What is Analysis?

A principal problem in systems development is the discovery, documentation and management of the system owner's requirements. A simplistic view of analysis requirements is that they state WHAT is required not HOW it is to be implemented. The HOW is design. Simple yes, but extremely important to separate analysis from design.

The transition from analysis to design is iterative. For example, a requirement might be to get people from one side of a river to the other. This is a statement of what is required. Clearly this must be known before we can proceed. We must now decide what means will be used to transfer the people across the river, e.g., a bridge, a ferry or a tunnel. Making this decision is applying a constraint to the requirement for crossing the river. Now if the decision is to use a bridge to span the river, this becomes a new requirement that must be analyzed.

What then are the requirements for a bridge? There must be access, egress, clearances, etc. The decision as to the type of bridge is another constraint or design decision, e.g., to build a suspension bridge, a movable bridge, etc.

The purpose of the requirements document is to specify what the system must do. Any design built into the requirements must always be satisfied. This may preclude other design decisions in the future because they would violate the requirements. Thus it is imperative to keep design out of the requirements specification.

What is a Requirements Document?

The requirements document is the official statement of the system requirements for all affected parties, i.e., the system stakeholders - the system owner, users, analysts, designers, and quality assurers. It should be a document that the system owner can easily understand and be sure of its correctness and completeness. Furthermore the designer must understand and interpret the requirements document in exactly the same way as the system owner.

What form should the requirements document take? A narrative is probably the worst form, yet this is the most common. We will see shortly that a model that has a diagrammatic rather than linguistic form will be the best choice. Diagrams should be used in a requirements document whenever it is necessary to specify structural relationships.

Textual description of a layout of a house versus a pictorial/graphical description

Imagine a situation in which you have contracted with an architect/builder for a new house. Assume there is no way of showing a model for this structure; all we have is textual narrative to describe the proposed house. No pictures, no plans, no visits to a similar "model" house, only a 100 page narrative. Then for example we might find the following description:
"The living room faces north-northwest, is trapezoidal in shape and will be 23 feet at its greatest width and 15 feet at the opposite wall. The west wall is 13 feet 4 inches long and abuts the southern portion of the east wall of the kitchen. ....."

Can you visualize this? Assuming that you can figure it out, you most certainly will have to draw it to get a sense of how this part of the house will look. Would you be satisfied with this type of narrative description? Would you be willing to give a go ahead on the construction based upon this type of specification? This is what is normally available to a user/owner of a software system being developed.

Why is software industry in the state that it is?

- What is the software crisis?
- Is the software that you use a quality product?
- Do you feel confident in the software that you use?
- Does it matter whether it is Commercial Off the Shelf (COTS) or custom built software?
- Does it do what was expected? Satisfy needs?

Manufactured (hard) items typically do what is expected, e.g., PC, car, or a toaster. We know how to construct "hard" items well; we have good engineering methodology for "hard" things. We have the technology for software, but don't do it well. Why?

- Analysis is concerned with WHAT, i.e., building the right product.
- Design is concerned with HOW to build it well - high quality, effective, efficient, highly maintainable.

For example, the best TV may not be the easiest (least expensive) to repair (maintain). If the TV design is modular, then replacing a module may be all that is required. If the design is not modular, then more extensive and time-consuming discovery is necessary to find the source of the problem. In a modular design, functionality should be isolated to one (or a few modules). Thus if the problem is "no sound", a replacement of the module that contains the audio logic is all that is required. If the design is not modular, finding and replacing components may also contribute to new problems (ripple effect). The concepts at work here are "high cohesion", "low coupling". These will be discussed later.

Software - an ill-conceived term

What is wrong with the term software? When you build a manufactured product, it is done. You can't change it after it is constructed (manufactured). For example, when a truck is built, no one expects that it can be modified for the particular owner (user). Yet with software, this is not the case. It is assumed that it can be modified easily and cheaply.
In the 1960's, labor was cheap and hardware was expensive. Thus it was assumed that to modify software was inexpensive since labor was cheap. It appeared to be relatively inexpensive to change the software. After all, that's what software is about.

Who wrote the first programs? They were techies (original hackers?) who did not understand the organization and its requirements, but could make the computer do its thing. This set a bad tone early on, and we are still suffering from this. Management left all computer decisions to these techies because they believed that they couldn't understand the technology.

We are almost totally dependent on IT for our very existence. (Note the recent pager satellite failure.) Almost everything we use is IT dependent.

The software industry got off on wrong the foot in believing that software was so easy to change that we did not have to plan and design carefully. It is not like working in clay instead of marble. There don't appear to be any wasted (expensive) materials if we have to re-code. It is hard to realize that large amounts of resources are wasted when we have to do much code redesign. Organizational management does not seem to get upset about large software project cancellations after millions are spent, but in a hard manufacturing operation, this would not be tolerated.

**Factory Construction Example**

An owner wants a factory built. An architect is called in (analyst) to capture the requirements. This process is similar to software development.

How does the building owner (system owner) know that the architect (analyst) has correctly and completely captured the requirements so that the right building (system) will be built (VALIDATION)?

In the traditional software development process, a document called a Functional Requirements Specification (FRS) is used as a contract between the system owner and the development organization. But the system owner cannot truly be sure that all the requirements are correct and complete. The FRS is a textual (narrative) document with a few graphical elements - screen (input) layouts and report (output) layouts. These are design, but the user can relate to them. There is no assurance that these layouts are correct.

In a building (factory) construction, the owner typically is given a scale model, artist renderings, blueprints (leveled diagram set), and some supporting narrative. The owner can relate to these and be able to validate that the requirements are captured correctly. The owner is knowledgeable about what architectural components are needed in a building construction, but not necessarily in an information system construction. (Note: this should not matter.)
There are few surprises when a building is constructed. There are many when a software system is constructed.

The following anecdote illustrates the traditional software development methodology applied to the factory construction.

The owner talks to the architect. As is typical in such interviews, the owner/user continually emphasizes one (or a few) aspect of the requirements. In this case, he explains that he selected this building site since it is near the interstate. Trucks come in and out to load and unload continuously. A large turnaround area is needed so trucks can easily come in, turn, go to loading bay, load or unload, and leave. Also there should be a means to efficiently move the raw materials to the manufacturing area and the finished goods to the warehouse area and the shipped goods to the trucks. Throughout the interview (fact-finding) process, the owner keeps stressing the importance of the turnaround area for the trucks.

The owner sees some text descriptions and some rough sketches and assumes the architect and his construction crews know what they are doing. He is given a specification document (FRS) that is almost 100% narrative (text), and is asked to sign off so that the project may proceed. So he signs off on the specification and goes away for nine months on an around the world cruise. (In a software system development, the owner/user there sees little evidence that anything is occurring during the development. The owner might just as well not be there.)

When he returns, the factory is completed. The architect is waiting at the entrance to the driveway. He wants to show the owner the wonderful drive and turnaround area for the trucks (remember that was very important to the owner). The architect is very excited. He shows the owner how easily the trucks will be able to turn around. The owner agrees that the trucks will have no trouble turning around. However, where the loading bays were supposed to have been, there is a blank wall. The owner asks the architect what happened to the loading bays. How will the goods be unloaded and loaded? The architect says that he vaguely remembers something about loading bays but it is not in the requirements specification. Sound familiar?

Now because the architect and the builder believed that there would be a blank wall at this location, they made it a load-bearing wall and placed the office wing behind it. (The specification did say that there should be no windows in the office area to reduce the noise and other distractions.) It is not possible to merely cut holes in the blank wall to install the loading bays. The entire structure and layout of the building needs to be redesigned and constructed. It is probably less expensive to tear the building down and start over. The cost to retrofit the loading bay requirement is many orders of magnitude greater to do now than at the requirements gathering stage.
Unfortunately, in software development this happens all the time. The rationale is that no "hard" materials are wasted as in the construction case. It is software after all. It turns out that the cost of redesigning the software is probably more than the materials cost in the building construction. Software is entirely labor-intensive.

Why did this happen?

The owner had no way to validate the requirements before the design was started. In the real world of construction, the user would have seen a scale model, artist renderings, and blueprints, all of which would have clearly shown that the loading bays were missing.

We need to provide better and clearer documentation for the user-owner of a software system so that the requirements can be validated by the system owner. We will see that the analysis methodology and modeling tools that will be discussed in this course will provide this. The documentation tool that we will use will be a Data Flow Diagram (DFD).

It is an order of magnitude more expensive to do something after a development phase is over. To make requirement change in design is an order of magnitude greater than had it been done in the analysis phase. To make a change in code after design is 1 to 2 orders of magnitude greater, etc.

Whose fault is it?

Analysis is an owner responsibility. The system owner is the knowledgeable expert about the system requirements and must take responsibility for the analysis. Organization management must budget resources so that the organizational staff will be available to participate in the analysis. This is almost never done. The organizational staff quickly becomes unavailable to the analysis team because of their normal work demands. They will often tell the analysts that they know more than enough about the system and can complete the requirements specifications without further owner/user input. This is the beginning of the end.

If any one group is to blame for the poor state of software development, it is organizational management. They have never assumed the same responsibility and oversight that would be normal in any other development activity. It is the responsibility of the systems development group to insist that organizational management be appropriately involved.