## Specification

There is no doubt that the mode of specification has much to do with the quality of solution.

Software engineers who have been forced to work with incomplete, inconsistent, or misleading specifications have experienced the frustration and confusion that invariably results. The quality, timeliness, and completeness of the software suffer as a consequence.

Specification, regardless of the mode through which we accomplish it, may be viewed as a representation process. Requirements are represented in a manner that ultimately leads to successful software implementation.

Software requirements may be specified in a variety of ways.

### Use case

A use case is a technique for documenting the potential requirements of a new system or software change. Each use case provides one or more scenarios that convey how the system should interact with the end-user or another system to achieve a specific business goal. Use cases typically avoid technical jargon, preferring instead the language of the end-user or domain expert. Use cases are often co-authored by requirements engineers and stakeholders.

Use cases are deceptively simple tools for describing the behavior of software or systems. A use case contains a textual description of all of the ways which the intended users could work with the software or system. Use cases do not describe any internal workings of the system, nor do they explain how that system will be implemented. They simply show the steps that a user follows to perform a task. All the ways that users interact with a system can be described in this manner.

### SRS

The Software Requirements Specification is produced at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by establishing a complete information description, a detailed functional description, a representation of system behavior, an indication of performance requirements and design constraints, appropriate validation criteria, and other information pertinent to requirements.

SRS is a complete description of the behavior of a system to be developed. It includes a set of use cases that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional (or supplementary) requirements. Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints).

A software requirements specification (SRS) is a comprehensive description of the intended purpose and environment for software under development. The SRS fully describes what the software will do and how it will be expected to perform.

An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost. A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations. Parameters such as operating speed, response time, availability, portability, maintainability, footprint, security and speed of recovery from adverse events are evaluated.

SRS serves as contract between customer & developer.

**Characteristics of a good SRS:**

1. Correct
2. Unambiguous
3. Complete
4. Consistent
5. Ranked for important and/or stability
6. Verifiable
7. Modifiable
8. Traceable

**Correct**: An SRS is correct if and only if every requirement stated therein is one that the software shall meet.

**Unambiguous**: An SRS is unambiguous if and only if, every requirement stated therein has only one interpretation.

**Complete**: An SRS is complete if and only if, it includes the following elements

(i) All significant requirements, whether related to functionality, performance, design constraints, attributes or external interfaces.

(ii) Responses to both valid & invalid inputs.

(iii) Full Label and references to all figures, tables and diagrams in the SRS and definition of all terms and units of measure.

**Consistent**: An SRS is consistent if and only if, no subset of individual requirements described in it conflict.

**Ranked for importance and/or Stability:** If an identifier is attached to every requirement to indicate either the importance or stability of that particular requirement.

**Verifiable:** An SRS is verifiable, if and only if, every requirement stated therein is verifiable.

**Modifiable:** An SRS is modifiable, if and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining structure and style.

**Traceable:** An SRS is traceable, if the origin of each of the requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation.

# Analysis Modeling

**What is it?**

Analysis modeling uses a combination of text and diagrammatic forms to depict requirements for data, function, and behavior in a way that is relatively easy to understand, and more important, straight forward to review for correctness, completeness, and consistency.

**Who does it?**

A software engineer (sometimes called an analyst) builds the model using requirements elicited from the customer.

**Why is it important?**

To validate software requirements, you need to examine them from a number of different points of view. Analysis modeling represents requirements in three “dimensions” thereby increasing the probability that errors will be found, that inconsistency will surface, and that omissions will be uncovered.

**What are the steps?**

Data, functional, and behavioral requirements are modeled using a number of different diagrammatic formats.

Data modeling defines data objects, attributes, and relationships.

Functional modeling indicates how data are transformed within a system.

Behavioral modeling depicts the impact of events.

Once preliminary models are created, they are refined and analyzed to assess their clarity, completeness, and consistency. A specification incorporating the model is created and then validated by both software engineers and customers/users.

**What is the work product?**

Data object descriptions, entity relationship diagrams, data flow diagrams, state transition diagrams, process specifications, and control specifications are created as part of the analysis modeling activity.

## Data modeling

Data modeling answers a set of specific questions that are relevant to any data processing application. What are the primary data objects to be processed by the system?

What is the composition of each data object and what attributes describe the object? Where do the objects currently reside? What are the relationships between each object and other objects? That is the relationships between the objects and the processes that transform them?

To answer these questions, data modeling methods make use of the entity relationship diagram.

Creating Entity Relationship Diagrams:

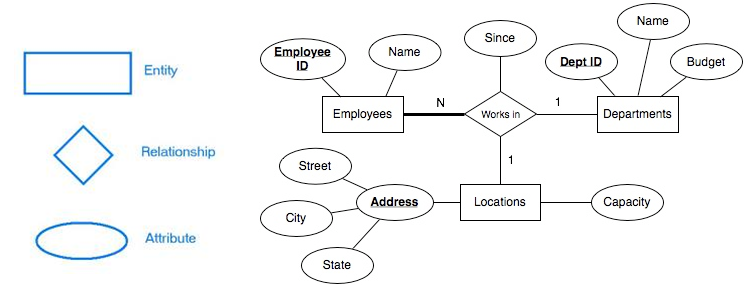
* Customer asked to list "things" that application addresses, these things evolve into input objects, output objects, and external entities
* Analyst and customer define connections between the objects
* One or more object-relationship pairs is created for each connection
* The cardinality and modality are determined for an object-relationship pair
* Attributes of each entity are defined
* The entity diagram is reviewed and refined

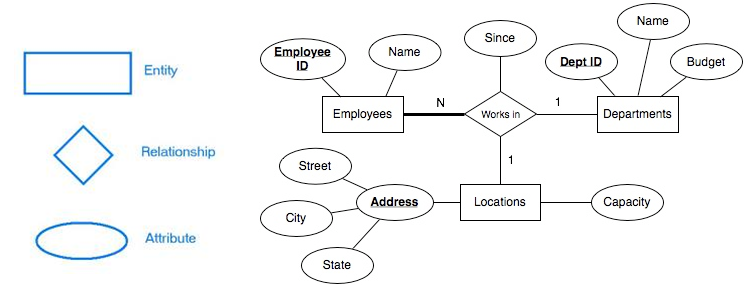
ERD elements:

* **Data object** - any person, organization, device, or software product that produces or consumes information
* **Attributes** - name a data object instance, describe its characteristics, or make reference to another data object
* **Relationships** - indicate the manner in which data objects are connected to one another
* Cardinality and Modality

**Cardinality** - in data modeling, cardinality specifies how the number of occurrences of one object is related to the number of occurrences of another object (1:1, 1: N, M: N)

**Modality** - zero (0) for an optional object relationship and one (1) for a mandatory relationship





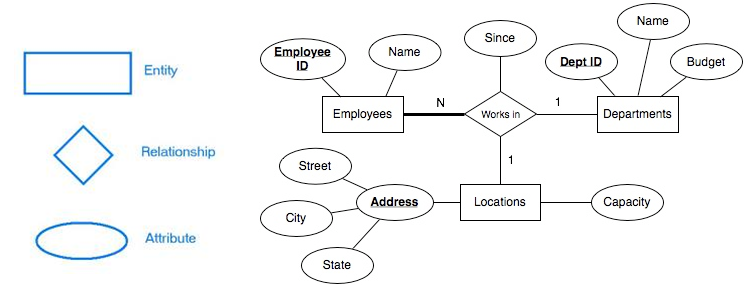
## Functional Modeling

Software transforms information, and in order to accomplish this, it must perform at least three generic functions: input, processing, and output. When functional models of an application are created, the software engineer focuses on problem specific functions. We use DFDs for functional modeling.

DFD: A **data flow diagram** (**DFD**) is a graphical representation of the "flow" of data through an information system. A DFD shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. DFD shows the relationships of external entities, process or transforms, data items, and data stores. DFD's cannot show procedural detail (e.g., conditionals or loops) only the flow of data through the software

Creating Data Flow Diagram

* Level 0 data flow diagram should depict the system as a single bubble
* Primary input and output should be carefully noted
* Refinement should begin by consolidating candidate processes, data objects, and data stores to be represented at the next level
* Label all arrows with meaningful names
* Information flow must be maintained from one level to level
* Refine one bubble at a time
* Write a PSPEC (a "mini-spec" written using English or another natural language or a program design language) for each bubble in the final DFD



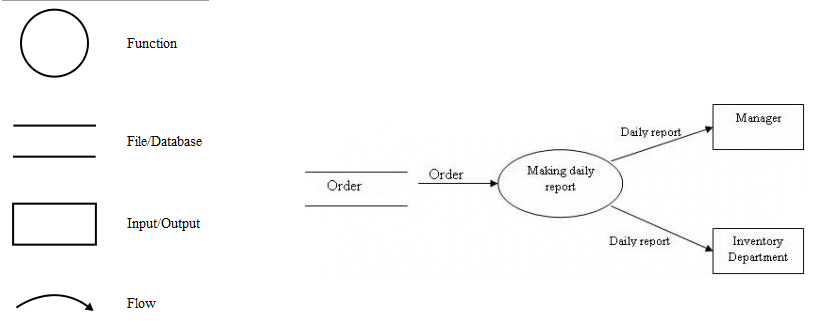
## Functional Modeling

Software transforms information, and in order to accomplish this, it must perform at least three generic functions: input, processing, and output. When functional models of an application are created, the software engineer focuses on problem specific functions. We use DFDs for functional modeling.

DFD: A **data flow diagram** (**DFD**) is a graphical representation of the "flow" of data through an information system. A DFD shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. DFD shows the relationships of external entities, process or transforms, data items, and data stores. DFD's cannot show procedural detail (e.g., conditionals or loops) only the flow of data through the software

Creating Data Flow Diagram

* Level 0 data flow diagram should depict the system as a single bubble
* Primary input and output should be carefully noted
* Refinement should begin by consolidating candidate processes, data objects, and data stores to be represented at the next level
* Label all arrows with meaningful names
* Information flow must be maintained from one level to level
* Refine one bubble at a time
* Write a PSPEC (a "mini-spec" written using English or another natural language or a program design language) for each bubble in the final DFD



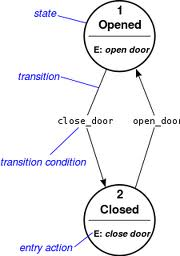
## Behavioral Modeling

Most software responds to events from the outside world. This response characteristic forms the basis of the behavioral model. A computer program always exists in some state—an externally observable mode of behavior (e.g., waiting, computing, printing) that is changed only when some event occurs. For example, software will remain in the wait state until (1) an internal clock indicates that some time interval has passed, (2) an external event (e.g., a mouse movement) causes an interrupt, or (3) an external system signals the software to act in some manner. A behavioral model creates a representation of the states of the software and the events that cause software to change state.

**The behavioral model indicates how software will respond to external events**. To create the model, the analyst must perform the following steps:

* Evaluate all use-cases to fully understand the sequence of interaction within the system.
* Identify events that drive the interaction sequence and understand how these events relate to specific objects.
* Create a sequence for each use-case.
* Build a state diagram for the system.
* Review the behavioral model to verify accuracy and consistency.

Sate Transition Diagram

* State transition diagrams represent the system states and events that trigger state transitions
* STD's indicate actions (e.g., process activation) taken as a consequence of a particular event
* A state is any observable mode of behavior
* UML sequence diagrams can also be used for behavioral modeling