ECE 298
System Design
Course Notes

Phil Anderson

Rev 2006

Table of Contents

Chapter 1 Introduction

Lesson 1

Any design, any process, any procedure depends on more than you. If you have an adversarial relationship with your team, or your boss, or your employees, with your suppliers or with your clients you will not achieve the success you could. If you can draw on the support of people outside work, it makes creative thought that much easier. The fundamental basis of all these relationships is respect for these people's parts in your work and your life.

I have been blessed through the years with many wonderful working associates and friends, and a marvellous family. One woman, my wife Chris, has given incredible support and encouragement, the benefit of her considered, gentle opinions and of her wisdom, two wonderful children and a life to be envied, while including a career of her own. It is to her this book is dedicated.

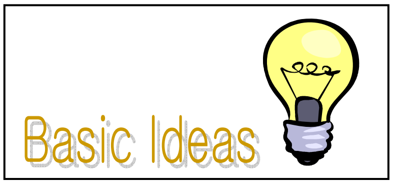
1.1 Overview

IEEE 1220: Systems Engineering

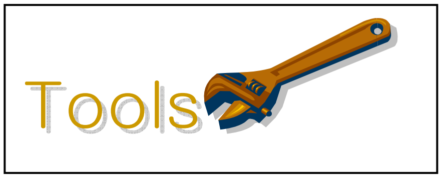
An interdisciplinary collaborative approach to derive, evolve, and verify a life cycle balanced system solution that satisfies customer expectations and meets public acceptability [2].

1.2 The Layout of the Book

Chapters are divided into areas that will be flagged with pictorials as follows:

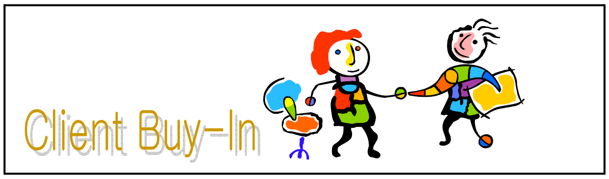


The text with this pictorial explains why the chapter is there – the aims, theories and practices used to achieve the end product of the chapter. The chapters follow the rough flow of a project from conception to delivery.



Often the end product of the chapter can be reached with intuitive work, but sometimes tools will help set a proper course and trigger ideas and thoroughness. A collection of tools to this end are flagged with the pictorial at the left.

Sometimes we get very involved in the details and miss things that are happening at a more global level. We need to know where the work, the current step and the project itself fit with the rest of the affected systems.



A client pays for the work and is the ultimate judge of whether the project was a success or failure. Dealing with the client constantly through the design and development process is important.

Summaries and applicability information are in boxes like this.

You will also see some of these boxes around

These are extra tips, hints and comments that would otherwise disrupt the flow of the thoughts in the main text body.

1.3 Some Basic Definitions

Artifact – This word is typically associated with archaeology (digging in ancient ruins looking for pot shards) but here we will use this word's more general meaning: something human-created. We will use it for all that computer and electrical engineers might produce – software programs, circuits, products, processes, etc. So a production procedure is an artifact, a toaster is an artifact, a sorting algorithm is an artifact.
We will also use product, design, system and other words to represent what it is that we are producing in our design project.

Stakeholder – A stakeholder is anyone with a "stake" in what is going on, anyone that is affected by the process and the result. It can be a very significant link, such as a patient may have to an anaesthetic machine during an operation, or a very minor link, such as an employee of an energy company might have to the anaesthetic machine because of its electricity use. We will expand on this in a later section.

User / Client – A specific stakeholder. The contact, the people paying the bills, the people employing the artifact directly. User and client will be used interchangeably.

1.4 About this course

An engineer works with technology, creating new processes and products in this area. To do so requires a knowledge of the technologies in use – teaching the theories and practicing the use of these technologies is the aim of most courses in an engineering curriculum. If you are a mechanical engineer, you are expected to know about flows and levers. A civil engineer should know about building coverings and bridge structures. As a computer engineer, you should know how to program in an object-oriented language and have knowledge of data structures and algorithms. As an electrical engineer, you should know about filters and how to use a circuit analysis program. In other kinds of engineering you would be expected to have other specific technical strengths.

However, as a practicing engineer, you will need additional, non-technical skills. All engineers require a combination of technical, communication and system engineering skills. Engineers with severely limited communication skills will stay at a low career level, unable to convince colleagues, management or clients of the worth of their opinions, and totally unable to handle the demands of management. Engineers with no system engineering skills will be kept from more interesting, creative work as they will not be trusted to make decisions in a wider context.

Engineering is about making decisions and evaluations in new circumstances. In the field you are not given the equation and told to shove numbers into it. You have to find the equation, sometimes you have to evolve the equation, you have to determine if the equation is appropriate and whether the results will be relevant. You have to question the numbers you are given and find the numbers you aren't given.

This course is deliberately open-ended, because we are exploring these other parts of being an engineer. We will not give you all the answers. You may have to deal with devices you know little about, with situations you have never encountered before and which are not completely described in the writeups, and with decisions that are not covered in class.

This is very much like the real world. People hiring want to see self-starters. They want their engineers to find the answers themselves, to see the whole picture and to work within it, to ask for help when they need it but not to expect to be told of every i to dot and t to cross. They want their engineers to be able to learn new things, create new things, do new things. They want their engineers to be able to communicate efficiently with other engineers on the team, with other people in the company and with the clients.

Let's look at communication for a moment.

But let's first think about calculus. You learn how to solve a differential equation, and then you get a problem set or an exam where a question is posed. You give the correct answer and get full marks for it. *If you cannot communicate the answer to the marker, you do not get the marks.* If you miscommunicate, that is if you put the answer down in the wrong form, or in a way that is imprecise, you will not get full marks. You must not only come up with the right answer, you must communicate the answer in an unambiguous, complete, expected way to the examiner.

And in the same way, system design is closely tied to communication. As an engineer you can expect to spend almost half your time in activities involving technical communication.

Example	**Form**	**From**	**To**	**Reason**
Interconnection Pinout Table	Table or computer file	Engineer of one device	Engineer of connecting device	Making sure that both devices will connect together
Progress Report	Report	Detail Engineer	Supervisor	Tracking project, solving problems early, client information
Request for Proposal	Document	Client	Development company or group	Gives client requirements for artifact. A Requirement Spec
Proposal	Document	Development company or group	Client	Defines how the group proposes to build the artifact – a System Spec
Maintenance Manual	Document	Development company or group	Those overseeing the artifact in use or when in need of repair / maintenance	Structure information, what to check, test procedures etc.
User's Manual	Document	Development company or group	User	How to operate / conduct the artifact. Cautions. First level of problem analysis.
Budget	Report	Development management	Purchaser or Executive levels	Go/Nogo decisions, tracking, cash flow determinations
Presentation of Proposal	Oral	Development team	Go/Nogo decision makers	Attempt to persuade these people to make a 'Go' decision
Memo about problem	Email	Engineer	Another	Transfer of information
Resume	Document	Applicant	Potential Employer	Getting an interview en route to getting employed

Table 1-1 Examples of Engineering Communication

Table 1-1 shows some of the typical communication activities. Some communications involve the technological details – software code and comments, schematic diagrams for example. Others are concerned with project management – progress reports and resumes. No engineer can avoid them; good engineers will be good at them.

In design the communications are done in English (at least for this course). These communications should be unambiguous, complete and in a form within the expectations of the other party, just like the answers to a calculus question.

In this course we will learn the basics of how design engineers communicate.

Along with practicing open-ended learning and decision making and along with learning about communication, we will also look at the basic design process, the formalism and organization required to do it right and some tools that will help give us the answers we need to make our designs appropriate, timely, cost-effective and solution-oriented.

1.5 Overall Themes

In various ways you will see the following themes throughout this book:

Creativity and Open-Ended Decision Making

Engineers design processes and products that did not previously exist. You cannot do this completely on a worn road – there must be a unique combination of technologies or new uses of technologies, perhaps from beyond your discipline and perhaps beyond engineering. Without this creativity, engineering could be done with just a photocopier.

System Awareness

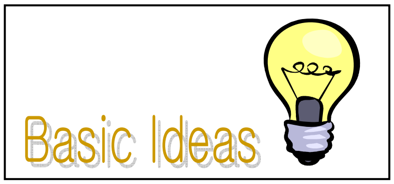
You must understand how the artifact you are creating will be used, why it will be used and what its effects will be on other people and systems.

Care and Analysis Early On

Good design is about avoiding mistakes and about finding mistakes at the earliest possible opportunity. The longer a mistake runs undetected, the greater the ultimate cost.

1.6 The Rest of this Text

In the next chapter a basic design flow will be introduced. We will follow this design flow in the rest of the book, looking at the tools and the strategies that work at each step.



Chapter 2 Start to Finish

Most design texts would have you believe that you design in a very regulated manner. First you establish the goal. Then you determine the requirements. Then you create a system design. And so on.

The truth is that engineers will start in the middle, the end, jumping backwards and forwards in their excitement about a new project and new problems. Designs are started on the backs of the paper placemats in diners, or on the backs of napkins, envelopes, whatever is available.

“Green” engineers believe that since this loosely organized method worked for their lab preparations in school and since this is how the ideas are initially developed, that more formalism isn’t required. Managers, on the other hand, latch onto the formalism. They force engineers away from the creative processes and into a strictly regimented set of steps with (usually) a very limited timeframe and an expectation of perfection as each step is completed. Real engineering is a blend of these activities, each supporting the other.

Answer this question: What is a long project?

Go back and consider the questions before going further.

Typically, a student considers a project that goes for a couple of 3 hour lab sessions to be “long”. They often see a very long project as taking the better part of a full course over a whole term. This view is quite reasonable under the circumstances.

In industry a short project could involve a half-dozen people over six months. A long project will probably take at least a couple of years, maybe even a decade, and could involve thousands of people. Some projects have 50 year schedules, some even longer.

The kind of planning you do for a lab or a project stretching over a couple of lab periods is trivial compared to the planning required for a larger project worth millions of dollars to a company. In the latter, you cannot afford to make even small mistakes that are not quickly and inexpensively corrected. A small mistake could have a large price tag if the result is mission failure or production recalls or people dying.

> *For example, you are aware from your software work that there are various sizes of variables that can be used in a computer. You should recognize at least a few of: short, long, integer, double, byte and so on. Each of these variable sizes has a certain capacity and thus will hold a certain range of numbers.*
>
> *After seven billion dollars of development work, a rocket and payload worth a half billion dollars was launched by the European Space Agency in 1996. The flight lasted under a minute: A large value held in a double-size variable was moved to a variable a quarter the size where it did not fit. This value, a horizontal velocity, was used to guide the rocket and, since it was not right, the guidance was misguided and the rocket had to be destroyed.[3] A small error, undetected and uncorrected, with a very expensive consequence.*

Our Starting Point

We can start with a very simple model of what we are doing in design, as shown in Figure 2-1. This is, of course, overly simplistic. However, it does convey the fundamental requirement of a design: that it deliver a solution to a need. It is the basis of the development model that follows.

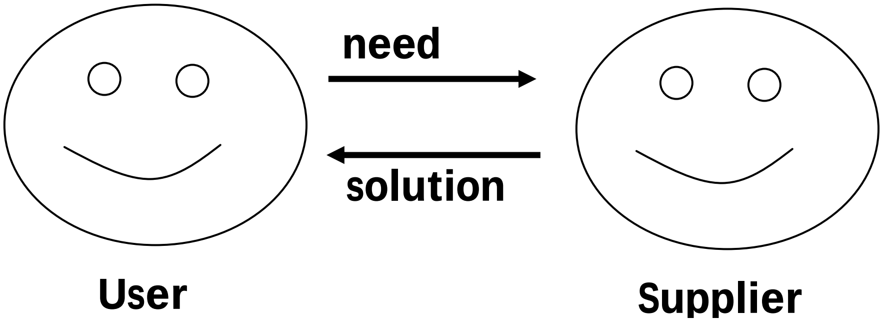
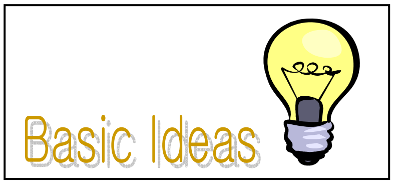


Figure 2-1 Simple Design Model



2.1 The Development Model

We will use the development model shown in Figure 2-2 following. It is flat, whereas design and development are not, and it is flawed, but it is simple and it is a reasonable starting point for more elaborate, more perfect models. Since it is a model, it is a tool for our development process.

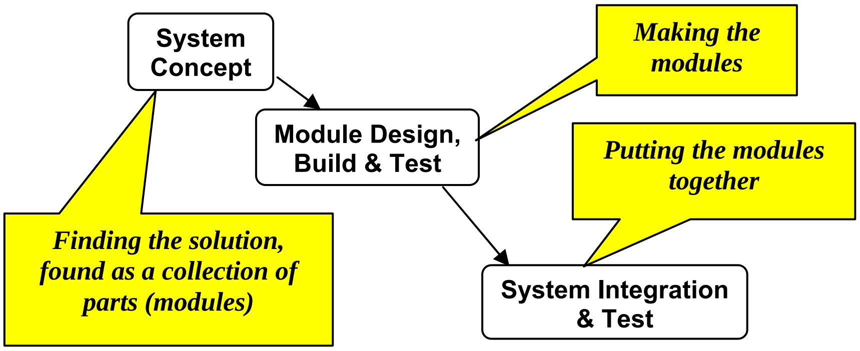

System Concept
Making the modules
Module Design, Build & Test
Putting the modules together
Finding the solution, found as a collection of parts (modules)
System Integration & Test

Figure 2-2 The Waterfall Model of Development

We have three stages to the model:

1. System Concept. Here we decide what the artifact (the product or process) must do in order to fulfill the aims of the user or client. The steps here involve determining what it is that the client wants and needs, the details of the requirements of an artifact to do this job and how we are going to confirm that the artifact in fact does these chores. Because the other two steps depend so much on this one, it is important that it be done right. This is where we will spend most of the time in this course and in these notes.

2. Module Design Build & Test. We build the artifact up in parts. Each of these we will design and test independently.

3. System Integration and Test. Here we bring the parts together and confirm that the artifact performs as desired.

Summary: The Development Model is introduced that will be used throughout most of this text.

Each one of the stages in this development model will be broken down into smaller steps. We will also discuss appropriate documentation that is generated in each of the steps.

This model is known as the Waterfall Model of development. You will see it embedded in stronger, more technology-appropriate models, particularly if you are interested in large software system development. Like all models it is to be used where it "works" (that is where it provides useful information) and discarded or augmented when this is appropriate.

2.2 Systems of systems

The need of the user is always a need that fits into a larger system context. A pencil is needed to allow the completion of an exam which is needed to complete a course which is needed to complete a degree which is needed as part of a life plan. A student's life therefore depends on a pencil (or some equivalent substitute solution to the primary need). A supplier of the solutions will know about the user's larger

Summary: Any system we produce is a part of another system and may be composed of systems as well.

system context and will order extra pencils in at exam times to satisfy the user's needs at that time.

The supplier of a Global Positioning System (GPS) for an aircraft should know about aircraft. The aircraft supplier should know about airlines. The airlines should know about transportation needs on a large scale. Systems within systems.

Decomposing a large system you usually encounter multiple subsystems. An airline requires a passenger ticketing system, an aircraft maintenance system, a scheduling system, an employee information system, and so on. The model, which ignores the fact that one system can be part of multiple systems, is shown in Figure 2-3.

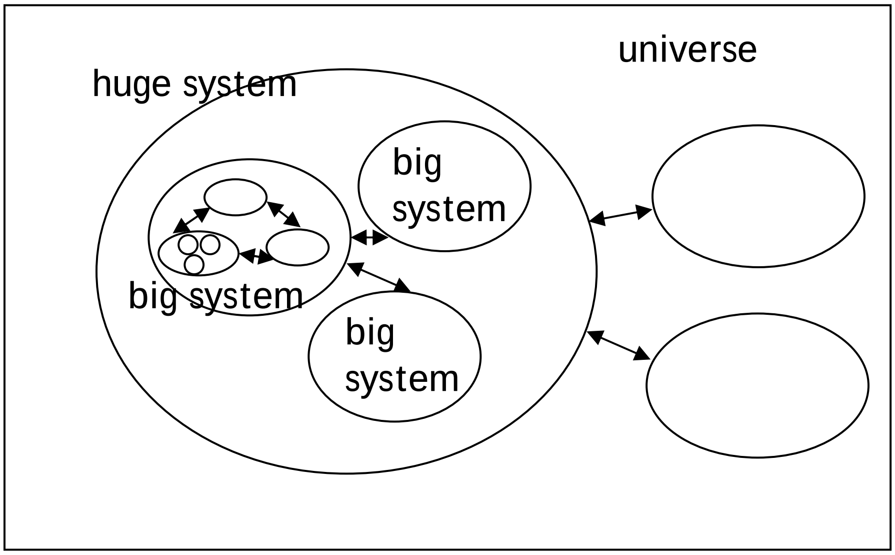


Figure 2-3 Systems within Systems

Do we need to understand completely every system that could be involved with our project?

No. A properly-designed larger system, say a **huge** system in Figure 2-3, will specify the **big** systems as to functionality: Big system A will do certain functions, big system B will do some other functions, and so on until the full requirements of the huge system are met by the collection of big systems. In implementing a **big** system, a further collection of smaller systems will be specified, and each one of the smaller system will perform functions that, collectively, perform the functions required of the **big** system.

At any level we should know a great deal about the system we are designing, enough about the system of which our system is a part to understand its place in that system, and enough about the systems that will make up our system to be able to reasonably allocate functions to these systems.

The further down this hierarchy we go the less we need to know about the largest systems. Tiny systems should need only general knowledge of the huge system. When a system design does not have this reduced knowledge requirement of much larger systems of which it is a part, or of the much smaller systems of which it is composed, is not a good system design. Keep this in mind as you create your own designs!

Consider a bus for local transit. In designing a bus we do not need to know the design details of the seating manufacture, only the operational characteristics of the resulting seat. Similarly, we do not need to know exacting details of every bus route, only how the circumstances of the routes the bus may take will reflect on how the bus might be designed.

2.3 Going in the right direction

Why we are doing all this work? With larger designs we need to be able to make sure at all times we are not straying far from our true target. If we aim for the wrong target or go a significant distance along a wrong path, we waste time and resources.

Summary: Early determination of correct direction will save time and resources.

Imagine a trek in the north woods. Our hiker in Figure 2-4 has several forks in the path. With a map and compass and with planning, the hiker is less likely to end up on the wrong route. Notice also in the diagram how an early mistake can lead to far worse consequences than a mistake made later.

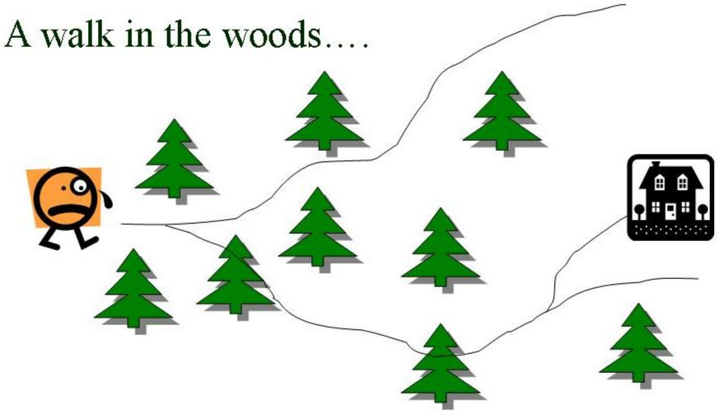
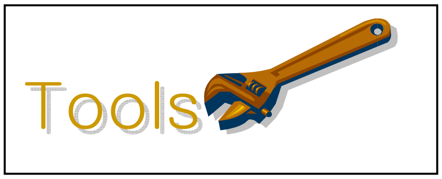


Figure 2-4 Consequences of Decision Making

Similarly, with design decisions, if we can increase the likelihood that we are following the best route, if we can increase the likelihood that we are producing what is wanted by the client, our designs are more likely to be successful. Furthermore, we will make our time and resource budgets more accurate and reduce our chances of a significant and costly surprise.



2.4 Mind Maps

One interesting tool that you should get some experience with is the Mind Map. While these can be done effectively on paper, they are extremely useful when done electronically where additions, rearrangements and copies are easily done. Happily, there is a freeware version you can download at http://freemind.sourceforge.net that works well and has many good features. Programs with more features and connectivity to other programs are available for purchase – one that I use is MindManager at www.mindjet.com.

What Mind Mapping does is link ideas together in a hierarchy. The central idea or main requirement is in the middle; branches out from the middle give the breakdown, branches out from these give further breakdown.

Summary: A mind map is a graphical tool to record interconnected information quickly.

Applicability: Generating Requirements, Generating possible System Designs, Reverse Engineering, organizing communications, organizing projects

Since these can be done on the fly they can be used to store brainstormed ideas. They can store lists and structures. They are useful for any sort of design work, including the design of a document, since they can hold pieces that can be rearranged and altered as your thoughts progress.

Figure 2-5 is a partial Mind Map of a toaster.

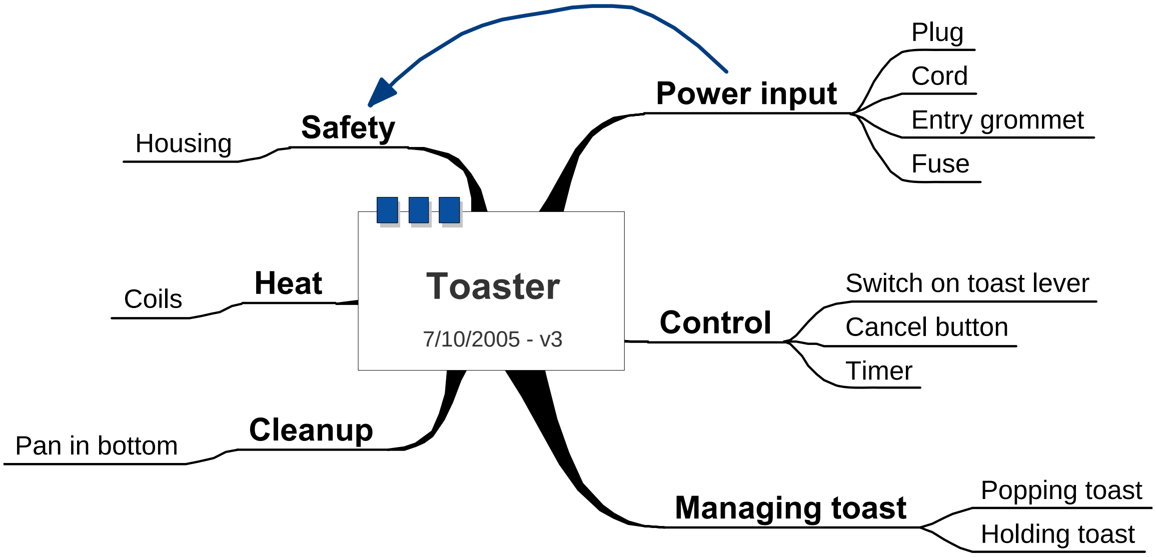


Figure 2-5 Mind Map of a Toaster

In the figure we are considering the parts of a toaster and dividing them by function. The functions “Power input” and “Safety” are linked, since the power input is at a dangerous voltage level and so safety is a factor. The function “Control” has a number of associated parts that implement the control of the toasting process – the timer which will cause the end of the cycle after a certain amount of time, a cancel button used by the user to stop the toaster before the end of the cycle, and the switch on the toast lever that indicates the start of the cycle.

2.5 Models

Engineers use models. You have already taken movement equations based on Newtonian mechanics (F=ma, v=d/t, and so on). You have probably taken some Chemistry and learned about atomic structures. You have learned calculus and how this can be applied to certain rate-of-change problems. At this point you are learning about logic gates.

All these are **models**.

A logic gate is a model that describes the statistically predominate behaviour of purified sand with doping under the influence of differential electronic charges. Or, in summary, a logic gate is something that makes thinking about circuitry easier for us.

Summary: Models are used throughout engineering. Realizing that you are using a model and being able to adjust that model and not be misled by that model are important to engineering design work.

Application: All stages of design work.

F=ma (force equals mass times acceleration) is a model (or a theory) of how force, mass and acceleration interact. It has shown to be incorrect, but for almost every circumstance where we might want to know this interaction, it works well enough.

This is the key: Working well enough.

Working well enough involves two criteria:

1. The model must give us some results
2. We have to know the model well enough to know how much we can trust the results, that is how applicable the results are to our present situation.

Graphically we can consider it as in

Figure 2-6. Models have areas where they work well, areas where they might give us bad information, and areas where we can be sure they are giving bad information. Most important is that we know where our model will deceive us.

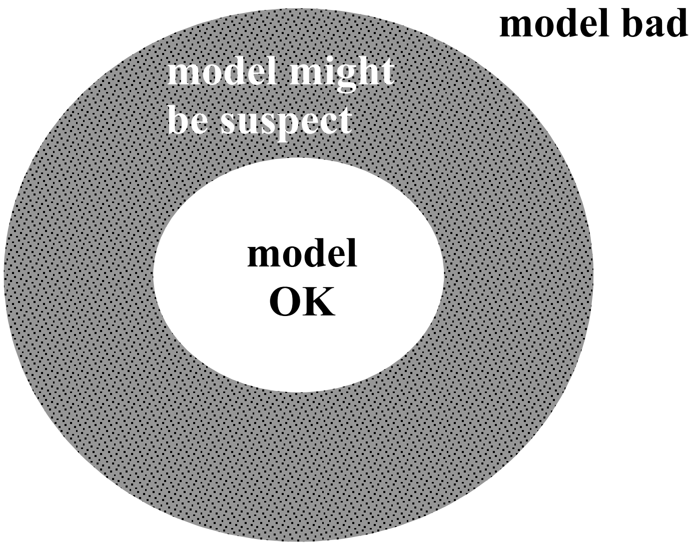


Figure 2-6 Models – Where they are Useful

Consider a logic gate. This is a model of what (say) silicon does. If we are using this model when designing a logic circuit, to be operated under conditions specified by the manufacturer, then the model is OK – it will (usually) give us the results we expect and so we can use the model to design a complex digital circuit.

However, if we supply no power to the power pin of the logic implementing the gate, this model is no longer valid. The silicon will perform some sort of function, but not the one predicted by the model. If we apply 120volts AC from the mains to our circuit, the circuit will no longer work as the model predicts (see Figure 2-7).

Figure 2-7

Digital Gate with 120VAC applied

If we have a very simplistic model, the “OK” area of Figure 2-6 will be small. A more complex model with finer refinements will give us a larger OK area.

Does this make a simplistic model useless? Not by a long shot. Engineers use these all the time, as do you.

Consider the problem of getting to your first class on time. If the class starts at 9:10 and it is 8:55 and you are in Scarborough, a great distance away, you immediately conclude that you are not going to make it on time. You use a simple model based on your travel capabilities. You do not say: “It will take me 5 minutes to get to the car and 1 minute to get the car started then 12 minutes to the highway and ….”. You do not need this complex model to tell you that you are going to be late.

Do we ever not use models? Only the actual produced items are not models! Every time we use an equation, build a prototype, construct a Use Case, or any other engineering activity, we are using models.

If, instead, you have 50 minutes to get to class, then you might refine your model to see if this is enough. If it is close, you might get on the internet and check out the traffic to see if a traffic block will make you later, or if you will be faster than normal due to a light traffic flow. Each iteration of the model will help you to determine whether it is likely that you get to class on time.

However, it is still a model. If you set out, the car may not start or could stop on the way, there could be an accident to delay you, the class might have been cancelled so you can never make it on time. If you don't set out, you can never be sure that you might not make it. At a very very small probability, the university might pass through a space warp and be frozen in time long enough for you to make it even though you had only a few minutes for a trip normally taking much longer. The model, whichever one you select, predicts a certain outcome based on its inherent assumptions of travel times, modes of travel, delays along the way and the likelihood of a suitable space warp.

When we start a design, we produce models, mostly in our heads at first. We take a cursory look at the time constraints, the budget constraints, the available talent and facilities and we quickly decide to continue to analyse the project, or to give up. If we continue, we continue to refine our project model until the design is done. Implementation will prove whether our model is, in fact, correct.

For example, you are asked to design a computer program to predict the performance of 10,000 stocks individually, and to have it ready by the end of the day. Without even knowing the data format, or the computer language or platform of implementation, or which stocks they are, you will probably dismiss the problem as undoable because of the tight timeframe.

If the time restriction was removed, but the remuneration was $100, you would (likely) reject the project as not worth your efforts.

If you felt that the project was possible and rewarding, you would move forward and begin to get the information you needed to understand the problem better and further refine your model. The complexity of the model you use will depend on the complexity of the problem. However, generally models are under-complex and the project will have problems that could have been anticipated early with proper models.

Models help us to reduce the loading on ourselves by handling some of the interactions and otherwise making things simpler to comprehend for us and our clients. Models predict behaviour which we can evaluate against what we would expect or desire, allowing us to evaluate a proposed design solution and to adjust parameters or requirements. Models help us to manage the project because they help us to understand the size the and complexity of the project elements.

2.6 Development Tools

Most of what we do now as electrical or software engineers involves the use of development tools and equipment. We look at signals on a digital oscilloscope, we design printed circuit boards using a routing program, we model filters using Matlab®, we write programs in C++ and compile them using someone's compiler.

We should consider these as situations involving models. A digital oscilloscope samples a signal at various points and displays it on a screen. Is this actually the signal that was there? No! First, we are sampling it at certain intervals. Events could happen faster than we are sampling and we would not see them. Secondly, how we sample makes a difference. Is a sample an average of the signal over the interval? Is it a sample at a specific moment in time? And even if we know this, is it an average power or an average voltage? How much actual time does it take to sample, even if 'at a specific moment'? What sort of intended and by-construct filters are altering the input before we even see it? What are the effects of the probe on the circuit? What we see on our oscilloscope screen is a model of what is actually happening.

Even in software we write a program that we compile and run. Our compiler is a tool – do we know exactly what it does, or does it filter and alter our input to give us something that will, under most circumstances, do what we want (assuming debugging and testing has been completed) but may generate some side effects we had not intended. One of the major ones here is data conversion. For example, consider:

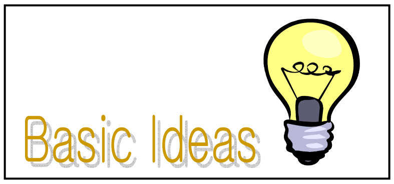
```
int A = 10;

int B = A * 10.6;
```

In this type of situation some compilers will cause the 10.6 to be converted to int before multiplication, and set B to 100. Others would convert the value from A to float, multiply, then convert back to int with the result that B would be set to 106. Either way, if you are expecting one instead of the other you might get a nasty surprise.

Warning: It is important that we realize that our development tools can lie to us and mislead us. Verilog®, a high level hardware descriptive language, allows us to design complex circuits easily. Most printed circuit board development programs have built-in verification tools. SPICE will allow us to design analog circuits quickly and determine their characteristics. But Verilog® can easily create monstrous circuits that were not what we intended because the language has masked what is actually happening in the design. A circuit board verification program can identify certain problems, but will not recognize that a very sensitive signal has been routed a great distance through some other very noisy circuits. SPICE may give us marginal circuits that work mathematically but not practically. Engineers experienced with these tools will realize the problems can occur and will make sure they don’t happen.

Chapter 3 The First Steps



3.1 First Stage Refinement

To be useful, each of the three stages of our design model must be further broken down to a number of steps. The upper right of Figure 3-1 shows the three stages and the lower left shows the steps that comprise the System Concept stage. The recording of the information generated at each step is designated in white, the work to create the information is shown in green boxes.

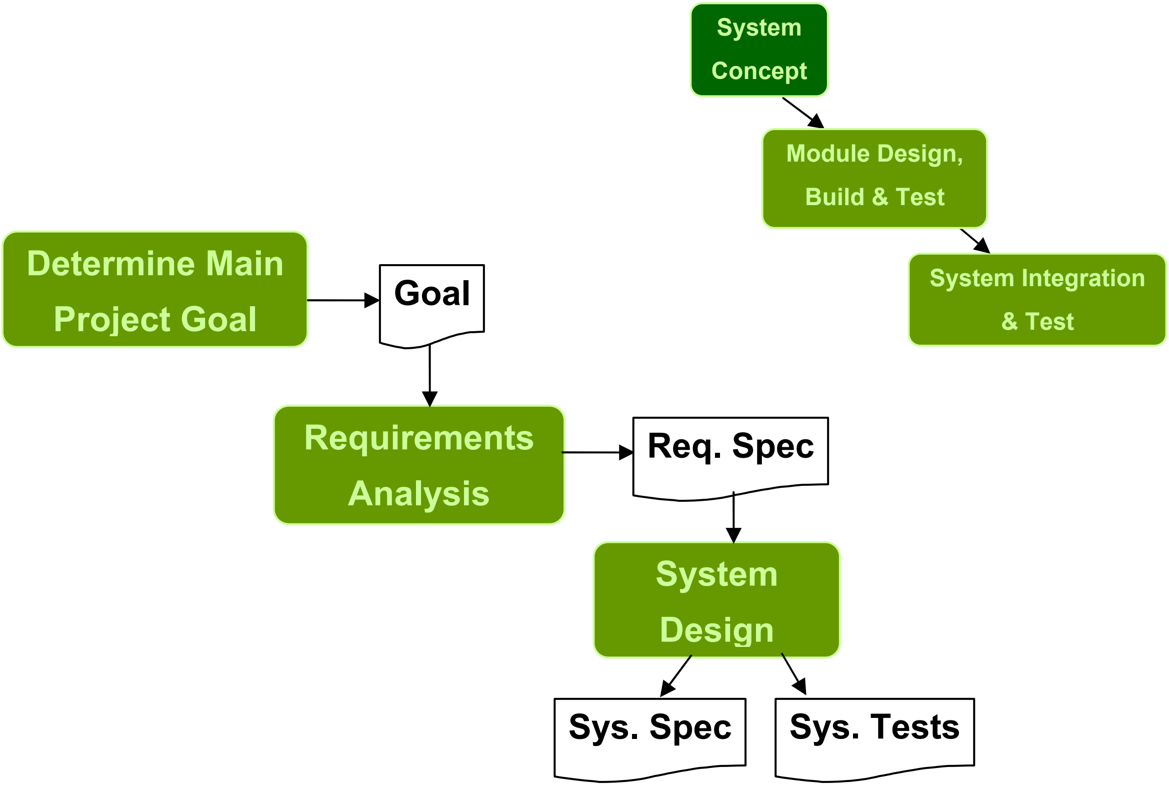


Figure 3-1 Refinement of the System Concept Stage of Development Process

Most of the planning is done in the steps shown here. Determining the goal starts the process off. The goal is the big-picture idea of what the project is about.

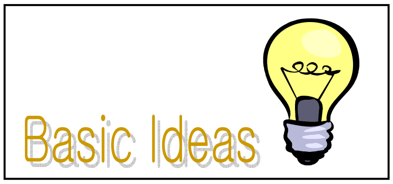
After the goal comes the Requirements Analysis, resulting in the Requirements Specification. This step is where we decide how the system is going to have to perform to meet the goal within the environment in which it will be used. We are not yet saying how the system will do this, only what it has to do. By meeting these requirements our design should embody all it needs of the system of which it is a part.

At the System Design step we actually select a design. There could be many choices as to how to meet the requirements and we want to weigh each carefully before selecting one. When we do, we will have selected a set of pieces (or modules) to work together in a certain way. This information we record in a system specification and we also create system tests that will verify that the system meets the requirements at this time as well.

In this chapter we will first look at the goal, as the goal launches the design project. Most of the rest of the text concerns establishing the requirements, so we know what the design has to do, and selecting an implementation, which is the design we propose that meets the requirements and the goal.

The next chapters are as follows:

Chapter 4	Determining the Context	The coming two chapters deal with the requirements. First we look at where the requirements come from, largely a context issue, and then we look at how to systematically put them together.
Chapter 5	Finishing the Requirements	
Chapter 6	Introduction to System Design	These next six chapters deal with the system design, The process is an iterative three-step process. Each step is covered in a chapter. The following chapter introduces Project Management, which is also done at this stage, and contributes to the selection of a design. The last of this section of chapters considers Risk, which is the ultimate determining factor of which design is chosen.
Chapter 7	Researching the Design	
Chapter 8	Determining the Design	
Chapter 9	Evaluating the Design	
Chapter 10	Managing the Design	
Chapter 11	Risk	
Chapter 12	Detail and Execution	Two chapters dedicated to the detail work on the design and making sure it does what it is supposed to do
Chapter 13	Testing	
Chapter 14	Breaking the Design Model	More realistic design models that are variations of the original
Chapter 15	Wrapping it Up	Concluding remarks on the process



3.2 General Structure-Goal

Some computer programs, like Google, are truly amazing. They take a collection of billions of pieces of information on billions of topics and usually find what you are looking for within a few tenths of a second or less. To do so, you must provide a focus, expressed in a word or phrase, that narrows the search.

Human brains are not that quick, but unlike Google, they can solve new problems and create new products and services unlike those that presently exist. However, they must start with a general focus as well.

Summary: The goal is the purpose of the design project. It is general, but gives direction to the project.

In design, the purpose of the design can usually be expressed in a short paragraph, or often just a sentence. This is called the **goal**.

The goal expresses the reason behind making the effort to do the design. It is not overly restrictive, it does not usually have numbers or limits and it does not indicate how the design will be implemented. It does, however, indicate how the success of the project will be measured.

This last is an important distinction, one that we will explore further when we talk about testing, verification and validation. In the requirement specifications we will go into all the various measures that will be made to make sure the design meets the goal (how fast, how high, how much power, and so on). But here the measure of success is whether the design works as intended by the larger system. For each goal we should be able to name the larger system and how we would measure success.

Good Examples of Goals

1. The group is to design a mid-sized car to target the low-purchase-price end of the buying public.

Larger system: Car manufacturer, potentially with multiple lines of cars.

Measure of success: Car, as described, able to be manufactured and sold at a profit.

2. I need to make it to my first lecture.

Larger system: Education of me, involving learning from the professors.

Measure of success: Arriving at the lecture before the professor begins to speak.

3. A designated engineer will produce the quality assurance program for the new production line.

Larger system: Manufacturing plant and, more specifically, the total production process of the products that come off this specific production line.

Measure of success: A quality assurance program that helps reduce the product failures to what is seen as an acceptable level.

Note that there are always some underlying assumptions in a goal statement and that the wording is usually not deeply technical. In the first example, it is assumed that we know what the buying public is, what a low cost car is, what a mid-sized car is, and so on. The specific details of these will be given in the requirement specification.

Note also that there is usually a person, team or company that is named as the creator of this design.

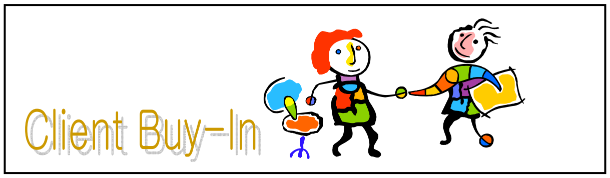
Let's look at example 2. Here the problem to be solved is to get from home to school so that the arrival time is before or at the time the lecture starts. The mode of travel is not specified, nor are any of the requirements (budgetary, safety, comfort, etc.)

In example 3 the goal is to create a "program". It is not clear what that program will have to do, other than have the products produced have a lower failure rate than they would without the program. Such details would come in the requirements specification. What is also very clear in this example is that this design is only part of a much larger system, that created to produce the products. This system, in turn, is part of a much larger system involving the entire plant. Digging down, implementing the quality control program, in its implementation, could involve the definition of other systems to do product testing, failure analysis and so on. This further refinement is where we will work through the balance of the system concept step of the design process.

Poor Examples of Goals

1. The team will build a small microwave oven with two microcontrollers in it. (Implementation details)
2. The team will create a new toy. (Too general to be useful in a formal design process. This might result in a cat toy or a baby toy – there is no clear measure of success)

3.3 Meeting the Client



At the goal stage you may first meet the client. The client can be another group in your company, either an engineering group developing a larger system, or a marketing group looking for a new product. The client may be from outside, which is typical of the engineering contracting environment. These clients may be looking for a process design, or a product design.

This is the time to develop rapport. This is the time to make the good first impression. It is also a time to listen carefully and to ask "big picture" questions that will allow you to get a good feeling for what is needed. Client relations are based on trust and respect, and this is the time to start establishing these.

3.4 Creativity

While a lot of this book is about formal methods, the key to design is creativity. This is a misunderstood term. Early programmers wore sandals and jeans instead of suits and were labelled "hippies" and "geeks". They claimed what they were doing was "art" yet the people they were doing it for considered it business. Much of this same culture difference still exists today, although there has been a blending of the groups as computers have become more ubiquitous. The difference between the extremes, however, is the understanding of creativity.

Summary: Creativity is the core of engineering design.

Painters are creative. Playwrights are creative. Music composers are creative. Painters, though, must understand their tools. They must know about paint compositions, about colours and about their canvases. Playwrights as well must know about actors, about stages and staging. Composers must create music that allows for the limitations of the musical instruments and of their players. In each case the artist uses these tools, respects these limitations and creates a new artifact to realize (or attempt to realize) a goal of the artist.

Engineers are also creative. Engineers use tools of technology to design new products, processes and procedures. So the geeks were right – they are creative artists. However, what is now better realized is that they must work towards a recognized, constrained goal that is in full agreement with the business needs of those wanting the product. This was often not the case when computing began.

If there is too much weight on the constraints and needs, creativity gets pinched. If there is too much boundless creativity, the goal is not realized. Engineers must be able to find the non-standard, better solutions, but they must be solutions.

It is not necessary to wear sandals and jeans, or pinstripes and vests. It is necessary to be able to venture into the absurd, and into other fields, for design direction and to maintain an open mind to the creative thoughts of others.

While true of early computer programming, the same type of comments as made here could be made for electronic engineers in the same period.

Often the questions are asked about what makes a computer engineer different from a computer scientist or an engineer different from a technician. Although one must be careful when painting a group with a general brush, the difference is often one of being able to harness creativity in practical multidisciplinary system design.

Chapter 4 Determining the Context

Getting to the requirements is a huge step. It is one thing to decide on a certain goal, for example that your client's company wants to add automated problem detection to their bottling line. Getting to the requirements involves answering all the questions that then arise: What problems are to be detected? What bottling line? Where can I put any equipment? How fast? How long will it run at a stretch? Who/what does any detection report to? What will then be done? How much can it cost? Who will maintain it? The questions early on seem without end, but they must all be answered before the Requirements Specification is complete.

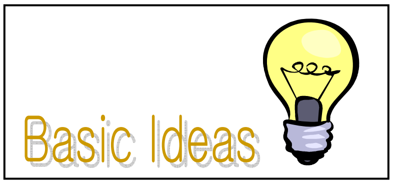
Summary: Context is introduced in preparation for generating design requirements.

Applicability: Requirements Generation

Many people and devices and systems will affect and will be affected by our artifact. This field of interactions I have chosen to call the context, but I could have called it the environment, the framework, the background, the situation or many other similar words. The number of words for the concept are a reflection of its importance. Some of these will be exact synonyms and others will have small variations in viewpoint and emphasis.

The details of the context are selected as part of the design process. If you visit the appliance section of a store and look at coffeemakers, you will find not one, but a large number to choose from. There are different sizes, different ways to add the water and the coffee grounds, different additional features such as timers and clocks, and different prices. There is no optimal design, there is only an optimal design for a given context. Some customers want a low price, others will pay to be able to set up the coffeemaker in the evening and have the coffee fresh-brewed when they first come into the kitchen in the morning. Coffeemaker manufacturers create products that are intended for a specific segment of the market, and their success depends upon a careful specification of the requirements.

The context involves the users and others affected by our artifact. It involves the existing similar artifacts already available. It involves what our company does and is and what our company is capable of doing.



4.1 Who the heck cares? Stakeholders!

Who the heck cares? Answer that and you know the stakeholders.

Stakeholders are any animate entity or group of animate entities (people, companies, organizations, interest groups, animal species,…) that are affected by the artifact. For example, if the artifact is a final functional check of a product before shipping, the stakeholders include the checker, the checker's supervisor, others in the company, the shipper, the user, the company shareholders, the government (in that the government may have regulations and certainly will get taxes on any corporate profits, which should change with fewer warranty expenses). The list is very long. Sometimes the list is very, very long.

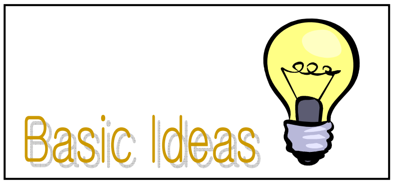
Summary: Stakeholders are those people and organizations that will be affected by the artifact, or that regulate aspects of the artifact.

Applicability: The effects of the artifact on the stakeholders should be addressed by the Design Requirements.

Because the list is long, being able to find the significance of the artifact to each stakeholder is important. For some, the effect of the artifact is minimal. For the preceding example, the company supplying the shipping containers for a product could be linked to a final functional check of the product, because the check will delay the shipping and thus delay the requirement for the shipping containers. But this functional-check-to-container-supplier link is very weak and there is probably no noticeable effect to the supply company for most changes in the functional check method.

For others, the effect of the artifact is primal. The employee doing the final functional check may have this as an only task. How the check is done, what tools are used, what time is to be taken are all fundamental to what that employee does all day.

Stakeholders are identified by a link to the artifact. In identifying these stakeholders we usually find a link, any link, and stop there. We don't worry about the details of the link, or of how many types of interactions comprise that link. In generating requirements for the design of an artifact, however, these interactions are important.



4.2 Where to Find the Information for Requirements

Unless you have already done considerable work in the field, you need to take the time to understand the area where the artifact is to be used. If the artifact is, for example, a payroll program, you will need to understand basic accounting and payroll deductions. You will need to understand the specific payroll needs of the client. You will have to understand how the governments work – what they need in terms of tax submissions and records kept.

If you are producing a sensor for a paper mill, you should be finding out about paper mills. How clean are they? What is the environment like, including temperature extremes, air quality, humidity and vibration? What will the interface be to the data recording device?

Here is a list of some of the places to look for information -

- **Books** Most significant background information has been filtered, organized and published in books
- **Online** Information is easy to find online. Unfortunately the reliability of this information is sometimes suspect.
- **Client** The client should provide you with needs and wants (which you will work with to form project functions and project objectives). You should welcome this input, but treat it as incomplete and sometimes misleading. Check other sources and get additional input until you feel confident that you are seeing the whole picture.

  The client should also be asked permission before going to others in the organization.
- **Users** The people (and the information documents of non-human entities) that interact with the artifact should be a primary source of information on how the artifact will be used and how it will fit in with existing products and procedures. Use Cases of the various stakeholders will give you a key as to whom to see. You might have to spend a reasonable amount of time with them to determine information that they consider so basic that they don't think they need to tell you, and to separate fact from misleading statements and from opinion. (A good source for more information on this is [4]).
- **Others in the field of use** Sometimes people close to the action will miss things that are more easily seen by people looking from a distance.
- **Other Engineers in your company** Likely your company has people that have already done projects similar to yours and have suffered because of the same mistakes you will make if you don't learn from their experiences.
- **People and products that interface with this artifact** The artifact must fit into a larger system. Your client will be pleased if the fit causes no problems with other parts of the system, although often this is not expressed as part of the needs and wants.
- **Patents and other IP listings** It is important that a new design not infringe on patented ideas or upon copyright. These documented ideas may also provide you with seeds of innovative design ideas of your own.

Many of the resources are human beings. For human resources, plan ahead – prepare your questions and their follow-ups; determine what information it is that you want to leave the meeting with; keep the meeting professional.

> Sometimes your best source of knowledge is on the front line. Janitors know a lot about the buildings they clean; the assemblers will know what works and what doesn't when cars are built; mill operators will know what types of steel are particularly difficult to roll. People who have been doing the work for a long time will have a wonderful historical perspective on the problems. New people will be able to describe some of the frustrations they had getting started – frustrations that might not be remembered by the "old hands".
>
> Sometimes you will get mislead by the people contracting for the artefact you are designing. Some engineers spend no time with the people that their decisions affect. This is counterproductive and a poor career decision.

4.3 Artifact Lifecycles

"To every thing there is a season". Whether you remember these as a Pete Seeger song sung by The Byrds, as part of a religious text, or as part of something some sage once told you, it is likely a familiar theme to you.

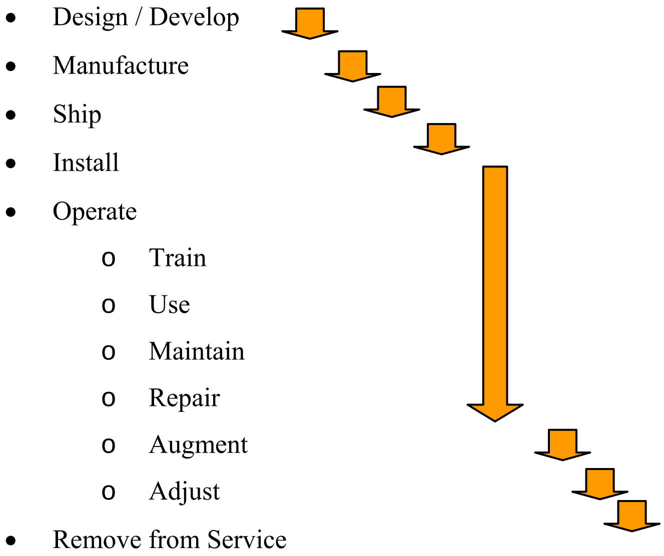
It certainly applies to engineering artifacts, excepting those that are so badly made or so badly misunderstood or so badly mistimed that they never have a season. These "in season" processes and products we create replace other "out of season" processes or products and will, in turn, be replaced or just discarded when their usefulness is done. The coming into season, being in season and getting out of season are the subject of this section.

> **Summary:** An artifact has a lifecycle – a beginning, middle and end with a number of definable stages in between. Our work as designers should take account of all of these.
>
> **Applicability:** Requirements and System Design

Engineers must consider the entire lifecycle of the artifacts they design. For example, if they design a new quality assurance (QA) plan for a plant, it is not enough to consider just how it will work when installed. One must consider as well how the QA plan will be introduced: what training is required, what facilities will have to be changed and the effects of downtime to the plant during the changes, the effects of the plan on the morale of the personnel, what hiring is required and so on. One must consider as well how the system will handle changes as the requirements of the QA plan change due to regulatory changes, changes to the plant and the products of the plant, and changes due to decisions from upper management. At the end, there may be records to be stored and equipment to be disposed of. All these are lifecycle considerations that must be reflected in our design.

For a product the cycle is more obvious. These artifacts will be designed, manufactured, shipped, installed, operated and removed from service.

A list of steps that might be involved in a product or process is shown in Figure 4-1.



- Design / Develop
- Manufacture
- Ship
- Install
- Operate
  - Train
  - Use
  - Maintain
  - Repair
  - Augment
  - Adjust
- Remove from Service

Figure 4-1 Lifecycle Considerations

And there is no universal answer as to how to treat any of the lifecycle stages. A design for a consumer product that is difficult to manufacture is likely to be rejected for a more appropriate design. On the other hand, if there is only a single artifact to be produced and if it is a "throw-away" item, the difficulty in manufacturing the artifact may not weigh heavily in our designs. In each case we have to look at difficulty in manufacturing in the context of the artifact we are producing.

Considering parts of the lifecycle will also allow us to reach a reasonable time and cost budget for the development and for the implementation / manufacture of the artifact. Such budgets are discussed in Chapter 10

Ultimately, most if not all business decisions have an economic base. We might feel this is sad; we might feel that there are decisions made for altruistic reasons. Ultimately the decision makers (i.e. those that control the money) are going to make the decision because they feel it will make the company money, save the company money or put the company in a better position to do either of these in the future. Does a company like General Motors develop a car locating system because they are dedicated to rider safety or because they feel this will help them sell more cars? Since they neither offer the design to other car manufacturers, nor provide it on all cars, this is clearly a decision to help sell cars (or to help sell them for more). The decisions we make about lifecycle (and all) design decisions should be moral, but should also respect the economics of the situation.

Typical lifestyle costs are shown in Figure 4-2. The production costs, which would include parts, tooling and staff, will far outweigh the costs of development. The costs to the user (which could be your own company or could be the client) will be far larger than the costs of production. Any work you can do during design to reduce costs in the rest of the lifecycle will often justify the effort.

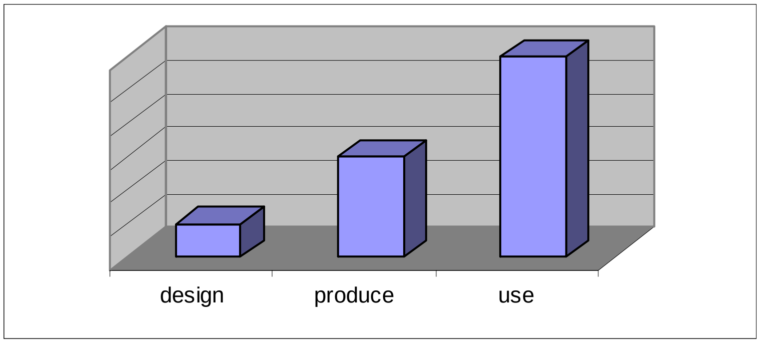


Figure 4-2 Typical Lifecycle Costs

For example, consider a part of a circuit design: a resistor, a simple circuit device whose cost is probably about a penny. If eliminated from the circuit by an engineer by spending a day of analysis and rework, the cost of this day of work may be $600[1]. If 100,000 units having this circuit are sold, the saving of only the penny per unit is $1,000. This is already worthwhile, but other costs have also disappeared, since the circuit board can be reduced in size since it no longer needs to include the resistor, we do not have to pay to mount the resistor, order the resistor, inventory the resistor, test the resistor, maintain the resistor and remove the resistor (along with the rest of the assembly) at the end of the artifact's life. Saving one resistor is easily worth a few days of engineering time in this case.

However, the distribution of Figure 4-2 is not universal. If only one or two copies of the artifact are produced, the cost patterns are different, and it may be well worth *adding* components if it will simplify the testing, increase the reliability or reduce the development time. Every situation must be considered independently.

Crafting a design to suit lifestyle considerations is very important, and this importance should be reflected in the Requirements document. Designs can have an orientation so they favour production, repair, cost and so on. Many of these are explored further in the section on "Design-Fors" to come later.

4.4 Artifact Families

Software and electronic development will usually be done through a series of versions and parallel offerings[2] to the customer base. The distinction usually made is that an offering is a release to a customer, with a certain set of attributes. Versions of an offering will be used to fix problems, reduce manufacturing cost or to add some features to keep the offering competitive. Note that in some cases the product(s) are all described as versions, with the major part of the version number changing when there is a significant change (1.xx, 2.xx, etc.) and the minor part of the version number changing when there is a fix or lesser change (1.2, 1.3, 1.45, etc.).

Figure 4-3 illustrates a typical overlapping set of offerings.

[1] This is based on about a $80,000 salary with overhead.

[2] These parallel offerings are usually called "models", but we will avoid confusion with our use of the word "models" in the development process described later in this chapter by avoiding the use of the word here.

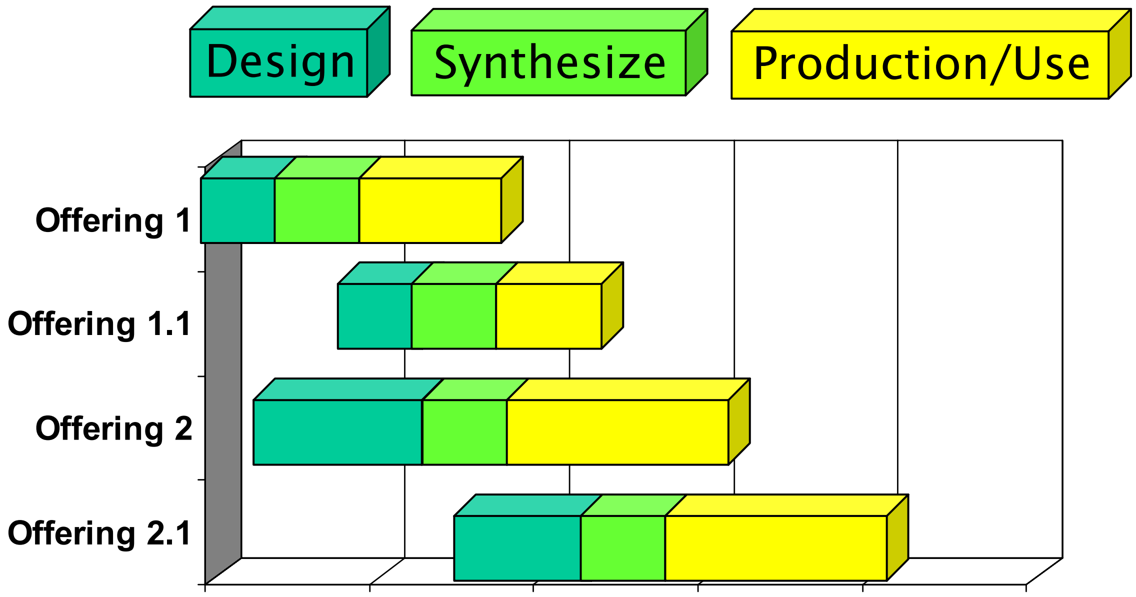


Figure 4-3 Sample Timescale of Product Offerings for Sale

Note that it is not uncommon that a new offering will begin development even while development is being completed on an older offering. As development continues, new ideas are emerge, competition may change the market and requirements may drift. Rather than continue with a never-ending development of a first release, ideas will be moved to a second development stream. Developers will also be moved to that stream as their work on the initial stream ends.

Summary: An artifact design usually is not created and then forever unchanged. Most are adjusted and tuned, paralleled with designs having different features and eventually replaced by something with the same basic functionality, but higher reward. Design of an artefact should keep this under consideration.

Applicability: Requirements and System Design

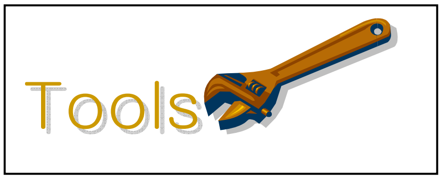
The realization that our designs are part of these artifact families should be reflected in our design work. Unimplemented design ideas coming from brainstorming or as a result of work on our artifact should flow forward to the next artifact designs. Our software and our hardware designs should include decisions and hooks that allow future generations to be more easily developed. Since we can expect to move to these new artifact development teams eventually, we are only supporting our own futures.

Listed below are some example choices that will support artifact families. These are not necessarily ideal choices in every situation, but should be considered.

- Clear and complete documentation of artifact development
- Programmable logic and processor-based development instead of highly-specific implementations
- Use of field-programmable devices and memories
- Module-based approaches with generalized, implementation-independent interfaces easily adapted to, whether software or hardware.
- Development in industry-standard languages with industry-standard products
- Use of well-supported, purchased subsystems instead of internally-developed systems.

- Use of established products for inclusion in the artifact and in development of the artifact that are (as well as you can tell) not near the end of their product lifetimes or in danger of supply problems.

Families can also be extended by addition. Often you will want to create requirements for future additions. Computers are a classic example where hooks and automatic installs are added to support features and devices not yet available.



4.5 Tools: Snapshots and Use Cases in Requirements Generation

Two tools work together to help us produce requirements that reflect what is needed by our client and other stakeholders. One is the "Snapshot" and the second is the "Use Case". Unlike the requirements themselves, these two tools develop descriptions that are highly specific to the context and usage of the artifact.

These two tools describe interplay between actors and the artifact, where the actors are either stakeholders or some other parts of the system. Note that, like the stakeholder, an actor can be an institution (such as a bank) or an organization or other selection of people (such as a lobby group) as well as an individual person. An actor can also be an object, such as an airplane or runway or database. Normally these objects would not be included in a set of stakeholders.

Both Snapshots and Use Cases were initially developed in the software realm, but have obvious applicability to other engineering disciplines. The Use Case has become an important part of UML [see Section 5.5], but can be used independently [21]. The Snapshot is an adaptation of the "Story" or "Lightweight Use Case" of Extreme Programming, a programming methodology [20].

Summary: Snapshots and Use Cases are introduced. These are invaluable methods of moving to requirements and of aiding further stages of the development.

Applicability: Through the development process.

The two tools differ from each other in their scope; we will look at the Snapshot first.

4.5.1 Snapshot

The Snapshot is a quick look at some aspect of an artifact in some part of its time from conception to disuse and disposal (the artifact's "lifecycle" which we will look at in detail later in this chapter). It is like a photograph – It contains no steps or states, and describes a single aspect of interaction between the artifact and the actor.

Examples are:

- toaster must fit under kitchen counters
- we must be able to read the display in direct sunlight
- it could be used at the beach and subject to sand and moisture
- the power source may be subject to brownouts and blackouts.
- service access will be required for any batteries
- the user may want to cancel the order at any stage in the order-taking procedure

Note that, unlike requirements which must be absolutely testable, a Snapshot can be missing the measurement criteria. So

The camera will have to be light enough to be carried easily.

is a good Snapshot, but would not be a good Requirement. When this ultimately became a Requirement it might be:

> The camera shall weigh no more than 200 grams.

Here we have decided that anything like a camera that is 200 grams or less is going to be “light enough to be carried easily” and we can test the final design by weighing it to see if it meets the Requirement.

Snapshots can be steps in a procedure (“The user must be identified”) or conditions (“The product may be used in the rain.”) or events (“The user might enter zero”).

What is the advantage of the Snapshot over the Requirement? There are two important advantages. First, we can generate them quickly, without a lot of research and background work. They are the natural product of the freeform, rapidly-changing environment of brainstorming and of the thoughts that flash through the mind as one considers the lifecycle of the artifact. Second, Snapshots are closely linked to the use of the artifact. Requirements, such as the example just above, are stripped of this link – Why must the camera weigh no more than 200 grams? Is it to avoid breaking the neck straps that it must work with? Is it because the assembly line can tolerate assemblies only up to this weight? Or is it a user requirement? The Requirement is no longer tied to a reason, as is clear with the Snapshot. If we had to revise our Requirements because a camera could not be designed to meet the other requirements and still be under 200 grams, having the reason for the requirement (i.e. the Snapshot) would allow us to make reasonable tradeoffs.

Note that because it is testable and because these tests will tell us whether the product complies, the Requirement, must ultimately be generated.

4.5.2 Use Case

The Use Case is similar to the Snapshot in that it links actors and the artifact, but it differs in complexity and scope. One of the actors will be considered the “primary” actor for the Use Case, and the Use Case is a description of how the artifact will provide the primary actor with a measurable, desired result consistent with the attainment of the product goal. [21]

Consider a banking machine as the artifact and a person with an account that uses the bank machine for deposits, withdrawals and other banking procedures as the primary actor. Making a deposit and making a withdrawal would be Use Cases for this primary actor. These are consistent with the goal of providing remote banking service without requiring a human service provider.

Establishing the account identification and enabling access through a card and PIN would be steps in the “withdraw money from the account” Use Case, but would not be considered a Use Case by itself for this primary actor. This is because the primary actor we are considering, the bank machine user, does not complete a reason for using the machine by just establishing identification. Getting secure account access is important to the withdrawal process in that the user wants the money kept inaccessible from other users, but it does not by itself fulfill the user’s goal in using the machine, which was to obtain money from the account.

If instead the artifact is the banking system and the primary actor is the banking machine, then “establishing the account and enabling access” **is** a Use Case, since it completes a goal of the banking machine. From this example we can see that there are many levels of Use Cases, depending on how we select the primary actor.

One can think of Use Cases as sections of a User’s Manual for the artifact. “The owner maintains the camera” is an example. Here the actor (the camera owner) will perform a series of tasks (maintenance). The Use Case would describe these tasks step by step.

Here are some more example Use Cases:

- Service people maintain the printer
- A customer has a printer serviced [this could be a Use Case for service centre software]
- Account Holder withdraws money from bank machine

- The LAN hub is installed
- The purchaser will undertake a formal acceptance procedure

Note that these are actually **titles** of Use Cases. The entire Use Case would have the step by step description, which can be a flow chart, pseudocode, text or any other suitable representation. We use the title as the name for the whole.

A typical short but complete textual example is shown in Figure 4-4. The title is followed by the Use Case body. Note the prerequisite conditions (including the action that triggers the Use Case) and the use of conditionals in this example. Some of the procedures are generalized and could be shared with other Use Cases – these are underlined. What this indicates is that this Use Case is composed of some lower-level functions.

Account Holder withdraws money from bank machine

- Prior condition: Account Holder establishes identity using PIN and bank card.
- Trigger: Account Holder selects "Withdraw funds" from menu

1. Account Holder identifies account for withdrawal
2. Account Holder indicates amount of withdrawal
3. If account has money then machine delivers money otherwise account holder is notified of deficiency
4. Machine asks Account Holder if other actions are desired. If not, disable all account accesses and return to idle situation.

Figure 4-4 Use case for a bank machine done in text representation

A Use Case must include not only the usual interaction but also variants of interaction that might also happen. For our bank machine withdrawal example this would include, for example, the actions if the machine was out of money, the actions if the bank card was rejected, and the actions if communication was lost with the central database. These variants can be noted using decision boxes or if-then-else constructs, or can be moved to a separate "Variations" section of the Use Case body.

A Use Case can also share processes with other Use Cases and these can, in a hierarchical system, be Use Cases themselves. The card-PIN verification of the user identification would be such a shared procedure, and when we implemented the system it is useful to know of such shared procedures so we can develop single modules shared by each instead of two separate modules. Careful development techniques will cause this sharing. Changing primary actors from the human bank machine user to the remote account database will identify user identification as a single Use Case from that perspective.

The Use Case can be done with varying levels of formality and completeness, depending on the stage and complexity of the development.

- The Use Case Title can be used to generate requirements. High level behaviour information that is part of the description, including the variants of interaction, can also be used during requirements generation.
- As we expand the Use Case, and as we generate subordinate, lower level Use Cases we are creating modules that will work together to provide the system behaviour we need. We are doing the System Design.
- When the Use Case information becomes detailed, we add sequence information and move to an algorithmic or pseudo-code structure, we are doing the Detailed Design.
- Since the Use Cases indicate the behavioural goals of the system, these are also very important as the basis of our system tests.

The Use Case, therefore, should be thought of as a dynamic tool used throughout the development process. Start with the Use Case Titles and expand and detail the Use Case as you move forward. During the requirements generation, one of the most useful steps is to use the Use Case as a springboard to Snapshots, as we will see next.

4.5.3 Use Case Diagram

We will expand on the Use Case by looking at the Use Case Diagram. Refer to section 4.5 for definitions.

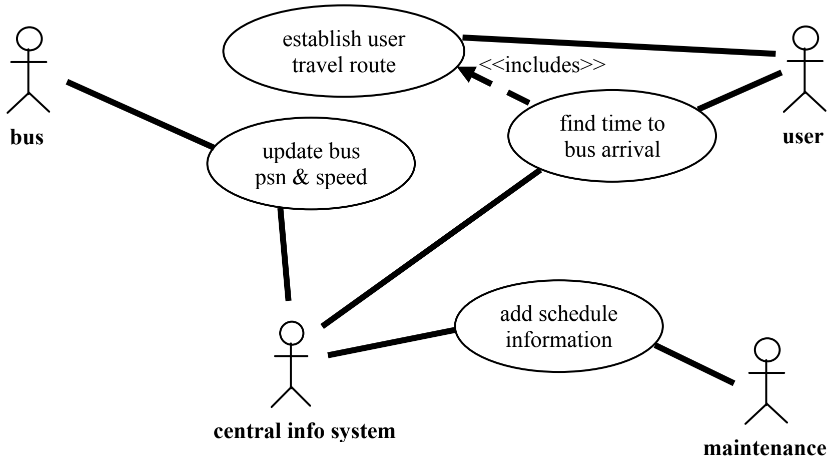


Figure 4-5 A Use Case Diagram of a Transit Information System

In Figure 4-7 we see a Use Case Diagram for a transit tracking system. In this system each bus communicates with the central station and lets the station know its current speed and position. Users can ask the central system where a certain bus is, or how long it will be until the bus arrives at a specific stop.

A system maintenance person will have to update the system by adding or removing buses and routes.

Use Cases, Modes and States

There is often confusion between Use Cases, modes and states.

A **state** is the lowest level. During use one can look at the artefact and determine distinctly the state. For example, the landing gear can be locked down or not. States have definite settings, levels, positions and values.

A **mode** involves a set of aspects involving an endeavour or purpose. For example, in landing mode the pilot will generally lower the nose of the aircraft, reduce speed, lower the landing gear and lower the flaps on the wings. However, if, for example, the landing gear is not down at a certain time does not mean that the plane is not in landing mode. It is a combination of some of these, combined with the pilot's intent to land the plane, that puts the plane into landing mode.

A **use case** is an interaction of 'actor' and artefact for a 'purpose'. The use case for this scenario is Pilot (an actor) lands (the purpose) Plane (another actor). This use case would make the designer realize that certain facilities (flap control, landing gear control) had to be available to the pilot to implement the use case.

There are several actors. The bus is an actor, the user is an actor, the central information system is an actor, the maintenance person is an actor. These are shown on the Use Case Diagram as stick figures with labels.

The actual Use Cases are shown as ovals with descriptions, and the system under consideration is to provide all the functionality described in the oval. Lines link each of the actors with the Use Case.

Each Use Case should be described in a parallel document to the Use Case Diagram. In this document further details of the case are given so that it is clear what the Use Case does, how the actors interact with the Use Case (for example, what information flows each way), when the Use

Case happens and how it is deemed complete.

We can also place “inclusions” by one Use Case of another. For example, “Finding Time to Bus Arrival” might include “Establish User Travel Route”. Inclusions are shown by linking the Use Cases with a directional, dashed arrow with ‘<<includes>>’ beside it.

Note that as we implement our design we also expand our Use Case Diagram. For example, “Establish User Travel Route” could be done either by establishing the user identification with an associated route already entered, or by entering a route or set of coordinates. These could have associated Use Cases.

Using the Use Case Diagram or some other Use Case description equivalent has the following five important results:

1. We identify the interaction of the stakeholders with the artifact.
2. We explore more than just the ‘ordinary’ interactions, but also the upkeep, installation and other important aspects to the design. (Some of these will be examined further when we look at the Lifecycle considerations of the artifact.)
3. We specify the requirements of the artifact, both directly and implicitly.
4. We can easily extend the Use Case into validation tests, functional decomposition, anomaly analysis and other areas – we will explore this later.

These Use Cases are the basic feed into the System Requirements as we will see next.

4.5.4 Snapshots / Use Cases / Requirements

Now let’s look at the interaction of Snapshots, Use Cases and Requirements during the development process.

Like most of the situations in development there is not a fully sequential process. Most of the flow will be Use Case Titles → Snapshots / Use Case Steps → Requirements, but some Snapshots may suggest other Use Cases, and even some Requirements may indicate other Snapshots or Use Cases. The Requirements for one Use Case may be developed before Snapshots are produced for other Use Cases.

Consider an example. An altimeter is an instrument that tells a pilot the height of the aircraft above sea level. It is used in determining a safe flight path and in conjunction with information about the height of the ground being overflown to determine the aircraft height above the ground. Here are some of the Use Cases.

- pilot reads the altimeter
- pilot resets the altimeter based on information from the airport
- the service crew maintains the altimeter
- the manufacturer assembles the altimeter
- the shipper ships the altimeter to the airline manufacturer

You should easily see from these example Use Cases for an altimeter that the number of Use Cases for this instrument is very large and only a fraction is shown here. We also might want to subdivide some of the Use Cases where there are subgoals accomplished by the primary actor. For example, “(the service crew) maintains the altimeter” could reasonably be divided into “cleans the altimeter”, “tests the altimeter” and “calibrates the altimeter”, since each of these would be a reasonable single operation that the service crew could perform on the altimeter in some circumstances.

As we move from the more general to the more specific as we refine the artifact requirements, the number of Use Cases will increase and their goals will be at a lower level.

We take the first Use Case from our list for the altimeter and diagrammatically produce some of the Snapshots and Requirements. Figure 4-6 shows a common form of representing the Use Case, the Use Case Diagram. Figure 4-7 extends this Use Case by showing some Snapshots derived from it.

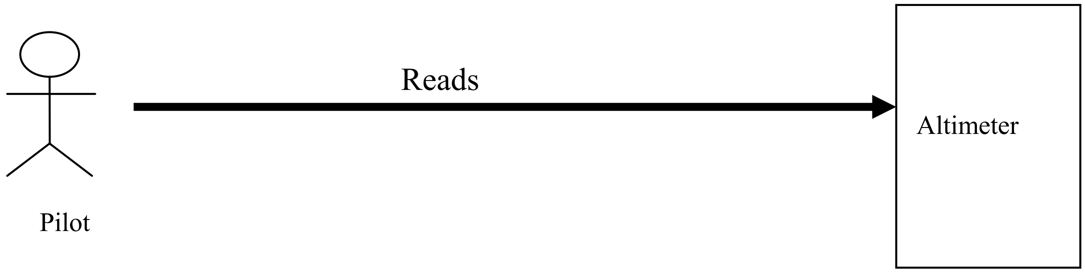


Figure 4-6 Read Altimeter Use Case

As was said, sometimes a Snapshot will suggest Use Cases and other Snapshots. In our example the Snapshot “blocked input tube” could suggest the Use Case “service crew cleans altimeter input” and the Snapshot “pilot must have a method of clearing the input or providing an alternate input to the altimeter during flight if tube is blocked”. [For your collection of trivialities: In non-digital altimeters which convert air pressure to altitude the pilot can break the cover glass on the altimeter to provide an alternate air pressure input if the external input is plugged.]

Entire books have been written on Use Cases [21] so there is much material being left out of this discussion. Further exploration is encouraged.

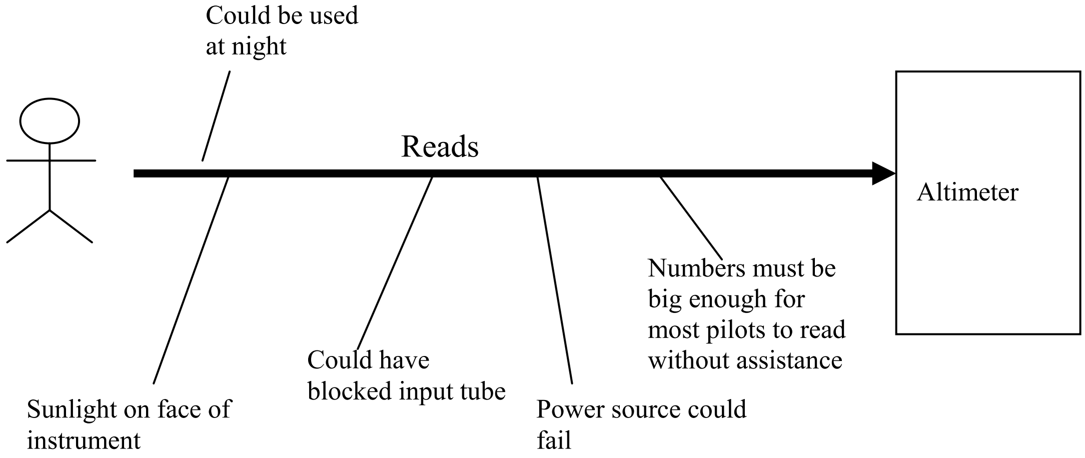

Could be used at night
Reads
Altimeter
Numbers must be big enough for most pilots to read without assistance
Could have blocked input tube
Sunlight on face of instrument
Power source could fail

Figure 4-7 Snapshots from the Read Altimeter Use Case

When can we be confident that we have a complete set of Use Cases and Snapshots? This has no algorithmic or absolute answer. In fact, we use the same set of “completeness” aids that we use for requirements, such as those in this chapter.

Let’s look at another example. Consider a design for a portable hairdryer. One of the more obvious Use Cases is “Wet-haired person dries hair with hairdryer, as shown in Figure 4-8.

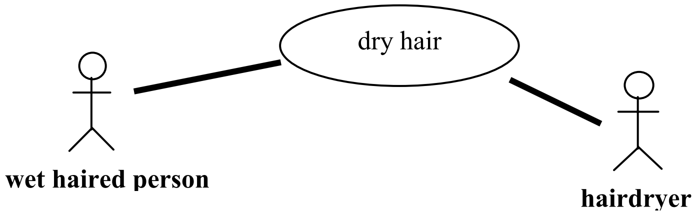


Figure 4-8 One Use Case for Hairdryer

The Use Case description could be as follows:

1. Wet-haired person (WHP) gets hairdryer from storage
2. WHP holds hairdryer, selects the heat level and starts hairdryer
3. WHP directs hairdryer to wet hair as required until hair is dry. This might be done section by section, or (more likely) hairdryer will be moved around drying all parts a bit at a time
4. WHP (now Dry-haired person) stops hairdryer and stores hairdryer away.

Since diagrams are faster than writing, let’s start with a slight modification of Figure 4-8 and add a the descriptive information.

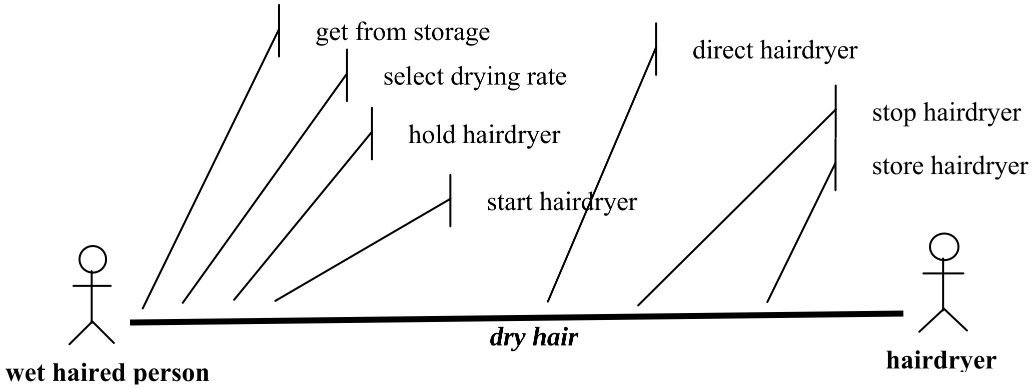


Figure 4-9 Use Case with Descriptive Information

From this figure some of the requirements become evident.

get from storage / store hairdryer	Function: Hairdryer shall be able to be stored close to place of use Objective: Hairdryer will not take much space in storage. Alterative: Hairdryer will not take much *shelf* space in storage.
select drying rate	Function: User shall be able to select one of at least two drying rates.
direct hairdryer	Function: User shall be able to direct drying to certain parts of hair

Thinking some more about what the WHP would want during this Use Case, we might add:

safety	Constraint: Hairdryer shall not cause harm to WHP under use
speed	Objective: Hairdryer will dry hair quickly

And we can go further: If we have to be able to direct the hairdryer, then the following requirements become evident:

direct hairdryer (cont)	Function: Hairdryer shall weigh no more than xxx kg. Objective: Weight of the hairdryer is low. Objective: Drying area large Objective: Minimize energy lost by misdirection (ie not on hair)

Note that no mention has been made of *how* we are going to dry the hair. Although every hairdryer you probably have ever encountered uses forced warmed air, this does not mean that it is the only way that will ever be devised to do so. When we choose how we are going to dry the hair, we will generate some side effects (see section 7.2.5, Force Flow Diagrams). We will come back to Use Cases yet again in the section on Risk to look at this.

Diagrammatically we can extend our drawing to show these functions, shown below for the "direct hairdryer" part of the Use Case.

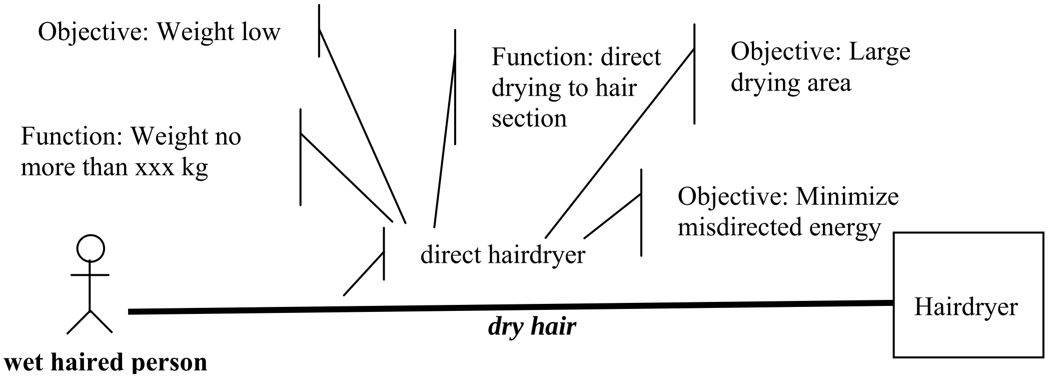
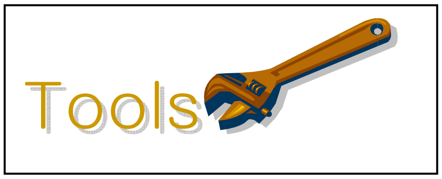

Objective: Weight low
Function: Weight no more than xxx kg
Function: direct drying to hair section
Objective: Large drying area
Objective: Minimize misdirected energy
direct hairdryer
Hairdryer
dry hair
wet haired person

Figure 4-10 Use Case with Requirements

This can be repeated for other Use Cases in conjunction with consideration of other viewpoints and "design-fors" as described in this chapter. This will net a comprehensive set of artifact requirements.



4.6 Tool: Environment Diagram

One way of determining what an artifact must do is to examine its environment. This includes the physical environment (will it get wet? will it be vibrated? will it get very warm? will it be in contact with seawater?), the connection environment (will it connect to the Internet? will it attach to a datalogger?) and the user environment (will it be controlled directly by a human?).

When we consider the interaction of an artifact with an environmental entity, we can associate a direction of flow. For example, consider an instrumentation buoy to be released from a ship in the ocean. Seawater will affect the buoy, but the buoy will negligibly affect the seawater. On the other hand, a radio-linked control/monitoring device would be affected by the buoy and would also affect the buoy.

Summary: The Environment Diagram describes the place of the system being designed in the physical environment(s). Requirements are developed from the interactions

The Environment Diagram (also known as a Context Diagram [5]) shows these directional effects. The centre is the "System of Interest". Each of the environmental entities has a box and a link with a single or double arrowhead to the System of Interest. Arrowheads show the direction or directions of effect. In Figure 4-11 the sun affects the buoy but the buoy does not affect the sun. If the buoy were not controlled, the arrow would go only from the buoy to the monitor. Since this is a two-way link, there is instead a double-headed arrow.

Note in the diagram that an operator is also an environmental entity. This is because the operator may change the battery, turn the device on, or perform other operations required by the system.

Where the diagram becomes useful is in the development of the Use Cases and the requirements from the interactions. The operator interactions just described will mean the operator must have a means of access to the battery compartment and power switch. The presence of seawater indicates the level of protection that would likely be required for batteries and switches.

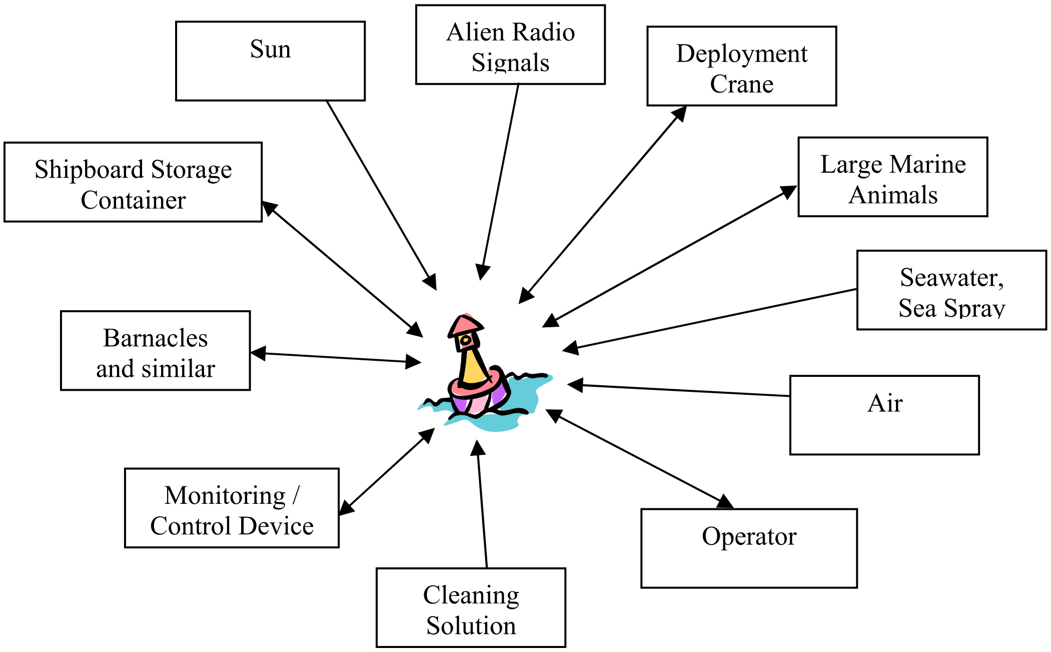


Figure 4-11 Partial Environmental Diagram for a Buoy

Consideration of some of the links will result in more requirements than others. For example, the radio-frequency link to the monitoring/controlling device will generate physical requirements, such as circuits and antenna to support the sending and receiving of the radio signals, and processing requirements such as the communication protocol, data formatting and command interpretation.

4.7 Design Fors – Introduction

Designs can be oriented to certain ends. Sometimes it is important that the design be ready on time. Perhaps it has to meet a certain cost target. There may be environmental concerns to manage. These are typical requirements as discussed first in Section 5.2 .

The following is a list of what you might "Design For":

Summary: There are usually a great many ways of achieving a required functionality. Choosing between these ways will often require consideration of other important ends. These are the "Design Fors"

Applicability: Very important "Design Fors" will become Requirements. Others will be considered during the System Design and Detailed Design.

Cost – Most design activity is run to a budget, so this is one cost concern that could direct how you design. The product itself (or the process) will have an associated cost to build (or to conduct in the case of a process). Keeping these costs down will increase company profitability and this is always a concern. In some cases the concern with be the cost over the entire lifecycle.

Reliability – A braking system on a car or a aircraft flight control must be highly reliable. This can be done through choice of components and through additions such as warning systems, backup systems and self-testing that allow failures to be few and to be detected and compensated for.

Environment – A camera that may get splashed will be built differently than a camera that must work under 200 feet of water, and both of these will be different than a camera that is expected to be kept from any dampness. Sample environmental concerns: chemical environments, wind, cold, heat, vibration, shock, high electrical noise, humidity and animal attack.

Manufacturing – Anything that will have to be built in multiple copies is likely to justify time to design for ease of assembly. This is a variant of 'cost' but one that is somewhat specialized.

Maintenance – There was a production car made once that required the removal of the engine in order to replace one of the sparkplugs (a routine automobile maintenance requirement). Clearly the engineering department was asleep during the design!
If the artifact is to be repaired or adjusted or tested it is wise to add access panels, connectors and sometimes even specialized components for this purpose. Modern automobiles have connectors for diagnostic equipment that speed repair of their computer-based control systems.

Ergonomics – How the user interacts with the artifact is often very important. Typical ergonomic considerations are installation height, size of buttons and placement of warnings.

Speed – In some cases, such as the design of a new processor version for a desktop computer, speed of operation of the artifact is critically important as it will greatly affect the volume of sales.

Power – In some cases, such as the design of a new processor for a laptop computer, speed may take second place to power. In a laptop one of the important considerations is battery life and so power conservation strategies and hardware support are important.

User Safety – Legislated and moral obligations require that what we design be safe for users. In some cases this will be a large factor in the design. This concern could affect the choice of component, the configuration of the design and the housing chosen. In a process, certain procedures will be put in place with the sole aim of keeping people away from harmful machinery or dangerous situations.

Quick product release – Many products have a short market life. Consider a cellphone. The first ones worked in a single mode (analog) and were about the size of a large chocolate bar (not counting the early radiophones that were the size of a shoebox!). Now cellphones fold, take pictures and movies, play games, have many rings, can download, work in other modes besides analog and have color displays. A state-of-the-art cellphone is old news 6 months later and probably off the market in 12 months. A cellphone designer cannot afford to take years to develop a cellphone.

The first cellphones did not have all these features for two reasons. First, delaying the features gave opportunity for selling replacements later[3]. Second, to develop these features would mean missing the first markets, which would mean giving up large revenues. Cellphones are built for quick product release – many of the internal workings and technology decisions are not optimal, but designed to get the product out to market quickly.

[3] I'll let you decide if this opinion is overly cynical.

Consideration of quick release can determine decisions of using an existing software library instead of creating faster or cheaper routines from scratch. It can determine whether you purchase or design a power supply.

Extensibility – As noted in section 4.4 , many products (especially software or electronics) appear in multiple versions. If such products are designed with extensions in mind, then later versions will be easier to produce and often faster to market.

Others – This list of “Design Fors” is not exhaustive. A designer should probe beneath the first remarks made by the customer and determine the customer’s motivations. Does the customer want the product quickly? Do they need it to be very easy to use? Is the customer on a limited budget? Are there additional features and extensions that will likely come in the future? How many copies of the artifact are being produced? With these type of questions answered the designer should look to the artifact production. Are there ways to increase the reward and decrease the risk for the company? These are some of the questions you must ask to determine your design orientation.

A number of tools for “Design For” will be introduced in Section 12.4 .

4.8 Corporate Context

The size of a company will affect the types of projects undertaken and can introduce requirements of the project and the development process. In general, small companies move quickly but cannot independently handle large projects. These companies can often make a profit on very small projects. Large companies move more slowly, but have the reserves and resources to handle larger projects. Small projects can be unprofitable due to the overhead of the management structure.

Large companies require structure in order to keep the numbers of communication paths tolerable. Considerable employee time is taken in communication at any time – having too many people in any one group makes communication an overwhelming, overly costly use of time. As a result, large companies often have unions and technical organizations, a large hierarchical organization chart and a “corporate ladder”, and specializations of function.

Summary: What the company can afford and its position in the marketplace will affect the kinds of projects that it can undertake and the method it will use.

Applicability: Corporate context can be reflected in the requirements and will also come into play later, during go-nogo decisions and implementation selection in the System Design step

Smaller companies have less specialization and less structure. It is more important that each employee be able to handle many types of work.

Typical “basement” startup companies have less than a half-dozen employees. The work is done by anyone available and skilled enough to do it.

A project proposal being handled by a large company has to run through levels of technical and budgetary approval at many levels and in many sections of the organization. Only a larger project that will ultimately pay the costs of this process and the costs of managing the project in the same way will be considered.

A project proposal handled in a very small company is at the opposite end of the spectrum. The budgetary information is simple to understand, the people involved can all sit at a table, and a project decision can be made in a few hours.

A useful tool for analysis of a new project is SWOT. This stands for:

- Strengths – What are the existing strengths of the company? This would include available cash and borrowing power, existing resources for development, production and sales, and existing market inroads.

- Weaknesses – What are the weaknesses of the company? What does not exist at the present? Particularly, what parts of the company are going to be strained by taking on this work?
- Opportunities – What are the rewards for undertaking this work? How will it increase the strength and position of the company in the future?
- Threats – What is the competition? What alternatives are available to the customers that will make the project a bad idea?

Typical requirements resulting from evaluations of corporate context would include an overall cost of development

4.9 Product Context

When considering the requirements for a design, besides its primary functions and the design-fors already examined, we need to consider a number of other factors. Some important ones are described in this section.

Summary: The product context, those characteristics of the particular product, affect the design process and the design requirements.

Applicability: Requirements analysis and System Design.

Production Volume

If we are producing only one item, or only a few, our costing structure and therefore our design strategies and methods are very different from if we are producing a high-volume consumer item.

In Table 4-1 we see a comparison of certain typical costs for categories of production volumes. The table examines the proportional costs of doing the development itself, of then producing all the necessary replicas of the artifact, of warranty, of a single artifact failure and of a design failure that is present in all artifacts and must be fixed. The indications given in the table are typical, but may differ in any particular project because of the other parts of the project context.

For a very large volume consumable[4], for instance, the cost of development is low compared to the total production cost. It therefore makes sense to spend extra effort in development to decrease the costs of production. Development will also include procedures for volume production and quality assurance testing.

For a product done in small volume, the development is a significant part of the total cost. If you can reduce the time in design, even if the design is non-optimal as a result, this is often the best approach.

(compared to total cost)	Very large volume consumable	Large volume non-consumable	Small volume	Very small volume
Devt cost	- - -	- -	+	+ + +
Repeat Production cost (total)	+ + +	+ + +	+	NA
Warranty cost	- -	-	+ +	+ + +
Cost of single product failure	+	+	+ + +	+ + + (?)
Cost of design failure	+ + +	+ + +	+ + +	+ + (?)

Table 4-1 Comparison of Costs for various Production Volumes

Proportional costs of a single product failure are generally low in the higher volume items than when you have only a few products in the field. However a design failure, a problem present in all products in the field, can be very expensive to rectify, regardless of the volume of production.

Operating Environment

Physical structures, including printed circuit board assemblies and even integrated circuits themselves, are subject to environmental agents. Temperature, chemical action, abrasive elements, vibration and shock all interact and can ultimately cause loss of function. A seemingly simple decision, like selecting a capacitor with connections at each end instead of connections at a single end can extend circuit life. The better-anchored capacitor will not be as sensitive to vibration and is less likely to work loose.

[4] A consumable is a product that is used up by the customer, like a bar of soap.

Some of these concerns are created by the components themselves. Fast circuits and high-powered circuits can run very hot – cooling fins and fans are required to keep them at operating temperatures.

Non-office environments, as in a mine, a sewer, a steel mill or deep space, place increased demands on a product that must be realized in the design requirements. Software ultimately interacts with physical devices as well, and therefore should take into account the environmental demands on these devices.

Design Requirements should include all environmental concerns.

Historical Context

Designs are almost always done in some sort of historical context. The new design may be a version of an older design, or a new model in an existing product line. Almost always an artifact is sold into an established customer base, with an established method of working.

The requirements should reflect this context.

A telephone system is an example where historical mixes with regulatory. New telephones must meet the electrical and communication standards established when automated telephone switching was first conceived, as well as added standards more recently added.

Similarly, people expect new Pentium processors chips to run software that was written for processors that were much older. Operating systems, like Windows®, are expected to run on older and newer processors, and to support older and newer application software.

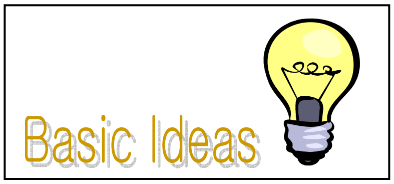
Legislative Context

The world of the electrical and computer engineer is one full of regulations and standards. Landline and cellular telephones must meet certain connectivity and use requirements. Anything with electricity must meet certain safety requirements. The use of any chemicals, such as those used to make printed circuit boards, requires careful use under environmental and safety regulations. Anything with higher frequency circuits must meet electrical emission requirements. The list goes on, and it gets larger when you include regulations in the U.S., the European Community and other governmental jurisdictions.

Any set of requirements must include compliance with all applicable regulations. Sample regulatory bodies:

- Workplace Environmental (WHMIS)
- Radio Frequency Emission (FCC)
- Electrical Safety (CSA, Ont. Hydro)
- Professional/Ethical (PEO)

Chapter 5 Getting Formal - Finishing the Requirements



5.1 General Structure - Requirements

Once you have the goal, the next step is to nail down the requirements.

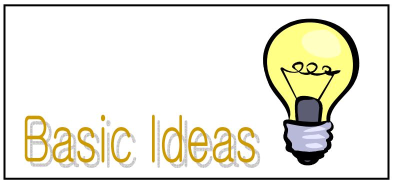
When you were throwing around ideas when the project was first introduced, what was it that made you decide that one idea was better than another, or that an idea should be rejected, or that an idea didn't go far enough? The answer is that you were developing a set of criteria to measure the ideas against. That one was too big, that one takes too much power, that one will take forever to produce, that one depends on some element that doesn't exist. You might express these orally, or you might just determine it in your head.

Summary: Requirements analysis and the Requirements Specification are studied and detailed.

For brainstorming, this analysis is a bad idea. See Section 8.3 about Brainstorming for more. However, the analysis is part of an engineers thought patterns, and if you can realize what it was that made you decide that some ideas were weak and others not, this will help you determine the requirements.

As we saw in the last chapter, specific requirements are usually developed in two steps. In the first step we determine some type of general evaluation, a Snapshot. The cellphone must be small, the horn must be loud, the stand must have legs. We MUST not stop here. The second step is to make the requirement measurable. The cellphone must fit in a shirt pocket, the horn must sound at 90dB or louder, the stand must have at least three legs that hold the piece 1 m from the ground. It is this final step that generates the actual requirement or requirements that will go into the Requirement Specification.

Not all requirements may go through the two steps. For example, if it is determined that the device must be green, then this is already measurable. However, many, such as "the horn must be loud", are usually first determined by finding the imprecise statement, then moving to the precise. "The horn must be loud" leads us to the question "How loud is loud?" and the final requirement "The horn shall sound at a minimum of 90 dB".



5.2 Functions, Objectives and Constraints

Requirements fall into various categories:

1. Constraints
2. Functions (or Needs)
3. Objectives (or Wants or "Measures of Effectiveness")

Constraints must be met and are strictly pass-fail. To pass the course you must have 50%. Not 49%. If you have 51%, that's nice, but does not make you pass any more than does a 50%. A constraint is determined by legislation or other forces outside the client.

Functions also are pass-fail, but these are client-based. The weight of the radio shall be under 1.0 kg. The TV remote control shall be wireless. Functions are what the artifact must have to meet the goal of the project. We also will use the word "needs" when our focus is the client-artifact relationship and "functions" when referring to the artifact development.

Summary: Requirements are based on stakeholder constraints, functions and objectives. Objectives are very important because they differentiate the design possibilities

Applicability: Requirements Design

Objectives can be either pass-fail or denoted by limit values or ranges. If practical, the toaster shall have a metal case. It would be best if the motor draws no more than 1 amp. Objectives need not be satisfied by the project, and so these are measures of "goodness" in the eyes of the user or of the producer of the artifact. Because, for each specific design possibility, these are evaluated and weighted to allow a design decisions, they are often called the **Measures of Effectiveness** of a design. We also will use the word "wants" when our focus is the client-artifact relationship and "objectives" or "project objectives" when referring to the artifact development.

The same characteristic can be in a constraint, a function and an objective. For example, the operating frequency must be below 1 KHz to avoid having to get a special radio emissions certification. That is a regulation and therefore a constraint. The product may need to have an operating frequency above 500 Hz to work – this would be a function or need. However, the client may want the device to work at the highest frequency possible. This would be an objective. Balancing other constraints, other functions and other objectives with these of the frequency would ultimately determine our actual frequency of operation in our artifact.

Looking ahead in the design process for this example: When we come to make implementation decisions at the System Design step, all three types of specifications will be taken into account. Because of the constraint and the function, the frequency must clearly be between 500 and 1000 Hertz. The designers should look at the costs and benefits of implementations at various frequencies in this range, including the benefit of meeting the client's "want" (objective) to be working at the highest possible frequency. When we choose the final design it will include a frequency target and the objective will convert from something not measurable ("frequency as high as possible") to something measurable ("frequency minimum of 900 Hertz"). All objectives must be converted during the System Design step in this way in order to be verifiable (i.e. testable).

What might be the benefits of meeting a client objective?

- Acceptance of your proposal over a competitive bid.
- Higher payment by the client.
- Additional contracts later since you responded to client wishes.
- Rapport with the client.

The relationship between stakeholder wants, needs and constraints is shown in the Venn diagram, Figure 5-1. Sometimes a similar drawing is used to represent only the *client generated* constraints, wants and needs, but we shall include the constraints, wants and needs of all stakeholders.

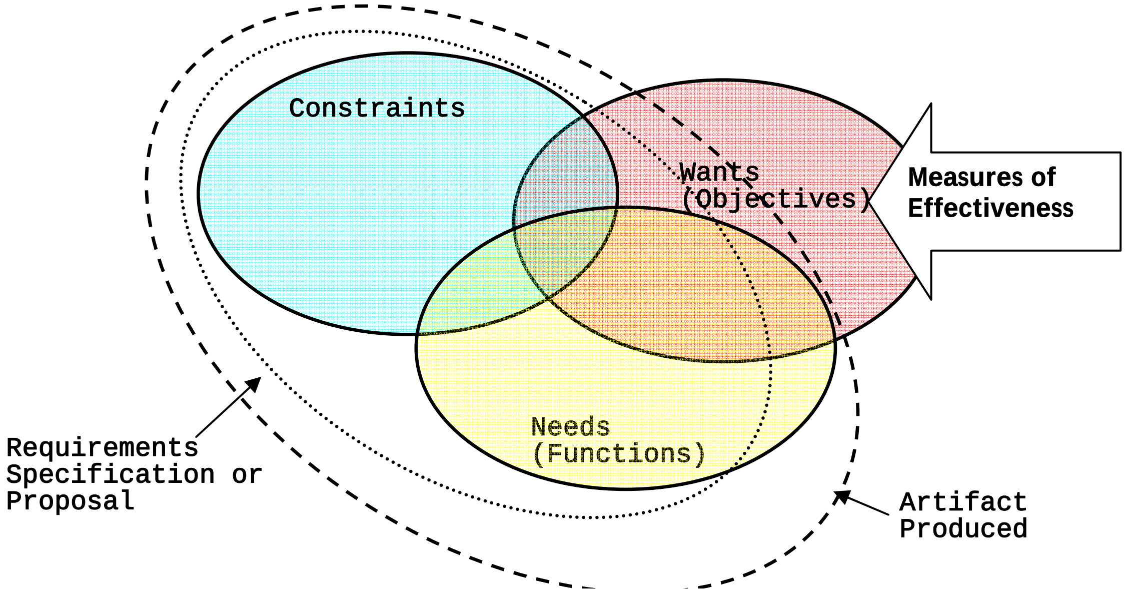


Figure 5-1 Relationships of Constraints, Wants and Needs

Concerning the clients:

- The clients will often express needs as wants
- The clients will often miss certain needs
- The clients will usually ask for more than just needs
- The clients will sometimes not inform you of all the constraints
- The clients will often specify how tasks are to be done when this is not necessary

Of importance at this stage is to access all of the client input, all the important regulatory information and all of the situational information (not told to us by the client but resulting from the use of the artifact). The result of this assessment will be the Requirement Specification.

The dotted line in the figure indicates what will appear in our Requirements Specification or in our Proposal to the clients (which is generally the same as a Requirements Specification, although a Proposal might leave out some of the requirements generated to suit stakeholders other than the client, such as our own company). Note that some of the Needs are omitted from the Requirements oval in the figure. This is because they haven't yet been recognized (a danger!) or because they are minutiae, and implicit in the understanding between clients and development team (less dangerous, but could still generate problems later!).

Eventually we will produce an artifact, as shown by the dashed line in the figure. At this time we should have included all of the features in the Requirements Specification, all the Constraints, all of the Needs and perhaps some more of the Wants in response to post-requirement requests from the clients. An artifact meeting these requirements will satisfy the clients; anything less will mean that the artifact stopped short of what was intended.

The Form of the Requirements

Constraints and project functions are generally expressed with a "shall":

> The outside surface shall be orange.
>
> The device shall respond to a signal at a frequency between 1.0 and 1.1 MHz.

"Will" may seem like synonym for "shall", but denotes a future tense and therefore has been argued to only be a characteristic that must be demonstrable sometime in the future, with the date unspecified. So "The outside surface will be orange." can be interpreted by the seller to mean that they will send someone around in the year 2504 to paint the product orange, if anyone cares at that future date.

Phrases such as "The surface shall be brightly coloured" are subject to interpretation and should usually not appear in a Requirements Specification. When they do, the client must be prepared to accept the interpretation of the phrase which the developer uses. Statements like "The operator would like the controls close together." have no built-in contractual obligation and only offer guidance that can be ignored. Objectives are often phrased in this way, and give the designer opportunity to make choices that increase the worth of the artifact to the client without compromising the functions and constraints.

Requirements must not have functions or objectives that specify the implementation. For example, say a client wants a program to keep track of employee names and departments (This would be the goal.) Some requirements might be:

1. The program shall be able to store up to 3000 employee names and their department names. [Function]
2. The program shall have a search facility to allow a particular employee to be found by name or any part of their name. [Function]
3. The program shall work on Windows XP on an Intel-based PC platform. [Constraint]

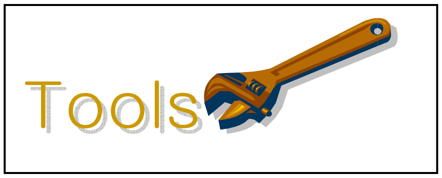
These requirements could be met by a spreadsheet program, a database program or a custom program. It could also be met by a simple text entry program! Any of these might be the best solution depending on the other requirements and on the worth of this solution's meeting of the Project objectives.

We would severely and perhaps unnecessarily limit our options if we specified the first requirement as

1. The program shall be written in MS Access® and shall store ….

If the client had a number of applications already in MS Access® the rewritten specification would make sense. It would not make sense if there was no compelling economic or logistical reason for making this choice of program at this stage. This does not preclude providing an MS Access® implementation; it just allows that there could be another, better way to provide the functionality.

Note that the third requirement above is such a client need. If not so limited we would have the option of suggesting a solution for a Linux operating system, or (arguably) for an index card system. Maybe a card system would be the best choice – not all the best solutions are high tech!



5.3 Determining Requirement Information

Requirement information is determined by considering the interactions of users and the artifact. We can often gain insights into these through certain efforts. In Section 4.2 we looked at efforts involving direct sources of information (printed information, user interviews and the like). Here are some other methods of creating these insights. These will result in Snapshots or Use Cases.

- **Itemize the borders of the project.** The artifact interacts with the environment, with other parts of a larger system, with the user. How will this be done? What are the requirements for protection of the parts from the environment and for information flow? (See the “environment diagram” in section 4.6)
- **Create first drafts or outlines of the users’ documentation.** By going over the details of how the artifact will be used and maintained you will often realize that certain facilities will have to be present in the artifact to make these user actions possible and efficient. In addition to giving well thought-out Use Case information to use in the requirements, this document will eventually have to be produced, so there is no lost time here.
- **Explore the attributes of the artifact.** Exploring the attributes we believe our artifact should have will lead us to other requirements. Figure 5-2 shows various attributes can be expanded for a soldering station to give us some requirements of a station. We want the soldering station to be safe; being safe requires that the soldering station have no sharp edges. Having no sharp edges is a measurable attribute and thus could be part of our requirement specification. The statement that the soldering station “Shall be safe” is too open-ended to be a suitable requirement.

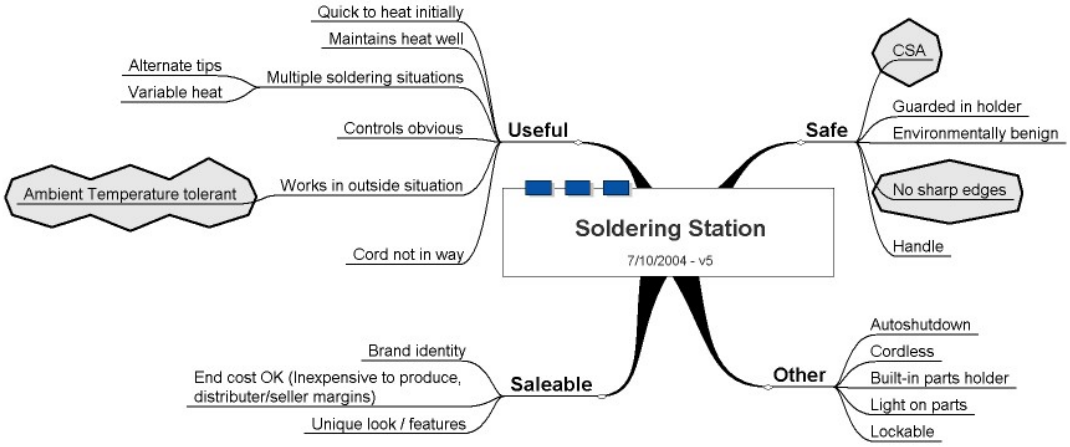
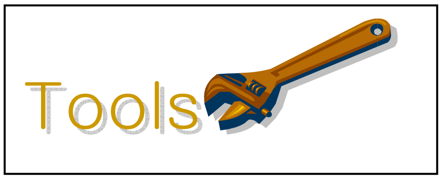


Figure 5-2 Attribute Breakdown Diagram of a Soldering Station

- **Use the UML diagrams (and related documentation) for requirements development.** These will be described later in this chapter.



5.4 Requirements Checklist

The following checks can improve your requirements.

Check #1: Requirements are Measurable

Requirements must be met or not met, which requires that they be measurable. Requirements that sound like advertising hype are not likely acceptable requirements. Table 5-1 gives examples of good and bad requirements and the measures they are associated with.

Good/bad	Example	Reason
Good	The exterior shall be painted yellow.	The colour can be seen to be either yellow (pass) or some other colour (fail)
Bad	The signal shall be low-pass filtered.	What is low pass? For audio this is a different filter than for video. How do we know if the filter is adequate?
Bad	The interface shall be user-friendly	Some people’s user friendly is other people’s boring and other people’s frustrating. There is no method of measuring ‘user friendly’. More on this below.
Good	The filter shall pass signals above 3 KHz with under 2% attenuation and reduce all signals 2 KHz and lower by at least 10 dB	We can run tests or simulations on the filter that will check to see if it meets these criteria.
Bad	The node shall respond quickly to the query	Is this seconds, milliseconds or nanoseconds? We can’t tell how to evaluate our response time here.
Bad	The product shall be light and portable	Light is a relative term. Portable just means movable. There is no clear pass or fail.
Good	The battery shall last a minimum of 5 days when device is in standby mode.	There is a simple test to see if the device meets this requirement. We can run the device on standby for 5 days on a battery and see if the device is still operable.
Good	The system will be down for no more than 4 hours in a year..	While there is no simple test for this, we do have a hard target. Careful failure analysis done in a method agreeable to the customer and to the producer would be needed to determine if the system met this requirement.

Table 5-1 Evaluations of some Sample Requirements

User Friendly

“User Friendly” deserves particular comment. It was initially used primarily for software, but has been extended to most devices with any human interface. It is a orientation of effort, but it is not adequate as a Requirement Specification. In order to differentiate good user interfaces from bad many metrics have been developed. Selection of colour, amount of text on a page, grouping of common controls – these are all design methods that can be qualified and quantified in ways that will make them testable. These testable qualifications and quantifications are what should be appearing as requirements.

Summary: A checklist for a Requirements Specification. Having measurable quantities is the heart of good Requirements Specifications.

Here are some general and specific measures that you can consider. First general ones (from [5]):

- Interfaces
- Context
- Performance / Function
- Modes & States
- Fuel & Waste (side effects)

Now some specific to the type of design -

Some Analog Measures	Some Digital Hardware Measures	Some Software Measures
o cost o signal to noise o input range / output range / bandwidth o power supply tolerance & requirements o heat generated o speed / skew / phase shift o size o component availability / manufacturability o testability	o cost o function o cost o size o states / modes o speed (throughput & response) o input tolerance o # bits in data paths o component availability / manufacturability o testability o flexibility / maintainability / dynamic update? o component availability / manufacturability o testability	o cost** o function o size o speed (average; minimum response) o flexibility / maintainability o input tolerance o "user friendliness" translated into testable items o testability

You should go through every specification and make sure there are associated measurable values with a pass/fail criterion. This is not simply adding numbers to your unspecific statements. There must also be a way to perform the test to verify the number. If you say the circuit shall respond in 2.2 picoseconds, you should be able to find a measuring device and devise a test that will show that the circuit meets this criterion. It also means that there must be access to the testpoints of the circuit or software. You may have to program extra monitoring routines or add circuits and interfaces to allow the testing to take place. This may add cost and change the characteristics of what it is you wish to measure. Ultimately these tests will be collected into a system test.

Check #2: Requirements are Free of Implementation Detail (Outside Constraints)

Consider the following:

1. The unit shall have an analogue-to-digital converter and microprocessor implementing a Fast Fourier Transform on the digital values to determine and output the value of the highest power frequency found in the input.
2. The unit shall determine and output the value of the highest power frequency found in the input.

The first specification constrains the design unnecessarily. Even though this could be the best way to implement the design, this may instead not be true. This is not the point in the design at which we should be restricting our thinking.

Examine your proposed requirements carefully to see if you have unnecessarily restricted them by specifying or implying a method of implementation.

Check #3 Interfaces All Specified

Since systems are parts of other systems, there will be interactions with those other parts. All of the interactions at the interfaces, or points of contact, should be specified. These interfaces can be other devices, human beings, power sources and environmental elements.

Consider a heart monitoring unit. It has interfaces with both the patient, and the medical staff. It also may have links to data logging equipment and computers for setup and periodic testing. It is plugged into the

AC system to get power. It also "interfaces" to the environment, which could include the vibration, humidity and temperature extremes of an arctic or tropical location, of an ambulance or a helicopter. We also need to consider anomalous interface problems. Do we want it to survive a drop? to survive having a coffee spilt on it?

Note that, in light of check #1, we will have to specify these interfaces in a measurable way.

Check #4: Feature Creep Controlled

If one person does some work, it takes a certain amount of time. If two people do the work it will usually take less time, but not half since the two people will have to spend time coordinating their activities. If a third person is added, there can be increasing benefit, but more coordination time is required. Beyond a certain group size the coordination time swamps any benefit from adding additional people.

Adding features and functions to a project will do the same to the project. Every new feature or function will require some additional software or circuits that must connect somehow with the rest. At certain points the additions will require significant changes to the underlying structure, causing significant delays and significant extra costs.

One area where this is most obvious is reliability – you can go a certain distance through careful selection, configuration, execution and testing, but eventually redundancy is required, a duplication of components and of cost.

These additions, this "feature creep", is common in the development process. Clients and developers both will see additional opportunities arise as the plan goes onto the table and will suggest additions. These additions will cost, and the cost will often not be taken into account.

The following will help control this phenomenon:

a. A clear Requirements Specification. Any additions will then be obvious and up front
b. Contractual recognition, remuneration and allowance for change.
c. Designs anticipating certain changes with structure that allows the changes to be done easily.
d. Using 'fast prototype' or 'spiral' development model where evaluations and signoffs are done on artifact concepts and interfaces.
e. Deferral of features and functions to the next version of the artifact.
f. Keeping the customer in the loop – see the following

Summary of Requirement Attributes [xxx]

A good set of requirements is SMART.

S – Specific. No vague requirements

M – Measurable. Testable targets.

A – Agreed to. Stakeholders all onside.

R – Realistic. Attainable given the company context

T – Time Oriented. Must be attainable in suitable time

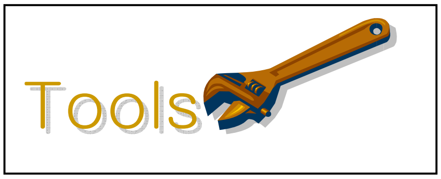
Check #5: Customer in the Loop

The previous section indicated the problems with customer-created feature creep. This problem and many customer-related problems during development can be reduced by keeping the customer "in the loop", that is well-informed of what is going on. This is particularly true in this part of the development work. If your customer is very aware of what you are planning at this stage, the chances of a "nasty surprise" on delivery are reduced.

Before committing large amounts of time and talent to the detailed design and synthesis, we want to make sure we are doing the right thing. The customer (whether internal or external) should be involved with the production of, and should sign off on

- the Requirements Specification
- a draft Users' Manual (covering installation, operation, first level problems)
- a list of deliverables
- a preliminary system test plan

If the customer wants to add an extra handle, or another button, if the customer wants to change an interface or get another document, if the customer wants to add a month-long acceptance test, these will be added early and made part of the contract cost. If added later there will be no question as to their effects on the schedule and cost.



5.5 UML2 (Cont)

UML2 is the short form for Universal Markup Language version 2.0.[18] UML is the result of software developer's efforts to manage large projects, particularly those to be created in "object-oriented" languages such as C++ and Java. It is a set of descriptive modelling tools that interrelate and that allow you to describe what is to go on in the artifact that is being designed.

Although created for software, UML is far too useful to be left as only a software tool. The UML models have applicability to design in general and to modelling processes in all human endeavours.

We will not look at every UML modelling tool. Some are somewhat specialized or software specific. Actually, UML itself does not attempt to model every situation. It is, by design, extendable by adding additional functionality and models to handle specific application areas. Look on what we look at here as a start – a set of useful tools that should be adapted and extended to best model the situation at hand.

The REASON we are doing all this is to describe the artifact to be developed well enough that we can anticipate all possible significant client and development concerns. We select the tools and use them in a manner that fits the work at hand.

Summary: UML is a set of "drawings" each of which models a system or a subsystem from a different perspective. Several are studied here.

Applicability: UML drawings can be useful for Requirements Analysis and for developing the System Design. They also have uses in engineering and general management, and in any other task requiring organization of interacting parts.

We will also not be necessarily rigorous or exact in our usage or descriptions of UML.

We already looked at the Use Case in section 4.5 . Use Cases are part of UML. In the sections ahead we will look at the following other components of UML:

- Activity Diagrams
- Interaction Diagrams
  - Sequence Diagrams
  - Communication Diagrams
  - Timing Diagrams
  - Combinations of these / Operations
- Interfaces (Composite Structure Diagrams)
  - Ports
  - Protocols
- State Machine Drawings
- Frames, and pulling it all together

Note that almost all these "drawings" will include more detailed information on the interactions depicted in the drawing.

5.5.1 Activity Diagrams

An Activity Diagram showing "Making Toast" is shown in Figure 5-3. This specifies one of the activities that might be noted on a Use Case Diagram for a toaster.

The Activity Diagram describes a sequence of activity. This could be from a Use Case; it could be a function description or process. Data flow and control can both be represented on this diagram.

To note on this diagram:

- The entry point of the diagram is shown with a large black dot. In order to enter, we must satisfy the pre-conditions. Here we must have put bread into the toaster.
- The exit point of the diagram is shown as a large black dot with a circle around it (or a hollow dot). When the exit point is reached the activity diagram is done.
- A dashed line encloses functions that are part of a process. Activities will go on in the process and the process will exit when certain conditions are satisfied. In our case, the process is the heating of the bread to make toast, and we exit normally when the toast is done.
- The "flag" shows an alternate way of leaving the process, other than through the usual route. Here the user can press a cancel button on the toaster to cause the toast to "pop" without regard to time, temperature or condition of the bread.
  Many procedures are aborted due to error or some intervention. These events should be represented or summarized on the activity diagram when they are important to the client or to the development. These could be shown using as part of the normal flow with conditionals, but the means shown makes diagrams that are generally more clear.
- The grey line shows a partition. Here the partition is between events and process that occur on the outside of the body of the toaster, and those that happen inside the body. Such divisions can be geographical, by actor or by any other criterion that will make the design and diagram clearer.
- The entire diagram is enclosed in a frame. Frames are used so that diagrams can be labelled and included in global diagrams, such as "Make Breakfast". We will look at this later.

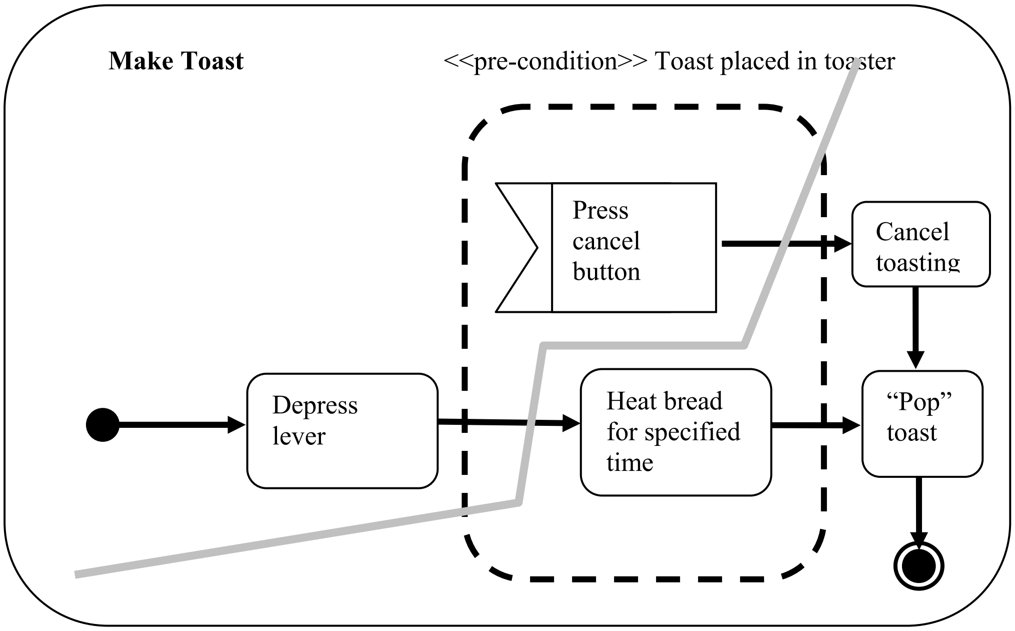

Make Toast
<<pre-condition>> Toast placed in toaster
Press cancel button
Cancel toasting
Depress lever
Heat bread for specified time
"Pop" toast

Figure 5-3 Activity Diagram for Making Toast

We can put conditional flow, decisions, loops and other process requirements on this diagram to make it suit the circumstances.

5.5.2 Interaction Diagrams

Interaction Diagrams describe how parts of the design work with each other. We will look at three different diagrams to handle this: sequential, time-based and communication diagrams. Sequence Diagrams describe the sequence of steps in an interaction. Timing Diagrams describe the timing of the interaction. Communication Diagrams describe the structure of the interaction links.

5.5.3 Interaction Diagrams - Sequence Diagrams

As with most UML diagrams, and as with Use Case diagrams in particular (Section 4.5.3), the sequence diagram uses the concepts of actors and their interactions with the artifact. Here the interactions are described as a sequence of tasks with a time order.

The drawings are set up with the actors along the top edge and these having vertical dashed lines coming down from them. An interaction is shown as a set of sequential communications between the actors, represented by arrows between the vertical lines. The order of the arrows from the top of the diagram down shows the sequence of the communications in the interaction.

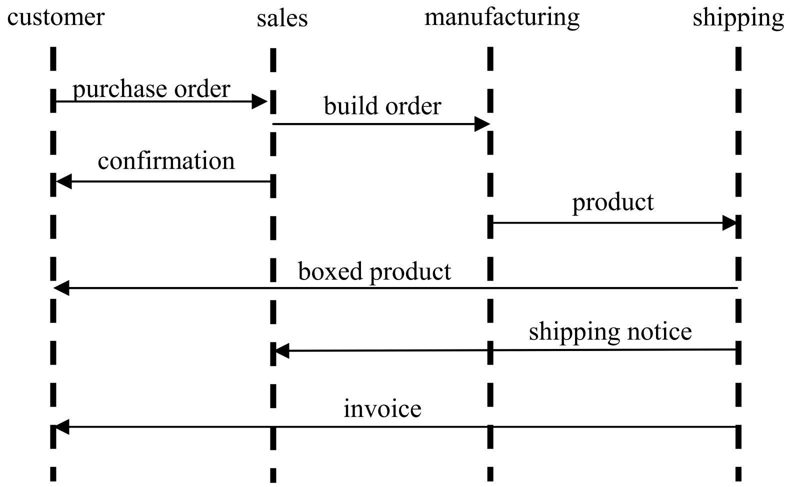


Figure 5-4 Sequence Diagram for Order Processing

These diagrams work for all sorts of interactions. For example, Figure 5-4 shows what might go on in filling a customer order. The process starts with a purchase order from customer to the sales office. The sales office sends a build order to manufacturing and a confirmation to the customer. When the product is finished it is sent to shipping where it is boxed and sent to the customer. Shipping also informs sales that the order is complete and sends an invoice to the customer so payment for product will be sent.

A similar type of diagram could be used to show a TCP/IP[5] interaction, a signal sequence in parts of a complex circuit or how a user would register for a web-based seminar.

Complex interactions can often be broken into smaller interactions that can be represented by parallel frames across the Sequence Drawing. Frames are described at the end of this UML section, but are basically boxes which are a higher level view of a set of detail.

You can add options, conditionals, alternatives, loops, breaks from loops, error processing, parallelism, critical regions and others as required to make your Sequence Diagram represent what has to happen (or not happen) in an interaction.

These Sequence Drawings allow us to determine what parts of a design are active at what times in order to interact with the other parts.

5.5.4 Interaction Diagrams - Communication Diagrams

Because UML was intended as an object-oriented software tool, Communication Diagrams consider interactions

There can easily be confusion here. Communication, in an Electrical / Computer Engineering sense, is what is called Interaction in UML. The Communications Diagram in UML is a Connection Drawing or System Overview in engineering. The UML terminology is used here.

[5] TCP/IP is a communications protocol that is used in network communications, including the Internet.

between software objects. However, this has easy extension to physical objects and thus is useful to describe the communication links between physical objects as well.

Figure 5-5 shows a high level Communication Diagram for a Personal Computer, monitor and printer.

It is important to note that the physical linkage and protocol used in interaction is not the same as the concern about the information content passed, which is shown in the Sequence Diagram. The sales and manufacturing data areas of Figure 5-4 might reside on the same computer, or could be miles apart with an internet connection between.

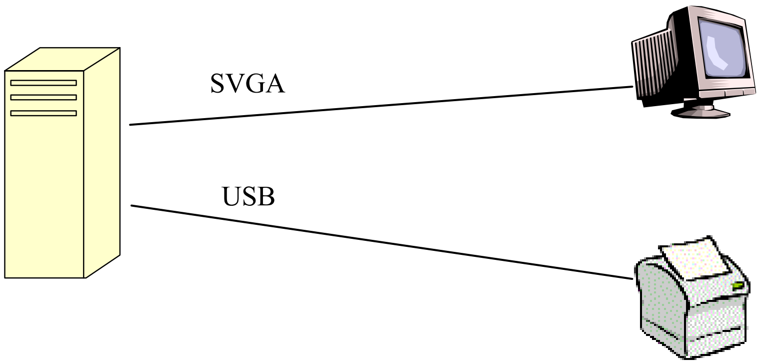


Figure 5-5 Simple Communication Drawing for a PC

5.5.5 Interaction Diagrams - Timing Diagrams

Timing Diagrams are just when you might expect. When there is a complicated requirement for the signal or interaction timing, it can be specified using a timing diagram – something that looks like an oscilloscope trace with time along the X axis, and something else (voltage, operating mode, state,…) along the Y axis. An example is in Figure 5-6.

Simple timing requirements could be put right on the sequence diagram.

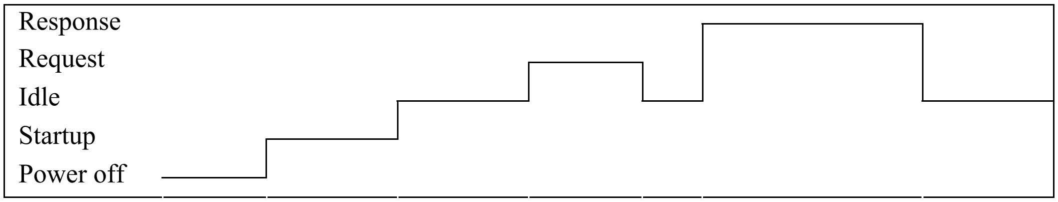


Figure 5-6 Timing Diagram of Simple Device Communication

5.5.6 Interaction Diagrams - System Interaction Diagrams & Frames

Timing, Sequence and Communication Diagrams (and others) can be grouped together in chains, each diagram represented by a Frame. An example Frame is shown in Figure 5-7. These are simply boxes with label information in the corner and content in the middle. The label information identifies the box uniquely (gives a name for this particular content). It identifies the type of box (Sequence Diagram, Timing Diagram, Activity Diagram, etc. – here sd for Sequence Diagram). It also specifies any parameters required by the content, such as the user's Personal Identification Number (PIN) in the figure.

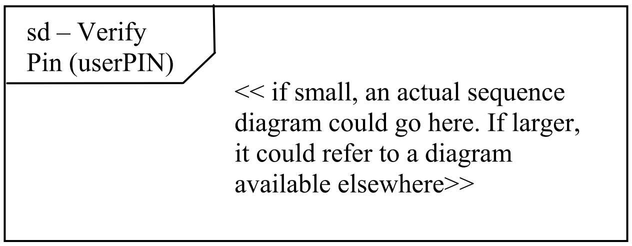

sd – Verify Pin (userPIN)
<< if small, an actual sequence diagram could go here. If larger, it could refer to a diagram available elsewhere>>

Figure 5-7 Frame for a Sequence Diagram

The chains link and group the Frames. The chain should represent a Use Case or similar. An example for an ATM cash withdrawal is shown in Figure 5-8. Here we chain together a number of sequence diagrams that get the user PIN, verify it, and allow the cash withdrawal if the PIN is correct.

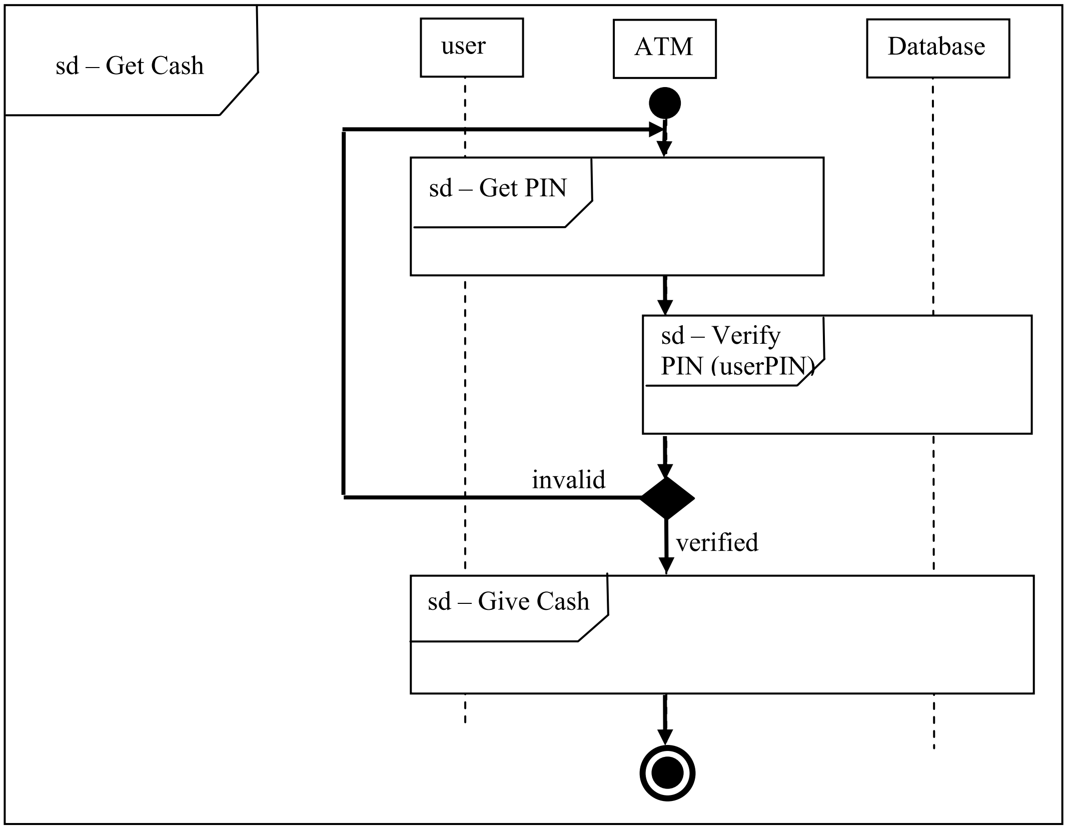


Figure 5-8 Sequence Drawing for ATM Cash Withdrawal

The figure also shows the use of the solid ball for the starting point, the double ball as the exit point, and the use of a conditional diamond. The GetPIN sequence involves the user and the ATM, whereas the Verify PIN sequence involves the ATM and the Database. All three are involved in the Give Cash sequence. Each of these sequences would be further defined in its own diagram.

5.5.7 Interfaces, Ports and Protocols

An interface is a requirement a module has to provide or to procure information from outside the module.

A port is an interaction point for a module; the port is what the outside world sees as the place to send information to and get information from the module. It has a specific information-exchange purpose and has associated with it an interaction method that fulfills that purpose.

UML2 differentiates between the interactions where the port for a module provides a service and the interactions where the port for a module requires a service in order to operate. A provided interface by a port is shown with a ball; a required interface of a port is shown with a cup. The way in which the interface transaction is handled is called a protocol.

A message protocol, such as TCP or that in Bluetooth, is often described using a sequence diagram.

An example of a protocol, a provider interface and a required interface is shown in Figure 5-9. The inventory system provides a port for access. The cash register requires an interface to the inventory system to record changes (purchases, for example, will deplete the inventory). The protocol used in this case is the Icomm protocol, which might be a proprietary method built up on TCP/IP, the usual local area network communications protocol, for example. The name of the protocol is written below the interface.

Although the word protocol is usually reserved for the software message passing method, it is useful to extend it even down to as low as the hardware signal level where we might specify maximum noise tolerance, signal rise and fall rates, voltages, currents and so on.

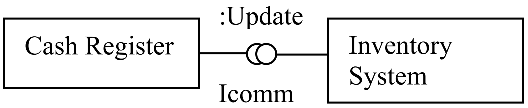


Figure 5-9 Example port/port/protocol

5.5.8 State Machine Drawings

UML also includes the concept of a State Machine. Electrical and Computer Engineers understand state machines. If you have not encountered them in your Digital Electronics course, you soon will.

In Digital Electronics we usually make a distinction between the “data path”, the movement of the information we are concerned about, and the “control path” which controls the sequencing of the movement. In UML the data path actions and the control path information are intermixed. The UML State Machine Drawing is more like a flowchart and the states are not nearly as carefully defined. A solid ball shows the point of entry and a solid ball with a ring around it is the exit point. Figure 5-10 shows a simple State Machine Drawing for a beverage machine.

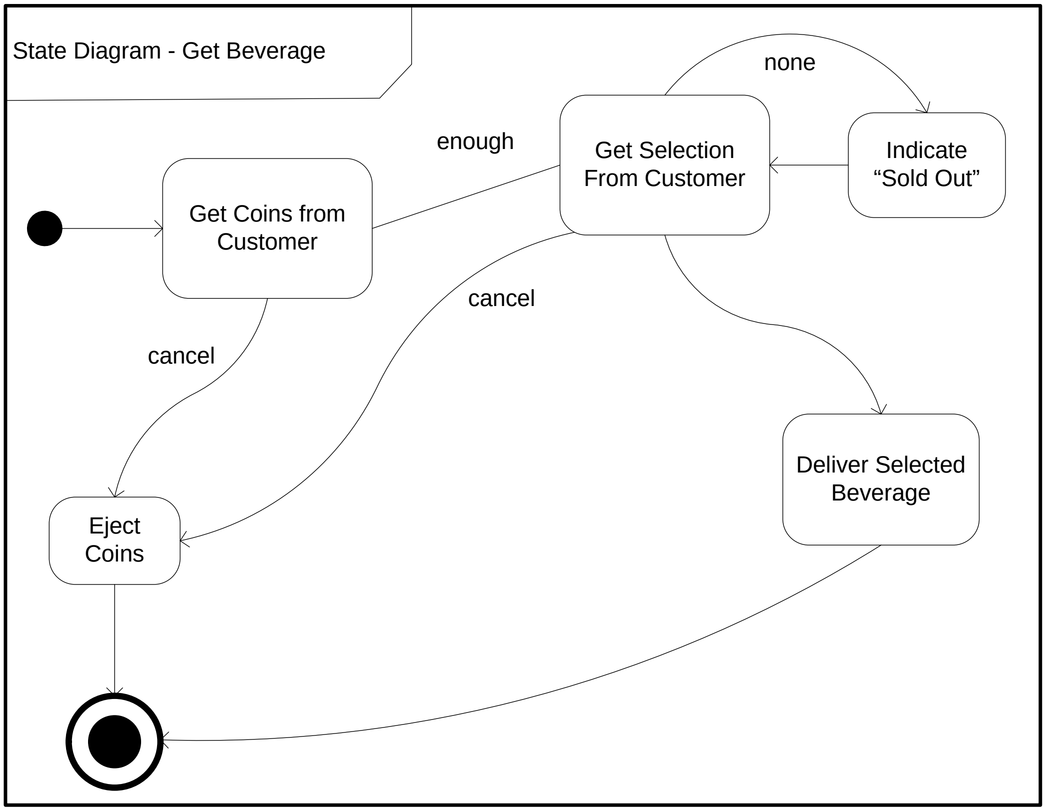

State Diagram - Get Beverage
Get Coins from Customer
enough
Get Selection From Customer
none
Indicate "Sold Out"
cancel
cancel
Eject Coins
Deliver Selected Beverage

Figure 5-10 Beverage Machine State Diagram

5.5.9 UML Not Covered Here

UML was developed as a method of producing software system representation, specifically object-oriented software system representation. There are a number of diagrams relating to classes, class interaction, class inheritance, class relationships and other class attributes that are not covered here. There are many areas of further definition that also have not been described. All of these, and the many UML extensions, are of great use in software development and are left for study should you decide to concentrate further in that field.

5.6 There's Too Much Paper: The Document as a Tool

"Goal document, Requirement Spec, System Spec, Progress Reports – I never have time to get any work done!" This is a common complaint from starting engineers. But ignoring these communications leads to problems. Parts won't fit; weeks will be spent on valueless work; the project will be cancelled because it is so far over budget and over time.

Properly done, these documents are really simply a *reporting* function. If you know what the project goal is, then you are just taking some time to write it down, or actually to copy it from where you already recorded it[6]. If you know all the requirements, then writing the requirements spec is almost trivial. If you are in reasonable command of your writing abilities, then this is not a major effort. Technical "writing" is different than the writing you learned in high school, but can be and should be mastered by the engineering student.

Summary: Paperwork often seems an impediment to getting "real" work done. Properly done it provides the necessary means to share information unambiguously with the rest of the team and to keep stakeholders onside.

What is confused by these engineers is the work of reporting and the work of generating the information to be reported on. Often the fact that you are have to organize the information for reporting will reveal gaping holes in your work, holes that take time to fill. If you do not fill these holes, if your background work is insufficient, then the document will not be complete. A superior will send it back to you for revision because it is not ready for a client.

A Document as a Design Problem

We know that a basic formal design stream has Goal, Requirements, Specification and Execution. A document is no different. If you take the time to plan, the document will be done more easily, the work can be more easily divided and there is less chance of the document coming back for major revisions.

First the Goal. Why are you writing this document? If the answer is "because my supervisor asked for it", then it is not likely that you will produce a good document. Right from the start you need to know who the document is for, and what its purpose is.

Documents with more than one distinct group of readers

If your document is for several distinct groups of people, perhaps you should think about designing your document with sections appealing to these distinct groups, rather than trying to produce a document that will please all everywhere. Another alternative: Produce two documents.

Example goals:

- ➢ This document is to provide a closed set of requirements to the client.
- ➢ This document is to detail a communications protocol between the input processor and the main processor.
- ➢ This document is to provide my supervisor with information about my progress on the project for budgeting purposes.

Next, the Requirements. You know who your document is intended for and what its purpose is. Now outline what will fulfill that purpose in the eyes of that audience. If this goes to your supervisor, what is it that your supervisor is most interested in? If it goes to the client, what does the client want to see?

Typical Concerns

Supervisor	**Client**
Task Timing	Expected functionality
Budgets	Done on time
Coordination	Done on budget
Human Resources	Confidence / Trust
Non-human Resources	Post-delivery support

A weekly progress report given to your supervisor is likely to include recently accomplished tasks, tasks with which you are presently involved and that you will be involved with shortly, and about the problems you are running into in

[6] Typically your engineering notebook. Don't have one? Get one!! (This is traditionally a paper notebook, but electronic forms are becoming more common. Either way, be careful to protect the information against loss.)

conducting the work. What else might your supervisor want to see? Dates of start and completion. Expected times of completion. Recommendations. Longer term concerns that are going to require preparation starting soon.

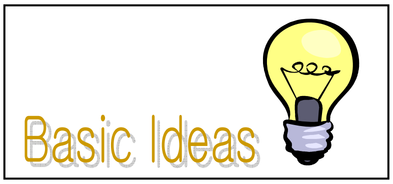
Also consider the document identification that is needed. Your name and the date at minimum will be required, along with the name of the document type. Are other identifying fields required to differentiate this document from other similar documents?

Length is also a requirement. Does your supervisor want to read a weekly report that is 100 pages long? Not likely. Will your supervisor be happy with a weekly report that is only a few lines long? Maybe, but likely not.

Specification. Here we just have the format of the piece. For a document, a fairly quick step.

Execution. Here is where we pick up time. We know what the requirements are, we should know what to record for each requirement [and if we don't, this is not a problem with the document, it is a problem with our planning!!]. We just have to fill in the blanks.

Regarding execution, most new engineers feel that if they are doing a writing exercise that it somehow involves only writing. This is an easy mistake. But think about what makes a good textbook work. Look at some product specifications or a user's manual for a DVD player. What do you see? Pictures, tables, graphs, figures, bullet lists,… . This is what engineering "writing" is about. Put the information into a form that makes it clear and concise. Don't worry about trying to make it fit the mould of an essay on "what I did last summer"!



Chapter 6 System Design

Here we are at Chapter 6 and we still do not have a design! We are about to change that.

It may seem that too much effort has been expended for no lines of code and no devices on order. When a system-engineering-knowledge-deprived supervisor measures your progress in these terms you may start to feel the effort was not worth the results. But persevere; our careful development of the requirements will help ensure we will select an appropriate design. Careful selection of the design will help ensure we end with an appropriate artifact. The cost of the development escalates quickly once we commit to a design, so careful early development will decidedly have its rewards later.

Summary: The designer, at the System Design step, proposes a design which will meet the requirements. The design is expressed as a set of modules that will be individually developed and tested. The system tests are also determined at this stage – these are the tests that will verify that the final product meets the requirements.

We are still in the first of the three stages of the design process (see Figure 6-1). In the System Design step we determine how we are going to implement the design. In this step the design is broken into a number of smaller pieces (called modules), each to be individually developed and tested. We determine what the modules will have to do, the implementation method and the module characteristics required to meet the system requirements. We also, at this stage, determine how we will test the design to make sure it meets these system requirements. In subsequent steps, we build and test the modules, bring these modules together into a system and test then deliver the system.

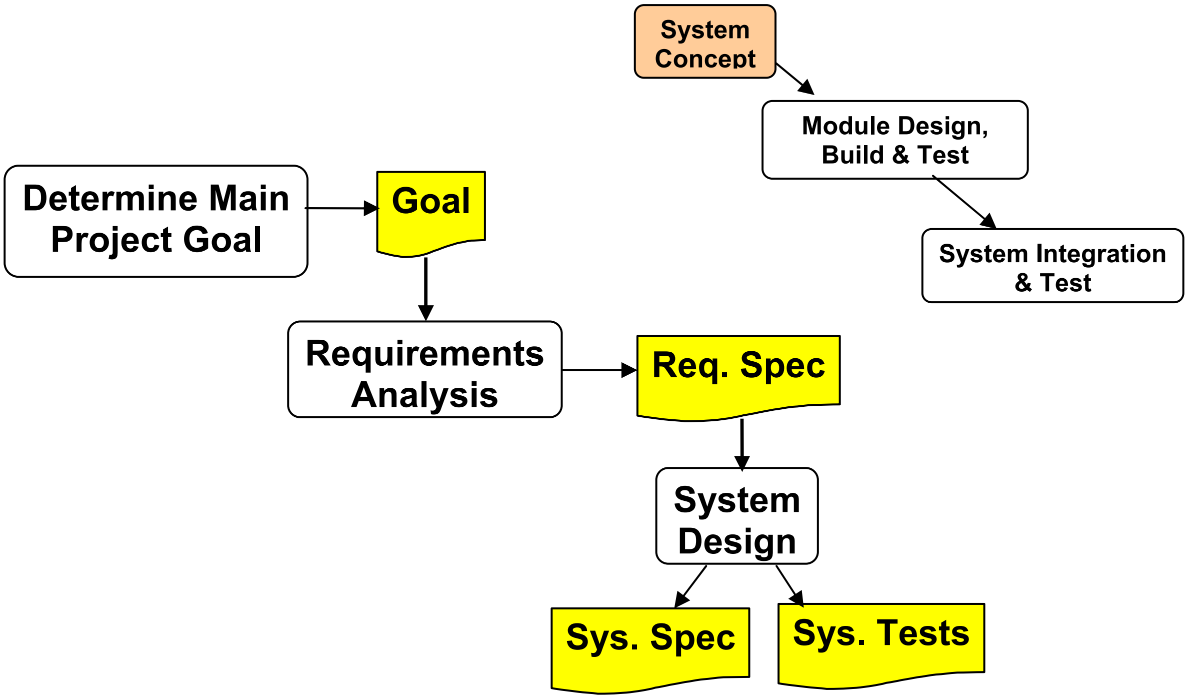


Figure 6-1 Design Process Step 1

In a large system, these modules may themselves be systems. In this case the implementation is not expressed but the requirements of the module are.

The output of the System Design process then is

- A set (or system) of module definitions and module requirements such that the modules together will fill the requirements of the artifact. This, plus any necessary background and other descriptive material, comprise the System Specification. In the case where the module is to be executed by the design team, the definition will include implementation methods.
- A set of tests to perform on the entire system once the modules are integrated. These tests will show and should attempt to guarantee that the implemented artifact will perform as required by the Requirement Specification.

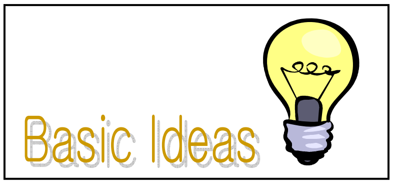
If you look at any consumer goal you quickly realize there is more than one marketable design. Attaching paper together into groupings, for example, you have clamps and clips, staples and binders, glues and thread, and this list is by no means exhaustive. These solutions exploit the Objectives in different ways and meet different consumer needs. Some are low cost, some are long lasting, some are quick to use, some are permanent while others are deliberately temporary. Similarly our System Design step can result in different solutions and we are going to have to choose a solution to move forward with.

This chapter and the next few chapters deal with the System Design process. In this chapter we will first look at some problems with dividing the project into a set of modules, the block diagram as a means of representation and what a System Specification might include. Finally we will look at finding a System Design in overview. We will find that the search is the iterative application of three steps: Research, Brainstorming and Evaluation. Each of these three steps, and some tools to make them effective, will be looked at in the chapters following this one. Chapter 10 then explains project management, including budgeting of resources. Chapter 11 focuses on risk determination and management. Resources and risk are other factors that will enter into the System Design process.

Often the system design process is the scariest part of design. In a complex project there is never absolute assurance that the goal can be reached, particularly with any one design that you might commit to. The tools in this chapter should help you reduce the problems to manageable, solvable parts.

If, at the end of the process, no suitable design is reached, first back up – is there a more general problem you can solve and thus avoid this impasse? Perhaps the bridge you cannot design and build in the required time is not the only way to cross the river.

If the solution still remains elusive, perhaps this is the point to cut your losses, with expenditures still low.



6.1 Potential Problems

Determining a design by dividing the design into modules is not a straight-forward task. Besides the problems of meeting the requirements, certain problems arise simply because you are dividing the project up. Recognizing that these problems can occur means actions can be taken to recognize and compensate for them.

Problem 1	Duplication of Resources	Our design may have software routines or hardware pieces that could be shared but are not because we have broken the design down such that each module developer will produce the same resource
Problem 2	Communication between modules unnecessarily complex	Our modules must communicate information to each other. It may be that we will make the communication more complex than either communicating module requires it to be

Problem 3	Module division at points where signals are very sensitive or very numerous	Module division can be done where the interface signals are very sensitive, have very fast transmission requirements, or are very numerous. Division at another point could reduce these problems.
Problem 4	Module structure unnecessary and costly	Some structures caused by division into modules are very costly to build, maintain and manage.
Problem 5	Incomplete analysis before proceeding	Sometimes modules go into detailed design and production before the system design is complete (such as building the box first). Some modules could prove impossible or impractical to complete, but the design goes ahead with these modules because of incomplete analysis.
Problem 6	Biased decision-making	Sometimes modules are designed around familiar but inefficient technology and sometimes around new technology simply because it is new. Either type of decision is potentially not the best.

6.2 Block Diagrams

This is the fundamental tool for describing system designs. The blocks are the modules we have been considering. Here are the attributes of a block -

- a function or set of closely-related functions
- an "information" connection to at least one other block
- possibly some inputs and outputs from/to the outside world

On the block diagram we describe briefly the major aspects of the inputs, outputs and information connections between the blocks. These aspects could include timing, sequencing, bandwidth, impedance, signal voltage and/or current, power requirements, number of connections, and communication protocol. There should be enough detail to make it evident what the block does, although this will still fall well short of fully describing the requirements of the block.

Summary: Block Diagrams are the fundamental tool for describing systems.

Applicability: System Design

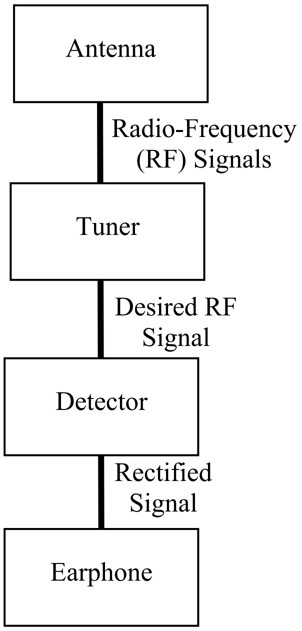


Figure 6-2 Block Diagram of a Crystal Radio

We divide a system into blocks at the physical interfaces and where the information is easily described. So divisions would not be made in the middle of a filter or a sorting routine; the filter or the sorting routine would be in a single block. Being able to name the function of every block and the information on every connection is a good indication that the blocks of the block diagram are correctly assigned.

A simple block diagram of a crystal radio is shown in Figure 6-2. There are four blocks that successively filter the incoming radio frequency signals until they reach the earphones where the audio part of the selected signal produces sound in the earphone.

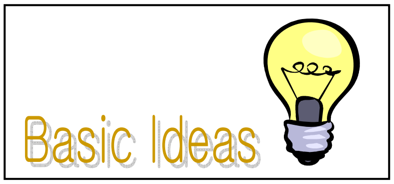
Many diagrams are extensions of the basic block diagram. A circuit diagram or a class diagram are specialized forms of block diagrams.

It is important that it be understood that this is an implementation decision being recorded in the block diagram. The order of the blocks, their functions and their interfaces are determined in the diagram. By using this design you are precluding any advantages offered by other solutions. You should only move to implement the design after careful evaluation of alternatives.

> When engineers design things on the back of a placemat, block diagrams are the means of communication. These are expanded and detailed as the ideas come into focus, and as more and more of the requirements are met

Block Diagrams of Larger Systems

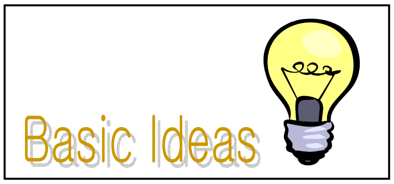
Larger systems are represented with a hierarchy of block diagrams. The higher-level diagrams have blocks that represent systems. Each system would have one or more additional diagrams associated with it where there was further refinement of the blocks. The lowest level blocks, the implementation level, should represent a single physical or logical unit with usually a single technology in use.



6.3 System Tests

The system design that we move forward with is one that meets all the requirements of the project, and gives certain expectations of the Objectives. These are attributes we can verify when we integrate the modules and complete the system. For example, we may have to meet a certain size requirement and the client also wants the weight to be low. Our system design will meet the size requirement, and we will have determined a weight that we should be able to meet with this design. We now can produce a System Test for this design that includes verification of the size and of the weight. When we complete the System Test we can be sure that we have met the requirements, and thus the artifact should be ready for delivery.

Why create the System Tests now? Doing so has several advantages. First, we may need test fixtures or extra software or test equipment or additional development team members which we must include in our budget and schedule. Second, we may need access points or test connections or integrated routines as part of the artifact in order to allow the testing. Third, knowing that these tests must be passed on system integration will make the module designers more aware of what parts of the design are important and what parts less so.



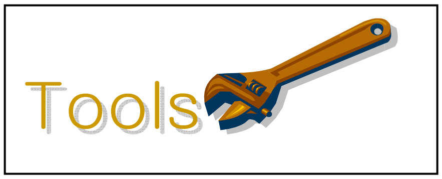
6.4 What is a System Design?

When we have selected the design to move forward with, we have determined that a certain set of interconnected modules will perform as a system to produce the required results. To ensure that these individual modules will work together properly, we have to issue requirements for each module. This, in fact, is generating a set of requirements specifications with associated metrics (measurable values) that mean that the module designs will perform as expected. For large projects, this becomes a smaller project for some other group. For small projects, the design specifics of the module will be determined by the design team itself, and the implementation method will usually be specified.

The system design will take each requirement and determine which module(s) are involved in meeting that requirement and to what extent. An overall weight limit, for example, will have contributions from all physical modules of an artifact. An electrical high-voltage safety requirement would have to be met by all components that used voltage classified as high-voltage. An input impedance limit on a specific external connector would be a requirement only of the module with that connection as a port. These restrictions form part of the requirements of the modules.

Also requirements of the modules are those requirements generated by the module-to-module interfaces. In order that the modules work as expected to perform their part of the overall system functions, they will have to meet the input requirements of the modules they output to and the output requirements of the modules from which they get input. Or, more simply, the modules that are connected to each other have to match. This can mean matching voltage, current, frequency, protocol, data format, physical connection and many other attributes.

Besides the module requirements, interconnections and general implementation, the System Design should also include some demonstration that the design will meet the System Requirements and goal. This can be done through calculations, high level simulation or a walkthrough of the Use Cases.



6.5 The Idea Generator

At every stage in a development you will encounter problems and have to solve them. When we are generating Use Cases or trying to list all the stakeholders, or determining what modules will go into an artifact or trying to minimize a path length to reduce noise sensitivity in a circuit, we are going to have to generate ideas and then accept or reject them based on their usefulness and reward. In this section we will introduce a process that will help you go through this process of finding ideas, particularly where the various alternatives are complex and multi-facetted. It is a three step process, that is usually done iteratively – we go back to previous stages as necessary until we are convinced we have found the most appropriate idea for our circumstances. Figure 6-3 shows this.

Summary: The idea generator is introduced. It is a three step process – research, brainstorming and evaluation – that is the backbone of the creative process.

Applicability: Implementation selection and other areas where choices must be made.

The first step is research. If we are already fairly well-versed in the problem area, this step might not be necessary. Usually, though, there will be some aspect about the situation that is unfamiliar to us and this is where we want to do some work. If the problems we are looking to solve are similar to solved problems in other fields (such as architecture, biology, medicine, dentistry, child-rearing), it is generally worth looking into those areas too.

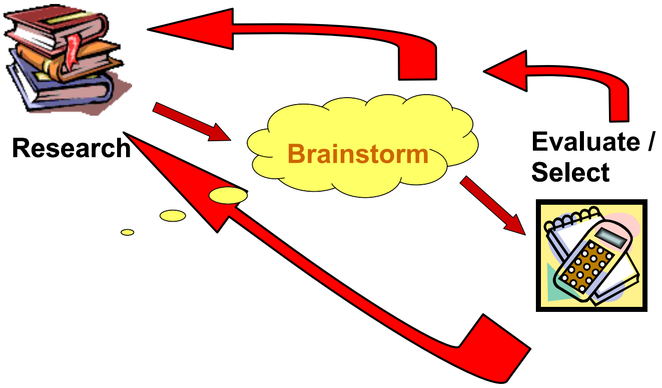
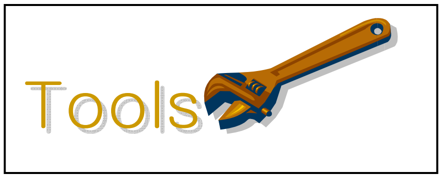


Figure 6-3 The Idea Generator

The middle step is what we normally associate with idea generation: the brainstorming. Note, however, that it does **not** include evaluation and it does **not** include research. More on this later.

Finally we evaluate the ideas that we found in brainstorming. Evaluation too soon may cause us to select a solution that is less than best for our situation, and may otherwise inhibit the formation of stronger ideas. If the evaluation doesn't give us a reasonable solution, we need to go back and repeat the process, using the knowledge we gained in this loop.



6.6 Keeping the crazy

Over the course of development work you will be required to do various amounts of formal reporting, formal analysis, formal testing and the like. However, the essence of good design is creativity. The best designs are not turn-the-crank copies of what already exists, but innovative combinations of technologies that fit the actual requirements of the situation very closely.

The key to the generation of good ideas is brainstorming and the key to good brainstorming is the generation of out-of-the-ordinary and off-the-wall ideas that can be reined in and used. To create these good ideas means that you must dare to be absurd, you must dare to be wrong, you must dare to be a bit crazy.

6.7 Summary

We are moving from a set of requirements to a system specification and associated system tests. It is an iterative process, involving research, brainstorming and analysis. These three will be looked at in the next three chapters. The best system design found will then go on to be implemented (unless analysis shows that no idea is suitable. This could be for budgetary reasons [Chapter 10] or because no design can be found that meets the technical requirements.)

Chapter 7 Researching the Design

7.1 Getting Information

Information is available through a large number of sources. Often engineers will feel they will be showing their incompetence by seeking information, but accomplished engineers are past this and welcome input from any source they can find to help them better their designs.

Here is a starting list that you will see is similar to the list for getting requirements information (see Section 4.2). As for the requirements: For human resources, plan ahead – prepare your questions and their follow-ups; determine what information it is that you want to leave the meeting with; keep the meetings professional.

Books

Most significant traditional solutions are found in books.

Journals and periodic publications

Recent information, usually more carefully screened, can be found in journals and periodic publications. Because there is less time elapsed to go to print, they tend to have newer information than is found in books. These sources are useful for finding the newest products and methods.

Online

This is the best source for information about current parts from manufacturers and similar information. Other types of information may be unrefereed and suspect.

Client

The client should have provided you with needs and wants, but may also be a good source of hints on how to provide the solution.

The client should also be asked permission before going to others in the organization. How you ask for these permissions and the reasons you give for asking for them should be carefully thought out.

Other Engineers in your company

The same engineers who have worked on similar projects that you consulted about the requirements may again be able to guide you to some good solution elements. They will also know what type of techniques the company is familiar with.

Similar artifacts

This is reverse engineering – which we will look at later in this chapter

Users, others in the field of use, people and products that interface with this artifact

These are the people you will already have approached for requirements information. Often they will offer implementation suggestions at the same time. Sometimes these solutions show incredible insight and sometimes they miss fundamental requirements or constraints and are not directly useable.

Often a single meeting with these people is not enough. The first meeting often just gets them thinking and a second meeting will often allow them to give you the results of this thought.

7.2 Reverse Engineering and Keeping Your Eyes Open

As an engineer you should be constantly looking at products and processes. School will not give you all the tools and techniques to use on a real-world engineering problem. There is too little time and the focus is often too narrow. Outside the classroom there are a wealth of systems – not only electronic and computer, but mechanical and biologic, social and religious, natural and human-made. Bringing all your knowledge of these systems into your arsenal will make you a master of problem-solution.

In addition to seeing **how** things are done, a proper reverse engineering exercise will tell you **why** they are done. You will, however informally, walk through the design process, but trying to trace down the thinking and decisions of another. These invaluable lessons teach you more about the specific design environment than you could ever learn, outside of doing a design yourself.

Summary: Reverse Engineering is the formal or informal method of finding out how something works. The method and some tools used in Reverse Engineering are introduced.

Application: Analysis of competitive product; adding to a personal repertoire of methods and processes; engineering amusement. Also useful in determining requirements.

In addition, you will exercise the skill of situation analysis, of walking in the shoes of the users and other stakeholders. When you turn to creating your own designs you will have to understand your stakeholders and their needs and viewpoints in order to determine the parameters that control your design choices.

But quite aside from all that, reverse engineering is about a mystery story waiting to be solved. This is what makes engineers discuss software strategies over coffee (and beer); it may be what gives life to the phrase “thinks like an engineer”.

Scott Adams, in The Dilbert Principle, expresses it as follows: “No engineer looks at a television remote control without wondering what it would take to turn it into a stun gun. No engineer can take a shower without wondering if some sort of Teflon coating would make showering unnecessary. To the engineer, the world is a toy box full of sub-optimized and feature-poor toys.”[14]

The underlying first requirement for turning a television remote control into a stun gun is to understand the television remote control. To understand the remote control, one must understand the reason a television exists and the remote control’s place in that reason.

Reverse engineering works on processes and software as well as physical products, but for ease of explanation we will use the word ‘product’ here.

As might be obvious upon short reflection, the steps in reverse engineering are almost the same as for development of a new product. Because there is a close parallel, many of the tools we will examine are applicable to both reverse engineering and new product development. A suitable set of places to concentrate our efforts are:

- Determine the product goal.
- Determine the product requirements.
- Examine the parts and interconnections and establish functions of these parts.
- Explore other avenues where product information can be found.
- Use analysis tools to model the product and uncover methodology.
- Solve the mysteries

We will look at these places of concentration in the next sections.

Of course if the plan is simply to find out how something was done you will direct your efforts to that aspect and stop when you have determined what you sought. On the other hand, if you are interested in assessing a competitive product, you will tend to use considerable effort in exploring every piece of the product.

7.2.1 Reverse Engineering Step 1 – Understanding the Artifact's Goal

With very few exceptions, and perhaps only one[7], every artifact is part of a system with a broader goal or purpose than the artifact alone. A hockey player's helmet protects the player's head. Staying protected is a need of the hockey player in order to contribute to the team for an appreciable amount of time.

The goal of a resistor is to limit the amount of current it passes for a given voltage. The amplification circuit the resistor is a part of has a need of such a restriction in order to correctly amplify the signal. The amplification circuit is required by the digital receiver in order to drive speakers. The digital receiver is part of an entertainment system used to satisfy the desire of a consumer for music and of the vendors for fulfilling that desire.

To properly reverse engineer something (an artifact) we must understand where the artifact fits in the larger system. This goal is something that we will also deal with when we create a new product. It must be expressed in a way that does not imply a specific implementation.

Just a note about the formality of this process: Do you always do reverse engineering step by step as presented here? Of course not!

Sometimes you are called upon to analyse and report on a competitor's product. In this case you want to walk through each step carefully. More often you see something that interests you technically and just want to figure out how it works. In these cases, knowing the steps will create a logical set of links between product features and perceived requirement without actually going through the steps in a formal way.

Consider the television remote control. The goal of a television remote control is to remotely control the television, that is, to make it unnecessary for the user of the television to leave the comfort of his/her couch and to walk the several metres across the room to the television to make channel adjustments, and then have to go all the way back to the couch again. Doing so might mean the loss of a few seconds of time that could be spent viewing a Seinfeld rerun, or even worse, could result in the user completely walking away from the television and doing something else.

To avoid the possibility of you leaving the television, the television manufacturer includes a remote control as part of the television system.

Notice that the specified goal, to remotely control the television, does not specify how the control will be made. Even further specifying the requirements will not do so, although further specification may eliminate some potential ways of producing the control. If the requirements state that the control must be easily lost under a seat cushion or must be able to be left in the kitchen if you go for a snack, having a remote control that is wired to the television directly will no longer be a possible solution. Analysis of these further requirements are the next step in reverse engineering.

7.2.2 Reverse Engineering Step 2 – Requirements Analysis

The goal states a broad aim of the design. It states where the artifact fits in the encompassing system and it does so in fairly high level, simple terms.

At the Requirements Analysis stage we look at the device and determine what characteristics are needed to meet the goal and its secondary requirements that make it useful in the system.

Looking at our television remote control again, we see that it will have size and weight contstraints, and certain functionality such as being able to adjust volume and change the channel. We probably also would require it to be wireless. We probably want it to work when pointed in the general direction of the television, and not precisely at a specific place. The buttons should be easy to press and arranged in some sort of obvious order. It should be able to be used in the dark or by people with bad close-vision eyesight, without having to count buttons – the shapes and positions of the buttons should give a clue.

[7] This "one" assumes that there is only one universe and that is the one all-encompassing system.

7.2.3 Reverse Engineering Step 3 – Examine the Parts and Interconnections; Determine Function

Personally, I consider this the fun part and it is what most people equate with reverse engineering. This is where you pull the artifact apart, as much as possible, and examine the pieces. For physical product you use screwdrivers, saws, crowbars and the like. For software, this might involve reverse-compilation tools[8].

Form and Function

One of the important differentiations to be made here is the difference between form and function. Function is what the part does. Form is how it does it. Typical functions are "calculates the square root", "changes the lens opening" and "verifies that the part works before packaging". Typical forms can be "software subroutine", "motor with cam attachment" and "verify the data on the CD after burning".

The form can also generate unwanted functions. Some of these are electrical noise, audible noise, heat, vibration and places where fingers could get pinched. The functions of some parts of the product will be to take care of the unwanted functions of other parts. A heat shield, for example, would be one of these, used to protect the user from the unwanted excess heat generated by a heating element.

Table 7-1 is a table of some categories of the functions you might see:

Table 7-1 Some Functions

Function Category	Example Function	Form that Executes Function
Main Product Goal	Adjust the volume	Volume buttons & hardware to transmit the request
Support Functions	Error-correcting infrared transmission	Processor and output hardware
User Safety	Use only low voltage power	Hardware implementation
Ergonomics	Portable	Case, hardware implementation
Purchaser appeal	Baseball motif for case and buttons	Case
Simpler manufacturing / testing / distribution	One screw holds case together	Case / screw
Environmental concerns	Recyclable plastic case	Case

Not only are the part functions themselves important, but also important is how these parts interconnect to the other parts. These interconnections may pass information or power, or they may just be physically supportive. Sometimes the interconnection will be the chief clue as to the purpose of one of the connected pieces: The chassis of a car, for example, is the return path for the electric circuits, something you would only determine by looking at the connections of some of the lights and of the battery to the chassis.

7.2.4 Reverse Engineering Step 4 – Find Other Information

The product itself can give you quite a bit of information, but if you don't have full access to the product, if you can't tear it apart or if tearing it apart doesn't yield all the answers you need, you have to look elsewhere. There are a number of sources of information that will aid your product dissection and that will give you information in addition to what you gain in product dissection.

Most products come with Users' Manuals, installation guides and so on. Often these will give you strong clues as to the product implementation. Furthermore, many products have online repair information, including parts lists and assembly diagrams, operational information, operational theory and the like. More general information can be found on websites such as www.howstuffworks.com or in books of a similar nature.

[8] Note that, according to some license agreements, reverse engineering of software is not allowed. Disassembly will often void warranties of products.

You should never underestimate users as an information source. Anyone who has performed the process or used the product will be able to give you insight into its internal workings. Certain procedures during usage will be dictated by the way parts have been designed.

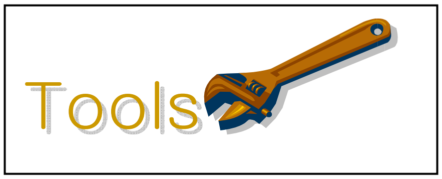
For example, inside my power grinder you will see two buttons, one on the side and one on the handle, that physically interact with each other. Why they do so might not be as quickly determined without the knowledge that you must press the button on the side, then pull the trigger on the handle to turn the grinder on. This is clearly a safety feature. It is much easier to understand the interaction of the buttons when you know beforehand how the grinder is used.

7.2.5 Reverse Engineering Step 5 – Use Analysis Methods

There are many different analysis methods one can use in reverse engineering a product. We will look at a few here:

- Parts Lists / Parts Descriptions
- Snapshots and Use Cases
- System Function Lists
- Electrical / Power Flow Analysis
- Function Analysis Systems Technique (FAST)

Again, it is important to realize that these same tools have use in the design and modification of new products as well as in reverse engineering existing products.



Parts Lists / Parts Descriptions

The parts list of a product is known as the Bill of Materials (BOM). A simple parts list, often omitting less relevant items like fasteners[9], will often be enough for a reverse engineering exercise. On the other hand, an in-depth reverse engineering exercise will often include a set of part description records, such as in Table 7-2, for every part.

Part # ______________________________

Part Name __________________________

Qty ________________________________

Function ____________________________

Weight _____________________________

Finish ______________________________

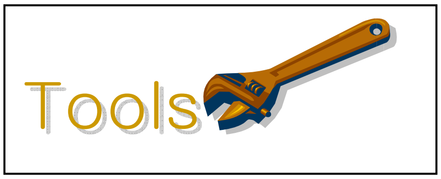
Dimensions __________________________

Manufacturing Info

Table 7-2 Part Description Record

The description shown is one reasonable for mechanical parts. Different information would be gathered for computer modules and for electronic parts.

[9] although fasteners can take time to determine and select during design!



Use Cases and Snapshots

Use cases and Snapshots were discussed in Section 4.6 and again in Section 4.5.3. Here we look at these as aids to determining product requirements and as clues to product implementation details.

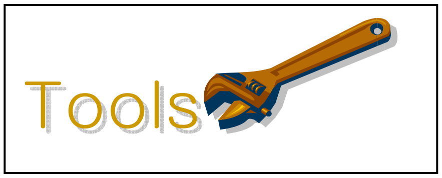
For a remote control we have the following Snapshots for the remote user:

- Insert / remove batteries
- Change channel up or down one step
- Change channel to specific requested channel
- Change channel to previously viewed channel
- Change audio volume up or down
- Mute audio
- Turn TV on / off
- Enter TV type to set the encoding method of the transmitted commands
- etc.

Summary: Various tools are introduced that will aid in the Reverse Engineering process. These tools have more general applicability.

Applicability: Reverse engineering and product development.

Knowing these Snapshots would make us look for the means these functions were implemented on the control. As knowledgeable consumers, in fact, if we are given a new control we 'reverse engineer' the new control by looking for the specific functionality we are already familiar with – we do not read the manual.



System Function Lists

We can also make a list of system functions. This is not the same as a Use Case list, but the System Function List is a list of those functions, which together, allow the product to perform the Use Cases. An attribute diagram such as Figure 2-5 and Figure 5-2 can be produced.

For our TV remote we have

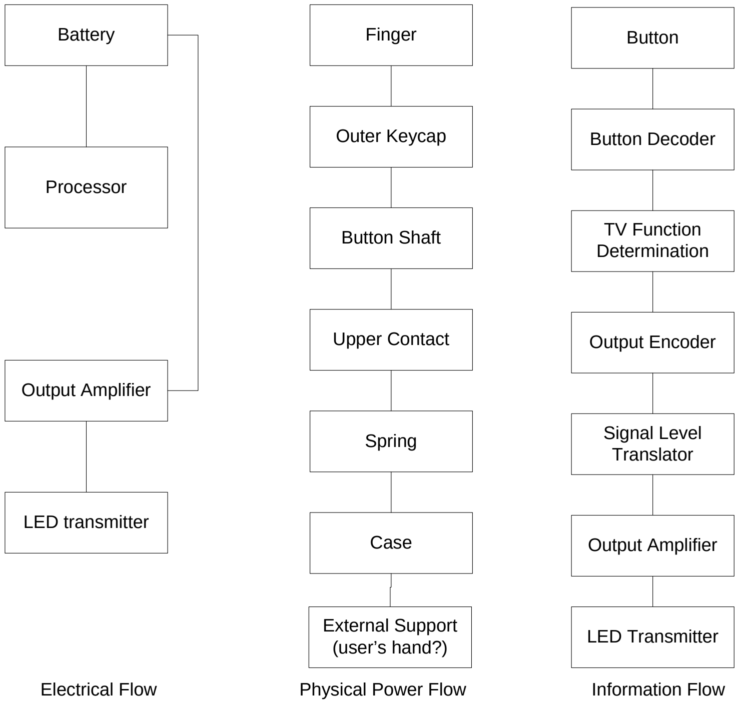
- Hold batteries
- Diagram battery orientation
- Cover batteries
- Determine button pressed
- Encode TV function selected
- Send encoded TV function selection to TV
- Remember encoding method used by TV
- etc.

Electrical / Power / Information Flow Analysis

In this section we are looking at three flowchart or schematic-style methods of representing information in our product. Power flow and electrical flow in a purely electronic product might be the same. Often, such as in our TV remote example, they are largely the same but have there may be flows that are non-electrical, such as the pushbutton mechanism on the user keypad. For a electric garage door opener there is an electrical system with a high power end including the motor and a low power end that controls the motor. There is a separate mechanical power transfer from the motor to the garage door. Sometimes we will want to treat the electrical and the mechanical systems separately and sometimes there will be advantage to keeping the two together in a single power flow diagram.

Information flow is different than power and electrical flow. Information can be held in a large number of different forms – as light, voltage or spin; encoded; differential – the list is endless. With an information flow analysis it is only important how it gets from one end to the other and how it is transformed and used.

Figure 7-1 shows a simple form of these three diagrams for our TV remote control. The electrical flow is not too exciting as it is fairly straight-forward – a battery providing the power to all the other assemblies. The flow of power and the electrical flow may often be the same. In the diagram we show the physical power flow, what happens when a person presses a key from a physical sense. In the last part of the diagram we see the information flow. This is probably the most important diagram to the electrical / computer engineer as it shows the various firmware (embedded software) and electronic stages in the device.

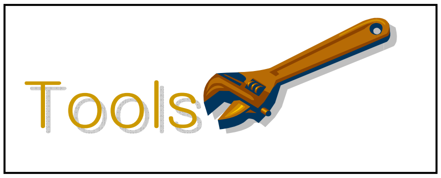


Electrical Flow

Physical Power Flow

Information Flow

Figure 7-1 Various Flow Diagrams for TV Remote Control



Function Analysis Systems Technique (FAST)[12]

Function Flow Analysis is a method of linking requirement to functional implementation. This analysis gets complex quickly, because, as well as the implementation of the main functions, there are additional functions that are the result of side effects of the main implementation. The Function Flow Analysis technique handles these as well.

Summary: Function Analysis Systems Technique breaks down the basic and supporting functions to show dependence

Application: Deciding function of various parts of an existing artefact; analysis of a proposed system design.

For example, consider the use of a motor in an electric mixer. The motor drives the gears that turn the beaters that mix the food. But the motor must be supplied with electricity, it must be housed so the user cannot directly touch the motor, it must be controlled (at a minimum be turned on and off), it generates heat and vibration, it consumes space and it has weight. Any design of an electric mixer must consider all of these factors. A Function Flow Analysis will indicate the interaction of the parts of the mixer and will show how secondary effects and requirements flow from the solutions to providing the primary functions.

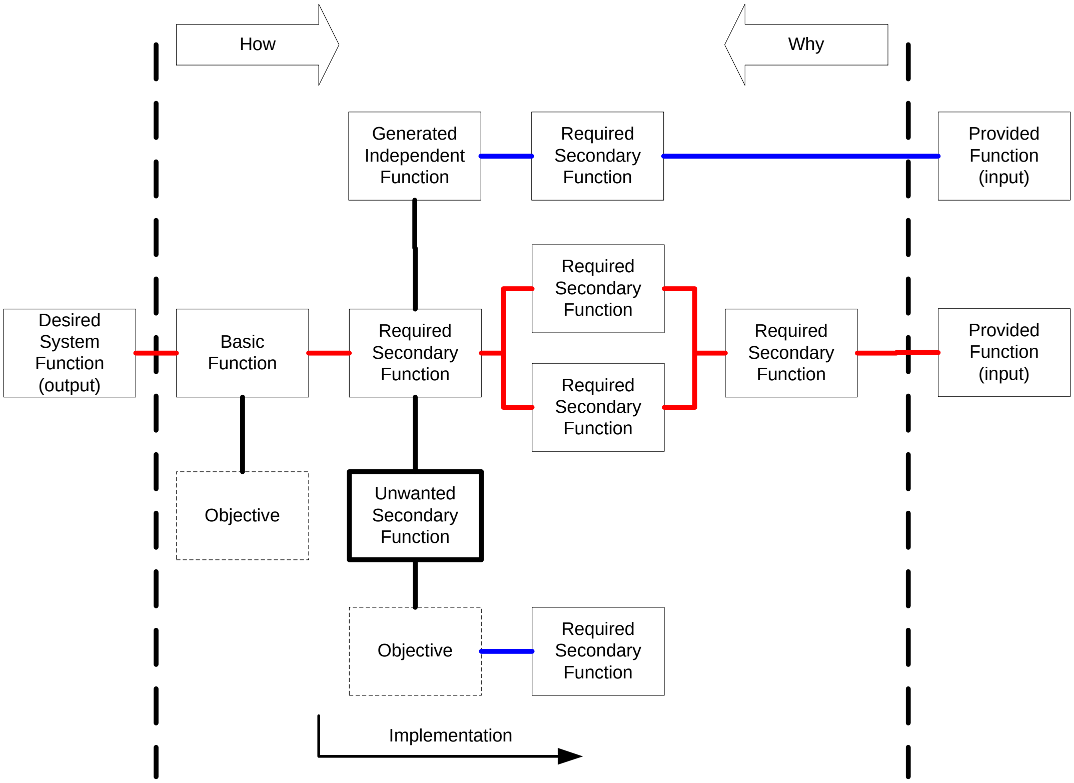

How
Why
Generated Independent Function
Required Secondary Function
Provided Function (input)
Required Secondary Function
Desired System Function (output)
Basic Function
Required Secondary Function
Required Secondary Function
Required Secondary Function
Provided Function (input)
Required Secondary Function
Objective
Unwanted Secondary Function
Objective
Required Secondary Function
Implementation

Figure 7-2 FAST Diagram Components

The Function Flow Analysis Diagram starts with a desired system function, then moves through the generated functions as shown in Figure 7-2. There is a major critical path running horizontally through the diagram, and minor critical paths that

Function Analysis Systems Technique (FAST) was introduced by Charles Bytheway of Sperry Rand Corporation as a way of removing non-critical elements from a design. The reference is to his original slide notes on the subject.

branch off from this.

Note the following:

- At the left side to the right of the first dashed line is the output of the artifact (or system). This is the objective or primary function of the entire system.
- The blocks between the dashed lines give the functions provided by the system by design or consequence.
- At the right side, to the right of the second dashed line, are any inputs that must be provided to the artifact. This could include, for example, electrical power and keypad input.
- The first block in from the left inside the system is the basic system function. Often linked to this are objectives of the basic function.
- In order to provide this basic function, other functions are required. These are to the right of the basic function. These functions will themselves require other functions. These are chained to the right of the functions that require them. The diagram shows one required function for the Basic Function, two functions required for that function and so on.
- Eventually this middle chain links to provided functions on the right side. This chain through the middle (shown with red links for illustration) is known as the **major critical path**, since this chain of functions is required to make the system provide the desired function.
- Some functions are required to support the objectives (these functions are not shown in our diagrams) or to support the functions on the critical path, but these are not themselves part of the major critical path. They start chains of functions of their own. Since they are still required for the system to work, these chains are called minor critical paths. These are the chains through the "Generated Independent Function" at the top of the diagram and to Required Secondary Function that links to the Unwanted Secondary Function at the bottom of the diagram.
- Sometimes there will be side effects of a function. An integrated circuit produces heat; a fan produces noise. These unwanted functions are shown in heavy boxes. They may generate further objectives, namely the minimization or elimination or bypassing of the problem, which then generates functional chains for these new purposes.

Best we now look at an example.

Figure 7-3 shows a partial breakdown of a garage door opener. A garage door opener is made to open and close the garage door, usually when the user presses a button on a wireless controller.

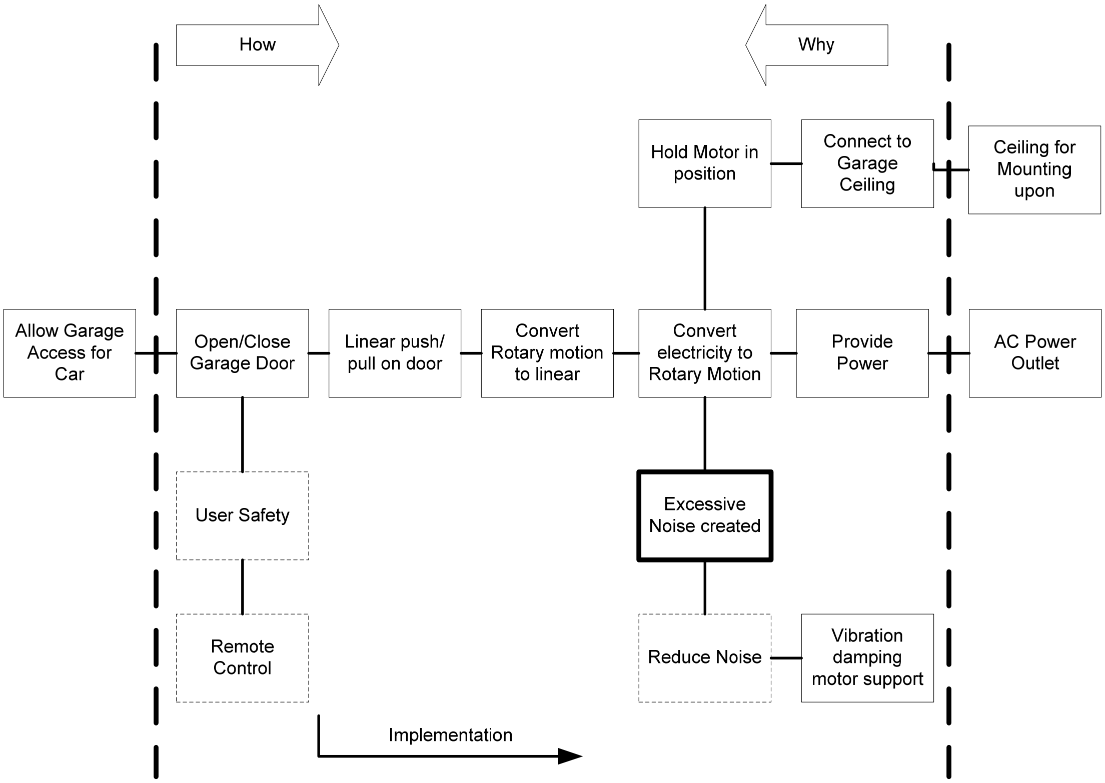

How
Why
Hold Motor in position
Connect to Garage Ceiling
Ceiling for Mounting upon
Allow Garage Access for Car
Open/Close Garage Door
Linear push/ pull on door
Convert Rotary motion to linear
Convert electricity to Rotary Motion
Provide Power
AC Power Outlet
User Safety
Excessive Noise created
Remote Control
Reduce Noise
Vibration damping motor support
Implementation

Figure 7-3 FAST Diagram of Garage Door Opener (Partial)

Of note in the diagram:

- the major critical path through the centre where the provided AC power eventually causes the desired action.
- the undesired creation of noise and the subsequent secondary chain to handle the problem
- the secondary chain to support the motor. Supporting the motor is not directly involved with moving the door, but is a necessary function to make the motor able to do its task.

Several chains are not shown. For example, the garage door opener has an objective that remote control be possible, however there is no function chain describing how this happens, or even how the motor is turned on and off or reversed in direction for up and down. Likewise, no safety functions are shown.

With a complete diagram you should be able to link every part of the system and every characteristic of every part of the system to some box in the diagram. Anything not found is not necessary to system function.

A complete drawing will also allow you to analyse the potential solutions. For our example of the Garage Door Opener we can see in Figure 7-3 the added complexity because of noise. This might lead us to solutions to creating the same end-to-end function but in a way that produced less noise and thus did not need the noise reduction measures. Motor selection, for example, might help this. Even a more expensive, but quieter, motor could prove most cost effective on a system-wide scale.

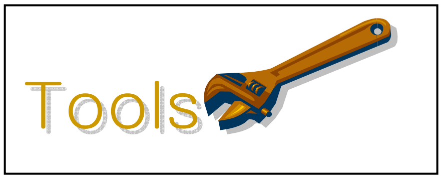
A complete drawing of even a simple device can be very complex. Something like a coffee machine or pencil sharpener can have many, many chains dealing with power, waste, heat, operator input, multiple operational modes, maintenance and so on. Often a partial drawing will be all that is necessary to determine a requirement, to decide on a functionality or to demonstrate an idea.

Another example and more details on the method can be found in [12].

7.2.6 Reverse Engineering Step 6 – Solve the Mysteries

At this point all the information is in and if you have not yet solved the mysteries of the product, it is time to do so. In doing so you might start to consider how it could be done differently, in particular how it could be done better if done differently.

The first TV remote I ever saw was hard-wired to the TV. It saved getting up to change the channel or volume, but was tethered. The cord limited the possibilities of passing it around and was a trip hazard[10]. For a while they used sound activation. This worked OK, but sometimes setting down keys or other sounds would make the TV change channels. Our dog would often change the channels by moving and jiggling his dog tags. Modern TV remotes use infra-red light transmissions of heavily encoded and redundant commands. Still, there is room for improvement. Even "universal" remotes do not self-teach, do not work at long distances and do not work particularly well if not aimed precisely. What other changes can you suggest?



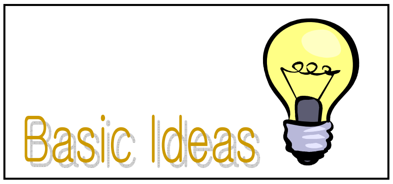
7.3 UML (again)

UML does not fit our development model well. At times it is dealing with requirements and at other times with implementation. However, since it is a set of models, one can pick the appropriate model for the task at hand.

The information will not be repeated here but should be reviewed by the reader for applicability to implementation.

[10] On the other hand, it never was carried away or lost in the seat cushions….

Chapter 8 Determining Designs



8.1 Something that Does What is Wanted

At this point we should be fairly sure of what we want (as expressed in our requirements), but we are not at all sure of how we are going to do it. The list of requirements can be daunting, and even if (as would be normal with engineers) we already have some thoughts on what we might do, it is hard to know if we have a workable solution, or the best of the workable solutions, within these thoughts.

Our potentially workable solutions will be combinations of pieces or modules that will each take care of some of our requirements. So, if our goal is something to make holes in wood, our solutions might include a drill bit with holding device that connects it to a motor, and a housing to provide user safety and user facility to guide the bit. Our solutions might include a laser with appropriate housing, or a nest of well-trained termites. (Remember, we still have not evaluated the designs at this point and want to keep ourselves open to all ideas.)

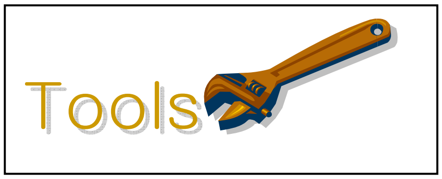
Summary: Potential implementations are proposed. This is the heart of the creative process, with its main method: Brainstorming.

Applicability: Systems design and any place a problem needs a solution

How we put these modules together constitutes an architecture. The architecture dictates how the modules interact and ultimately how the artifact performs, how much trouble it is to change the design, and how difficult the artifact will be to test and maintain in the field. If poorly chosen, awkward interactions and difficult usage will be unalterable characteristics of the design and ultimately lead to its failure to meet expectations.

Designs also have orientations. Beyond the hard requirements and constraints are the Measures of Evaluation. We can orient a design to increase certain measures; these we call “Design For’s” since we are designing for a certain strength in these attributes.

In this chapter we will look at some basic architectural models that might be considered, we will look methods of coming up with designs using brainstorming and we will look at some common “Design For’s”.



8.2 Architectures

It is often useful to consider the breakdown of a system into modules that conform to various design architectures. Here we will present eight common architectures used in process, electronic and software artifacts. Note that these are not independent or otherwise exclusive. They can be used effectively hierarchically and jointly. Some of the examples demonstrate multiple architectures.

Also please note that the descriptions are far from comprehensive. Anyone interested in exploring the architectures further should consult more detailed information on the architecture(s) of interest.

8.2.1 Functional Breakdown

This is very similar to a Work Breakdown Structure that we will examine in detail in Section 10.2 .

The starting point of a functional breakdown is usually a block diagram. Each function of the artifact is given a block (there could be further functional breakdown of complex items later.) Figure 8-1 shows how a doorbell could be represented as a block diagram.

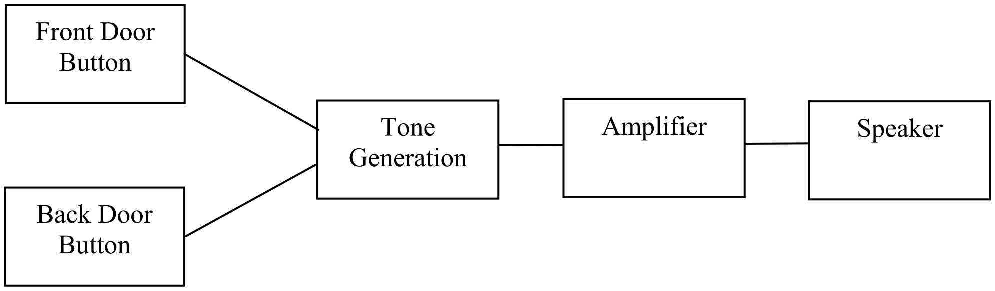


Figure 8-1 Block Diagram of a Doorbell System

8.2.2 State machine

The state machine is a representation of anything that depends on history as well as current inputs. Our doorbell as presented in Section 8.2.1 is not such a device, since the doorbell ringing only depends on the present input from the pushbutton. The USB interface from Section 5.5.4, would be a state machine since it implements a protocol and the actions at any particular time depend on where it is in the protocol, in other words, depend on the history to that point.

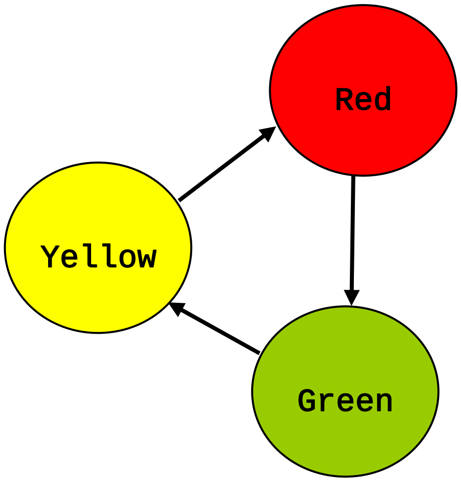


Figure 8-2 State Diagram of a Simple Traffic Signal

Figure 8-2 is a state diagram of a traffic signal. The reason for the state transition is often shown on the arrows.

State diagrams model control processes well. Often the historical requirements of the process can be modelled in a few states and easily shown. State diagrams also translate easily to hardware or software structures that are flexible and easily understood.

8.2.3 Implicit Invocation

Some system functions are implicitly invocated under certain circumstances. For example, the landing gear of an aircraft could be extended automatically when the altitude is under 300m.

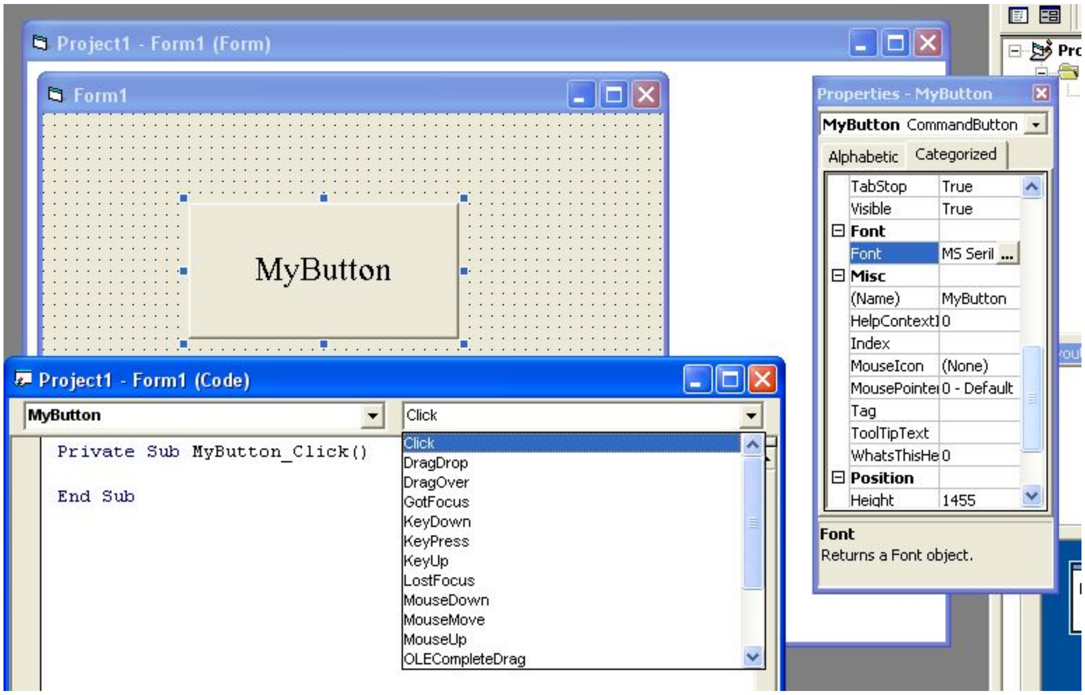
In software you may have seen this in Java and C++ with automatic invocation of initialization procedures when a new instance is created, or with implicit conversions of data from integer to floating point.

In hardware, there often are protection circuits, such as the crowbar on a power supply that lowers the voltage output when current demand would otherwise cause power supply failure.

8.2.4 Event Architecture

Operating systems like Windows® and applications such as a word processor are often built around an event architecture. What happens is a reaction to an event such as a keypress, a mouse click or some processor-generated event like a time-out or real-time clock “tick”. Errors or exceptions and input/output are other events that can generate reactions.

Figure 8-3 shows a typical programming step for implicit invocation architectures. Here a user button is being programmed. Routines can be specified for various events, such as click, dragdrop, dragover, gotfocus and so on as shown in the drop-down list of the figure.



```
Private Sub MyButton_Click()

End Sub
```

TabStop	True
Visible	True
⊟ **Font**	
Font	MS Seril ...
⊟ **Misc**	
(Name)	MyButton
HelpContextI	0
Index	
MouseIcon	(None)
MousePointe	0 - Default
Tag	
ToolTipText	
WhatsThisHe	0
⊟ **Position**	
Height	1455

Font
Returns a Font object.

Figure 8-3 Programming Environment for Implicit Invocation

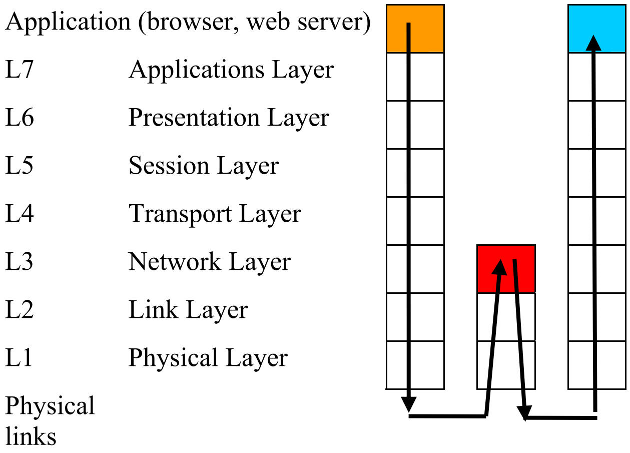
8.2.5 Structure & Process Hiding

When we do a block diagram (Section 6.2) we represent a chunk of our final design with a block. Ideally, each block gets and/or provides information to other blocks without having to know how the other blocks perform their functions. This has the advantage that each block (or module) can be developed independently. We can use the same technique of hiding information within a module to give us the same advantage.

Here we will look at three variants of data and process hiding.

Layers

The OSI communications data model, used in TCP/IP for Internet communications, for example, is one place this architecture is used. Figure 8-4 shows this model. Each of the layers is responsible for a certain abstraction of the communication process. One application that wants to communicate with another doesn't have to concern itself with error correction or dividing up the information into packets or packet sequences or the physical media being used. Lower levels handle these concerns, but don't have to worry about what the information means that is being transmitted. There can be intermediate stages, such as routers, that need only the levels that allow them to determine the final destinations so they can tell where the packets go. A communication with an intermediate routing stage is shown in the figure. Taken together, the layers perform all the communication requirements.



Adapted from: http://www.erg.abdn.ac.uk/users/gorry/course/intro-pages/osi.html

Figure 8-4 OSI Communication Model

This makes the entire, complex communication function easier to adapt to changes and variants. To each higher level layer the layer below seeminglyprovides all of the functionality for all the layers below, in other words the layer below finishes the communication function. The higher level layer does not need to know how this function is completed. Similarly, a layer will provide the communicated information to a higher level. This providing layer does not have to worry about what the information content is or how it will be used. Thus each layer can be programmed and maintained independently, and it is much easier to move from an Ethernet to a DSL to a dialup to an optical link since the higher levels are programmed without needing to know the media in use.

Hiding Abstraction Levels

Closely related to this is the hiding of levels of abstraction.

We do this in hardware. A digital electronic circuit is, at a lowest level, a method of creating a situation where electrons behave in a highly probabilistic way in a manner we determine. We hide the quantum effects with a classical model of the electron behavior, and this with a transistor-level model of what is going on, and this with a gate-level model of transistor circuits and this with a logic circuit model with flipflops, counters, etc. and so on. Each level hides the details of what goes on in the level below. The cost of this hiding is that we can no longer manipulate events at the granularity offered at the level below.

This same type of structure is also true in analog hardware where we have opamps, filters and mathematical models of filters.

Software does this through routines and subroutines and with classes and subclasses. Software is an abstraction level of digital hardware (which is an abstraction level of analog hardware which is an abstraction of quatum physical effects).

Functional decomposition is the generic form of this type of activity, although with less formality and functional hiding that with the abstractions just considered. Which brings us to the next topic….

… Data and Function Hiding

Another way of looking at layers and at abstraction is as a form of data and function hiding.

C++, Java and (to a lesser extent) C programming languages formalize this way of looking at layers and abstraction. There are local variables and procedures hidden from the general view. This makes our programming easier and safer, safer since the outside processing cannot affect the local store except through interfaces we have control over. We localize the activities and store to keep the data and processes from malicious or inadvertent alteration by the outside.

Operating systems extend hardware support for these separations.

8.2.6 Data flow, pipes & filters

Our block diagram, such as in Section 6.2 , gives a functional linkage between parts. With a data flow architecture we are concerned with the information linkage between parts.

Figure 8-5 shows the data flow in a crystal radio, where the 'data' is electronic signals. Each of the blocks filters the input signals then sends it to the next block. Eventually we have only the desired signal, filtered so the earphone can reproduce the original audio that was sent as a radio signal.

Similar filtering and "piping" (sending over a conduit from one module to another) is done in software. This type of architecture is the basis of the Unix/Linux OS user interface. Parallism and streaming are also natural outcomes of this representation.

In business, significant gains in productivity occurred when industry changed their model from a functional one, where order scheduling was a side issue of the network of machines and processes, to one where the order drove the processes and thus scheduling was a natural result. Like most techniques, this architectural representation is useful in other areas beyond electronic and software design.

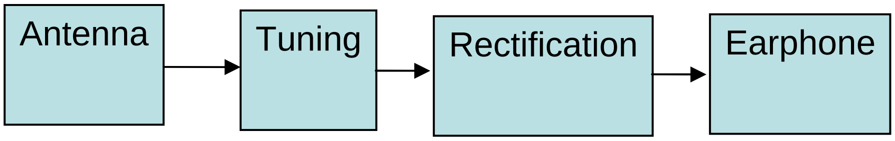


Figure 8-5 Data Flow in a Crystal Radio

8.2.7 Data structure

Sometimes the key to a part you are designing is not the movement of information, but the store and the filtered retrieval of the information. When there is a lot of information to be retrieved in a somewhat non-deterministic way, this is a database. When there is less information and the method of retrieval is reflected in the store, the information store type is named depending on its characteristics: queue, stack, array, binary tree and so on. In all cases we are concerned about saving the information and how we might get to it.

Databases, such a student information database, are organized in records (say one record per student) that are organized into tables (such as a mark record table or address information table) that are linked together by some key (say a student number). Concerns usually revolve around speed of searching and sorting, around security of information and around accuracy of information.

8.2.8 Object oriented ("OO")

Object-oriented software development rose from concerns that functional breakdown methods were not meeting the complexity needs of larger systems. Objects have characteristics based on what they are, capabilities based on what they are and state based on their history. They spring from the Use Cases, which makes them easier to verify as correct. The creation, destruction and work done by an object are more easily understood and controlled. As a result, much of the new large systems are developed in object-oriented software languages like C++ and Java.

Some weaknesses have appeared. At times the methodology is cumbersome and, in the general case, the programs generated have a difficult time-behaviour to predict due to the large amount of implicit behaviour done by the computer to support the objects. As a result, many scientific programs and small programs are done in a non-object-oriented language, such as C. However, research is continuing to reduce these weaknesses.

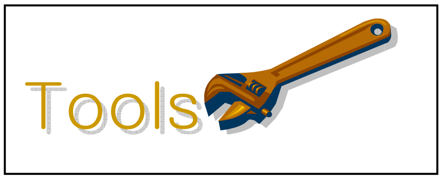
Even a worthwhile partial description of object-oriented development is beyond the scope of this book, but well worth the while of anyone considering an involvement with large software systems. For others, consideration of this type of approach can lead to a simpler understanding of complex problems, and perhaps will generate potential implementations that might not otherwise come to mind.

8.2.9 Examples of Architecture

Consider the design of a cellphone. You can see the following architectures (amongst others):

Functional Breakdown	display, case, microphone, speaker, antenna, etc
State Machine	on/off, connected/not connected, …
Implicit Invocation	store of last number for redial
Layers / Data Hiding	communication protocol
Data Flow / Pipes / Filters	determination of your link on amongst all the other cell links on the air

Question: Do you also have the same "architectures" in a document??



8.3 Brainstorming

Everyone likes to think they know how to brainstorm. But to most people the word is synonymous with "think about". Sometimes they believe the word means "find a solution". As a result, the thinking stops prematurely or never gets properly started. Students will "brainstorm a solution to a calculus question", but really just be finding an answer. Worse, they will brainstorm to find an answer for a design project, and stop when the first workable solution presents itself.

What does it matter? In a lot of situations, it doesn't. However to do good design you must be creative. This is not about just throwing together the design outlined by your professor or in a book or on a website. This is about understanding the fundamental creative element and the place of brainstorming in this fundamental creative element.

Summary: Brainstorming is a deliberate technique used to form ideas. Various methods and aids are described.

Applicability: Idea generation

8.3.1 Brainstorming in Groups

Brainstorming in groups is best done when there is no ownership of ideas, no supervisors or other hierarchy, no "hidden agendas", no interruptions and no (severe) time constraints. Hearing an idea from another will often trigger new thought trails in one's own head. There are typically 1-12 people in a group; although a group of one seems strange, adherence to the group rules will make even solo brainstorming work better. There is generally a facilitator – the facilitator will keep the times during formalized brainstorming with fixed-time periods of activity, and will sometimes step in if the work is going off-track, if participants are not adhering to the rules or if summaries are needed. Participants may keep their own notes, or there may be a "recorder" who keeps notes for everyone.

What rules must be adhered to? Not many, since it is important to keep the participants loose and creative. Here is a quick list:

- no bosses – everyone is an equal and all ideas have equal weight
- no decisions. Quick evaluations may be done, but only as a means of generating new ideas that work even better, because….
- … there must be no criticism. All ideas are welcome, even and particularly if they are absurd as long as they are at least partial solutions to the problem at hand.
- no cellphones, email, visitations or any other interruptions. Yes, turn it right off!

- build on ideas. Even small variations are valuable (and remember: no criticism!)
- don't stop at the first apparent solution – the more ideas the better
- allow misinterpretation – keep ideas brief and perhaps a bit hazy
- A good brainstorming session will bring smiles and laughter. The mood should be light and open.

Certain formal methods exist, generally based on the process of creating some ideas then expanding them or using them to spawn other ideas. Two general formal methods are:

  - Each participant writes down on a piece of paper as many ideas as they can in 10 minutes. The papers are passed left and then each participant tries to expand the ideas on the paper they are given and record any new ideas they have as a result of seeing these other ideas. The ideas are then presented to the whole group to work on.
  - Participants go around the table introducing one idea each or a variant on an idea already expressed, then contribute ideas at random until the 10 minute mark. The group then determines the 5 ideas most likely to result in a final solution and work on expanding those ideas.

8.3.2 Informal Brainstorming

At the beginning of this section on brainstorming, we talked about brainstorming as not being the same as "thinking about". The reason for this is that "thinking about" usually means finding a single possible solution under heavily constrained requirements (many of them unanalyzed and over-restrictive) and doing this with instant analysis and acceptance or rejection of every idea generated. This is not brainstorming.

If, instead, your ideas run more freely, if analysis is done sparingly and late, if you dare to be absurd, if you are periodically laughing at the silliness of some of the ideas that come up, if you are generating ideas that "come from left field" and are not standard fare and are often beyond what any single person in the group might have thought of alone – if these are true, then you are brainstorming.

Like any activity, exercise is important in brainstorming. With brainstorming this practice will mean that your mind quickly starts exploring the cracks and corners of an idea space, and the solutions will come faster and more easily.

Often the solution is standard fare. You must not fail to record the established ideas that time has tested. But problems are usually generated because standard fare is not providing the easy answers or because the answers generated are not fully satisfactory. Standard fare in standard combinations builds the standard artifact – certainly not a product to make inroads in the marketplace. In these cases the solutions will often then be a generally-unused, or a different application of, an idea or selection of ideas.

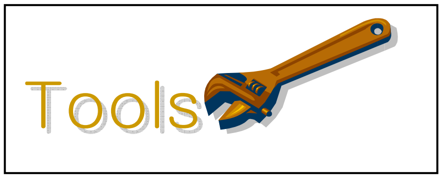
8.3.3 Aids to Brainstorming

Sometimes things get "stuck". New ideas stop flowing before there is the general feeling that enough ideas have been generated to adequately cover the solution possibilities. If so, try the following:

- Think about the opposite – instead of trying to figure out how to make it move, figure out how to keep it from moving. This may give you clues as to what to do to allow it to move
- Find solutions around random words, a noun, verb or modifier of these. Find a solution with the word "pill" or "dog" or "eat". Not all will work, but one of them might trigger a new chain of thoughts.
- Relax the constraints. What choices might become available if the message only had to be delivered correctly half the time instead of all the time, or if you didn't have to protect the user from high voltage. Maybe you can find a solution to the reduced problem, then extend it to cover the constraint that was relaxed.
- Change seats, take a break and listen to some gentle classical music or take a walk or a shower or do a crossword or have a game of solitaire, divide into smaller groups for a while to work – Generally change things, at least for a while, since nothing is coming from the way things are.

- Review and existing close product and suggest alternatives for every part (see SCAMPER, below)
- Explain the problem to an outside party

There are a number of advanced techniques for creative thinking, such as those found at [19].



8.4 Function-Physical Decomposition

The key to creating the system design is to produce a set of functional modules that fulfill the requirements and can be passed on to the next stages for detailed design. We have just learned how to do a function flow diagram in the last chapter. We probably have some idea of how to structure the artifact physically. We can iterate between the two to come to a final system design. Figure 8-6 shows this.

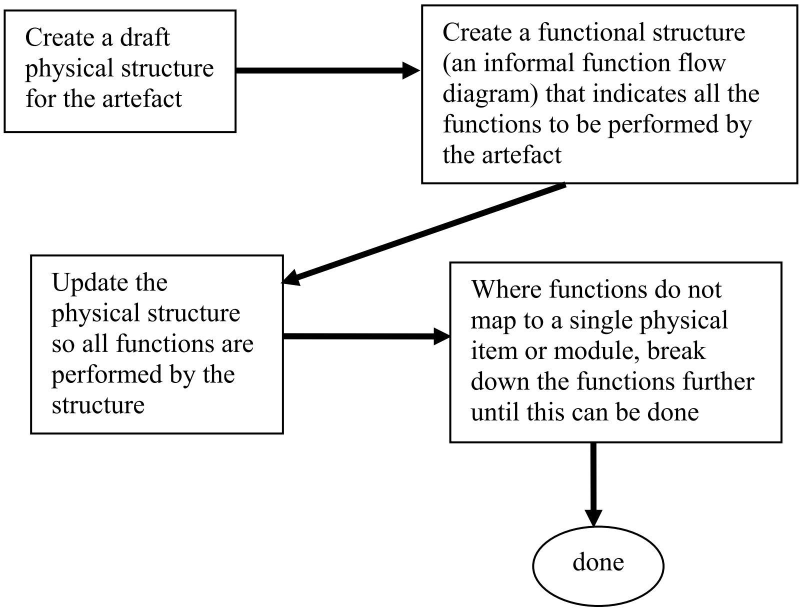

Create a draft physical structure for the artefact
Create a functional structure (an informal function flow diagram) that indicates all the functions to be performed by the artefact
Update the physical structure so all functions are performed by the structure
Where functions do not map to a single physical item or module, break down the functions further until this can be done
done

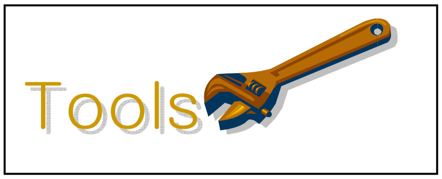
Figure 8-6 Function - Physical Decomposition

The purpose of the procedure is to have no functions performed by more than one module. If we find this is the case, we subdivide the functions until this purpose is fulfilled.

At the end we have a physical structure that will support all of the desired functions, divided into sections such that no function extends beyond the module. Thus each module can be designed relatively independently and tested against the assigned functions.

Note that the same analysis also applies to software and mixed software-hardware projects. For software projects the physical structure is equivalent to a software system architecture and the physical item a definable part of the software.

Summary: Functional / Physical decomposition is a method to associate the required functionality with a single physical entity or module to simplify analysis and futher development.



8.5 Finding new artifacts from old - SCAMPER

One way to create new products or processes from old is called SCAMPER, which is an acronym for:

S – Substitute – take the old and replace components, materials and people

C – Combine – take two existing products and processes and combine them into one. What advantages does this generate?

A – Adapt – put the product or process into a different set of circumstances. How would it work in extreme cold or hot, or out in the rain?

M – Modify – take the existing product or process and make it bigger or smaller, change the colour or weight, alter the shape

P – Put to another use – try to use an existing product or process in a different application. Change it as you need to make it fit.

E – Eliminate – remove parts from the product or process. What can you do with what is left? Can you imagine a situation where you don’t need certain elements of the products or process?

Basic Ideas

Chapter 9 Evaluating the Design

Research and creative idea generation will result in a number of design possibilities. Different architectures and technologies can often be used to fill the same set of requirements. Objectives, those attributes of the artifact that are seen as valuable to the stakeholders, can also be met at various levels. As we shall see in Chapter 11 , the risk and thus the final reward can vary and thus lead to different potential implementations.

This is the evaluation step of our three step process (research-brainstorm-evaluate; see Section 6.5). Cross-fertilization of ideas is also possible – it may be that the best solution is not one of the proposed solutions, but contains components of many of them. Figure 9-1shows this process. Reminder: This is only one iteration of what could be many iterations of the process to find the solution with which we want to proceed. This is a view of the evaluation part of Figure 6-3.

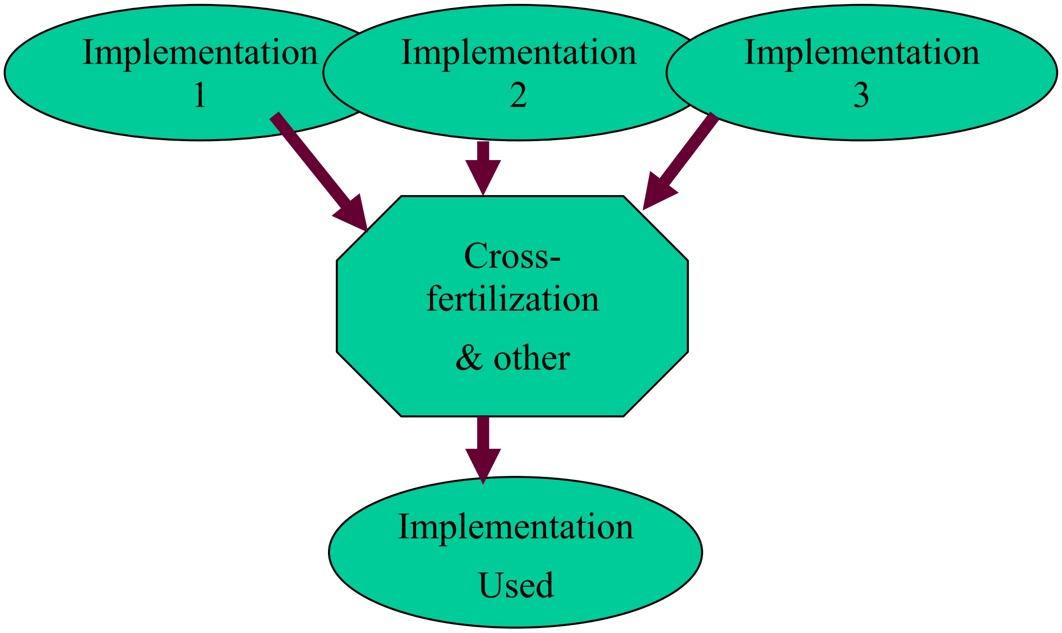
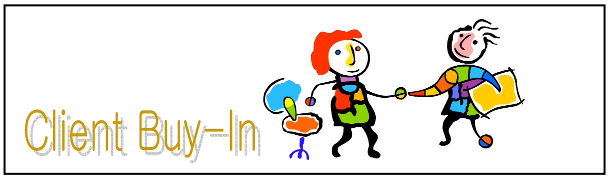


Figure 9-1 Evaluations and Cross-Fertilization

Note that all implementations in the figure meet the requirements specification. They differ in how they satisfy the project objectives. Our evaluations in determining the outcome solution involve cost, risk and reward, which will be covered in the chapters to follow.

In this chapter we will look at how we can evaluate these implementations using the House of Quality method. In the next chapter we look at budgeting, then in the chapter after that look at risk-based evaluations.

In Figure 6-3 we see diagrammatically the step we are taking at this stage. Our work has netted us several possible implementations. Each of these should be able to meet the Requirements Specification – now we want to choose implementation that we will move forward with into the detailed design phase. Should we be dissatisfied with the choice, another loop through research and creation of potential implementation is indicated.



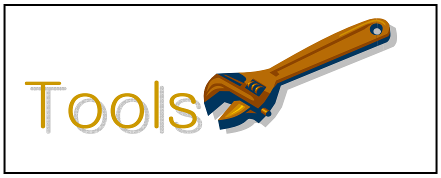
9.1 Involving the Stakeholders

Sometimes the temptation is to make System Design evaluations within the design team, since the design team knows the consequences of the decisions to development. This is a mistake. Often the client (which could be the company itself, if this is a production item) can assume that certain features are impossible or prohibitively expensive. Sometimes certain stakeholders will have sensitivities to price or performance that have not been adequately conveyed to the designers up to this point. Very often the circumstances may have shifted during the time it has taken to get to this stage. Trends may have developed that make certain design decisions preferred over others.

The designers should involve the client in the decisions at this stage. A presentation to a group is a good idea, with many groups represented. These could include (depending on the type of project):

- the users of the system
- sales, since they will have to persuade someone to purchase copies of the system
- marketing, since they will have to create a market for the system
- the person that gave the go-ahead to the project
- finance, since they are funding the development and will see the financial results of the sales
- representation of the clients technical group who will install and maintain the artifact

Many of these you will already have consulted in determining the System Requirements.



9.2 House of Quality

The House of Quality [13] was created to allow the comparative evaluations of existing and proposed solutions to a set of requirements that can be considered Project objectives. These solutions are placed side by side in the house, along with weighting information as to the importance of each differing attribute.

Many of the parts of the House of Quality can be used independently to compare implementations and to help determine a choice.

Summary: The House of Quality method allows the comparison of proposed and existing methods.

Application: Choosing and evaluating the system design to implement; comparison of competitive designs when looking for market niches.

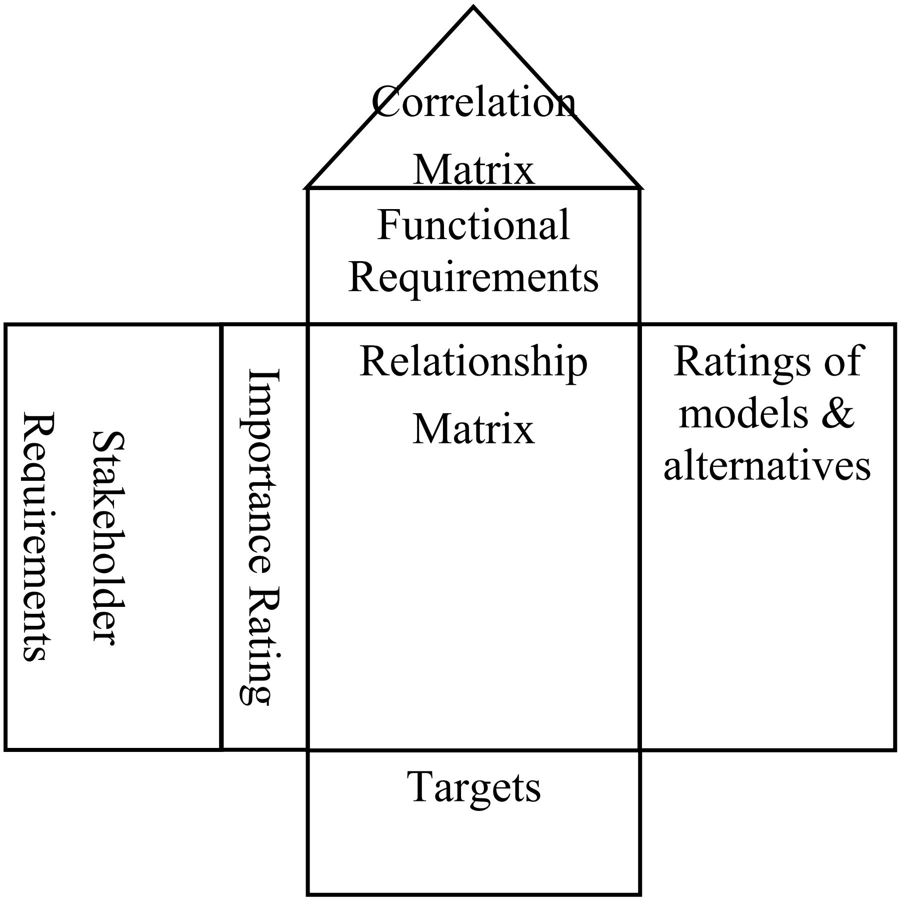


Figure 9-2 House of Quality

Figure 9-2 shows the layout of the House and we will go through filling the various parts of the house in order, as adapted from [6]. The first two steps identify the artifact through its requirements:

1. List the user requirements at the left of the matrix. By this stage in the design process (or acquisition process, since this works well for a purchasing decision as well) you should have determined the stakeholders, Use Cases and from them derived the requirements.
2. Put in the relative importance, say 0-10. These are just numbers to indicate the importance of each of the requirements.

You may want to eliminate any solutions that do not meet the project constraints and functions. This will reduce the need to list these in the user requirements area, so that all we are worrying about here are the Project objectives, those requirements (like product cost) where there may be flexibility.

We will work through the House with the example of a TV remote controller. The tables will only be partially complete to save space. As in all uses of the House, the values assigned are for particular target users and could be different for another set of target users.

The user requirements and importance areas for the TV Remote are shown in Table 9-1. As shown by the importance assignments, “Portable” is looked on as the most important requirement. We might want to sacrifice “Long life battery”, for instance, if it will increase portability since this function is not rated as highly.

Works more than just the TV	5
Long range	4
Low cost	7
Programmable for new function	2
Portable	9
Long life battery	4

Table 9-1 Stakeholder Requirements & Importance Rating for TV Remote

Next we fill in the Functional Requirements and the Relationship Matrix in these next two steps.

3. Fill in the Functional (or engineering) Requirements by determining how each of the stakeholder requirements can be filled. Add arrows or +/- to show direction of improvement.
4. Fill in the Relationship Matrix to show strong (!), substantial (*), some (-) or no ( ) correlation between stakeholder requirement and functional requirement.

In Table 9-2 are shown the Functional Requirements (top row in white) and the Relationship Matrix (balance of white) for the TV remote control. The Stakeholder Requirements are shown on the left side.

Stakeholder Requirement	Output Power +	Battery hours +	Button Size +	Overall Size -	Internal "Smarts" +
Works more than just the TV					!
Long range	!				-
Low cost		*			*
Programmable for new function					!
Portable		!		*	
Long life battery	!	!			-

Table 9-2 Functional Requirements & Relationship Matrix for TV Remote

We see that Overall Size increase is not to advantage, it is better if it goes down, thus we have the – sign where we enter the Overall Size in the functional requirements. On the other hand, we want big buttons, so Button Size gets better if it goes up.

In the matrix area we relate stakeholder requirements to functions. Output power (the power of the signal sent to the TV from the remote) is strongly related to the stakeholder requirements of Long Life Battery [in a negative way] and to Long Range [in a positive way]. This is because increasing the output power will drain the batteries faster and lower battery life, but should allow us to be further from the TV and still be able to control it.

In the next step we introduce the solutions that we want to compare. This could include models already produced by our company and by the competition, and it could include proposed new models. Comparison of existing models may help us to determine a niche where our artifact would do well despite competition in the marketplace.

5. Do competitive analysis by completing the Ratings of Models and Alternatives. Beside each Stakeholder Requirement rate each model as to how well it fills that function.

Stakeholder Requirement	Model ABC	Model XYZ
Works more than just the TV	**8**	**5**
Long range	**5**	**4**
Low cost	**4**	**3**
Programmable for new function	**0**	**6**
Portable	**9**	**9**
Long life battery	**7**	**4**

Table 9-3 Ratings of Models and Alternatives for TV Remote

In Table 9-3 we rate two models of TV Remote. They are both equally and highly portable. Model ABC lasts longer on a set of batteries, but has no new function programmability. Neither is particularly low cost – perhaps this is an area that could be exploited in a new design.

The area of the House from left to right through the middle is now complete. It deals with the stakeholder requirements of the artifact.

The area up and down through the middle of the House deals with the technical problems. We are going to now concentrate on this area, starting with the first of four parts of the "Targets" area of the House.

6. In the targets area, indicate the technical difficulty of each function.

We identified a number of functional requirements in step 3. Some of these will be trivial to implement, others will be very difficult. In this step we indicate this difficulty.

	Output Power +	Battery hours +	Button Size +	Size -	Internal "Smarts" +
Difficulty	**7**	**3**	**4**	**7**	**9**

Table 9-4 Ratings of Function Difficulty for TV Remote

For the TV Remote, for our company, the difficulty to add Internal Smarts (some sort of higher level intelligence) is much more difficult than simply adding battery hours. Table 9-4 shows this.

What we do next is to show how the functions interrelate.

7. Correlate the functions in the matrix "roof" area of the House. Strong negative correlation (making one better for the design will make the other worse) is shown with a double negative sign. Weaker negative correlation is shown by a single negative sign. Positive and strong positive correlations are shown with single and with double positive signs respectively.

Some of the functions we have determined for our TV Remote will interfere strongly. For example, as shown in Table 9-5, as our button size goes up (which was identified as a positive) our overall size goes up (which we identified as a negative). This strong correlation is shown by a double positive sign in the cell of intersection of the two functions, as shown by the arrows in the figure.

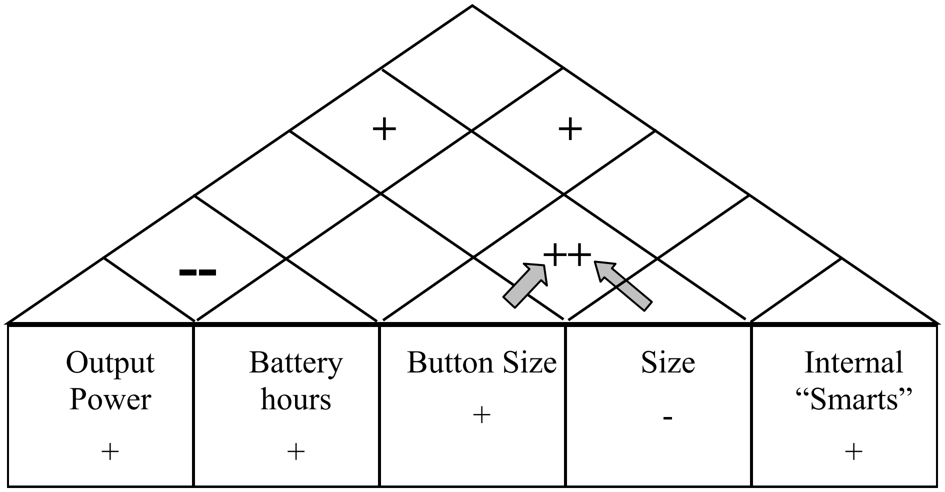


Table 9-5 Correlation Matrix of Functions for TV Remote

On the other hand, increasing the output power will reduce battery life.

One interesting cell here is the battery hours vs. internal smarts cell. Although internal smarts will use power, internal smarts could allow some smart signal switching and low power modes that could increase the battery life, hence the plus sign (+) in that call of the table.

Our final inputs are in the Target area where we add up to three more sets of entries. The first set of entries are actual target values for each of the functions. This is placed below the Difficulties area first shown in Table 9-4.

8. In the targets area, (using the importance and the difficulty and what you are hoping to achieve) list the target values

	Output Power +	Battery hours +	Button Size +	Size -	Internal "Smarts" +
Difficulty	**7**	**3**	**4**	**7**	**9**
Targets	**Max 2 mW**	**Min 10K**	**Min 50mm x 50mm**	**Max 20cm x 4 cm x 1 cm**	**High**

Table 9-6 Target Values of Functions for TV Remote

Lastly you can add two more sets of information. First entries about the various solutions. Secondly, about the technical importance.

For each model being compared, add a row of cells to the bottom of the Target area and fill in the values of that model for each functional requirement.

9. Add a last row of cells to the target area indicating the technical importance of the functional requirement.

	Output Power +	Battery hours +	Button Size +	Size -	Internal "Smarts" +
Difficulty	**7**	**3**	**4**	**7**	**9**
Targets	**2 mW**	**10K**	**50mm x 50mm**	**Max 20 cm x** **4 cm x 1 cm**	**High**
Model ABC	**1.5 mW**	**6K**	**55mm x 55mm**	**24 cm x** **3.5 cm x** **1.5 cm**	**Med**
Model XYZ	**2.2 mW**	**7K**	**30mm x 30mm**	**19 cm x** **5 cm x 1 cm**	**Med**
Technical Importance	**H**	**M**	**L**	**L**	**H**

Table 9-7 Values of Functions achieved for various models, and Technical Importance of each Function, for TV Remote

In Table 9-7 we have added the values for our hypothetical models. Technical importance indicates where we would best concentrate efforts to improve; technical difficulty indicates the effort to do so.

We are done, but what have we done?? The House of Quality we have constructed helps us to answer many questions concerning our System Design, among them:

Is our design any different from any other? Other things being equal, if the market doesn't see any huge difference in our design, the market prospects are not good. From the House we can see where our design is different, where it is the same, and this should lead us to ways to produce a unique entry.

How much effort will be required to add a certain feature or to augment a certain characteristic? Some features can be added with little effort, some will require huge efforts. This information is in the House. We can also see the value of making this particular change. We want to concentrate on areas with high technical importance and lower technical difficulty to get best return for our efforts.

What will be the anticipated return on concentrating on a certain design feature? We want to put effort into those Stakeholder Requirements that will result in the highest return in perceived value. Our House shows us which ones these are.

What will be the tradeoff if we stress a certain feature in our design? For our TV remote we want big buttons, but a small total size. With these sorts of dependent functional requirements we need to decide how much to trade one for the other.

Does the comparison with the competition raise any red flags concerning our proposal? If we are using the same technologies as the competition, huge differences in the functional characteristics may indicate a miscalculation in our design proposal.

9.3 Moving Forward

A good design effort will bring a number of possible implementations and variations to the table. These will have to be evaluated to determine what is the best design to move to detailed design.

One of the inputs into this decision is the cost and time for finishing the project for each of the implementations. The creation of these estimates and the use of the resulting budgets for managing the project as it moves forward are the topics of the next chapter.

Chapter 10 Managing the Design

Basic Ideas

10.1.1 Project Management

When you are preparing with your partner as a student for a three hour lab, a minimum amount of organization needs to be done. It can probably be arranged orally and adapted on the fly. If a mistake is made, the general consequence is a bit more time in preparing or in executing the lab.

When you are launching the detailed design of a two year project involving dozens of engineers, the organization has to be more careful. If a mistake is made it could easily cost thousands if not millions of dollars. It could mean the premature end of the project or even the end of the company.

Summary: Project Management is important to link the project to available resources in the company and to the design targets of time and cost.

Although there are timing and cost and other resource considerations during the requirements and system design stages of design, the bulk of the time, money and other resources will be used during the coming stages. For this reason, the Project Plan is created at the same time as the system design, and project management continues through the rest of the project, as shown in Figure 10-1. The projected costs and resources for a given proposed system design are also a significant part of the evaluation of that design.

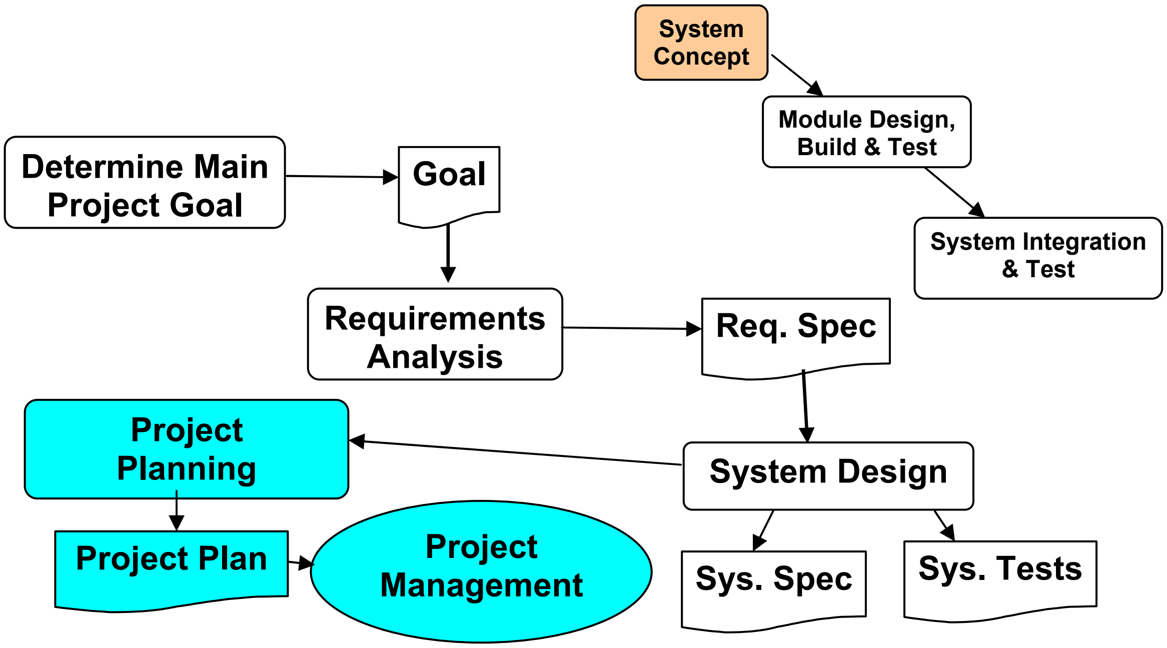


Figure 10-1 Creation of the Project Plan / Project Management

Project management is the management of resources. People, equipment, time and money are all typical of the managed resources. The management of these involves the prediction of how the project will proceed and how much of each resource will be used or consumed as the project progresses – these predictions are called budgets. Our study of project management will be primarily a study of how to create these budgets without making (as many of) the common mistakes.

Project management involves a plan. The elements of a plan and some of the tools to manage the plan are shown in Figure 10-2.We start with a work definition. This divides the work into packages, each of which can be assigned a time, a set of resources and cost to complete. These times and costs go into time and cost schedules.

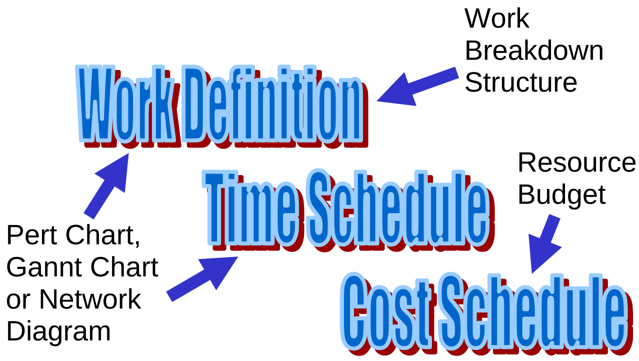


Figure 10-2 Elements of a Plan and Tools to Manage these Elements

The figure also shows some common tools used to create, display and manage these elements. We will describe these later in this chapter.

As was stated at the end of the last chapter, Project Management and System Design are closely linked. Project Management is the management of the implementation of the modules determined by the System Design and of the next steps where the modules are brought together to make the solution system.

As we move from System Design forward we see seven major motivators for the Project Planning step:

Design Tuning As we evaluate the design we will see ways to increase performance at low cost, to reduce development and other costs and to increase the likelihood that the design will work. We see where the resources are going and how we might make changes to reduce the resources required.

Closure Designs often are subject to “feature creep”, particularly designs in electronics and software. Once the first prototypes are produced it becomes easier to see how certain features would enhance the products. Budgets keep the design in check and bounded. Any work on extra features and functions will clearly add to the cost and the time and thus will be carefully considered.

Evaluation We need to know about the design itself. How well will it meet the Project objectives – some designs will be better all-round than others, but usually there is a tradeoff, a poorer result for some objectives but better results for others. We can determine the design’s impact on the environment, the cost of production, and other concerns.

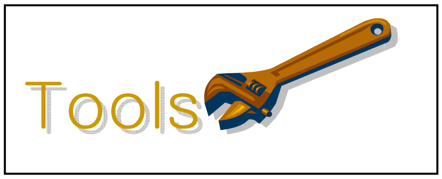
Go-ahead Decision Since we are going to commit the highest percentage of resources we will use on the project, we want to make sure the resources are well-spent. Is the project feasible; can we find the resources to develop, produce and maintain the artifact? Is it likely to generate increased revenue for the company?

Preparations for the Next Steps Going into module design will increase the number of people working on the project. Our System Design will help to get them all on the same page, since it will be important that they understand the system requirements, even as they concentrate on producing their modules. For example, if a team finds a way to produce their module at well under the specified weight target, this may relieve another group of having to meet theirs

Corporate resource planning A resource like a human or a machine cannot be two places at once. A resource like money, if it runs out, may spell the end of the project. Knowing how and when the project will affect available resources will keep their use efficient. It will also help eliminate surprises and unexpected delays due to unavailable resources. A part of preparations for the next steps is planning out the scheduling and costs of the resources.

Development management By comparing reality to the projected times and costs, you can bring extra resources where required to help bring a project back on track.

.



10.2 Work Breakdown Structure

The Work Breakdown Structure (WBS) is used to reduce a project to a set of increasingly smaller subprojects, and eventually to a set of tasks. Each of these tasks should be well-defined, done by a single person, outside contractor or small, focussed group. A task should be small enough that the timeframe is hours to days. A task should be done at a single location and be able to be done without waiting for resources. Table 10-1 shows some good and bad examples of these tasks.

Example	**Comments**
Create a subroutine in C++ to sort up to 300 16-bit integers into ascending order	Example of good task
Assemble the printed circuit board	Example of good task.
Design and produce the circuit board	Alright if given as a project to an outside contractor (who would then break it down further). Too general and done at too many places over too large a time to be a good example for an in-house design member.
Assemble the printed circuit board and deliver it to the customer	This is a bad task allocation. There are clearly two tasks involving two sets of equipment.
Assemble what you can of the printed circuit board, then finish it when the missing parts are delivered.	This is a bad task allocation as well. The task cannot be done at a single time since the resources (the missing parts) are not there.

Table 10-1 Examples of Tasks for a Work Breakdown Structure

An example structure is shown at the left of Figure 10-3, and at the right of the figure is a partial example. The top level shown here is large enough to be called a "program" and there are many levels of breakdown. Smaller projects would have fewer levels – the preparation for a three hour lab might have only two or three.

Summary: The Work Breakdown Structure is introduced. This is a hierarchical representation of the work required on the project and ultimately breaks the project into a series of tasks that are done by a single person or group without waiting for additional resources.

Applicability: Project Plan

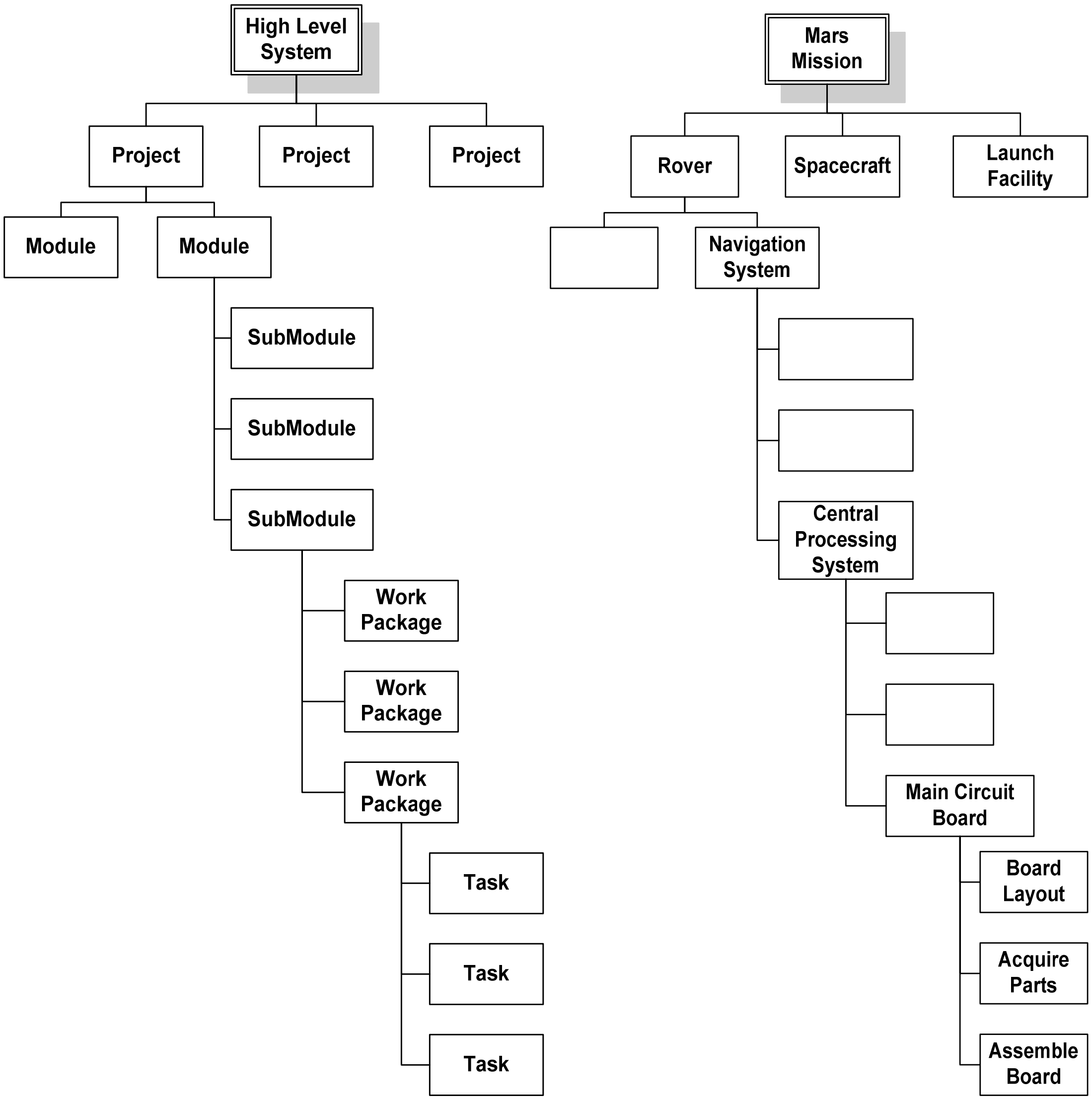


Figure 10-3 WBS for Complex System and an Example Complex System

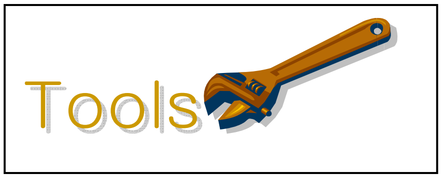
In the example, the entire system is a mission to explore Mars. This system would have many major modules, including a Rover (a robotic semi-autonomous unmanned vehicle that would travel over the surface of Mars performing experiments), a spacecraft to get it there, and a launch pad from which to send it from earth.

Each of these projects would be divided into modules. One of the modules for the Rover would be a navigation system to allow geographic position and geographic orientation to be determined. The drive system and earth communication system are two other example modules in the Rover.

Because even the navigation system is very complex, it would be divided into submodules; shown is a Central Processing System that would take various input data and algorithmically determine the outputs.

The Central Processing System would be divided into Work Packages. Work Packages would further be divided into Tasks. Tasks were described at the beginning of this section. For our Rover, a circuit board could be considered a Work Package. The embedded software might be considered another Work Package. These would be further broken down to tasks – some of the tasks for the circuit board would be to lay out the printed circuit board connections, to obtain the parts for the board, and to assemble the board and parts.

Reducing the large systems into a set of tasks is important for Project Management. Once we know what tasks are involved, we can assign resources to them, we can determine how much they will cost and we can estimate the times involved to complete them. These three, resources-cost-time, are the basic input to the next stage of project planning: creating the schedules.



10.3 Scheduling

Once we have our project even partially dissected we begin to be concerned about the order of the events. Some tasks can happen in parallel (if we can find resources to do both at once). Some must happen in a certain sequence. All are going to take time, money and tools. We need to worry about the order to find out what resources are needed when.

Looking at the simple project of ‘getting dressed’ we can see many ways of doing so, two of which are shown in Figure 10-4. Also shown is a method that will not work, as (usually, at any rate) we cannot put on our socks after our shoes. Similarly, we cannot test something before it is built, we cannot build something before the parts arrive, and so on. However, we can walk and chew gum at the same time (with a few exceptions). Similarly we can have one lab partner answering the prelab questions while the other writes the code for the lab. Finding the necessary ordering and the potential parallel tasks are important to determining a schedule.

A workable sequence

shirt	underwear	pants	left sock	left shoe	right sock	right shoe

Another workable sequence

underwear	left sock	right sock	pants	left shoe	shirt	right shoe

A sequence that doesn’t work

shirt	right shoe	pants	left shoe	underwear	right sock	left sock

Figure 10-4 Sequences of Getting Dressed

The parallelism of activities depends on the numbers of resources. Suppose we are making a tomato sandwich. We need to slice the bread, butter the bread, wash the tomato, slice the tomato, wash the lettuce, assemble the sandwich and clean up. Three sandwich production sequences are shown in Figure 10-5 using different numbers of people to produce the sandwich. If there is only one person (or one knife) we would have to do the tasks serially as in the left-hand sequence. If there are two people, we can parallel activity as shown in the second sequence of the figure. Three people could do it as in the rightmost sequence.

Summary: Scheduling involves determining the sequence of tasks and their timing.

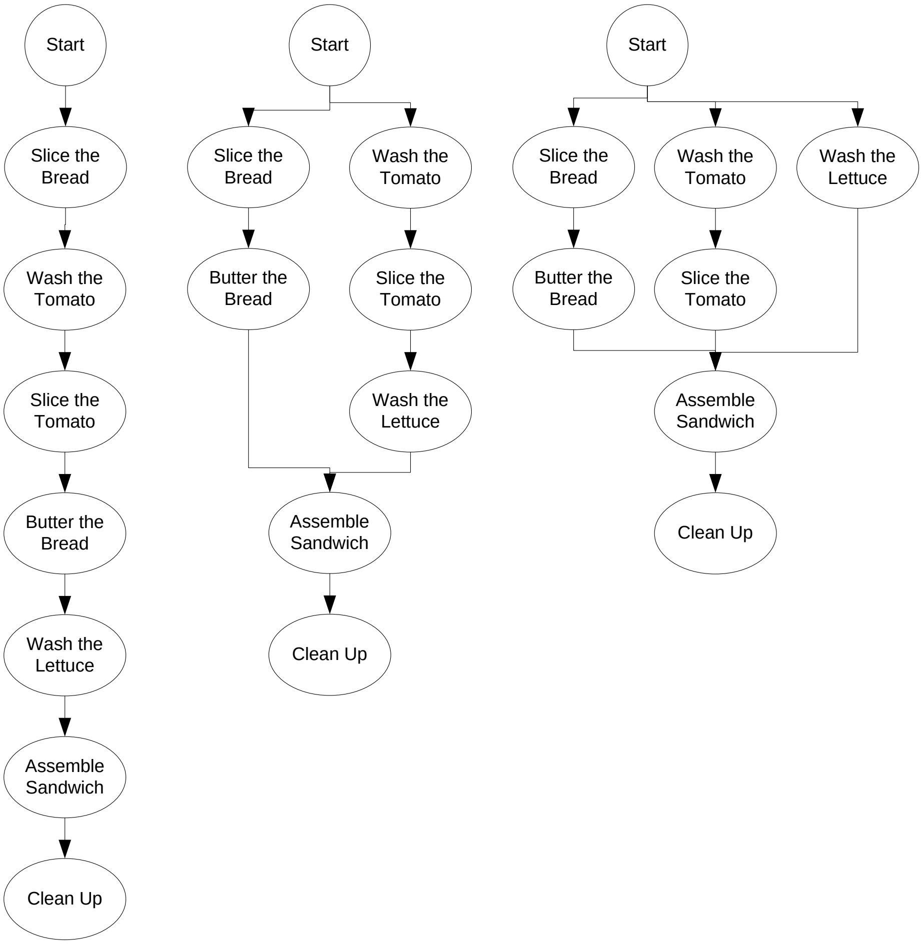


Figure 10-5 Sandwich production with one, two and three producers

There are a few interesting revelations in our simple example. First note that the time is not cut in half by doubling the number of people from one to two. In fact, we still have not reached half the number of steps in time when we add a third person. Increasing the number of producers past three is of no advantage since there are no more tasks that can be paralleled.

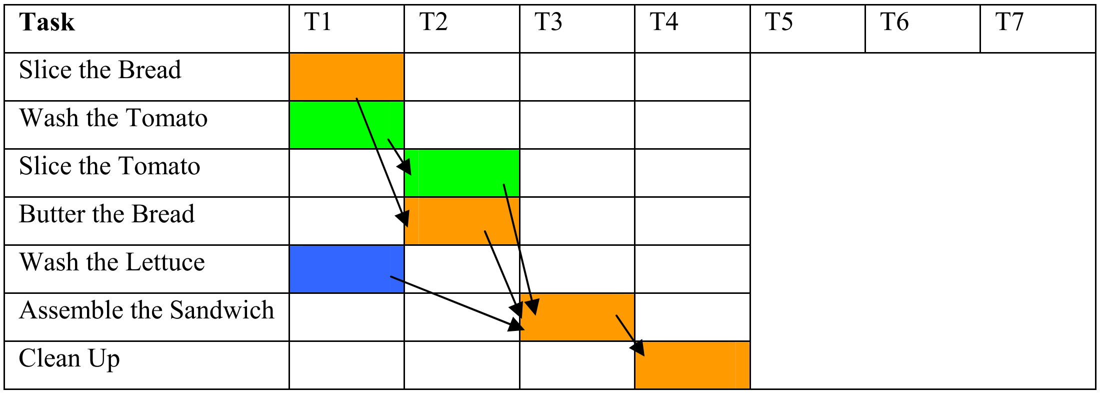
Note also that the cost changes for each of these scenarios. If there is one of us we are rewarded with a full sandwich. If two, the sandwich is done faster but we have to share the sandwich. With three the sandwich gets split three ways and with four people we get the sandwich done no faster, but must share four ways.

Task	T1	T2	T3	T4	T5	T6	T7
Slice the Bread	🟧						
Wash the Tomato		🟧					
Slice the Tomato			🟧				
Butter the Bread				🟧			
Wash the Lettuce					🟧		
Assemble the Sandwich						🟧	
Clean Up							🟧

Figure 10-6 Gannt Chart - Sandwich Production with One Producer

Task	T1	T2	T3	T4	T5	T6	T7
Slice the Bread	🟧						
Wash the Tomato	🟩						
Slice the Tomato		🟩					
Butter the Bread		🟧					
Wash the Lettuce			🟩				
Assemble the Sandwich				🟧			
Clean Up					🟧		

Figure 10-7 Gannt Chart - Sandwich Production with Two Producers



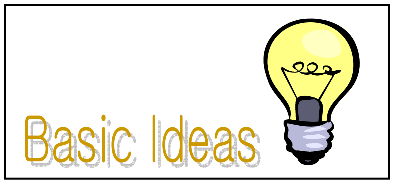
Task	T1	T2	T3	T4	T5	T6	T7
Slice the Bread	🟧						
Wash the Tomato	🟩						
Slice the Tomato		🟩					
Butter the Bread		🟧					
Wash the Lettuce	🟦						
Assemble the Sandwich			🟧				
Clean Up				🟧			

Figure 10-8 Gannt Chart - Sandwich Production with Three Producers, showing Dependency

Figure 10-6, Figure 10-7 and Figure 10-8 are **Gannt charts**. They show the sequence of the tasks (or of higher levels of breakdown) and the parallelism. Often a time scale is added so the Gannt chart also shows the time allocations and requirements.

We can also show **task dependency** with arrows as shown in the last of these figures. Any task at an arrow head is dependent on the predecessor at the base of the arrow being completed first.

The same information can also be represented in Pert charts (a flowchart type modelling) or in Network diagrams (where blocks show timing as well). A simple Pert chart is shown in Figure 10-9.



10.4 Resource Considerations

The following can be seen in Figure 10-9.

Slack Time No, this is not summer vacation. Because certain tasks or sets of tasks can be done in parallel with other tasks or sets of tasks, often one of the parallel paths will be shorter in time than the other. The difference between the path taking the longer time and the path taking the shorter time is "slack time", and this value is an attribute of the shorter path. The bottom path in the figure has a slack time of one day. This means that if the tasks on this path together take an extra day it will not affect the length of time to finish the product.

Critical Path The critical path is the path from project start to finish (or between a set of endpoints common to the paths under consideration) that use all the available time of the shortest-time path, and thus is the path that has no slack time. The critical path in the figure is shown in red. Any time slippage in the tasks on the critical path will change the overall time to finish the product.

Resources Resources are anything you need to get the work done. People are resources, but have to be separated according to their skills – having 5 plumbers on a project is not going to help you get your walls plastered. Tools are resources. Sometimes they can be associated with the people that use them, as you might expect plumbers to have pipe wrenches and soldering equipment. Sometimes there is a specialized piece of equipment that is treated as a resource – a crane, for example, would have to be available when you needed materials moved to the top of a high-rise building under construction.

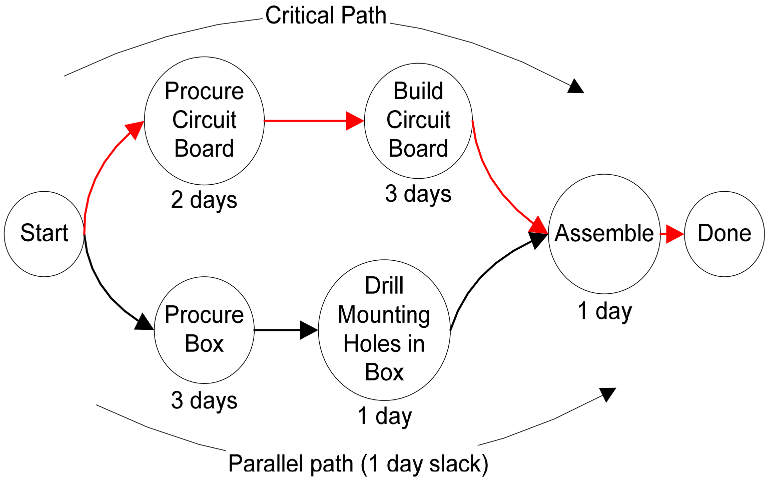


Figure 10-9 PERT chart of product assembly with two paths

Levelling Resources are assigned to tasks on a project. Often these resources are people, but sometimes they are equipment or facilities. By balancing the assignments between resources with the same capabilities (as far as any individual task is concerned) we can maximize the productivity. We can also determine if resources are over-allocated, in other words if we are expecting them to do more work in a given period than they are capable of.

Switching allocations to give the most uniform work allocation is known as **Resource Levelling**. At its best, levelling will result in all resources being used 100% of the time while we are paying for them. In actuality this seldom works.

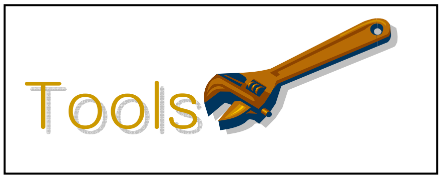
Resource levelling also applies to distribution of a database over a server farm for such companies as Google and allocations of funds in a stock portfolio, as well as making tomato sandwiches and assigning tasks to front end loaders.

There are two more important concepts that relate to the monitoring of the project. The first is the **milestone**. A milestone is not an task or set of tasks, but an event. It is the event of completing some major piece of work, such as a circuit board or a software library. Up to these points the work is partially complete, and how much work remains is not precisely measurable. However, since the milestone is associated with 100% completion of the work, it is a good measurement point.

The second important concept is a **gate**. These are milestones at which project decisions are made. Such decisions involve many stakeholders of the project, and are typically go/nogo decisions or decisions about the path the development will take, usually determining the resources, money and time to be devoted to the project.

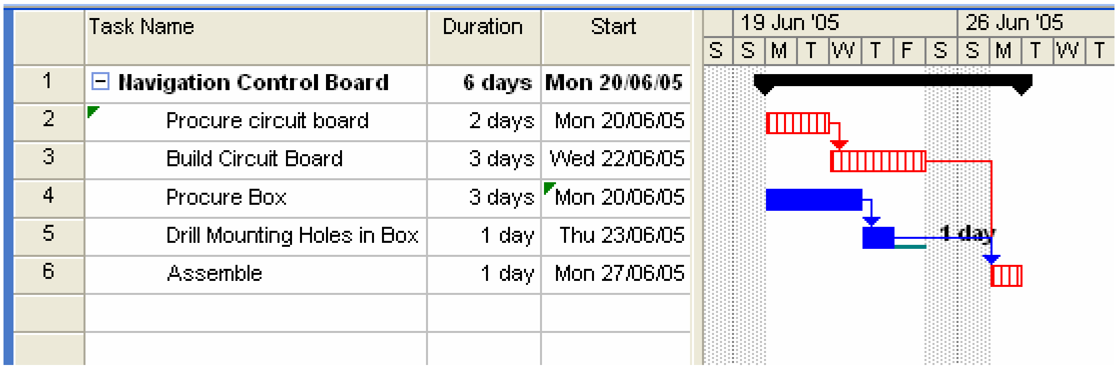
By clearly defining milestones and gates the developers are freer to make the many small decisions involved with the work without constant schedule-based justifications to management, and the managers have points of time where it will be clear whether or not the developers are to schedule, and any necessary adjustments can be made to bring the project back into line.

10.5 Project Management Software



There are many kinds of project management software. One example is Microsoft Project® (MP). Others work in a similar way, some having additional features specific to certain industries.

In Figure 10-10 we can see the way MP shows a Work Breakdown. The dark black bar in the figure shows that "Navigation Control Board" breaks down into the tasks from Figure 10-9. It also shows the sequence of these tasks and the Gannt chart at the right shows the dependencies generated because of sequence.



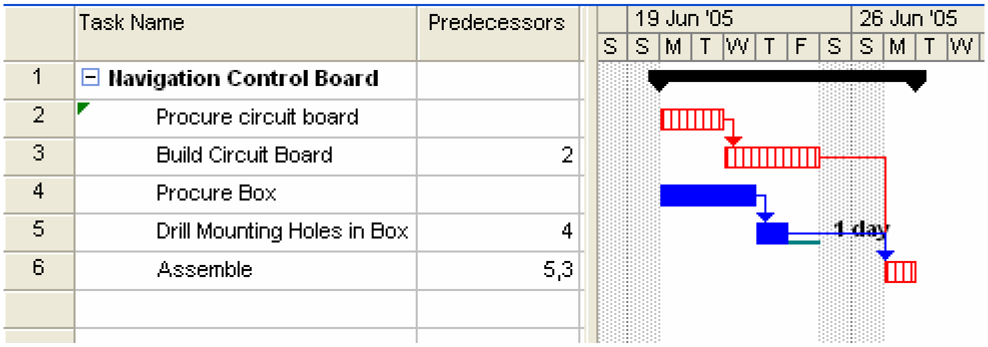
	Task Name	Duration	Start
1	⊟ **Navigation Control Board**	**6 days**	**Mon 20/06/05**
2	Procure circuit board	2 days	Mon 20/06/05
3	Build Circuit Board	3 days	Wed 22/06/05
4	Procure Box	3 days	Mon 20/06/05
5	Drill Mounting Holes in Box	1 day	Thu 23/06/05
6	Assemble	1 day	Mon 27/06/05

Figure 10-10 Example of partial project breakdown in MP

In the figure we have assigned durations for each task. Note that this might be different than our loading on any particular resource. For example, 'procure the circuit board' may take the two days assigned as duration because it takes two days to get a board delivered from the board manufacturer. The person that placed the order is free to do something else while the board is en route.

Note that the critical path is striped, and the non-critical path has a slack time of one day as noted on the Gannt chart.

As well as the duration of each task we also have to supply sequencing information. This is done by specifying the predecessors of each task. This is shown for our example in Figure 10-11. For example, we cannot assemble the device until we have the holes in the box and the circuit board is built, so the "Assemble" task has these two tasks, with IDs 5 and 3, as predecessors.



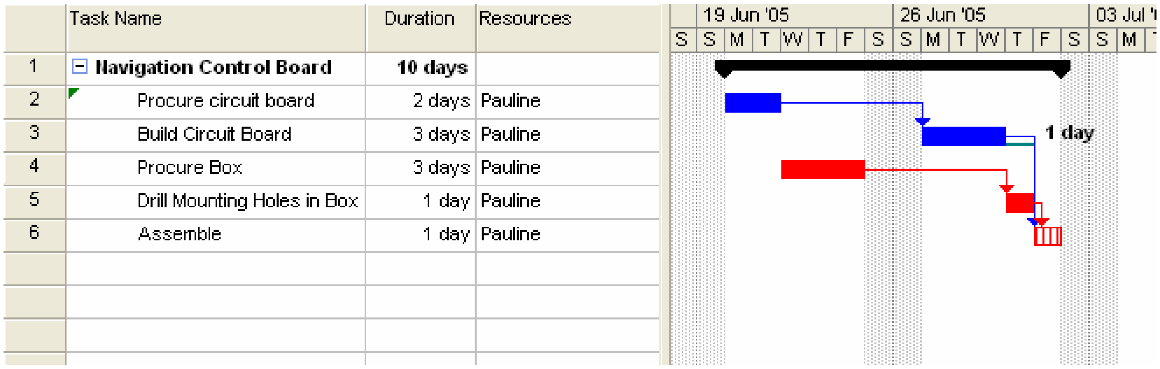
	Task Name	Predecessors
1	⊟ **Navigation Control Board**	
2	Procure circuit board	
3	Build Circuit Board	2
4	Procure Box	
5	Drill Mounting Holes in Box	4
6	Assemble	5,3

Figure 10-11 Microsoft Project Example - Sequencing

So Figure 10-11 indicates that we can do the project in 6 working days. However this assumes that we have the resources to be able to do any tasks in parallel where this is possible. What would happen if we were not able to do this?

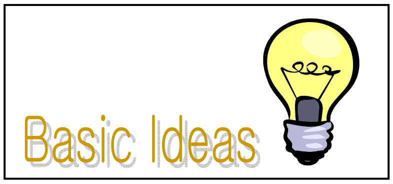
In Figure 10-12 we have a resource, the employee Pauline, who is going to do all the work. We assume that all the "Duration" times must be done by Pauline. We then ask MS to level, meaning that it will try to assign the resources optimally, but since there is only a single resource, the tasks end up being done one after the other. This revised schedule has our board taking 10 working days instead of 6. The critical path now has no meaning because every possible sequence from start to end will take the maximum time. There is no parallelism possible without adding more resources.

Let’s hope Pauline doesn’t get sick!



	Task Name	Duration	Resources
1	⊟ **Navigation Control Board**	**10 days**	
2	Procure circuit board	2 days	Pauline
3	Build Circuit Board	3 days	Pauline
4	Procure Box	3 days	Pauline
5	Drill Mounting Holes in Box	1 day	Pauline
6	Assemble	1 day	Pauline

Figure 10-12 Microsoft Project Example - Resources and Leveling



10.6 Project Management (again)

There are a lot of tools available for product management. You have seen a few, and should not underestimate the power of a spreadsheet or word processor in your toolset.

Developing the project plan is a lot of work. The concepts are not that difficult, but finding all the numbers and plugging them in is very time-intensive and is not as easy as you might first think. Usually it is done badly – a study from the Standish Group [7] indicated that of IT projects about 20% were on time, on budget with full functionality. About 30% were cancelled because the overruns were so bad. Of the rest, they averaged 100% over the allocated budget, 100% over the allocated time, and delivery of only 66% of the functions originally to be delivered.

The reasons for these failures are many. Sometimes the requirements are wrong, and the wrong artifact is built. Often the budgets are wrong and it takes significantly more cash, more resources and more time than was allocated. All these can be reasonable, however, and the project will fail due to human factors: poor team morale and lack of interest in the project. Common factors are lack of executive support for the project, over or under management by project managers and interpersonal conflict. The human factors manifest themselves in lower productivity, in transfers of people out of the project and by inattention to detail.

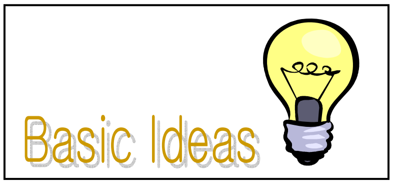
Planning and Costs

Planning, of course, takes place at all stages of the project. Here we are only looking at the planning involved with managing the detailed design.

Creating the plan and managing the project according to the plan now consume almost the same amount of time. Typical planning time for a project are now in the range of 35 to 55 percent of the total management hours spent [1].

We will not cover these human aspects here, or the methods of managing people. Anyone coming into a position of authority should learn these skills [1].

As a last word on project management: By the time we reach this stage, we have spent a considerable amount of time on setting up a project. You have to continue the effort through the life of the project. You want to know when the predictions have proved inaccurate. There are two reasons for this. First, these problems can result in a change in strategy. There is no point in having a plumber come if the building to be plumbed does not yet exist. The second reason is that this project will not be the only one you do. The lessons you learn because of problems that occurred should be applied to future projects. Everyone gets burned – the wise person only gets burned once.



10.7 Producing Better Estimates

If, as we saw in the last section, creating a WBS, and time and cost budgets is difficult and prone to omissions, what can we do about it? Here are some “Use Case”-type considerations to help you produce better estimates:

- Usual operation. This one is generally complete – everyone knows how their hardware or software will be used when up and operational and being used as expected
- Erroneous inputs. Users and connected devices will often misbehave. Your system must be able to tolerate inputs that are not what is expected.
- Infrequent conditions. A word processor is used almost all the time for entering letters. Very little time is spent, for example, in changing a heading style. In creating the word processor program, however, you will spend probably as much time on the routines to adjust the styles as the routines to do normal text entry. In estimating time you must consider the time to do the design work, not the proportion of time in which the design work will be used.
- Startup. One infrequent condition of special note is startup. Hardware and software can often start in strange modes if not properly initialized. Ensuring the proper conditions after startup can be a long task that should not be ignored.
- Testing. Often testing is assumed to be a simple task, yet often the creation of tests and of test facilities will take as long as the development of the untested artifact itself. In some cases, banking software for example, there is a significant multiple of the development time spent in testing

Personal Time Management

You can make the best use of time by doing the following:

- Handle email, letters and small problems once – deal with them right away so you don’t have to worry about them again.
- Handle short-term, important items first then long-term important items. Leave short-term less important items for later. Ignore long-term less important items → time for a break!

- Informing, organizing and management. If your job is entirely involved with completing technical tasks, assume you will spend at least 40% of your time on communication-related tasks. This will go up as you become head of a team, or a project.

Here are some common mistakes from those doing their first estimates:

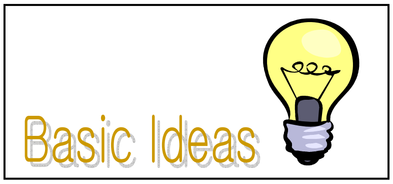
- assuming 100% of time is on development. [See the last point above.]
- assuming perfection first time through. [in practice you can expect to do considerable rework & adjustments as problems and oversights are found]
- assuming no change in specs [“Didn’t we tell you the customer changed his mind again??”]
- assuming simultaneous activity from same people [“When I asked you how long you needed for each of these tasks you said a week, and it’s now been a week….”]
- assuming that required resources are available [“So what if the parts aren’t here – build it anyway!”]

And here is a checklist of considerations for your budgets:

- □ include all of the resources for all of the time for the project, including contingency time. Be sure to include all personnel including supervisory
- □ differentiate between elapsed & “hands-on” time, with no multiple allocations
- □ consider taxes & duties & shipping & storage
- □ allow for breakage, “stealing”, damaged parts
- □ allow for long shipping times
- □ **allow for mistakes**

Repeating: Budgeting is not easy. Some more information to help you with budgets is in Appendix C.

10.8 Use of Budgets Moving Forward



These charts and budgets are of use getting funding and assembling the necessary resources to get the project started. They are also of use in tracking the project to make sure we do not fall behind. The project manager should review the project as it goes, checking at each milestone to make sure the costs and times are under control. The customer should be aware of the overall progress of the work, particularly if there is any hint that the project might be becoming irretrievably behind schedule.

At any point in the schedule one can ask the following questions:

- At this point in the schedule of events if everything were done as planned, what should I have spent to get the scheduled work done? This could include work not done that should have been done.
  [Budgeted cost of work scheduled, **BCWS**]
- At this point in time, how much have I spent to get what was done done? Only work done will be included in this calculation.
  [Actual cost of work performed, **ACWP**]
- At this point in time, how much did I budget for getting what was done done? Again, only work done will be included in this calculation.
  [Budgeted cost of work performed, **BCWP**]

We can calculate a number of values that will help us determine how well we are tracking our estimates and schedule.

CV = BCWP-ACWP This difference is known as the cost variance, and it is how much you have you overestimated the costs for the work so far. Of course, if this number is negative, you have overrun your budget.

SV = BCWP-BCWS This difference is the schedule variance. We can have a cost variance of zero, meaning we are paying what we expected for everything, but have a non-zero schedule variance because tasks are being done faster than we expected (giving a positive schedule variance) or we are behind schedule (and have a negative schedule variance).

Since these variances are numbers that will vary with the size of the project, it is often useful to work with the percentage of the variance of the budgeted values:

CV% = CV / BCWP and **SV% = SV / BCWS**

or by using ratios, known as performance indexes:

Cost performance index [CPI] = BCWP / ACWP

Schedule performance index [SPI] = BCWP / BCWS

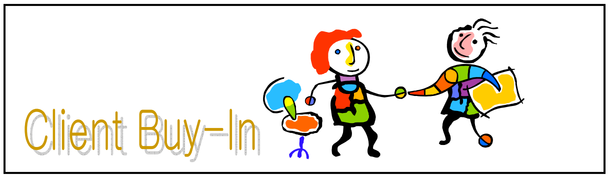
If these indexes are greater than 1.0 we should be happy.

BCWP is difficult to determine, since normally the tasks are partly done and estimation of amount complete is difficult. See Murphy and Pareto under Laws, Appendix D.

When you determine that the project is behind, first realize that you are in good company – most projects get behind. Then determine if you need to do anything about it. If, realistically, the variation is not going to be important, you may wish to leave things as they are. If not, there is work to be done.

Generally speaking you can trade money for time and time for money. If your time schedule shows the artifact delivery being late, you can usually spend more and get it less late. This may mean overtime, extra people or more equipment. There is a limit to this, since equipment won't arrive instantly, new people need training and there is a limit to the amount of overtime a person can or will work. It also becomes increasingly expensive, particularly if the matter is urgent. Trading time for money generally involves letting the delays pile up and dealing with the fact that the project will not be done on time.

All projects run into the unexpected and the uncontrollable. Stating the obvious: If you plan well, these problems are mere speed bumps. If you don't plan well, the problems will become crises and the crises will take over and define your project, and the consequences will be generally discouraging. Also very important is to determine if you have a simple slippage or a trend that will result in increasingly greater slippage as the project progresses.



If you determine that your project is slightly behind schedule and you can easily catch up, the client need not know. If you determine that your project will be delivered significantly behind schedule despite any extra efforts, make sure your client knows early. You probably should deliver the news along with information about the causes and with descriptions of the solutions you are mounting to minimize the delay in delivery. Your client, who is creating a system which includes your system as a component, will want to make adjustments and it will be easier for your client to make these adjustments if there is more time to do so.

10.9 Go/Nogo Decisions and Determination of Price to Client

For an internal project this is the stage where a decision is made to continue with the project, or to stop it as it is not in the company's best interests. For an external project, the decision often is whether to accept the project or not, or is what the contract price to the client will be. To make these decisions we turn back to our basic key elements: money, resources and time.

Money At all points of the project life, the amount of cash and credit available to your company must be enough to sustain the project (and all other company activities). At the end, the company should be in a better position financially than they would be had the project not taken place.

Resources At all points of the project life, the company must be able to put together the resources to be able to sustain the project. If there are not enough qualified people, not enough equipment of the right type, and not enough space and similar, and these cannot be found in time or within budget, the project should not be continued.

Time Most projects must meet a completion time for the whole project and for certain major milestones. If these cannot be met, the project should be reconsidered.

A note on situations where you are deciding on a contract price to a client: Engineers often do this very poorly. *The contract cost should be based on the cost of the alternative.* Notice that this does NOT mention how much it will cost your company to fulfill the contract. The cost to your company determines the go/nogo decision. If, as the only alternative, the client can purchase ABC software program to do the job for $250,000, then your price should be less than this. If this will not make your company profit, you should exercise a nogo decision. If the costs to your company are much lower than the alternative, that's great – you have a lot of room to play with and may want to give the client a better price to buy goodwill, but there is no particular

Summary: Budgets and Project Management are involved in determining if the project will proceed and keeping the project on track.

reason to lower the price otherwise. In brief: The cost of the alternative determines the price to the client; the cost of the work in relation to this price determines the go/nogo decision.

A project does not fail if no suitable solution is found. It is a great mistake to move forward with a design where the risk of failure is too high for the company to tolerate, and a credit to the decision-makers and to the engineers involved to stop the project at an early stage. The basic budgetary information is fundamental to the go/nogo decision, as is the determination of risk as covered in the next chapter.

Basic Ideas

Chapter 11 Risk Analysis and Risk-Based Decisions

Engineering decisions should first be moral. When our decisions put the public in danger or are fraudulent, these are not good decisions. However, most decisions will generally become monetary decisions.

Often the technology will exist to solve almost every technical problem that you try to find a solution for. However, not all problems can be solved for under a given cost.

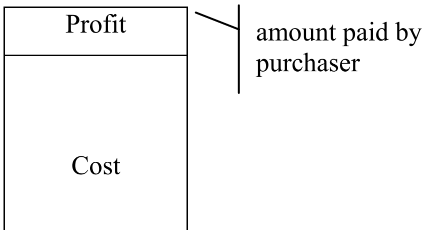


Figure 11-1 Cost and Profit with no Risk

When we evaluate a decision we should take risk into account. If there is no risk, the simple model shown in Figure 11-1 applies. Doing the work has a cost, which is fixed because there is no variability (which is risk). If we take risk into account, as in Figure 11-2, we could end up with a larger profit as at the left side of the figure (since some less likely events happened in our favour), or the cost could even exceed the amount paid by the purchaser, in which case we have lost money, as at the right side of the figure. Not understanding risk means we are not sure how much of a chance we are taking, and how much we could potentially lose.

In this chapter we are going to take a first look at risk. We will look at how to calculate it and how to reduce and avoid risk in common electrical/computer engineering situations.

Every time you move from one step to the next in a project you take a risk. At best, you are risking your time. At worst, you are risking the success of the project, and the resulting consequences of this failure. If your preparations to take the step are inadequate, you may ultimately have to throw away work done and backtrack.

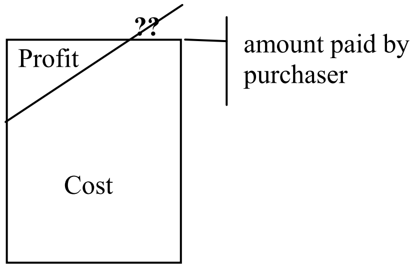


Figure 11-2 Cost and Profit with Risk

Every time you stop at one step and decide not to move to the next right away, you also take a risk. Again, the success or failure of the project may be total time critical and while you pause you are using time and resources and might be losing sales.

How do you make the decisions? You could spend a lot of time researching, exploring the design and the necessary parts in great detail, thus reducing the risk that the design is not adequate but using up considerable time and resources to do so. Alternately, you could cross your fingers, make your best guess with very little work and hope for the best. This approach would have higher risk that the design could fail, but would get you going faster. A compromise between these positions, based on the risks involved, is the best approach.

Consider that you are working on an experiment to be included in the payload of a rocket to be launched on a certain date. If you do a very meticulous testing, you will have a strong guarantee that the experiment will work as desired and a very strong guarantee that you will miss the launch. If you do no testing, you will have a strong guarantee that the experiment will be launched, and very weak assurance that the experiment will work. The best course of action will be one that meets the launch deadline, but with a payload that has a very good chance of performing correctly.

Summary: All engineering situations involve risk. Risk can be reduced by avoidance and by technique.

Applicability: Risk can be used to evaluate designs and to make project management decisions about a design.

Avoidance vs. Mitigation

There are two ways of dealing with risk. First, you can find an alternative that avoids the risk. Instead of jumping over the hole you walk around it. Second, you can mitigate (reduce) the risk. Practice jumping until your likelihood of getting over the hole when you attempt to jump it is higher. When reducing overall risk a combination of the techniques is used to reduce the risk to an acceptable level. Risk can never be reduced to zero.

Where is the risk?

The risks in a project can have various sources. They can come from outside the company, from within the company but outside the project and from within the project itself. Here are some project risks:

1. Failure to deliver. The project is cancelled because of an internal problem or by the customer due to a circumstance beyond your control. For example, the customer could cancel half the order or a bank could cancel a credit line so there is no funding to finish the project. Often a project is undertaken because of a certain market expectation, with the risk that this market expectation will not be realized.
2. Failure to perform. Many projects are delivered with less than all requirements met. Sometimes this is a time constraint (a delivery time comes and not all the functionality has been implemented) or it could be a technical constraint (a technology did not work as promised, or there were unexpected problems, such as noise, that limited the functionality).
3. Client dissatisfaction. Even though a project may meet its requirements, if the performance is less than the client expected, this is not desirable. This is an issue that started with inadequate requirements generation and with not keeping the client adequately involved early in the development process. You should bear in mind that a good deal of your companies business will be repeat business with existing clients, or will be with clients referred to you by existing clients. Unhappy clients cut these links.
4. Overruns. Unanticipated problems, extra costs or slower-than-expected development result in more time or more cost than intended.
5. Changes in constraints. Laws change and there could be environmental, reporting, safety and other requirements that must be met with money and resources that were not factored into the original estimates.

Table 11-1 gives some typical sources of risk and the consequences. Note that these sources extend through the lifecycle of the artifact.

Where	Example Problems	Consequences
In Design	• requirements unattainable in timeframe/budget • component delivery slow / components unavailable • requirements inadequate	• failure to deliver • delivery late or over budget • hard to produce / maintain
In the Field	• requirements do not match customer needs or expectations • system does not meet requirements • failure of system components	• mission failure or partial failure • downtime • repair / rework costs
In Production	• failure to perform assembly tasks • failure of parts • damage / failure in shipping or storage	• warranty • liability • no resale's

Table 11-1 Risk Sources and Consequences

Reducing Risk

Risk can be reduced. Part of taking engineering courses is to enter into a design situation with the basic knowledge and abilities that will help you to make lower-risk designs. Risk can also be reduced by following a careful design process with good requirements generation that keeps the customer close to the process. Risk can be reduced by careful, methodical testing that reaches back to the original goal and requirements of the project. Often designers are too close to the project to see potential problems – a third party review will often reveal these problems and reduce the risk. In summary, technique and training are your main platforms for risk management.

Most of the risk over which we have control as engineers is technical risk. Here are some other techniques for technical risk reduction. Explore them further in projects where they could prove worthwhile:

COTS COTS is Commercial-Off-The-Shelf, those components that are already produced and standard stock. Using components that are readily available can avoid delays in design and in production.

Reuse If you already have the code or the circuit designed, use the same design again. It is generally far faster than development from scratch. Caution: designs have failed because of reuse when the reused piece was not thoroughly checked against the requirements.
A version of reuse is “Design from Inventory” where you try to use parts already used by other products in the company.

Co-design Most system designs have hardware, software and mechanical components, each being designed by separate teams. There should be constant interaction and tradeoffs taking place among the teams.

There should be involvement with other groups that are part of the product lifecycle, including sales, manufacturing, maintenance and design testing.

Prototyping Building intermediate, reduced-function prototypes is of great advantage in reducing risk. The stakeholders get a chance to make comment on the design at a stage where change is still not overly costly (this is particularly important for user interface designs). You can check out alternative implementations of modules, or make sure that a certain design concept is going to perform as expected. You can more easily see potential problem sources and design to avoid them or reduce their impact.

One good design technique is fast prototyping where the development efforts are streamed to producing prototypes early in the project for customer evaluations and comment and for concept testing.

Design for Change Assume that your design will have to change in response to changes in user or regulatory requirements, or because of unanticipated problems. How can you select components or create an architecture that makes changes in design easier to implement? What documentation must be in place to allow you to make these changes later when the project is not current in your mind?

Programmable parts (processors and programmable logic) are conducive to future change. Generalized and under-utilized interfaces will also make changes easier.

Contingency Planning Contingency planning involves having a plan ready for problems that develop. Are the testing tools and the people ready to move on a problem should it happen?

To do this properly, you need to look at the “what if”s, examining potential sources of failure and creating the strategies to handle them.

Sample problems:

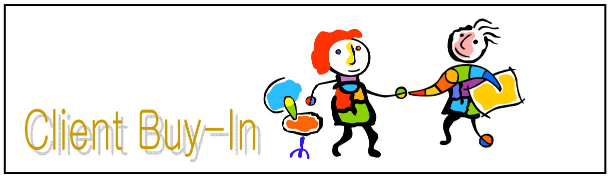
- parts don’t arrive
- software is incomplete

- parts or software purchased does not meet the requirements
- failure of parts or programs
- changes in specifications
- delays in some part of the development

Fault Tolerance: Soft failure, error recovery, limp-along and redundant design

There are methods of design that allow “soft failure” where there is not a catastrophic problem created when a failure happens. Nuclear plants, for example, are made to shut down carefully when a problem is detected. Some errors can be trapped and a system recovery is possible. You can also design systems that maintain some functionality in the event of a failure (aircraft are built this way) and where a failure is masked by redundant circuits. These are fault-tolerant designs.

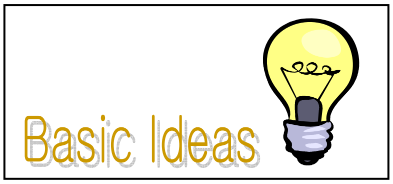
Fault tolerance involves detection and at least partial compensation.



At several places so far we have already talked about customer / user / stakeholder involvement. This can not be over-emphasized. Although there may be some day-to-day problems that you do not want your customer to know about, and although there also may be proprietary information that your company does not want generally known, keeping your customer in the loop will usually contribute positively to the ultimate success of your project.

The benefits?:

- better requirements specifications
- better concentration of development efforts on the objectives that are important to the customer
- faster capture of problems related to inconsistencies between design and customer expectations
- understanding of customer of development problems
- customer buy-in to decisions made



Calculation of Risk

If everything went wrong on a project, the company will (almost always) lose money. If we assumed that everything will go wrong, we would never start a project. Similarly, we would never buy a lottery ticket if we assumed there was no chance at all of winning.

But people do buy lottery tickets because there is a very small, but non-zero, chance of winning. And companies do take on projects because the probability of profit is seen as reasonable.

Probability has a concept called “expectation”. If we purchase a lottery ticket for $2, and the chances of winning are one in ten million of winning five million dollars, the expected return on our ticket is

Expected return = (win)*(probability of win) = $5,000,000 * (1/10,000,000) = $0.50

In other words, we have an expected loss of $2.00 - $0.50 = $1.50 on our ticket.

Projects are usually expected to be profitable. This gives us a way of looking at Table 11-1 so that instead of having a sloping line we can draw the financial picture as in Figure 11-3. The Risk Budget is determined as the sum of all the potential costs due to all the sources of risk times the probability of that happening, so it is the expected increase in costs due to failures.

Risk Budget = ∑ (risk cost) * (probability of risk)

There is a risk that a tornado will level the design labs and the project will have to be restarted. The cost of this is enormous, but the probability is very, very low, so the contribution to the risk budget is extremely small (particularly if the risk is mitigated through insurance coverage). On the other hand, the failure of a component could cause the system to fail, and thus cause mission failure – that is failure of the system of which this system is a part. The costs of the failure are again enormous, but the probability is not low enough to make this contribution insignificant. The product of this total cost and this probability will be a more significant contributor to our risk budget than the tornado.

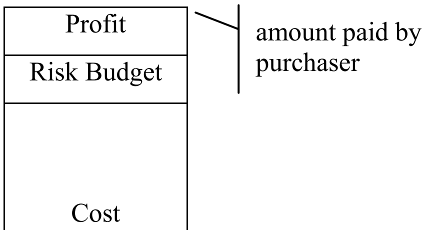
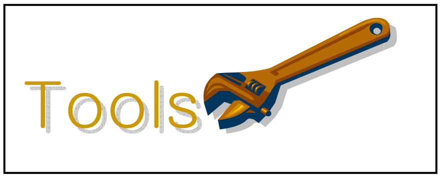


Figure 11-3 Cost and Profit with Risk Budget

Risk and the System Design

Like many aspects of the design of the artifact, use of risk in selecting a system design will be an iterative process. A quick determination of cost, risk and profit are measured against a client's expected payment for the artifact. A go-nogo decision is made. This is done over and over again with increasing amounts of detail in the risk model. The system design is chosen based on getting acceptable returns with acceptable risk. Acceptable risk involves iteratively choosing new designs or altered designs that avoid or mitigate risk. Eventually a design is chosen with acceptable risk and return, or a nogo decision is reached as the most cost-effective route to follow.



11.1 Snapshots, Use Cases and Risk

The usefulness of the Snapshot and the Use Case does not end with the system requirements. Once the system design is chosen, the Use Case diagram can be used to generate the functions that come about because of the implementation. For example, in the case of a hairdryer that works with 120V electrically-heated forced hot air we can revisit the Use Case discussed in **Error! Reference source not found.** to determine additional implementation-based requirements, such as

- shall not allow heat to build to a level where it might harm / damage the user (hand or scalp or hair) or any tool (such as a hairbrush) that might be used in conjunction with the hairdryer
- shall not allow user to be endangered by high voltages used in the device

Most of these will result from anomalous behaviour, that is behaviour out of the ordinary, although some may be expected to occur without precaution and some may be expected to occur with fair regularity.

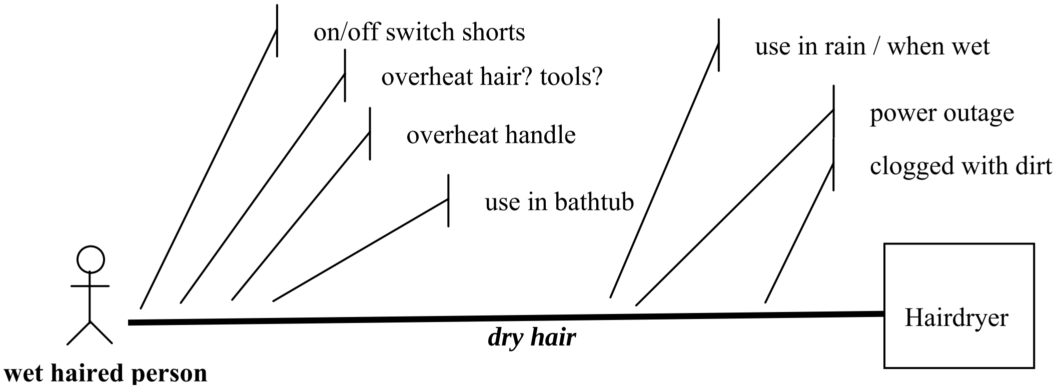


Figure 11-4 Risk Situations Connected with Use Case

Figure 11-4 shows some of the risk situations that might be associated with drying hair with a hairdryer. As can be seen, they can be expressed most naturally as Snapshots or low-level Use Cases. Sometimes the actual risk is implied ("use in the bathtub" implies the risk situation "gets electrocuted because of use in the bathtub").

Note that this only brings the analysis a certain distance by identifying the specific situation with risk. The next steps would be:

A. evaluate the cost of the occurrence

B. determine the probability of the occurrence

C. given the risk (cost of occurrence times probability of occurrence), decide how to deal with it:

D. ignore (incur the cost should it occur)

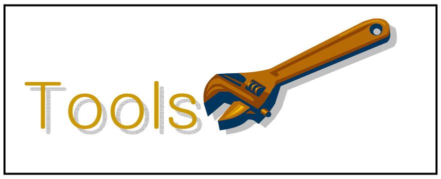
E. insure (a method of mitigation where the product will not change but someone else will incur the financial cost should the problem occur; note that there may be other costs that are virtually uninsurable such as loss of sales)

F. avoid (use batteries)

G. mitigate (build in an automatic shutdown, for example)

H. warn (in the instructions nobody reads about how to use a hairdryer)

11.2 Fault Trees and Implementation Evaluation



A fault tree [8] is used to show how a problem can lead to a consequence. The tree includes information about the circumstances under which the problem can result in the consequences. These diagrams are often used in forensic activity, meaning that we have a consequence (ex: the airplane crashed) and we are trying to find out the root causes. The consequences, therefore are normally listed at the top of the page and the potential causes at the bottom. We will maintain this orientation, but will create our trees from cause to consequence.

Cause events, also called basic events, are shown in circles. These are events or faults that ultimately may cause a problem.

Sometimes these causes will have no consequences because the problem goes no further. For example, if my computer is off it does not matter that my cat walks across my keyboard. On the other hand, if I am working on this chapter on my word processor, the cat walking across the keyboard may insert 0ponvbt54rzx which is not likely what I wanted at that point in my document. Such conditioning events, such as the computer being on, are shown in ovals.

Summary: A Fault Tree is used to illustrate the causes of failure. By integrating cost we can determine risk of failure.

Applicability: This technique can be used to mitigate risk by determining where a design should change and to evaluate potential system designs.

The cat-on-keyboard problem is shown in Figure 11-5.

Similar to conditioning events are "external events", also called "house events", which are events that are expected to happen. So if the cat presses the Caps Lock button there will be no consequence until I PRESS SOME KEYS. Pressing the keys is an external event. These are shown in houses, as with "keys are pressed" in the figure.

Since these diagrams can become very involved, a triangle is used to indicate continuation in a sub-tree.

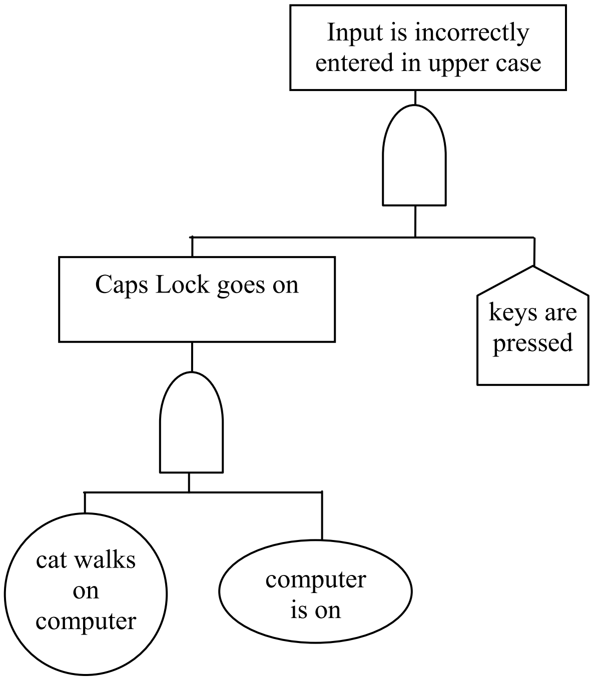


Figure 11-5 Example Fault Tree

A diamond is used to represent a set of events that is not further resolved. This could be because alternate methods of analysis are used for that type of event.

Consequences are given in rectangles.

The events are linked together using logic symbols, the same ones used in basic digital logic: AND, OR, NOR, NAND, XOR, NOT.

The diagram of the cat pressing the caps-lock key says that if the computer is on (conditioning event) –and– the cat walks across the keyboard (cause event) then the Caps Lock goes on. If this is true –and- keys are pressed (external event) then the input is incorrectly entered in upper case.

We can expand Fault Trees in a method of evaluation as in Figure 11-6. In this Risk Analysis Diagram the flow is reversed from the Fault Tree, with the consequences appearing below the events.

The top portion of the diagram is a Fault Tree. The upper nodes are relabelled Failure Mechanisms. This is done because most often we will relate a consequence to some type of initial mechanism. For example, the consequence “airplane crashes” may have a failure mechanism of “pilot fails to lower landing gear”.

In the diagram Failure Mechanism A results in three cause events: A, C and D. Cause event C also can result from Failure Mechanism B.

Further, cause event B results from cause event A.

Cause event B causes consequence A and cause event C causes consequence B.

To have consequence C we need to have cause event D (caused by Failure Mechanism A), and we must have cause event E (caused by Failure Mechanism B) and we must have External Event A.

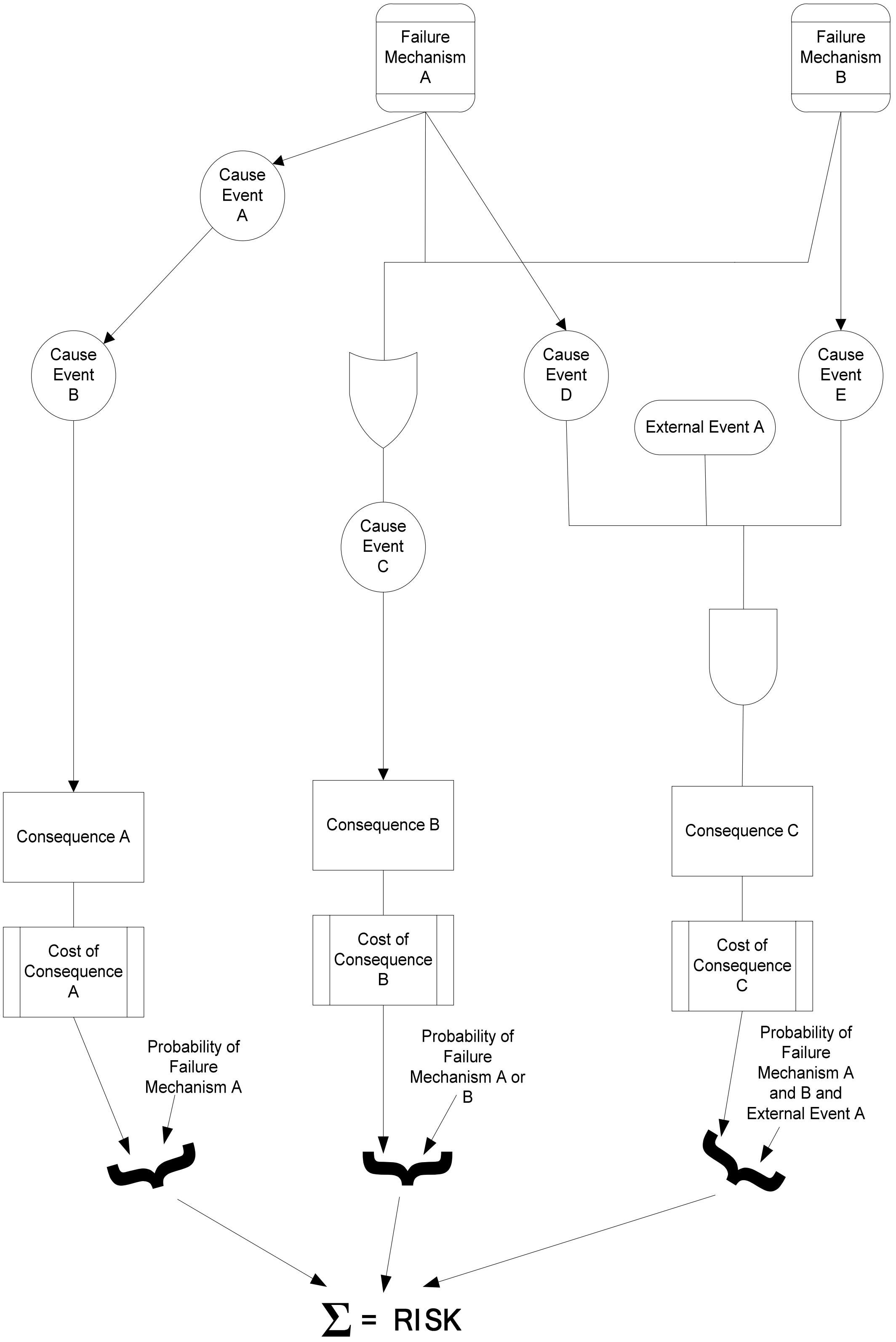


Figure 11-6 Risk Analysis Diagram

The consequences are what cost the project money. If the fan fails on your Pentium chip, a replacement fan may be only a few dollars. You probably will never be directly aware that the fan has failed. If the failing of the fan results in the failure of the Pentium chip you will notice very quickly, however, and the failure will be quite costly. It may generate other problems such as lost files and it could have larger costs due to lost opportunities since your computer is down.

Now we can determine the risk from the Risk Analysis Diagram:

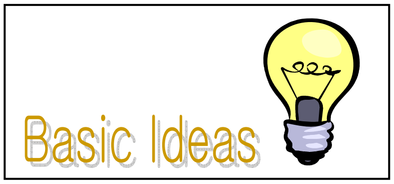
- To each of these consequences we can apply a cost if the consequence happens. This is the last set of boxes in the diagram.
- If we backtrack through the **and** and the **or** gates back to the failure mechanisms, and find out the probability of those failure mechanisms, we can find the probability of the consequence. In the figure consequence A only happens when we have Failure Mechanism A, so the probabilities of these two are the same. Consequence B can result from either Failure Mechanism A or Failure Mechanism B, so the probability is the sum of the two (assuming they are independent failures). For Consequence C we need to have Failure Mechanisms A and B and External Event A. Again assuming independence, the product of the probabilities of these three events will be the probability we assign to Consequence C.
- For each consequence we now multiply the cost of the consequence and the probability of the consequence happening. These are our expected risk due to that consequence.
- Summing the risks from all the consequences gives us the expected risk for this implementation.

We can do the same for other implementations to compare them.

We can also change the risk by making alterations.

1. We can reduce the probability of the failure mechanisms by using better parts or construction techniques. This will reduce the value of the probability used in the risk calculation.
2. We can introduce detection and shutdown mechanisms that prevent the failure mechanism from propagating downwards. For example, we can introduce buzzers and lights if the landing gear is not down when the flaps are used (as is normal when landing), or we can automatically put down the landing gear when the altitude is within 100m of the altitude of the destination airport.
3. We can change the architecture to remove the failure mechanisms or cause events. For instance, we can design an aircraft that always has the gear extended. This has other ramifications, of course.
4. We can introduce redundancy so that, for at least some of the event sequences, multiple failures must occur for the consequences to happen. If we have 18 wheels on our transport trailer, the consequences of a single tire blowout are greatly reduced.

The example in the next section will show how some of these alterations are done.



11.3 Example: Working with Risk

Let's look quickly at a sample design where we want to reduce risk.

Say you design a circuit involved in sequencing dynamite charges that will bring down old buildings. Part of the design involves a firing circuit to cause the dynamite to explode. How do you design that circuit? If there is a 1% chance that the circuit will fail and blow the dynamite up prematurely, is that acceptable? Is 0.1% acceptable? 0.01%? How can we tell? Is 0% the only acceptable figure, and can that be achieved?

Our circuit diagram is a model of the actual circuit. We can analyze the failure rates of the components, predict the outcomes, combine all these probabilities of failure causing detonation of our dynamite and come to some conclusion as to how probable it is that the circuit fails. In Figure 11-7 we have a switch (in a fancy plunger box) which we close to complete the electrical circuit and blow up the dynamite. We find the probability that the switch fails closed (for now we will not include the risk that it is prematurely pressed) and the probability that connecting wires short[11]). This value is the probability of failure of this circuit in a catastrophic way.

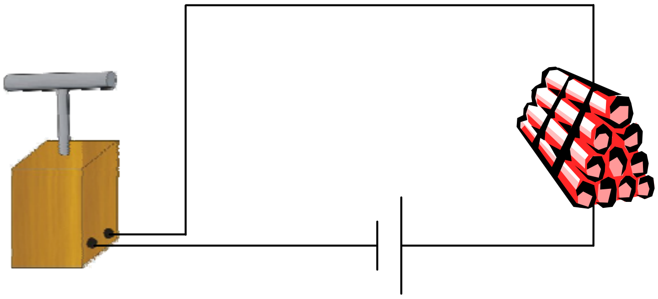


Figure 11-7 Circuit to Trigger Dynamite

The **risk** is the cost of the failure times the probability of the failure. What is the cost if the dynamite blows up when it wasn't intended? There could be deaths, probably lawsuits, property damage, loss of the circuit-provider's company reputation and possibly the end of the company as well. The cost could be measured in the millions, some of which would be covered by insurance, some not.

We will assume, to make analysis easier, that all of the risk is associated with the switch sticking closed, so that the dynamite will blow as soon as the switch is attached and before we want it to.

If the final cost of the failure is $5,000,000 (to pick a number) and the probability of failure is 0.1%, then the risk is 0.1% of $5,000,000 or $5,000. This would have to be added to the cost of the circuit to the customer, meaning the customer would be paying over $5,000 for a switch and wires[12].

This is likely unacceptable. Since the cost of failure will not change, we need to drive down the probability of failure or the cost of the failure.

One route to go is to duplicate the circuit. Using our figures from before, the circuit of Figure 11-8 has a probability of failure of 0.1%*0.1% or 0.01%[13] so our risk would come down to $500.

We could also use fuses instead of electrical detonation and completely eliminate the risk caused by a faulty switch. Using fuses would, however, create other risks and may not be appropriate for the problem of bringing down an old building.

[11] We will ignore the overlap of probability that they both fail closed together, since coincident failures is generally very small UNLESS there is a common cause (like flooding, an earthquake or malicious actions.

[12] Actually, we haven't included the costs of the physical components since they are so small in comparison to this liability figure.

[13] Again assuming independence of failure

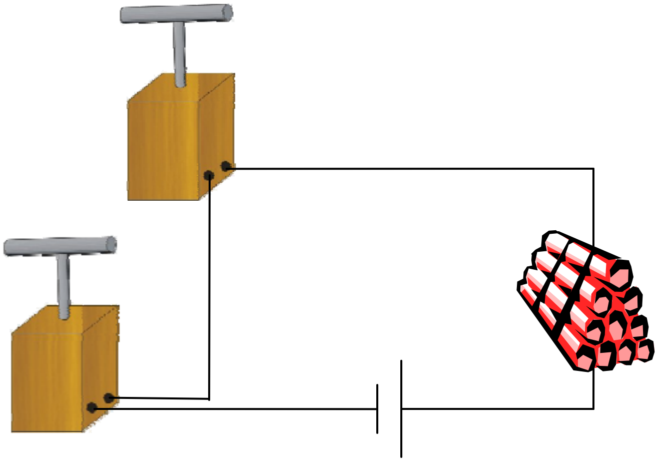


Figure 11-8 Double Switch Circuit

A better route is to include a test circuit in the design that shows the operator that there is a short before the dynamite is connected. Then a failure would have to occur after the connection. A simple modification to do this is shown in Figure 11-9. With a more sophisticated circuit than shown a failure could be detected and the circuit made inoperative.

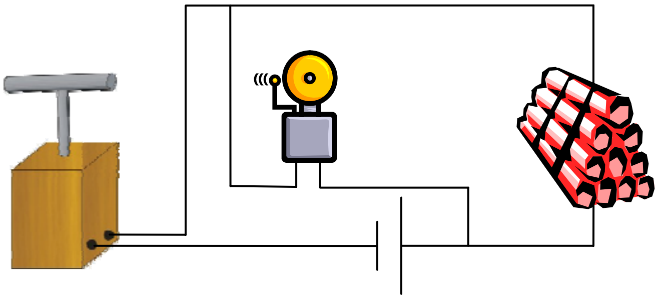


Figure 11-9 Circuit with Failure Indicator

We can also reduce the cost of the failure by operating procedure. We can ensure that, before we connect the switch, the dynamite be placed, and the area cleared. The risk is less because the cost of failure is less.

Do these reduce the failure probability and therefore the risk to zero? Again, no. There is no method of reducing risk to zero, only ways to make the risk very small.

This simple example shows how we can deal with risk through mitigation, and some types of risk through avoidance. It should also be clear that no method is going to be risk-free.

Chapter 12 Detail and Execution

12.1 General Structure - Detailed Design

Looking at our development model, shown in Figure 12-1, we can see the flow from the previous stage into this, our second stage of the three-stage model. From the previous stage we have decided, at a system level, what our solution will be. This includes the breakdown of the solution into a number of modules which will work together to effect a solution. The next step in finishing the project is to complete the design and implementation of these individual modules.

Summary: At the Detailed Specification stage the modules are designed in detail, synthesized (built) and tested independently. The requirements for the modules flow down from the system design.

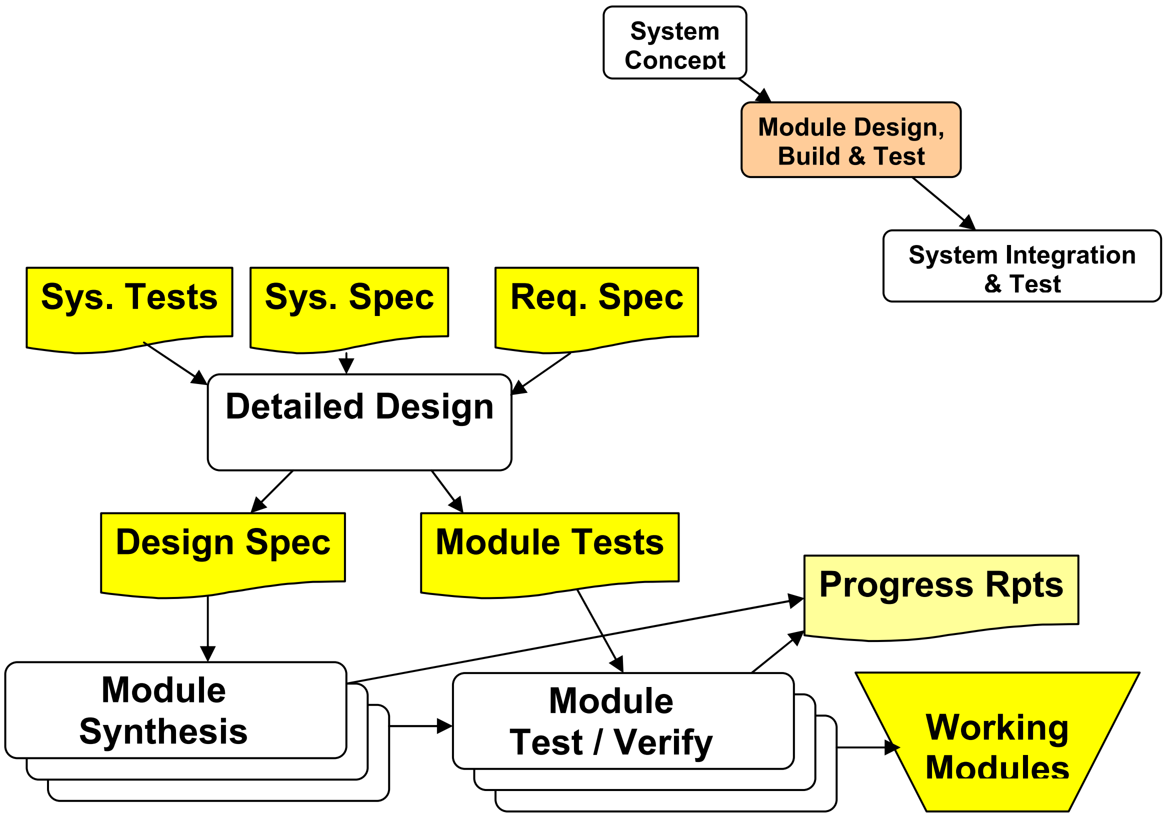


Figure 12-1 Module Design, Build and Test

To define the modules we had to know the limits of the technologies we would be using and place reasonable expectations on the results. In the Detailed Design we must go “hands-on”. We will need detailed working knowledge of our development tools, we will have to calculate precise component values, we will have to create the schematics and we will have to create the software programs. The level of knowledge required changes from a system level to a detailed level.

The System Specification defines these modules. The design of any module must also conform to the Requirement Specification and must allow the System Tests, as determined in the previous stage.

We will synthesize these modules, then test and verify them independently. This will usually take much longer than the work to this point, and can be handled by engineers with less broad-system experience. This move away from the system to the specific emphasizes the need to have done the previous work well. Any design flaws will now be implemented, and will probably not become apparent until a significant amount of work has been done.

Since we are now striving for parallelism and independence in the design/test of our modules, we pass requirements on to each module team. Each module becomes its own design or acquisition project. For a large system, these module designs will become formal design projects for the group that does them. They will go through their own stages of Requirements Analysis, Systems Design, Detailed Design and System Integration.

We must maintain communication with others at the system level. Even a small project will generally involve periodically informing others of progress. In a larger project this is usually done on a more formal level with Progress Reports that might be done weekly. These allow problems to be identified before they turn critical. They also allow project managers to take care of budgets and timing. Lastly, they provide a way for project managers to interact with their people, to address their concerns with the development.

Real Life: Good designers usually realize that even excellent specifications may have shortcomings. They will meet with other designers responsible for adjoining modules to make sure they have accurate and complete information, and sometimes to make tradeoffs to speed the system development, or to make the system better.

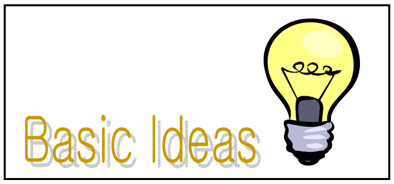
Module Tests

The Module Tests are developed in this stage. These module tests should

- test basic functionality
- verify that that the module meets the Design Specification for that module

If the Design Specification was written correctly then adherence to these requirements will mean that the system meets its requirements.

Testing will be discussed in greater detail in the next chapter.



12.2 General Structure - System Integration

Once the modules are done we put them together. The theory is that the system then works as expected; the practice is that there are generally some oversights that have to be dealt with. If we did our early work well, however, these oversights will be small and we can move from the System Integration step to testing and verification and from there to delivery. In Figure 12-2 we see the details of these steps.

Summary: At this final stage the modules are brought together and the system is made to work.

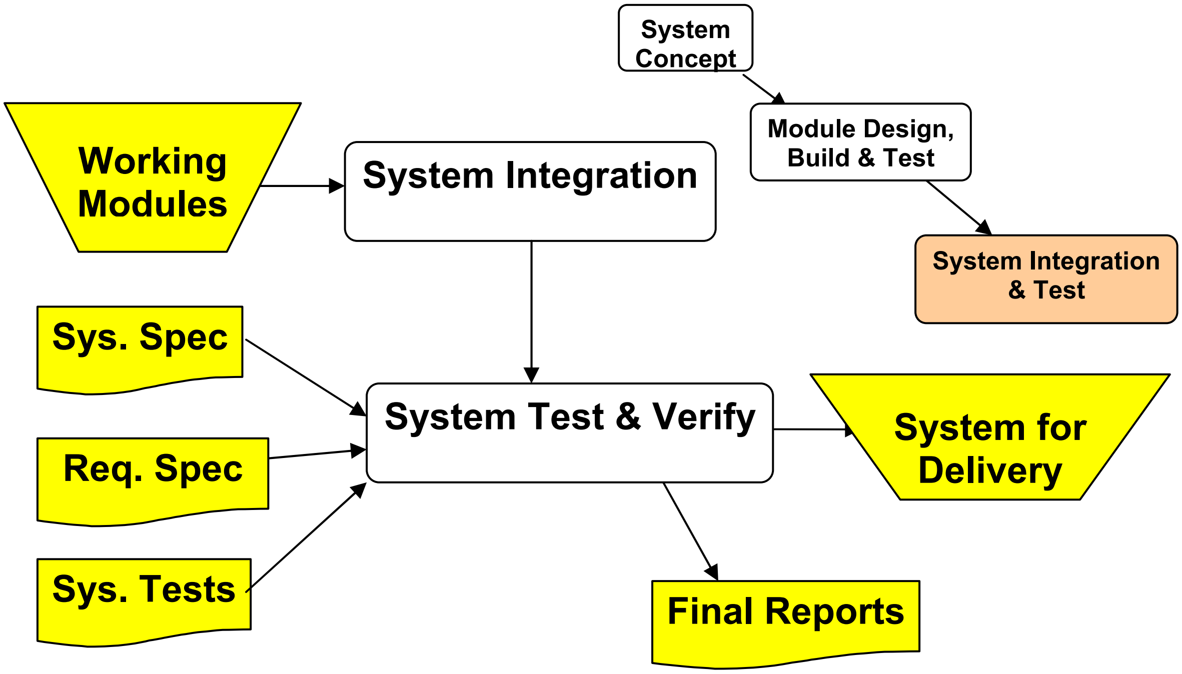
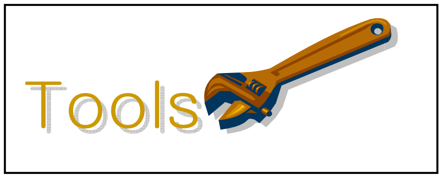


Figure 12-2 System Integration and Test

One of the outcomes of the process are the final reports. There are many different types of final report, depending on the artifact and the requirements. Technical, development, user and maintenance manuals could all be considered here. Production information could be included. Recommendations for future development could be a part. Any concluding communications that tie off the development can be included here. Oral presentations, conference papers and job applications could all result from the end of a project.

The Test and Verify step is very important. This step could include client acceptance tests or other stakeholder signoffs. It definitely will include the checks to make sure that the system fulfills all of the requirements. There is a backward thread linking this step to all the others; we will explore this in the next chapter.



12.3 Development tools

Development tools are very important to the engineer during the detailed design step. These include very common tools such as oscilloscopes and compilers, and also some very expensive ones such as CASE (Computer Aided Software Engineering) tools and code analysis tools. While the discussion of each could be a textbook by itself and so is beyond the scope of this book, some brief comments are in order.

Buy or Not to Buy Development tools cost money. However, measured against the cost of engineering time, and the cost of project time slippage, the purchase cost of these tools is generally not the most important part of the decision. More important is usually the cost of the learning curve measured against the cost savings through tool use. Cost, in this case, should include the time difference as well as the money difference.

Tools are not Optimal Excepting almost trivial examples: No compiler emits a program that cannot be hand optimized to be made more efficient. No automated circuit layout program gives the perfect circuit. No oscilloscope doesn't also load the circuit under test and filter the actual signal. No hardware simulator perfectly models the components.

Knowing the limitations of your tools will help you design your testing and your designs to minimize the effects of the shortcomings of these tools.

Using the Tools You Have There are three aspects to using what tools you have.

1. If you don’t have a certain tool, and if getting that tool is very costly or will delay the work a significant amount of time, see if you can’t make due with the tools you have. This is “ingenuity”, which has the same roots as “engineer”.
2. Only when you have been in outside of school and perhaps when you have tried to save some time yourself will you understand the devastating effects that sometimes come of skipping a complete test after even a small change. If there is a verification tool associated with your development suite, use it and use it last. Never release a product after a design change without a thorough test.

Software IP (libraries) & other IP resources Many times you can purchase/lease IP (intellectual property) to help you in your development. Whether this is a good idea is often a complex decision.

Factors in the decision include:

1. The initial and per unit cost of the IP.
2. The amount the IP will have to be altered for use, and whether you have the right to do so
3. The amount of adjustment in your own work to fit with the IP interface

12.4 Design Fors Revisited

Design Fors were introduced in Section 4.7 where they were part of the process to generate the system requirements. Now we will look at them again, this time when we are considering the details of the design. Figure 12-3 shows some ‘Design Fors’.

Figure 12-3 Some "Design Fors" and Typical Decisions involved with them.

Notes on some of the entries:

Autoinsertion Machines will do the assembly. You may need to accommodate the abilities of the machine in your design

Access You may have to get to certain parts for manufacture, testing and repair

Fastener Type Your design may have to accommodate certain types of fasteners

Mass programmability Are there parts that must be programmed (embedded memories or programmable logic, for example)? How is this to be done efficiently in production?

Part reuse Using parts already being purchased by the company may make purchase and inventory logistics easier.

COTS Commercial-Off-the-Shelf items are those parts that are already being produced in quantity, perhaps by multiple manufacturers. If you include these parts in your design you will (usually) find them easier and obtain them quicker.

Test connections Anticipate the need for testing during design, manufacture and maintenance.

Good enough If a design is done quickly it may make the company more money than it would make with a more optimal design that took more time to produce.

FPGA Field Programmable Gate Array – a user programmable logic chip for implementation of logic networks.

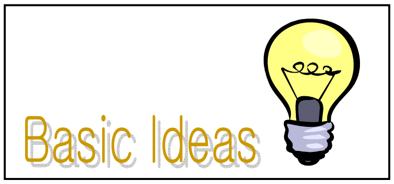
ASIC Application Specific Integrated Circuit – a logic chip that is not user programmable, but is made for a specific function. A Pentium chip would be an example of an ASIC

JTAG JTAG [15,16] is a development standard where four extra wires go to each flipflop. These wires allow the modification and monitoring of every flipflop in a serial fashion so tests can be set up, run and the results compared to what was expected.

Design Fors vs Requirements

Many of the “Design Fors” will not be expressed in the requirements discussed with the client because they are Objectives where the benefit is to the design company.

Chapter 13 Making Sure it Works



13.1 Testing Verification Validation

Testing, verification and validation are different aspects of making sure that the system is eventually correct. This process of "making sure" should trace back to the early system definitions as shown by the larger arrows in Figure 13-1.

Testing Any time we exercise a certain function of any part of the system to see if it works as expected, that is a test. Testing is part of verification and validation, but does not indicate any completeness or sufficiency that is part of these.

Verification Verification demonstrates compliance with a specification. It can be through a test, or it could be simply a check off of function or characteristic, such as verifying that the colour is red.

We verify at two places. We verify that each module fills the requirements of that module as designated in the Detailed Design. This is done by the Module Tests. We also verify that the system meets the design requirements as given in the Requirement Specification. This is done by the System Tests.

Summary: Testing, Verification and Validation are three methods by which we check the functionality of our artefact.

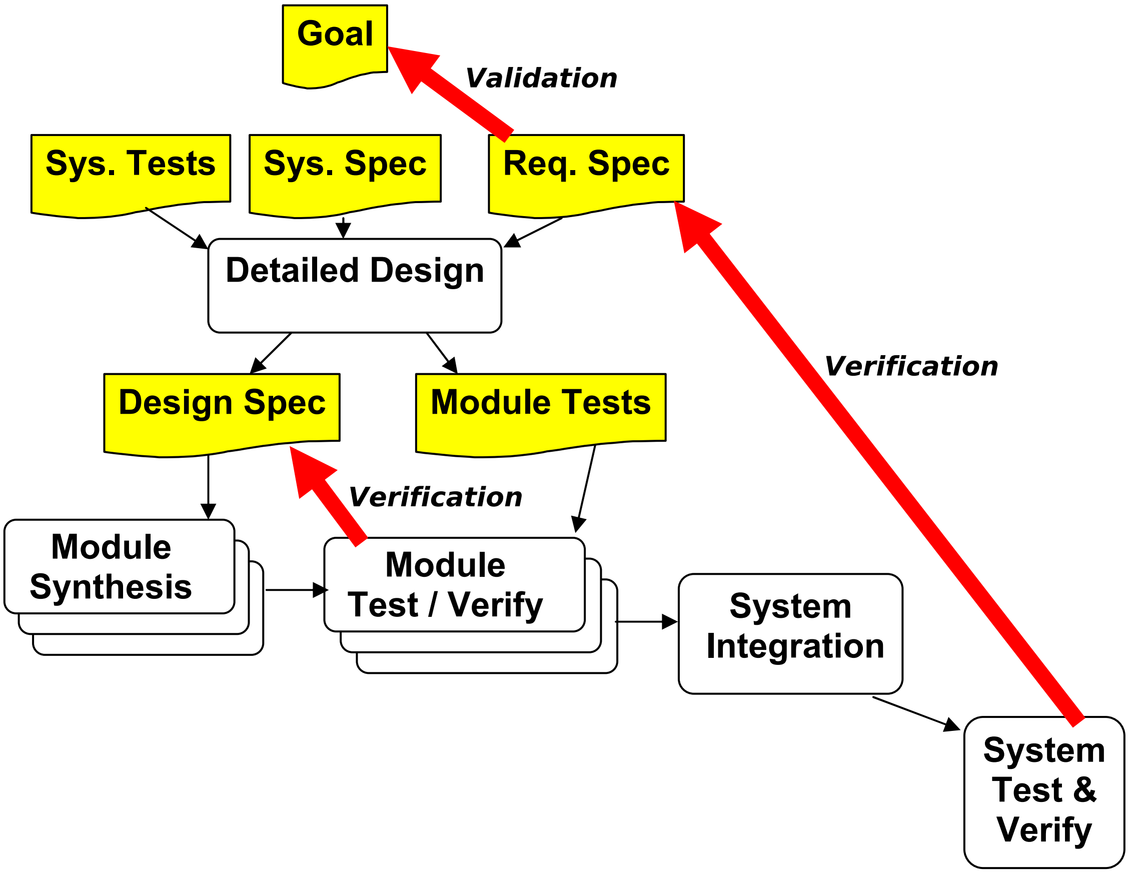


Figure 13-1 Verification and Validation

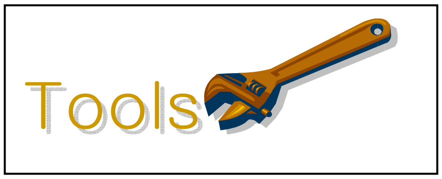
Validation Meeting the Requirement Specifications does not mean that the system it will work in the larger system of which it is a part. In theory, it should. In practice the Requirement Specification is never absolutely complete and certain critical elements may have been left out or be incorrect. Validation is determining that the system does what the larger system, of which it is a part, requires it to do.

Acceptance Tests

Actual projects will often have test sets to verify to a client that there is compliance or achievement of a milestone. These are called "Acceptance Tests" and are seldom full verification or validation tests. They may have several parts, including tests of the user interface, stress and environmental tests and performance tests[22]. Since they are client-generated, they will reflect client concerns about the behaviour of the artefact. These should have been captured in Snapshots and use cases in the early design steps, which underscores these Snapshots and use cases as test devices.

Consider this incorrect specification: Say you are developing an air traffic control system (ATCS) to track aircraft traffic around an airport. The aircraft must legally fly at least 500 feet above the ground unless landing or taking off. The Requirements Specification might say that the ATCS must filter out bad images, including those outside legal flying space. This could be a reasonable filter against building reflections or advertising balloons. However, in practice, an aircraft in trouble could well approach below the legal altitude and a proper ATCS should track this aircraft regardless of its height. The well-intentioned specification was wrong.

The difference between Verification and Validation is exemplified by this cry: "I know that's what I said, but it's not what I meant!"



13.2 Tests and Use Cases

If Use Cases were used during the design process (and the author strongly recommends that they be used!) then generation of validation and verification tests are easier. The Use Cases themselves can be performed to validate the design and the requirements that have been built from the breakdown of the Use Case can provide the verification tests.

Figure 13-2 is a Use Case for a hairdryer with some of the requirements generated from it that was first seen in Chapter 5

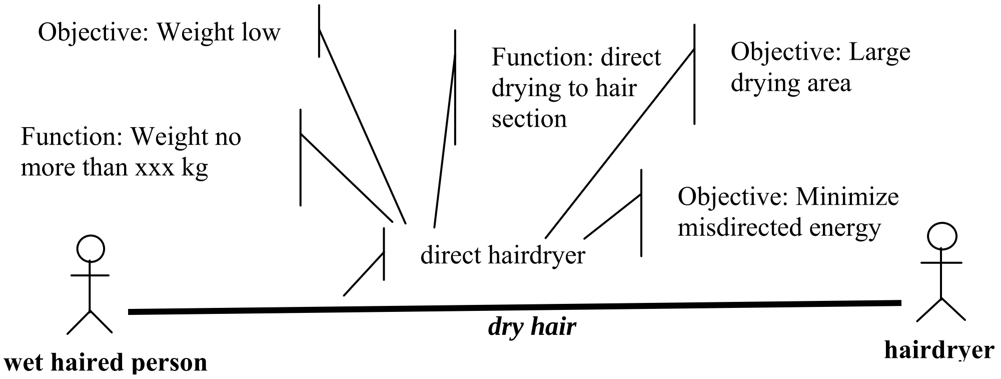
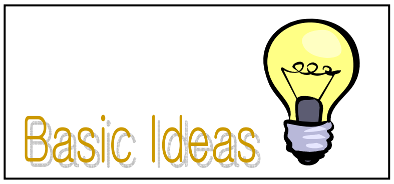

Objective: Weight low
Function: Weight no more than xxx kg
Function: direct drying to hair section
Objective: Large drying area
Objective: Minimize misdirected energy
direct hairdryer
dry hair
wet haired person
hairdryer

Figure 13-2 Use Case with Requirements

From this we can generate the following tests for our hairdryer product resulting from the design cycle. Note that by the final system tests the objectives we generated at the requirements phase of our design activity were turned into functions == our design made the tradeoffs in a manner we felt best and the tradeoff levels became the design targets:

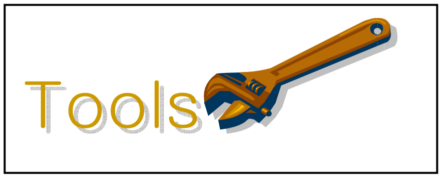
Validation Tests	
Wet haired people use the product and check that their hair is dried in a time and method acceptable to them	
Verification Tests	
Function: Weight no more than xxx kg (now includes the objective “weight low”)	Test: Weigh the product
Function: direct drying to hair section	Test: Direction possible? (yes/no)
Function: Drying area minimum xx cm^2	Test: Measure drying area
Function: Misdirected energy under 5% of total	Test: Measure portion of total energy directed to drying area. (Determine using test above, plus measure of input electrical energy.)



13.3 Tests & Models

Models were first discussed in Section 2.5 . There we saw that our drawings and descriptions were models, and that our test instruments also gave us an inexact modelled value. Certainly we take a risk using models (as we must) in that our modelled values might be incorrect. For this reason, every time we do a test we must add in the potential inexactness of the measurement and of the model to our value before comparing it to what we want to achieve.

For example, if we want a maximum of 1.00 volts on an output of a production prototype, and we use a measuring device that is +/-.01 volts in accuracy, we must work for a maximum of 1.00-.01 or 0.99 volts. The requirement for 1.00 volts on the output should already reflect the fact that this value could vary due to variations in component tolerances during production. If there is an absolute requirement for this maximum, a test should be included on every production device built. (The production test procedure should be part of our development deliverables!)



13.4 Debugging

Hit a big problem? Here are some tips.

Contributors:

- “The Power of the Most Likely”, Bob Colwell, Computer 2002 [Intel chief architect PII to PIV]
- http://vergil.chemistry.gatech.edu/resources/programming/c-tutorial/debug.html
- http://oopweb.com/CPP/Documents/DebugCPP/VolumeFrames.html?/CPP/Documents/DebugCPP/Volume/techniques.html
- Tal Akermanis, Chris Schneider, Sewer Depot
- Paul Chow, University of Toronto
- Phil Anderson, University of Toronto

– **H**=hardware, **S**=software

1. **H&S** Best: Avoidance through module testing, careful design, systematic development methods, consideration of all possible circumstances and building to handle them. Use all design-rule checks available in your tools and use them after the last change in your design.
   Document your work in a way that will help you in a debugging / test cycle.
   Note: Plan to spend as long on the verification test as on the code or circuit design. Plan to spend at least as long in development working on handling the boundary and error conditions as you do for the intended data / process flow.
2. **H&S** Read the rest of this completely and carefully. Debugging is a process of forming and testing hypotheses about problem causes using a variety of techniques. The more techniques you know, the faster you will be.

Summary: This is a set of techniques that can help you find that elusive problem.

3. **H&S** Look for hardware and software aids for checking and debugging. The major variable (i.e. alterable) cost of development is human – the less time you spend debugging, the lower the cost. Saving a day or two will easily pay for many hardware or software aids. Look online for free advice about your hardware or software tools.
4. **H** Check the power supply and the connectors. This includes the fuses and the wall plug.
5. **S** Software version of the above: Make sure the listing you have is the code that you are debugging.
6. **S** Read and understand all error and warning messages. Eliminate all messages by filtering the ones you can ignore[14] (by setting a high warning filter level in your compiler) and taking care of the ones you can’t ignore. You won’t find an important message in pages of messages you don’t care about.
7. **H&S** Use a somewhat-binary division of the circuit / software to isolate the problem (a corollary of: Find out where the problem isn’t)
8. **H&S** Isolate or remove pieces of the code / circuit that appear to work or not to be involved & see if this is true. Figure out what it can’t be … but be sure of that.
9. **H&S** Solve the easy problems first (particularly if there are possibly or definitely multiple problems)

[14] Some design managers will insist that you change your code so no warnings at any level are produced so there is never a possibility of a filtered, important message.

10. **H&S** Build in monitors / test points / error traps / assertions etc. For software, use #ifdef - #endif directives around test elements.
11. **H&S** Carefully determine the situations where the problem happens. Keep accurate lab notes.
12. **H&S** Read the manuals for your test equipment & software. There are often sections which suggest how to find certain problems in your development work. Look for features in the compiler / chips to aid debugging (clock outputs, debugging modes).
13. **S** Learn & use breakpoint and trace features in your monitors & emulators
14. **S** Learn to use specialized tools for “loading”, profiling, etc.
15. **H&S** Test your test equipment / test code.
16. **H&S** When it does not make sense, assume nothing, verify the basics first
17. **H&S** Walk through your code or circuit in great detail (“bench check”)
18. **H** See if any of the components are too hot to touch. See if any of the components are abnormally cool. Use caution – you can burn yourself on a working circuit; components with heat sinks are expected to be hot.
19. **H** Run it with a low and with a high voltage power source in a noisy environment
20. **H&S** Look for patterns in the misbehaviour, or a lack of patterns
21. **H&S** Components are not perfect or ideal: Be aware of 2nd order effects, tolerances, rise times, propagation delays, internal inductance or capacitance or resistance etc etc
22. **H** Check the voltage source for brownout, ripple, etc. Also, a slow voltage ramp up of a power supply may cause microprocessor-based designs to fail.
23. **H&S** Design for test
24. **H&S** Change one thing at a time. Document your changes. Don’t make them irreversible because you have lost the trail or the old copy. Use a revision control system. Keep copies of older versions.
25. **H** Some power supplies will not work without a (significant) load.
26. **S** Problems which happen after “long” use: Look for
    a. memory leaks
    b. stack overflows
    c. relationships to # of users
27. **S** Problems after adding an innocent line of code. The “innocent line of code” may have uncovered a latent problem.
    a. But first: really look at the code. It may have side effects or not be so innocent
    b. did you really do more than add the line of code
    c. stack overflow
    d. misaligned variables
    e. memory overflow
    f. uninitialized variables / bad pointers
    g. new use of a “tested” functionality
28. **H** Problems in certain physical situations
    a. magnetic fields

   b. electrical noise

   c. grounding problems

29. **H&S** Explain the problem to someone else. They may not be able to appreciate the problem, but you may come to a solution is organizing the explanation. [This is known to some as “ANWB debugging”. ANWB being a Dutch organization that helps with car trouble. In Canada I guess you would call the CAA and explain your problem to them.]

30. **H&S** If the problem does not get solved quickly, make plans for the long haul. (Ex: Don’t wait a week to get a logic analyzer or storage ‘scope or to get help from the company guru). Don’t expect to solve a 1 day problem in 5 minute sessions spread over a week.

31. **H&S** When you find and fix the problem, think about other problems that probably exist because of similar techniques or assumptions you used in the development, or where you used code and circuit copies. Fix those problems before retesting. Add tests to your verification process that will catch this type of bug.

32. **H&S** When you hit a wall, take a walk, a drive or a shower. Let the subconscious work it over

13.5 The Case for Testing

Testing to most inexperienced engineers is almost an afterthought. But let us consider an example – a software error in the embedded software on a device.

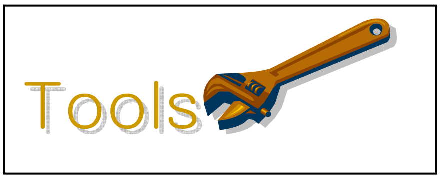
If our first test is a review of the system requirements and the error is caught, it will only take a clerical change to correct – a few seconds. If we review code as we enter it and find the problem during the review, we may have to restructure some of the code and it could take a few minutes, or perhaps a bit more, to make it right. If the problem is uncovered during a module test, the problem is still quite local, but it could take up to a few days to make the change and bring the module back through testing. When the modules are brought together we potentially affect all of the teams developing all of the modules. The cost per hour goes up and the time could be hours or even weeks. If we are so unfortunate as to find the problem after the release of the artifact it could be days, weeks or months to fix, it could involve multiple trips to sites and it could involve a large number of people. As well, there is a huge cost in lost trust from the customer. Table 13-1 summarizes this increasing cost

Problem Type	Where it Appears	Cost
Syntax and similar errors	Debugging	Very Low (minutes to hours)
Detailed design flaws	Module testing	Low (hours to days)
System design flaws	System Integration	Medium (days to weeks)
Requirement flaws	Delivery	Very High (weeks / months)

Table 13-1 Problem Appearance and Cost

The costs shown are costs in time, but have associated financial costs with the same jumps in magnitude as the problems are found later in the lifecycle of the artifact. Some flaws found late, such as the Pentium math 'bug' [9,10], or Therac 25 programming flaw[11], have huge costs including loss of life.

The conclusion, unhappily relearned by engineers over and over again: Test early, test often, test well.



13.6 Prototyping / Simulation / Analysis

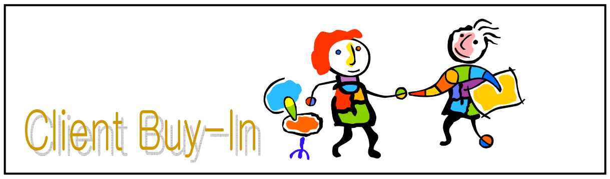
Prototyping, simulation and analysis can help at this stage to determine requirements. By producing a fast prototype, for example, one can become aware of failure mechanisms early and of customer acceptance of the features and requirements you are considering. The technique is so important that many advanced development models extend or replace the Waterfall model with models based on iterative prototyping, simulation and analysis followed by a reworking of the requirements and the design.

Summary: Prototyping, simulation and analysis can allow early interaction with the client to determine requirements and later, during the System Design step, to help in the evaluation of potential implementations.

Applicability: Requirements analysis and system analysis steps

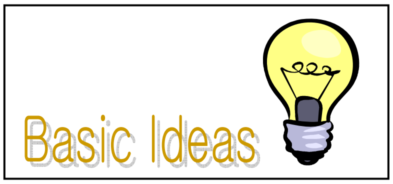
It is a methodology falling within the general theme of catching mistakes at the earliest possible moment. In prototyping you build a stripped-down version of the actual artifact. With simulation, you build a model of the artifact and test it. With analysis , you model the artifact with equations and see what they reveal. In all cases you are attempting to determine the operating characteristics of the artifact before you actually produce the final version and to make changes in the requirements and the implementation where problems or client dissatisfaction are found.

For very large systems, jet fighter aircraft for example, the client will often fund the development of prototypes by several competing companies, then purchase multiple copies of a further revised design from the company producing the prototype seen as best.



One of the most important things that prototyping does is to reveal miscommunication between designer and client. There are often unexpressed client needs and wants, misinterpretation of certain requests.

Prototyping is also useful in determining potential problems with other stakeholders: Are there changes that can be made easily that will help it to be easy to manufacture, test, maintain, evolve to new models or keep records about the artifact?



13.7 Test Methods; Types of Tests

Can we test a non-trivial system completely? No. Sometimes we can test the model of a system completely, where "completely" only includes the tests available with the modeling system. We cannot test for every type of failure or error in every type of use situation in every combination of environmental circumstance. The number of potential faults caused by failures that we do test for, as compared with the total number of potential failures, is known as the fault coverage.

Some functionality of any delivered item we cannot test at all – consider a stick of dynamite: We cannot make sure the dynamite works before we give it to the customer because it only works once – we would destroy it in the testing. The same is also true of high-voltage destructive testing of inputs for obtaining a CSA (Canadian Standards Association) rating; we cannot subject every production item to this test.

Summary: 100% testing is almost never practical or possible. Testing to high levels of confidence is possible with the right techniques.

How then can we test? Certain strategies are used to do adequate testing in the face of the fact that complete testing of a design or in production is impossible. They are based on probability, the complete study of which we will not do here, however it amounts to this: If we test in certain ways, sometimes with samples, sometimes with every unit produced, we can get very good fault coverage. In fact, by calculating probabilities we can determine the level of testing we need to do to get any level of fault coverage (short of 100%).

Looking at the testing requirements at every stage, including the early ones, is of great advantage.

Table 13-2 shows a typical requirement and the verification done to show compliance. Note that some verifications are only complete in a probabilistic sense (see Appendix A on probability).

System Requirement	System Verification
Shall address all memory locations	Address subset of all memory locations as provided by customer
Shall have a mass under 20 kg	Verify weight of 10 samples are all under 19.8 kg (+/- .2 kg)
Shall have resistors within 5% of nominal resistance	Demonstrate compliance statistically to 99.99% with a 95% confidence level. The vendor of the resistors shall provide proof of compliance.

Table 13-2 Requirements and Verification Examples

Example Test Process

Lens Controller for Imax Camera in Shuttle Mission: The following test and risk reduction methods were used.

- 3 levels of complete system test (min) done by three separate groups
- Pre-classification of components; selection only of environmentally pre-qualified components
- Component selection & test in production.
- Special design considerations to reduce risk of failure (coatings, extra mechanical fastenings)
- Building of spares for destructive testing & failure replacement during testing

Certain methods help us to cover large numbers of situations with a relatively small number of tests and to increase the significance of these tests to the development process.

1. **Test against requirement specs.** The requirement specification defines the system you are building. It should have measurable targets. Your tests need to prove that your artifact meets these targets as proof you have completed the design.
2. **Case testing, including random destructive tests and stress tests.** The Use Cases you generated during the design phase give operational situations where your system must perform. These Use Cases should be simulated or performed to show compliance.
3. **Alpha / Beta releases** Alpha and Beta releases to selected customers will help to find usage problems that you have not predicted in prior analysis and its associated testing. Alpha releases are often internal, to a group specifically formed for testing. Beta releases follow, and usually go to a selected group of actual users.
4. **Test against non-independent fault combinations** Often a single test will cover many potential faults and failures. By choosing tests that exercise independent pieces of the system, overlap is avoided and more faults and failures are covered with fewer tests.
5. **Test corner cases and boundary conditions** Testing at the extremes of inputs, environments and other factors will generally handle the ordinary levels of these factors as well. Using these extremes in combinations, the “corners” of the factors in combination, will reduce the number of tests. Figure 13-3 shows seven tests done for two input parameters, which test the four corner cases, two cases which would be outside the acceptable range and one case where everything is well within range.

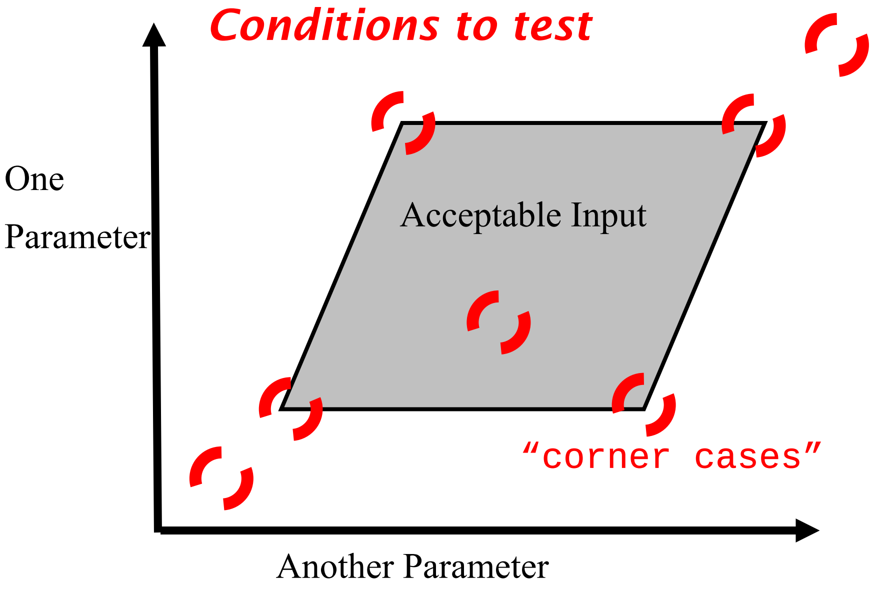


Figure 13-3 A Typical Test Set for Two Parameters, Showing Corner Cases

Chapter 14 Breaking the Development Model

You may already have noticed the basic waterfall model (goal→requirements→system design→detailed design→integration) being stretched during the parts. Already we have repeatedly looked at iteration during the design steps, something that is not in the basic waterfall model as strictly interpreted.

Traditional engineering products, such as bridges, chemical plants and buildings, require close adherence to a development model such as the waterfall model since you are building one copy of a very expensive item. If you build a bridge and it is not quite right, you cannot easily and cheaply add a traffic lane, or increase a load factor. You must carefully plan for all of the needs, weigh the objectives and decide fairly closely exactly what to proceed with to the implementation stage. Once the concrete is poured it is not easily ripped up and changed.

This is not true of the products of electrical and computer engineering. Software is fairly simply changed during the development cycles, and new versions with more and different features can be built comparatively simply from the old. Hardware designs are similar, up to a point, and trial designs can be built, tweeked, evaluated and compared under most circumstances. Some devices, such as field programmable gate arrays (FPGAs) are made to support these processes in the digital hardware realm.

Of course this is not universally true. Software, once released, is sometimes difficult or impossible to adjust, and some hardware devices such as integrated circuits and very high speed circuits are costly to produce and therefore best done only once if possible.

Why would we move from the waterfall model? There are four motivations.

- First, it allows us to exploit the talent we have at the time we have it. Developers are not waiting while requirements are assembled for all sections of the project.
- Second, it naturally supports the development of only a subset of features (Use Cases), with the others held back for future versions, or next rounds of development.
- Third, it supports the testing in isolation of potentially problematic pieces of the system design, and allows the early modification of the system design should alternative strategies be necessary
- Fourth, and possibly most important, it can allow early interaction with the client and users, so that requirement deficiencies can be quickly remedied.

Here we are going to look at design models that exploit a situation where changes during the design process are possible at comparatively lower cost.

14.1 Alternate Models: Evolutionary and Incremental

For projects that have relatively standard-practice implementations to match a well-defined set of requirements, the waterfall model suffices. There may be several implementations to choose from, and the implementation may be long and difficult to effect, but whether any of the implementation choices is doable will not be a question.

Many projects are not like this.

Evolutionary Model

The requirements may be vague because the client is not fully aware of what is required, or because the client is changing the requirements on the fly. These are very common situations. When the requirements are vague, the strict waterfall model stalls and may never get fully started.

An evolutionary model will help in these situations. In this model the artifact is developed with a set of attributes or features, evaluated, then another development cycle takes place. The artifact may be released to the client or to production repeatedly as well. With each cycle the aim is to determine the actual requirements better and to extend the functionality of the artifact.

Spiral development, risk-driven development, continuous design and extreme programming / test-driven design are all types of evolutionary design that share this cyclic design pattern, shown in Figure 14-1. Going through these cycles means that the development path is chosen to match the development and changes in the requirements.

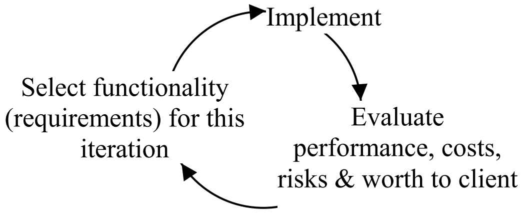

Implement
Evaluate performance, costs, risks & worth to client
Select functionality (requirements) for this iteration

Figure 14-1 Evolutionary Development

For example, you might be asked to produce a programming editor for a development platform, but might only be given some very loose information about what it would have to do. It might be quite likely that the last requirements will come very close to the expected delivery date, so waiting for them before starting is not reasonable. Instead, you would begin the work, perhaps starting with the basic search and replace functions, open and save operations and so on. Later, as the requirements came in, you would add to this base.

It should be immediately evident from this example that careful consideration of the design of the "base" of the editor is going to be very important. If the base is unable to reasonably easily accommodate the additions but instead requires significant alterations or a complete rewrite, there is little advantage in using the evolutionary model. Planning of the technology is very important. Many software techniques and computer languages are focussed on managing this evolution.

Evolutionary methods are also called "Adaptive", and a method called "Agile" pushes this to a very small time window.[23]

Incremental Model

Sometimes the requirements are well-understood and fixed, but the method of implementation is not standard practice. In these cases, detailing a complete system design built around a technical uncertainty will often be wasted effort when the technology proves deficient.

The incremental design model is of use here. Unlike the waterfall or evolutionary models, where a full system design is produced, here we produce only a part of the design for evaluation. We may do several iterations of the part in order to find a design that 'works' or to be able to compare designs to find which is 'best' under the circumstances. Often the designs of other parts of the system will depend on this part, and so they will have to wait until a successful design for this part is found, although often some other parts can be fully designed (as they are not dependent on the part subject to incremental design) and others can be started in a way that allows later change to adapt.

The strict waterfall model has one determining a goal, then a set of requirements carefully constructed so as to be free of implementation details. This is often not the case in the real world.

Figure 14-2 is closer to what would actually happen. Partial implementations are proposed, then analyzed, and from this analysis comes requirements. Say one idea for implementation is found inadequate because the artifact must work in the cold. The designers realize they must add "shall work in temperatures below minus 40°C" to the requirements. These iterative thought exercises where we propose solutions and consider the consequences of using that solution help us create stronger requirements.

"Fast prototyping" is a name that is often used for the incremental approach when it involves actual production of working pieces of the design.

One area where these techniques are very useful are in the development of a user interface, and graphical user interfaces (GUIs) in particular. Often clients will not be able to fully express the attributes and functionality of the user interface. Production of trial interfaces and then presenting and discussing them with the client and users is usually the most efficient method of finding a suitable interface.

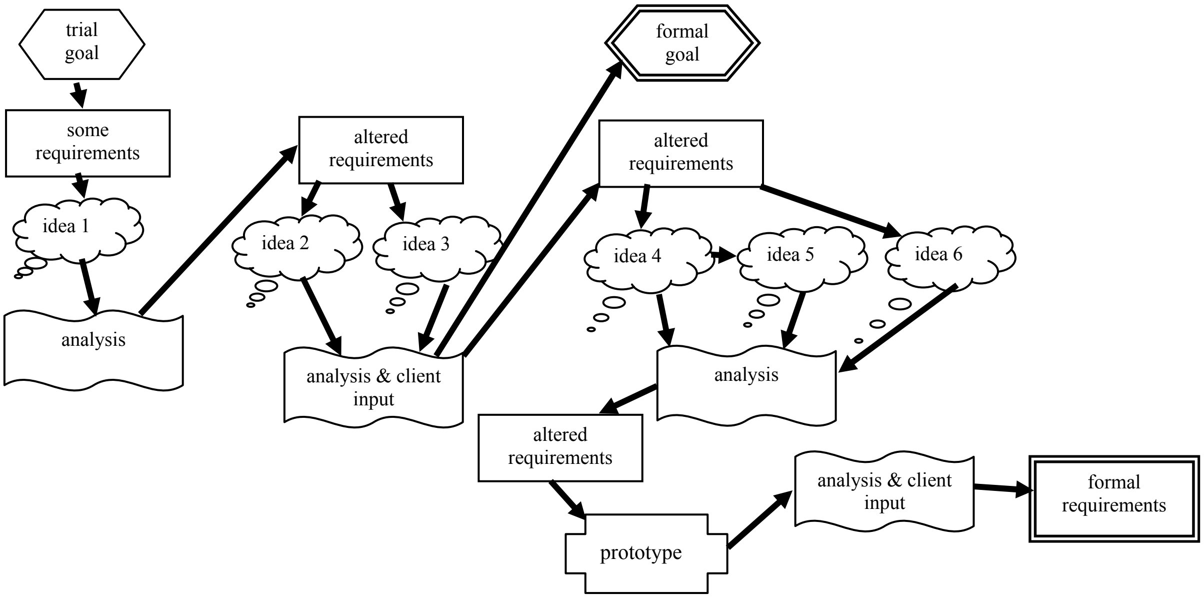


Figure 14-2 Part of an Example Design Sequence

14.2 Caveats and Conclusions

One must not conclude that requirements are not needed if one uses an incremental or evolutionary (or other) approach. Efforts must still be strongly directed to satisfy the goals of the client and society. Every cycle must be analysed and evaluated and these actions must relate to the goal and "big picture" client requirements. By the end of the design process the artifact must answer the goal of the client and meet the constraints and functions. These models, including the waterfall model, are guides to help create this success by focusing efforts and simplifying coordination.

An incomplete or incorrect set of requirements is the leading problem with development projects Chapter 1 .

For very simple projects such formalism is not likely necessary. For very complex projects with a long timescale and many developers they will require additional formalism and the associated additional control. Most straight-forward projects with unchanging, clear requirements can be done well with the waterfall model. Combinations of other techniques may be required if requirements are changing and/or vague or if technological unknowns could change the course of the design partway through.

Chapter 15 Wrapping it Up

15.1 Final Remarks

It is hard to conclude, because this is only the start to an education in design. If you have read to here and have some grasp of the basic waterfall model, of some of the issues you are going to encounter in design and of some of the system-level problems you are going to encounter, then you are well-grounded to move on.

I would suggest reviewing the overall themes in Section 1.5 , but rather than repeat these we will look at some basic strategies to good design – fundamentals that will help guide your choices in the open-ended situations basic to design.

And no matter what happens with your designs, as explained in [7]: “Failure begets knowledge. Out of knowledge you gain wisdom, and it is with wisdom that you can become truly successful.”

15.2 Phundamentals from Fil

Here are some fundamentals to design

i. Keep the Crazy

Often the best designs spring from combinations of ideas that might first seem absurd. Keep the spark, keep the weird. You will enjoy engineering more and be a better engineer.

ii. Catch Mistakes Early

If there is any lesson repeated at length in this book and so important in design work, it is to plan, test and organize to catch mistakes early and to keep them from happening.

iii. Get Close – Move Back

There are details you miss if your view is too distant, and interactions you miss if your view is too close. You cannot exclusively do either and hope to have a good design.

iv. The Design is Probably Bigger than First Thought

Very few overestimate the work involved in a design. Consider this during budgeting and when making design choices.

v. Create Closure

It is very easy to keep a project open-ended, since the planning is greatly reduced. However this leads to “feature creep”, time and cost overruns and general dissatisfaction. If you erect a boundary fence or draw a line it is much easier to see what is on one side and what is on the other. This will make your design decisions easier, the design management easier and the negotiation of changes with the client easier (and those changes will come!).

vi. Follow the Money

If you aren’t sure, look at the money. Clients are happy when they are making money. Bosses are happy when they make or preserve money. Projects die when someone stops the money. Love and passion make life worth living but keep an eye on the funds and love and passion (at least in your engineering life) will have a place to stay.

vii. Concentrate on the Careabouts

Some designers will spend a tremendous amount of time working on features, functions and characteristics that have little or no value. Determine what matters (a stakeholder “careabout”) and direct your efforts there.

Appendix A References and Reading

[1] Kerzner, H., In Search of Excellence in Project Management, New York:Van Nostrand Reinhold, 1998

[2] "IEEE Std. 1220-1998: IEEE Standard for Application and Management of the Systems Engineering Process", 1998, available through The Systems and Software Consortium, Inc., [Online] HTTP: http://www.software.org/quagmire/descriptions/ieee1220.asp

[3] D. Arnold (last modifier)," The Explosion of the Ariane 5",Institute for Mathematics and its Applications, [Online document], August 23, 2000, [cited July 22,2005], Available HTTP: http://www.ima.umn.edu/~arnold/disasters/ariane.html

[4] A. Sutcliffe, User-Centred Requirements Engineering, Theory and Practice, London: Sprin ger-Verlag London Limited, 2002

[5] Course Notes, System Engineering, Project Performance Australia, August 2004

[6] Otto and Wood, Product Design, Prentice Hall, 2001

[7] "T23E-T10E Standish Group Report", Standish Group, 2005. Available online in many places, including HTTP: http://dimsboiv.uqac.ca/~sboivin/C2005/8INF814_Ete05/raw/CHAOS_1994.pdf

[8] U.S Nuclear Regulatory Industry, Fault Tree Handbook,Washington:U.S.Government Printing Office, 1981, available online HTTP: http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/sr0492.pdf

[9] A. Edelman, "The mathematics of the Pentium division bug", SIAM Review 39(March 1997):54-67

[10] L. M. Fisher, "Flaw reported in new Intel chip", New York Times, May 5,1997

[11] N. Leveson and C. S. Turner, "An Investigation of the Therac-25 Accidents", IEEE Computer, Vol. 25, No. 7, pp. 18-41, , July 1993. Available online http://courses.cs.vt.edu/~cs3604/lib/Therac_25/Therac_1.html

[12] J. R. Wixson "Function Analysis and Decomposistion using Function Analysis Systems Technique", CVS, CMfgE, Lockheed-Martin Idaho Technologies Company, Inc., P.O. Box 1625. Idaho Falls, ID 83415-3634

[13] Hauser and Clausing, "Tool to build the relationships: 'House of Quality'", Harvard Business Review, May-June 1988

[14] S. Adams, The Dilbert Principle, New York:HarperCollins, 1996 p174-175

[15] 1149.1-2001 IEEE Standard Test Access Port and Boundary-Scan Architecture, Standard IEEE Product No:SH94949 IEEE Standard No:1149.1-2001, ISBN:0-7381-2944-5, 2001

[16] R. Oshana, "Introduction to JTAG", Embedded Systems Programming, October 29, 2002

[17] AbsoluteAstronomy.com,[Online reference],HTTP: http://www.absoluteastronomy.com/Reference.htm

[18] "UML™ Resource Page", Object Management Group, Inc., [Online document], January 19, 2005, [cited August 11, 2005], Available HTTP: http://www.uml.org/#UML2.0

[19] "Free Brainstorming Training", Infinite Innovations Ltd., [Online document], 2003, [cited August 2, 2005], Available HTTP: http://www.brainstorming.co.uk/tutorials/tutorialcontents.html

[20] Yonat Sharon, "Extreme Programming", Object Orientation Tips, [Online documents], April 15, 1999, [cited July 10, 2006], Available HTTP: http://ootips.org/xp.html

[21] A. Cockburn, Writing Effective Use Cases, Addison Wesley, 2001

[22] J. Whittaker, How to Break Software, Addison-Wesley, 2003

[23] "Agile software development", Wikipedia, [Online document], August 15, 2006, [cited August 17, 2006], Available HTTP: http://en.wikipedia.org/wiki/Agile_software_development

[24] M. Mannion, B. Keepence "SMART Requirements", ACM SIGSOFT, SE Notes Vol 20, No. 2, April 1995, Also available: [cited August 22, 2006] HTTP: http://www.win.tue.nl/~mchaudro/sa2004/SMART.DOC

Appendix B Introduction to Probability

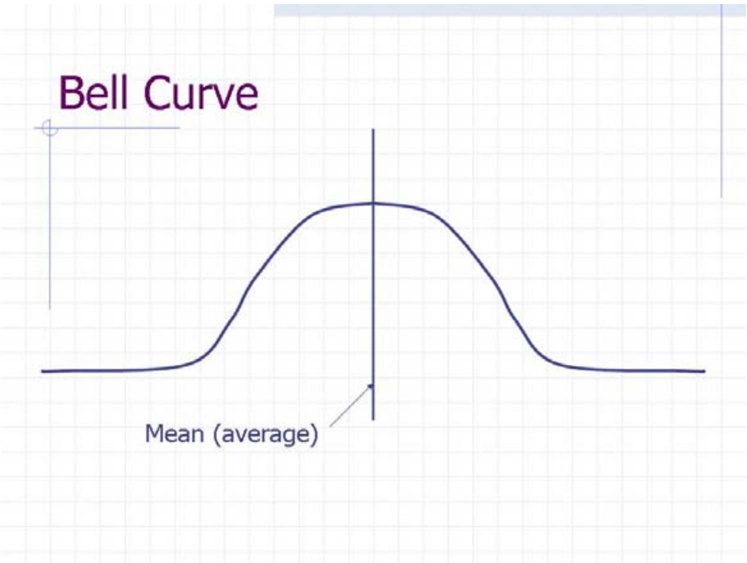


Figure 15-1 Normal or Bell Curve

Summary: Probability is the basis of decisions when you aren't sure but want to make a selection that is most likely successful. Here we introduce the Normal probability distribution which is often used as a representation of the probability of parameters of natural items.

If you look at values from many processes, for example student marks and IQ, they will fit a certain curve. This curve, called the Normal or Bell Curve, is shown in Figure 15-1. Most of the values are found near the midpoint which is the average or mean, with only proportionately few values out at the extremes of the curve.

With a Normal Curve, the spread is important. If the spread is wide, many of the values will be far from the mean. A tight spread indicates that the mean will be a reasonable representation of all values.

One of the ways of indicating the spread is by the standard deviation, often represented by a sigma (σ). This value is the square root of the sum of the squares of the differences from the mean:

$$\sigma = (\Sigma\ (\text{mean-value})^2)^{1/2}$$

There are many aspects of probability analysis that we are not dealing with here for the sake of brevity. Hopefully your education will include a course on probability and statistics and this will make the picture more complete and useful.

For example, say we have values of 6,7,8,12,13 and 14. The mean is 10. The standard deviation is root(4^2+3^2+2^2+2^2+3^2+4^2)=root(58)=7.6

Wider distributions will have more values further from the mean and thus larger standard deviations. Tighter distributions will have fewer values far from the mean, and thus smaller standard deviations.

When we plot the standard deviation as a distance from the mean on our Normal Curve we see what is shown in Figure 15-2. The values falling between one standard deviation to the left and one standard deviation to the right are 68% of all the values in the distribution. If you go to +/- three standard deviations you get 99.7% of the values. Some manufacturing companies announced "Six Sigma" programs, where they aimed for failure rates outside with a mean six sigma from the specified limit, or about 3.4 outside the limit in a million parts (for a specific test measurement).

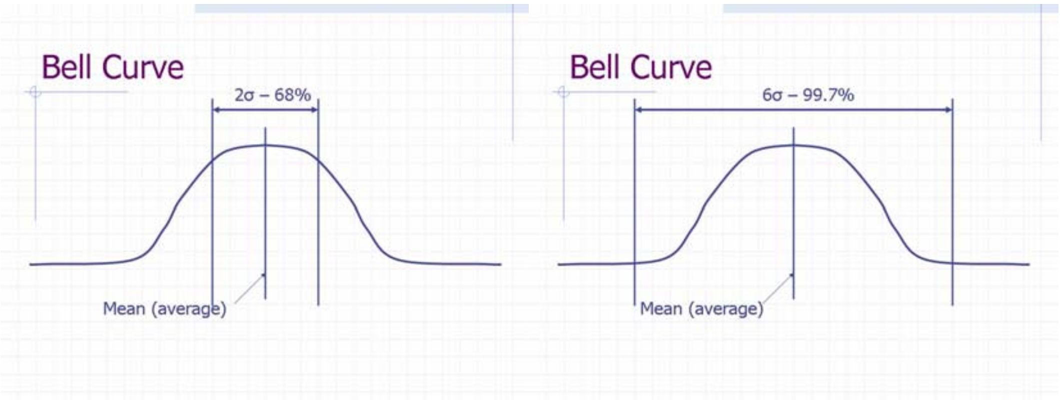


Figure 15-2 Normal Curves with Standard Deviations Shown

Probability methods allow us to test a sample from a space, say a number of resistors from a manufacturer, then to determine the confidence we would have that these samples came from any particular distribution, including the Normal Curve. You often hear this when elections are near and they indicate that 28% of people support the Liberals, and the poll is within +/-2% nine times in ten. This means that 28% of the people polled liked the Liberals, and that this would be within 2% of the true Liberal support in (at least) 9 out of 10 polls done with this sample size. Similarly, we can select a set of resistors, test them, and determine, with some confidence, how well all the resistors are going to perform in the test.

Appendix C Introduction to Economic Terms & Cost Estimation

The costs and the timing of the costs are very important to a project. One of the first concerns of a business is **cash flow**. This is making sure that the people that owe or that are lending you money pay on time ("cash in") and that this brings in enough money to pay the people that you owe or are lending money to ("cash out"). Some delays can be tolerated, but it brings ill will and stress. Often loans from a lender are tied to milestones reached, which means a project must proceed as expected or cash flow will suffer.

At the end of the project we hope to make money. If costs exceed the income (or savings) from the project we are losing money and a company that does this often will often go bankrupt.

To make it easier to understand and to control the costs, the costs of a business are generally divided into a number of categories. Thus each project can be individually monitored and within each project we can determine what costs we have the greatest control over.

A business divides its costs into

- **Fixed Costs** These are costs that will not change with the quantity of work (within limits). They are almost unavoidable costs like salaries, equipment payments and leases, heat and rent. Certainly you can change fixed costs (say through firings or moving into a smaller place if in a manufacturing environment), but this is not done easily and would not be considered part of the usual day-to-day business flow.
- **Variable Costs.** These are costs related to quantity of goods produced or work done.

For example, if you have borrowed money for a soldering machine, the loan payments must be made regardless of how much you use the machine. This is a fixed cost. The solder used, however, will depend on how many boards are soldered. The cost of the solder is a variable cost.

A project has similar types of divisions:

- **Indirect Costs** Business resources that are used directly by the project should be assigned as costs of the project. Such costs include employee time, equipment costs and expenses shared with other projects. This will be done as a percentage of the time and facilities used by the project.
  So if the business requires an engineer to supervise the project 50% of the time, the indirect cost would be half the employee cost.
  Indirect costs are like fixed business costs – they happen regardless of the progress of the project.
- **Direct Costs** Direct costs come about because of the progress of the project. Worker wages and benefits are direct costs for workers involved only in this project. Equipment rented or purchased only for this project, parts, phone charges, services and other bills that are specifically for this project are direct costs.
- **Overhead.** The project must support the business of which it is a part. Every business has costs and often departments to handle sales, marketing, human resources and accounting. There will be management staff and a network of administrative help to make sure the corporate, government and shareholder needs are met, and all these people have wages. In addition, there are fixed costs to provide workplace resources for these people. A project will be assigned a percentage of the total of these costs, known as overhead.
- **Profit.** The shareholders of a company, who are its owners, expect the company to earn more money than they spend. Projects must produce or contribute to this excess of earnings over costs, which is the profit of the company.

A quick note on employee cost: Depending on how you want to arrange the costs, you may want to consider costs as being the employee wage, or you may want to include other costs of having an employee including sickness, training and other paid, non-productive time. This is about a thirty percent increase over wages. You may also want to include employee costs such as stationery and administration, which makes the cost even higher.

Often you will find people playing games with the costs – moving items between categories or delaying so profitability is artificially boosted. From a global perspective, though, you cannot create new money in a company (unless you are the government!) and profit will always be the difference between what comes in and what goes out. Sometimes these people hide the games; sometimes these people are caught; sometimes these people go to jail or are fired.

Estimating costs at the start of a project is not easy. There are an extremely large number of factors, incomplete information, uncertain events and uncontrolled circumstances. However estimation must be done before a company can decide whether or not to work on a project. To do a good estimate use

- experience (if you don't have it, find it)
- industry tables and logs of costs
- walkthroughs and Use Case analysis
- results from previous projects
- expenses of major items with % increase for rest

Don't forget

- wastage (damaged or stolen parts)
- employee wages, including idle time & meeting time
- false starts and reworking
- installation and training and early maintenance
- shipping
- airfare, hotels and food for on-site and other visits by clients and by your people
- monetary exchange, taxes, certifications

Appendix D Laws

System design works under a number of inescapable laws. This set can be found at [17].

One is best to plan so their effects are minimized, or to make them work in your favour.

Murphy's

if anything can go wrong it will.

Parkinson's

things expand to fill the available space

[work to fill the available time / budget]

[data to fill the hard drive]

[traffic to fill the bandwidth]

etc.

Clarke's

From Arthur C. Clarke, the science fiction writer.
1. When a distinguished but elderly scientist states that something is possible, he is almost certainly right. When he states that something is impossible, he is very probably wrong.
2. The only way of discovering the limits of the possible is to venture a little way past them into the impossible.
3. Any sufficiently advanced technology is indistinguishable from magic.

Pareto's

(also known as the 80/20 rule)

80% of the work/time is on 20% of the project.

80% of the profit comes from 20% of the activity

You can use it in a lot of areas: 80% of an exam is on 20% of the material??

Sturgeon's

From Theodore Sturgeon, another science fiction writer.

Ninety percent of everything is crud.

Infernal Dynamics

1. An object in motion will be moving in the wrong direction.
2. An object at rest will be in the wrong place.
3. The energy required to move an object in the correct direction, or put it in the right place, will be more than you wish to expend but not so much as to make the task impossible.