## Analysis Techniques

Now that we have seen each phase of the engineering design process described in simple terms, let us explore each phase in more detail.

The chances of a product being developed on time and within budget are somewhat slim unless the members of the software development team agree on what the software product will do. Only after a clear picture of the present situation has been gained can the team attempt to answer the critical question:

> *"What must the new product be able to do?"*

The real objective of the requirements phase is to determine what software the client **needs**. This problem is exacerbated by the fact that many clients do not know what they need.

Furthermore, even if the client has a good idea of what is needed, he or she may have difficulty in articulating these ideas to the developers, because most clients are less computer literate than the members of the development team. There is a famous quote uttered by a US politician back in the 60's as he realised he'd dropped a gaff and was rapidly trying to back-peddle his way out.

> *"…I know you believe you understood what you think I said, but I am not sure you realise that what you heard is not what I meant!"*

This excuse applies equally well to the issue of requirements analysis. The developers hear their client's requests, but what they hear is not what the client should be saying.

### Analysis Techniques

The principle objective of analysis is to uncover an understanding the flow of information or data in the system, and/or its behaviour. That is, we seek an understanding of how information is transformed by the system, during its passage from input to output. We also seek to understand the systems interface characteristics and constraints e.g. this machine does not accept 25 cent coins.

Analysis then attempts to uncover '***what'*** the product does, and '***how it behaves***' it does **not** concentrate in any way on ***how*** the software will be implemented. That comes later. For example, analysis might concentrate on the following issues:-

- What data or events are input/recognised by the system?
- What functions/transformations must the system provide?
- What interfaces have been defined?
- What constraints/limits apply?

## Analysis Techniques

There is no one single analysis technique that is guaranteed to work and lead to a professional developed end product. Rather, several different techniques exist that are often combined, with each yielding up new information and understanding to the analyst. Here are some the most popular that have a proven track record.

### Technique 1 - Meetings and Discussions

One of the best *initial* analysis techniques is *communication*. Hold frequent *discussions* and *reviews* with the customer. This may sound obvious, but it is important to ask the right questions if it going to be of use. Try using a combination of both open and closed questions with your customer. For example, an open question might ask them to explain what is wrong with their present arrangement, or to explain how the system should work. This will get them talking about the system and its solution from their point of view. You could also ask them to take you a step-by-step sequence of typical operations, actions and/or calculations performed by the system, from initial input to final output.

Closed questions are designed to seek *detailed clarification* of specific points. For example, what resolution graphics card will be required? What is the format of the report produced by the program? How fast should the system be in dealing with data? As your understanding of the system grows, you will be able to pose lots of 'What should happens when 'X' takes place' questions to uncover the systems behaviour.

Both types of questions can lead to an increased understanding of the system for both customer and developer alike. The result of such meetings is usually a written report, which the developer and customer agree or refine.

Most important of all, don't make *assumptions* about how the software should work, in the absence of a clear explanation. If necessary, go back to the customer, discuss any problems with them and seek clarification of any ambiguities.

### Technique 2 - Generate a Questionnaire

Another way of gaining understanding is to send a questionnaire to the relevant members of the client organisation. This technique is useful when the opinions of, say, hundreds of individuals (or stakeholders) need to be determined.

Furthermore, a carefully thought-out written answer may be more accurate than an immediate verbal response to a question posed by an analyst. However, a meeting conducted by a methodical interviewer who listens carefully and poses questions that expand on initial responses will usually yield far better information than a thoughtfully worded questionnaire. However, because questionnaires are not interactive, there is no way that a new or modified question can be posed in response to an answer.

**Technique 3 - Analyse the Customers Current Methods**

A different way of obtaining information, particularly in a business, production, or process control environment, is to examine the various *forms*, both input and output that are used by the client. For example, a form in a print shop might reflect press number, paper roll size, humidity, ink temperature, paper tension, and so on. A form used in a test facility might include a graphical print out for each component. This information exists for a reason and will help you to construct questions about its origin, purpose and importance to the system

Other documents, such as operating procedures and job descriptions, can also be powerful tools for finding out exactly how and what is done. Comprehensive information regarding how the client currently does business, can be extraordinarily helpful in determining the client's needs.

Another method of obtaining such information is to set up *video cameras* within the workplace to record exactly what is being done.

**Technique 4 - Facilities, Application, Specification Techniques or FAST**

This technique encourages the customer and developer to work as a team to promote a rapid understanding of the system and its behaviour, the problems it poses and to propose solutions to them. The attraction of this scheme is that it produces information, which is of assistance in modelling the software, and will thus be useful to the designer/programmer later

- **Stage 1**- The customer and analyst develop a "*product request*" which is essentially a brief description of the system and what it will do in fairly loose terms.

- **Stage 2**- Before a meeting is held, each member of the team, which includes the customer, is asked to examine *the "product request*" to create a list of ***objects*** (nouns) that are evident in the proposal. For example, consider the following extract:

> *"The user enters their data via a keyboard and the invoice appears on the printer".*

The objects thus identify the things the system will *interact* with i.e. the external devices or data in the system. Next, a list of ***operations*** is identified (verbs). For example

> *"The system processes the data, prior to formatting and displaying on the printer".*

In essence these things identify *operations* the software must perform and act on the object identified previously. A similar process occurs in an attempt to identify any ***constraints*** imposed on the system for example:

> *"The system only accepts 5, 10 and 25 cent coins".*

Lastly, ***performance criteria*** are assessed, For example:

*"The system must generate a response within 20mS".*

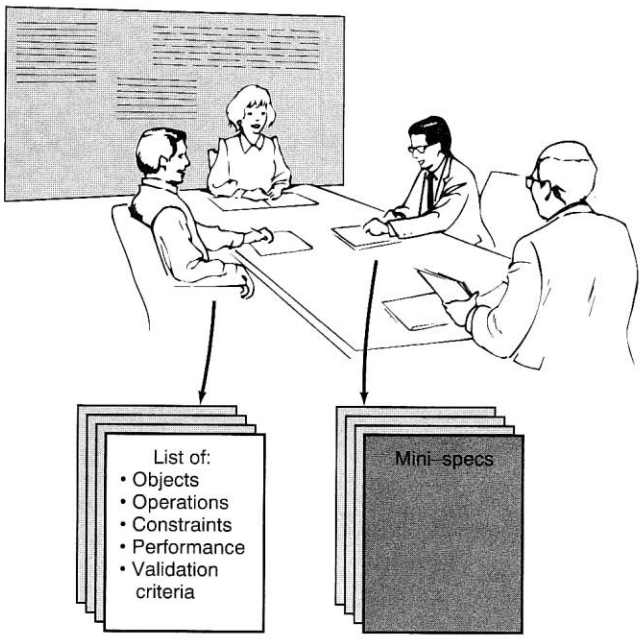
- **Stage 3** - The members of the team (users, customers and developers) then meet and sit around a table and each in turn presents their findings. Common elements of each list are grouped together and refined into more detail in an attempt to elaborate the description of the item.

  This process also uncovers new objects, new behaviour, new constraints and new performance criteria, or highlights ambiguities and/or deficiencies in existing elements. The discussion may raise issues that may not be answerable at that meeting, thus an *issues list* is kept to record all unresolved details or questions for future meetings.

- **Stage 4** - Each member of the team then produces validation criteria against which the finished product can be assessed. That is data and results that can be used to check that the system performs according to the specification.

- **Stage 5** - Finally, someone from the team is given the job of drafting a mini-specification taking on board all aspects of the meeting.

The outcome of a FAST meeting is a **mini-specification** document as shown below**.**




**Technique 5 - Design a Prototype.**

A powerful technique for uncovering detail from the customer is to explore the concept of a *prototype*. Here the analyst presents a prototype model for the customer to evaluate. This could be something as simple as a *paper and pencil* model of the system showing how the user interacts with it. It could include a *sequence* of screen shots showing the appearance of the system performing critical operations, and perhaps the users *input* and the system *responses* to it.

At a more detailed level, it could include a *part-working prototype* showing how the system might behave (although you should be careful to ensure that the customer realises that this is not the finished product). If the system is an enhanced version of a previous one, the prototype might then consist simply of a discussion of the enhancements/changes to the product.

Prototyping 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 required. A "*quick design*" then occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the user. For example, inputs, outputs, operation and behaviour. Performance is not usually assessed at this stage.

The quick design leads to the construction of a prototype. The prototype is evaluated by the customer/user and is used to refine requirements for the software to be developed. A process of iteration occurs

**Reasons for Prototyping**

- Customers often understand (or think they understand) the problem domain, that is, what they want to achieve with the system, but often have little appreciation of what the finished product will look like or how it will behave. Creating a prototype *educates* the customer and gives them an indication of what they can expect from the finished system.

- Customers *hate surprises* and frequently reject software based on how they see the software working and how effectively it solves their problems, not on how clever the code or the designers have been in creating it. Again, creating a prototype educates the customer and gives them an indication of what they can expect from the finished system.

- Seeing a prototype often serves to enhance an understanding of the problem domain and to uncover anomalies and ambiguities in its behaviour/operation. The earlier this is detected, the better for everyone.

**Prototyping Techniques**

***Step 1****- Evaluate the suitability of the product for prototyping.*

Not all software is suitable. Those that are best are usually those that interact heavily with an operator, presenting/displaying data graphically and/or those that involve a large amount of combinatorial processing.

The usefulness of a prototype must be weighed against the cost, time and effort required producing it, which after all is only a prototype, not the basis of the final product. For example if the simulation/prototype requires 50,000+ lines of code it is unlikely to be suitable.

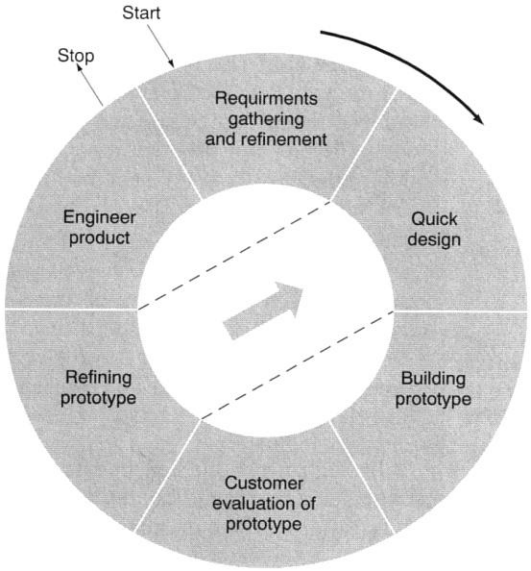
***Step 2*** - *Develop a simplified representation of the system in terms of its requirements and specification.*

Before the prototype can be built, a simplified model of the software must be developed so that certain aspects of it can be modelled without getting involved in the complexities of otherwise hidden detail. Prototypes for the other components can follow later.

For example, the development of a prototype car would only attempt to model the aspects of the car the user interacts with. Essential but otherwise hidden details such as the engine gearbox etc. can be hidden or simplified..

***Step 3*** – *Develop the prototype and present it to the customer.*

The customer will examine the design and propose modifications and or enhancements to functionality, operation, behaviour etc. which will then lead to the design being refined. *Step 3* is then repeated iteratively until the customer is happy with the design. The sequence of events for the prototyping technique is illustrated below.




The introduction of 4th Generation programming tools can help here to provide a rapid user interface for the user to view and see how the system looks e.g. **Visual basic** and **Visual C++** are both rapid development tools and quickly generate a user interface.

**Step 4 - Throw away the prototype**

The results of prototyping can be very important to the final design of the software. In fact, if the prototype is accurate enough in modelling the behaviour and functionality of the required system, it may only be necessary to say to the designer, "*design a system that work like this*".

However, it is important to appreciate that if the prototype was in the form of software it has to be **"thrown away",** as it was probably only held together by luck and flaky code in the first place.

It is vital that we do not lead the customer (or the developer) to believe that with a bit of effort, the prototype can be coerced into becoming the finished product. This would raise false expectations about delivery dates of the finished product and might ask them to question how you can justify charging so much for a developing the finished product when with just a few weeks of effort the product looks like it might already exist.