure that the **requirements remain complete** through implementation. Conducting traceability activities early and throughout the project ensures that errors and omissions are caught as soon as possible, in time for correction and inclusion.

Traceability makes sure that the **phases in the project life cycle are linked** – the outputs of one phase are the inputs to the next. Traceability can link each phase effectively.

Traceability supports **change control**. As stated above, there are often dependencies between requirements. When a requirement, use case, model or design element is changed, the related components also need to reflect the change. Traceability provides a quick method to determine all the interdependent relationships where a change will need to be implemented.

Traceability also helps analysts **confirm the results of requirements elicitation**. We need to confirm that the requirements match stakeholders' understanding of the problem and need. We can do this by tracing functional requirements "backwards" to project objectives and benefits. We can do this by tracing functional requirements "backwards" to stakeholder requirements. And we can do this by continuously checking to make sure that project artifacts concerning data (data models) trace to requirements (functional requirements or use cases).

Requirements begin to make their appearance in the initiation phase as project objectives. In the analysis phase, as the stakeholders and users provide you with their requirements, they are still high level and ambiguous. As the business analyst progresses through the analysis and conceptual design phases, these stakeholder requirements need to be turned into high quality, traceable, functional requirements.

Best practices acknowledge that requirements never really settle down. Rather, they are continuously and iteratively rediscovered and refined throughout the project life cycle. That is why business analysts continue to be involved with, but not primarily responsible for, the technical phases of the SDLC. This is also why the pure linear path through the project seldom is sufficient. In practice, the phases of the project are revisited time and again as requirements discovery and the needs of the project dictate.

Module 7 System Views

System Views

In this module, you will:

- *Learn the history of the UML*
- *Review the five system perspectives of UML*
- *Draw a boundary between conceptual design and technical design*

The UML System Architecture Viewpoints

In Module 3, you learned about three of the different viewpoints of the UML system architecture. Recall them and write them in the space below with a brief explanation of each.

System Views

The Five Perspectives

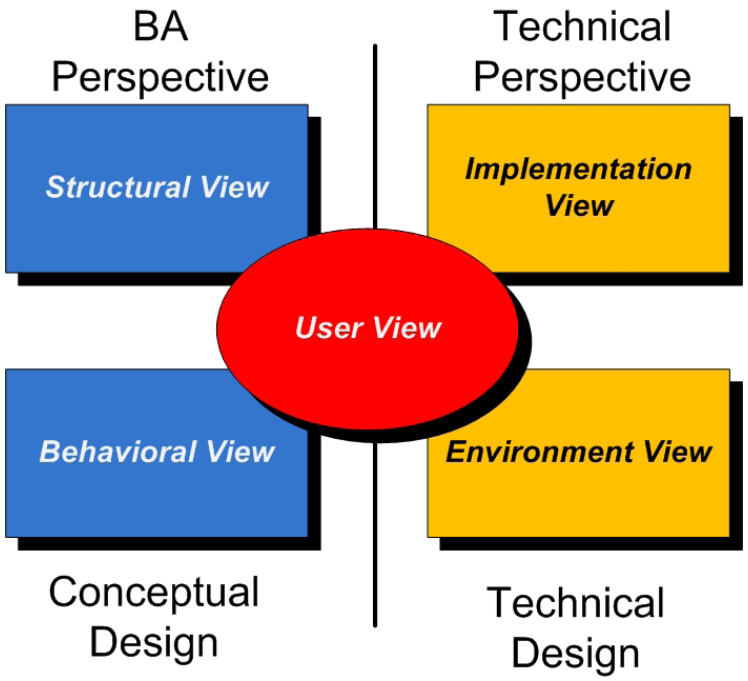


Figure 7.1 The UML System Architecture Model

No one model is sufficient to understand and document a system. UML is designed to support software systems requirements and designs from multiple perspectives or *viewpoints*. There are many modeling techniques for each viewpoint, and it is important to understand how and when to apply one over another.

We reviewed three perspectives already:

Structural view – Models that represent the **nouns** of a system; these are the static elements that represent either conceptual or physical things. The class and object diagrams are examples of structural views.
Behavioral view – Models the **verbs** or the dynamic elements that represent system processes and functions. Activity diagrams and use cases are examples of this view.

User view – Models the behavioral things as seen by the **system's observers**. The business use-case is an example of the user view.

System Views

Differentiation between conceptual and technical design can now be detailed. The conceptual design, in the realm of business analysis includes the three perspectives we have reviewed. The technical perspective includes the last two perspectives:

Implementation view – Encompasses the components used to assemble and release the physical system. The tools and techniques applied in this view are as follow:

- Component diagrams
- Code libraries
- Source code
- Configuration management
- Interaction diagrams with physical details
- State chart diagrams with physical details
- Activity diagrams with physical details

Environment view – Addresses the physical system hardware on which the computer system runs. The tools and techniques applied in this view are as follow:

- Networks and nodes
- Deployment diagrams
- Operating system requirements
- Hardware requirements

Business analysts may achieve competence with UML in line with their role in a project. This role generally is focused on the initiation, analysis, and conceptual design phases of the SDLC. It is easy to get started using UML to elicit requirements and visualize conceptual designs; the hard part is knowing when to stop!

It is generally accepted that a fuzzy boundary exists between the conceptual and technical design phases in the SDLC. A business analyst approaches this boundary from the conceptual design side, and the technical expert approaches it from the technical design side.

Interaction diagrams and activity diagrams can be considered part of the fuzzy boundary, since they are used to document a system on both sides.

The figure on the preceding page renders the boundary more sharply than it should, but the intent is to emphasize the technical perspective as distinct from the business analyst perspective.

System Views

The History of UML

UML is a powerful language. Its power comes from a formal syntax and extraordinarily rich semantics. Its syntactic and semantic depth is a result of its history, which is an interesting story.

Ivar Jacobson developed the concept of software components in 1967 while designing a new generation of software for Ericsson telephone switches. This important insight led him to develop *sequence diagrams* and *collaboration diagrams* as well. In the late 1980s he founded Objective Systems to further his work with object-oriented analysis, design, and implementation.

In the early 1990s, Grady Booch was known for an object-oriented modeling technique developed for use with the Ada programming language. The Booch components were widely applied in Ada programming because they supported analysis, design, and implementation using a rich set of symbols and tools, such as those which modeled collections of software components called *packages*.

Around the same time, Jim Rumbaugh led the development of the Object Modeling Technique (OMT) at General Electric's Research and Development Center.

The development of UML began in the mid 1990s, when Grady Booch, Jim Rumbaugh, and Ivar Jacobson all found themselves at Rational Software Corporation. The *Unified Modeling Language* is so-named because it unifies the theory and practice of these three original thinkers.

Today, UML is an international standard maintained by the Object Management Group (OMG). It is intended to support all of the following for a software-intensive system:

- ☑ Visualizing
- ☑ Specifying
- ☑ Constructing
- ☑ Documenting

In addition, today many parts of UML are used in systems projects, regardless of whether object-orientation is the chosen methodology. Use cases and activity diagrams are common tools of the business analyst. Even the swim lane diagrams used in ***Fast Start in Business Analysis©*** have their origin in UML.

Module 8

Conceptual Data Models

Conceptual Data Models

In this module, you will:

- *Rediscover the principle of abstraction*
- *Learn about information engineering*
- *Produce a conceptual data model from requirements*

Conceptual Data Models

The Principle of Abstraction

You first encountered the principle of ***abstraction*** in Module 2. You learned that abstractions are important to business analysis and modeling. What is the definition of abstraction?

You touched on structural models through the brief discussion of objects and classes. In this module, we will return to structural models to document the data requirements of a system.

Conceptual Data Models

Information Engineering

> ***Information engineering approaches system development from the perspective of process and data combined: it defines the processes the system must perform, and the data that is used by the processes.***

Information engineering (IE) was developed by James Martin.[15] IE is commonly used to model data requirements, based on the details found in functional requirements.

A data model is a structural system model based on entities.

Entity:

> ***An entity is a person, place, thing, or concept that has characteristics of interest to the enterprise. In other words, an entity is an abstraction of something of value about which data are stored.***

Business analysts discover entities – ***nouns*** – by analyzing stakeholder and functional requirements. Processes are described as ***verbs***. For data models, the focus is on *who* and *what*, exclusively. Time-dependent information is not a factor. In fact, introducing time into a data model is incorrect.

Each entity or noun –

- ☑ Has a valid name that follows an established naming convention
- ☑ Must be completely documented in a dictionary
- ☑ Usually has a relationship with at least one other entity

Just as for great buildings, great data models are built up from simple, stable foundations. We begin with *conceptual* data modeling as the first step.

[15] To consult the original source, see James Martin, Information Engineering: Book II Planning and Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1990.

Conceptual Models

A conceptual data model depicts a general idea of the collection of entities and their relationships. A good conceptual data model establishes the boundary for communicating data requirements. In other words, it tells you which things (entities) are necessary to understanding and documenting the requirements, and what their basic relationships might be.

Conceptual models are the first step in data modeling. They are often done in parallel with identifying stakeholder requirements, or as the functional requirements begin to come alive. As such, a conceptual model is often based on a partial understanding of the requirements.

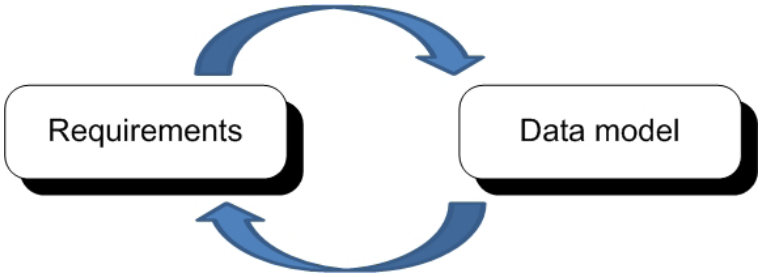


Figure 8.1Iterative nature of data modeling

As we have stated in all of the previous workshops, eliciting and documenting requirements is iterative. This is also true for data modeling. As requirements are uncovered, the data models are updated. And as the data model begins to be produced, it will likely uncover additional requirements.

Conceptual Data Models

Entity Relationship Diagrams

Data models are rendered as ***entity relationship diagrams***. Abbreviated ***ERD,*** an entity relationship diagram is a powerful graphical tool developed originally by Peter Chen at MIT in 1976.[16] Relational theory began its rise to dominance for data modeling and database management system implementation around this time. The techniques of data modeling were developed to a high art during the last quarter of the 20th century, and their development paralleled the technical and commercial success of relational database management systems such as Oracle, DB2 and SQL.

There are direct parallels between ERDs and the class diagram discussed previously. In Module 4, using UML, we called the system "things" or nouns ***classes***. IE calls these nouns ***entities***. We saw ***class diagrams*** produced by UML. IE produces ***ERD***s.

Consider the following requirement:

> ***A customer may submit many purchase orders, and every purchase order is uniquely associated with only one customer.***

In this requirement, there are two obvious entities: customer and purchase order. This is illustrated in the simple ERD below.

Figure 8.2 Simple Conceptual ERD

[16] Peter Chen, "The Entity-Relationship Model: Towards a Unified View of Data", *ACM Transactions on Database Systems*, Vol. 1, No. 1, March 1976, pages 9-36.

Here's how IE data modeling works:

1. Each **entity** is represented by a rectangle containing the entity name. The name, always a singular noun, may be simple (CUSTOMER) or compound (PURCHASE_ORDER).
2. A **relationship** is a line representing a relationship between two entities. A relationship always is bi-directional, and is interpreted from the perspective of both entities. That is, the relationship is read from both left to right, and right to left.
3. There are many types of relationships, represented by a **connector** on the relationship line.
4. The symbol of the connector denotes the **cardinality**[17] of the relationship.
5. Cardinality can be zero, one or many.

Let's look at some examples to understand the bi-directionality of modeling. Take the example ERD on the previous page and break it into its two components:

> ***A customer may submit many purchase orders, and every purchase order is uniquely associated with only one customer.***

Reading from left to right: For one customer record, there may be zero associated purchase order records at a minimum, and there may be any number of them at a maximum.

Reading from right to left: One purchase order record must have one and only one associated customer record.

Figure 8.3 ERD Fragments

[17] UML uses the word *multiplicity*

Conceptual Data Models

Cardinality:

Cardinality defines the relationship between entities and is documented through the use of symbols on the connector.

- *A connector belongs to the entity at the opposite end of the line.*
- *There are two parts to each connector: the minimum cardinality is nearest the middle of the line, and the maximum cardinality is nearest the end of the line.*

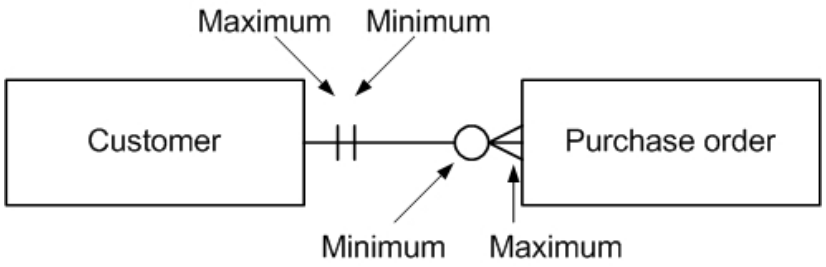


Figure 8.4 Components of an ERD

The following figure provides a quick reference for cardinality semantics in information engineering.

IE Cardinality Symbol	Explanation
	Simple relationship (used at conceptual level, only)
	One to many (used at conceptual level, only)
	Many (used at conceptual level, only)
	One or more; required many
	None or more; optional many
	One and only one; exactly one
	None or one; optional one

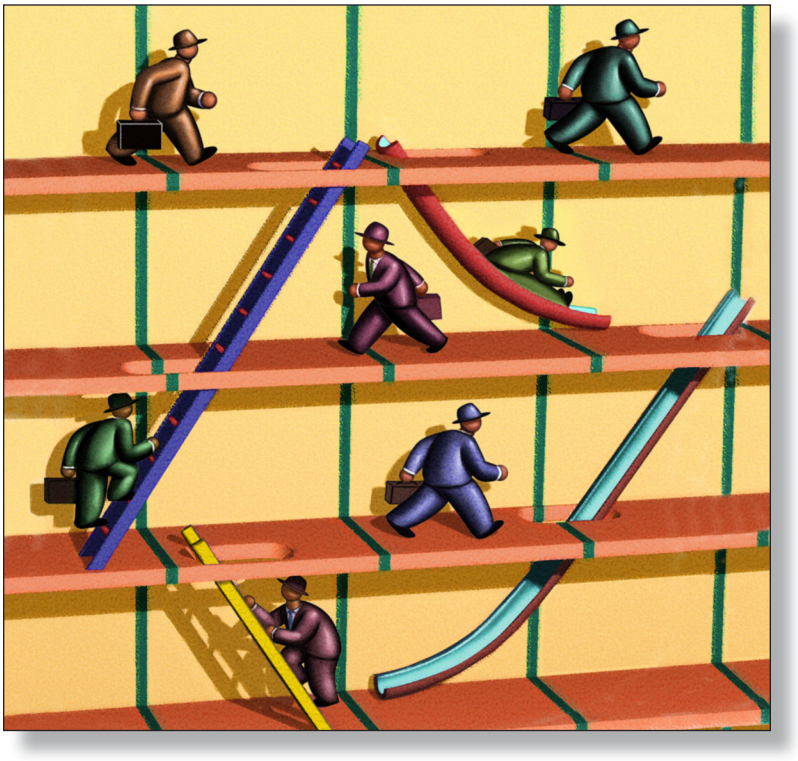
Figure 8.5 IE Cardinality Semantics

Conceptual Data Models

At this point, you have rediscovered the principle of abstraction and applied it to the problem of conceptual data modeling.

Your next challenge is to learn and apply the prescribed methodology to logical data modeling.

Module 11 Value Added Modeling



Value Added Modeling

In this module, you will:

- *Understand the value of requirements management and confirmation*
- *Think about how to use and apply models from various approaches*
- *Discover requirements prioritizations schemes*
- *Understand requirements re-use*

We have covered a lot thus far:

- ☑ *Modeling and traceability can effectively support the requirements management and elicitation processes*
- ☑ *Behavioral modeling can represent system processes*
- ☑ *Structural modeling can represent data requirements*
- ☑ *The link between requirements and data models is iterative and produces value to the project*
- ☑ *The quality of the data model will improve the quality of the requirements as a whol*e

There are a few more important concepts remaining to ensure that this workshop lives up to its name as an advanced workshop.

Value Added Modeling

Value added modeling is the use and application of models to manage project requirements and ultimately improve project quality and success. The act of producing a model is not enough. Good business analysts must analyze, synthesize and interpret models for effectively managing and confirming requirements. And as stated many times before, managing requirements involves eliciting information, analyzing and documenting it, and finally communicating the information for validation and confirmation to a diverse audience.

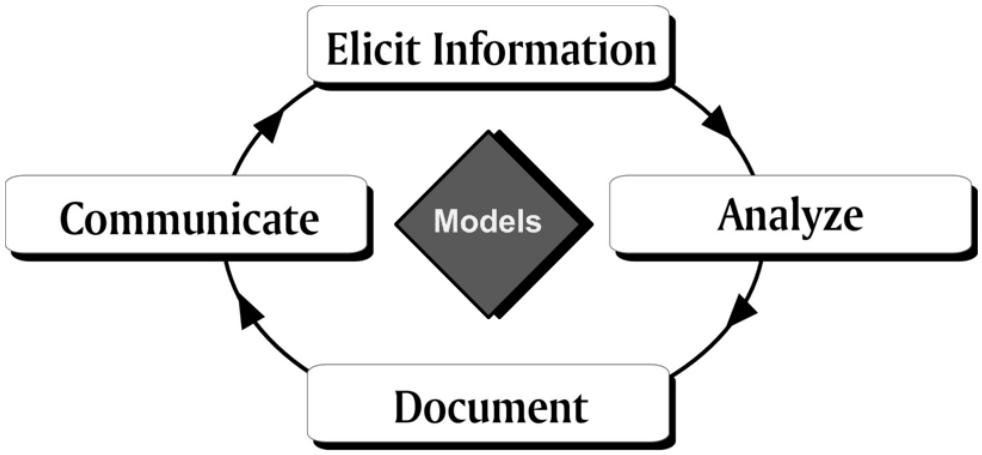


Figure 11.1 Value Added Modeling

Value added modeling is the construction and application of models to:

- Confirm requirements effectively
- Perform requirements traceability to ensure that the solution delivered is the solution desired
- Prioritize requirements so that resources can be appropriate assigned
- Ensure that requirements can be re-used by future projects
- Support the transfer of knowledge from one project to another through the use of a dictionary

Value Added Modeling

Requirements Prioritization

Requirements are important to the success of a systems project. As you well know, a project can have many, many requirements. But how do we know which ones are more important than others? (Yes – we know that stakeholders will claim that all are equally important!)

To prioritize requirements, criteria must be identified and agreed. Priority may be based on criteria such as:

- Relative customer value
- Potential revenue
- Risk of not doing it
- Resource considerations
- Difficulty of implementation
- Competitive pressure
- Generic or neutral rankings such as
    - must have / nice to have / wish list
    - high / medium / low

Use the scheme that makes most sense to you and your stakeholders. But recognize that evaluating requirements against any criteria is subjective. An effective business analyst may be asked to facilitate prioritization meetings amongst stakeholders. The first step is to agree how to prioritize, and to make sure it is well documented. Remember, that prioritization may occur frequently throughout a project, as business needs change.

At the end of the process, each requirement should have an assigned priority. These are then used to determine which requirements should be implemented first; which requirements should have resources assigned to them.

Requirements Re-use

We have stated many times in this workshop that requirements, in their various forms, can be documented for re-use in future projects to save time and money:

- ☑ Classes or entities
- ☑ Objects
- ☑ Dictionary entries
- ☑ Data models
- ☑ Use cases

Object orientation, through its use of UML is particularly good for requirements re-use. Classes and objects can be re-used by other projects, once defined. Use cases, through the three relationships defined (generalization, include, extend) are also valuable in the effort to re-use requirements.

The logical data models, whether they are UML class diagrams or IE entity relationship diagrams, typically remain static and are used consistently by many systems in an organization. Once the entities (or classes) are defined and documented in the dictionary, many projects can take advantage of the data requirements embedded in these models.

Many requirements are written for vendor requests for information, quotes or proposals. Quite often these include both functional and non-functional requirements. In this arena, many of the requirements can form a repository of "standard" requirements against which to evaluate vendors.

Similarly, requirements surrounding security, privacy, system performance, business continuity and disaster recovery may be written once and used across many projects. Even requirements related to compliance to laws and policies, such as HIPAA and Sarbanes Oxley, can be re-used by future projects.

When these types of requirements are discovered, best practice would dictate that they be documented carefully and stored in a central repository for access and re-use in the future.

Value Added Modeling

Isomorphism

A business analyst can be expected to integrate multiple models in a project so that a meaningful story about the requirements can be told. It is neither necessary nor practical to use just one modeling technique. As powerful as UML is, there still are story-telling deficiencies lurking in the syntactic structure and semantic expressiveness of the language. The same may be said for every other modeling technique.

That is why this workshop presented IE along with UML. Previous workshops introduced other models (functional flows, swim lanes, process scripts, context models); future workshops will do the same (storyboards). ***Our objective is to provide you with choice.***

To successfully complete requirements analysis and transition to the technical design phase of a project, the models, regardless of their approach, must be consistent, unambiguous, traceable, and understandable by all project stakeholders. Whether words or graphic symbols, a business analyst must construct a doorway into the requirements documentation through which stakeholders will want to enter. In short, a business analyst must invite interested parties by integrating structural and behavioral models through views relevant to the stakeholders.

Stakeholders want to see something of value as soon as they pass through this doorway.

The concept of isomorphism can help this process.

Isomorphism:

> ***Isomorphism refers to the concept that regardless of the technique used, the meaning remains the same. It can be said that there is a one-to-one correspondence between two models; one can be converted into another without loss of information.***

Isomorphism helps to convey meaning to project stakeholders in multiple ways. If one particular model does not align with the observer's point of view, then another, equivalent model might.

The critical example of isomorphism in this workshop are class and entity relationship diagrams. Even though the syntax is different between the two approaches, the meaning remains the same.

So Which Approach?

You have learned two approaches to modeling in this workshop: object orientation using UML and information engineering.

The choice of one modeling approach over another is driven by several factors:

- ☑ Organization methodology
- ☑ Individual skill level and style of analyzing information
- ☑ Audience and their expectations and particular style of receiving information
- ☑ Project constraints
- ☑ Practical considerations such as available software

Many organizations integrate the two approaches as follows:

- Employ UML modeling techniques for behavioral modeling
  - Use cases
  - Activity diagrams (sometimes known as swim lanes or process flows)
- Utilize IE for modeling data taking advantage of the powerful process of normalization, quality rules, and semantic analysis
- Create a unified dictionary containing all the necessary elements
  - Project specific vocabulary
  - UML elements such as use cases and activity diagrams
  - Requirements traceability matrixes
  - Data dictionary to support data models

Regardless of the approach or modeling technique used, requirements management can be effectively accomplished.

Value Added Modeling

We hope you have enjoyed the journey through modeling.

As you can now see, creating a model is only the first step. Using models to confirm project objectives and requirements, although an iterative process is how models can add value to the project's outcome.

Good luck as you continue your journey.