Systems Engineering Summary

1. Basic Systems Engineering concepts

Before we delve into the world of Systems Engineering, we first need to get to know some basic concepts. What is Systems Engineering? What tools does it have? And what is a market?

1.1 Introduction to Systems Engineering

1.1.1 What are Systems?

A **system** is a combination of elements. These elements interact with each other to accomplish an objective. Systems can be very big (like the Apollo Project) or very small (like a USB-stick).

There is also a distinction between closed and open systems. A **closed system** has no relationship with the environment at all. On the other hand, a **open system** does have influence on/gets influenced by the environment.

Finally, we can distinguish systems, based on their behaviour. **Deterministic systems** behave in a fully predictable way. We can always predict what they will do. **Stochastic systems**, on the other hand, have an element of chance in them. We can only estimate the probability that a certain output will occur. Finally, when dealing with **emergent systems**, we can not predict anything. Nothing is known about the behaviour of such systems.

1.1.2 What is Systems Engineering?

In **Systems Engineering** (SE) we examine how we can make succesful systems. There are a lot of tools that can be used for that. We will examine several in this summary.

Over the past decades, quite some good ideas have come forth from Systems Engineering. Processes like **Parts Traceability** (knowing where parts come from) and **Formal Interface Control** (making sure parts fit together) are all part of the SE toolbox.

1.1.3 System Hierarchy Diagram

As we just saw, a system consists of **elements**. These elements often again consist of **subsystems**. Below these subsystems, we can distinguish **assemblies**, **subassemblies**, **components** and **parts**. The parts are the lowest level of separately identifiable items. There are often also links between elements. These links are called **relations**.

The set of relations between the system, its elements, the subsystems, etcetera, is called the **System Hierarchy**. This hierarchy is often displayed in a **System Hierarchy Diagram** (SHD). An example System Hierarchy Diagram of a camera can be seen in figure 1.1.

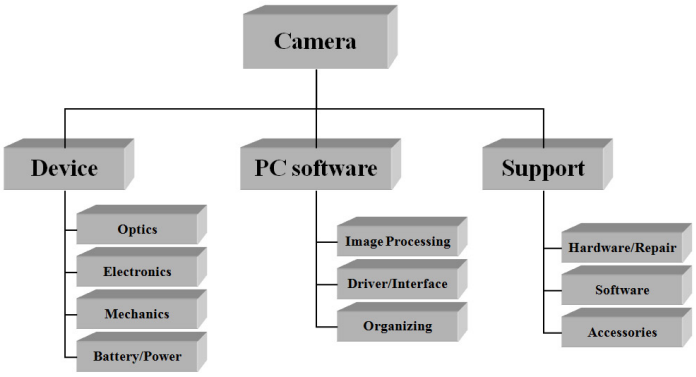

Figure 1.1: A System Hierarchy Diagram for a camera.

1.2 Systems Engineering tools

1.2.1 The Project Objective Statement

Systems Engineering has several tools, with which systems can be evaluated. The first tool we examine is the **Project Objective Statement** (POS).

Let's suppose we're developing a system. It would be nice to know what this system should do, and how it should do this. This is where the POS comes in. The POS mentions, in about 25 words...

- the **expected result**.
- the **available resources** to reach this result.
- the **time span** in which the result should be reached.

An example POS for the Delfly is: *Impress the jury of the first US-European Micro UAV Competition by designing a flapping wing, vision based MAV, using commercially off the shelf products within a budget of 5000 euro, by 11 students in 10 weeks time.*

Something similar to the POS is the **Mission Need Statement** (MNS). The MNS states what the system (product) should do.

1.2.2 The Work Flow Diagram

A system usually has a certain input. By using this input, the system should give a certain output. To do this, the system often needs to do quite a lot of things. The steps necessary to get the output can be visualized in a **Work Flow Diagram** (WFD). A general example of such a diagram can be seen in figure 1.2.

In a WFD, we always start with the input, given in hexagonal blocks. The work steps are indicated by rectangle blocks. At every work step, there is an indication of the responsible person, the effort (in hours) necessary to perform the step, and the throughput time. Eventually, we arrive at the output of the system, which is again in an hexagonal block.

1.2.3 The Work Breakdown Structure

A Work Flow Diagram contains several tasks that need to be done. Sometimes, these tasks require further clarification. They can thus be examined more closely. A **Work Breakdown Structure** (WBS) is useful for this.

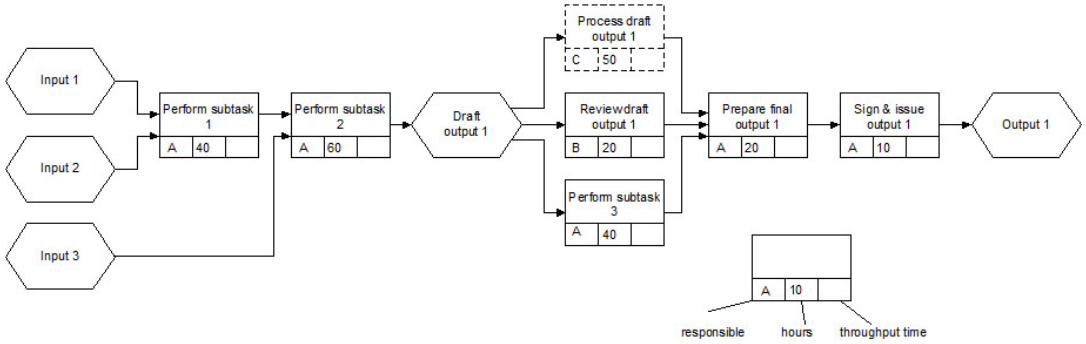

Figure 1.2: A general example of a Work Flow Diagram.

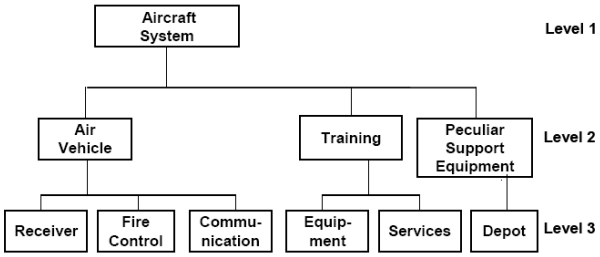

Figure 1.3: An example of a Work Breakdown Structure.

In a WBS, a task is split up into several 'sub-tasks', called **work packages**. Every single work package must be completed, before the actual task is completed. (The WBS is therefore an 'and'-tree. Only when all sub-tasks have been completed, will the task itself be completed.) An example of a WBS can be seen in figure 1.3.

1.3 Examining markets

1.3.1 What is a market?

A **market** is a place where the demand and the supply of a product come together. This isn't always a physical place. The internet can be considered a market too.

So how do we describe a market? To do that, we have to look at certain properties. Every market has certain **customers**. It also has certain **products**. We could split this up even more. Because every product also has a certain **function** and a certain **technology** in it.

Splitting up a market like this is called **market segmentation**. Every sub-group of the market is called a **market segment**. Companies often apply different marketing strategies to different market segments.

There is a nice method to visualize market segments. It is called the **Multidimensional Market Definition** (MMD). It consists of a 3D graph. The three axes of this graph indicate the customer, the function, and the technology/material of the product. An example of an MMD (applied for the bottling market) can be seen in figure 1.4.

Let's take a closer look at the MMD. The positions in the graph are so-called **cells**. Each cell indicates a certain combination of customer, function and technology. A cell, or a combination of cells, thus indicates

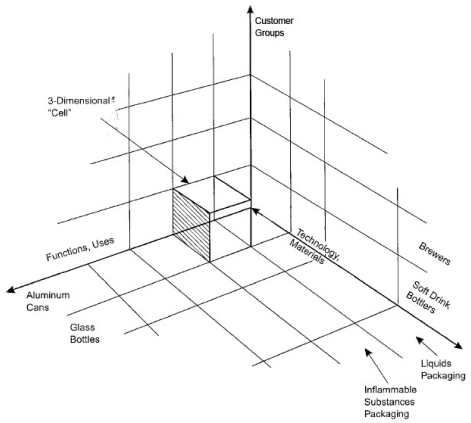

Figure 1.4: An example of a Multidimensional Market Definition.

a market segment.

1.3.2 The Supply Chain

Products don't just arrive at the market. They have to be supplied to it. This supplying can be visualized by a **Supply Chain**. An example of such a Supply Chain can be seen in figure 1.5.

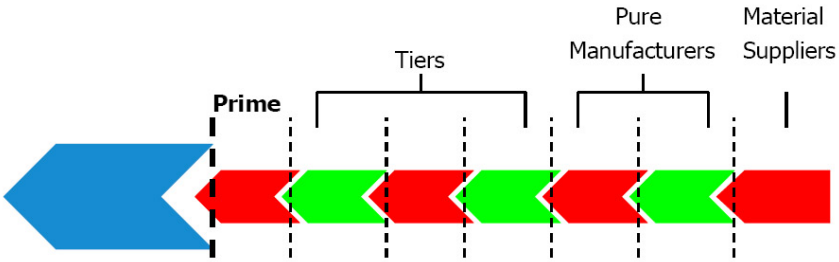

Figure 1.5: An example of a Supply Chain.

In a Supply Chain, the steps a product goes through, before it reaches the market, are displayed. First we start at the **raw material suppliers**. The raw materials are then turned into basic parts by **pure manufacturers**. These basic parts are eventually assembled, and ready to be supplied to the market.

1.3.3 Market Analysis

Let's suppose we know what our market looks like. Now we want to position our product onto the market. To do that in an optimal way, we need to perform a **Market Analysis**.

An important tool to perform a market analysis, is the **SWOT analysis**. We want to give our product a certain position on the market. In a SWOT analysis, we look at the **Strengths**, the **Weaknesses**, the **Opportunities** and the **Threats** caused by this. If we know these four things, we might be able to improve the position of our product on the market.

There is another way to examine the position of our product in the market. We can also look at the **Product Life Cycle** (PLC). This cycle describes the sales of the product, while it is on the market. An example of a Product Life Cycle graph can be seen in figure 1.6.

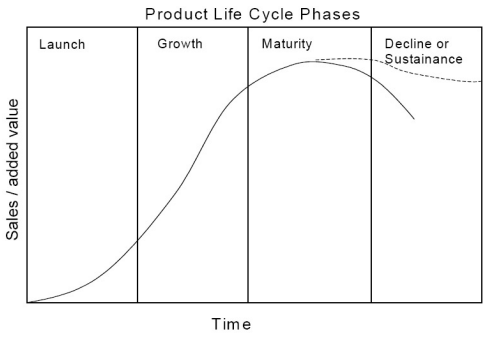

Figure 1.6: The general shape of the Product Life Cycle graph.

Let's take a closer look at the PLC. Initially, in the **introduction/launch stage**, sales are low. However, growth is accelerating, and soon we reach the **growth phase**. When this growth is starting to halt, we reach the **maturity phase**. Finally, the product either leaves the market (**decline**) or will have a steady sale level (**sustenance**).

2. System functions, requirements and resources

In this chapter, we examine the kind of functions systems can have. We also look at how requirements are generated, and how we should deal with resources.

2.1 System functions

2.1.1 The Functional Flow Diagram

Every system has functions: It should do certain things. To accomplish these functions, several steps (sub-functions) need to be taken. A good way to visualize this, is by using a **Functional Flow Diagram** (FFD). An example of an FFD for an automobile can be seen in figure 2.1.

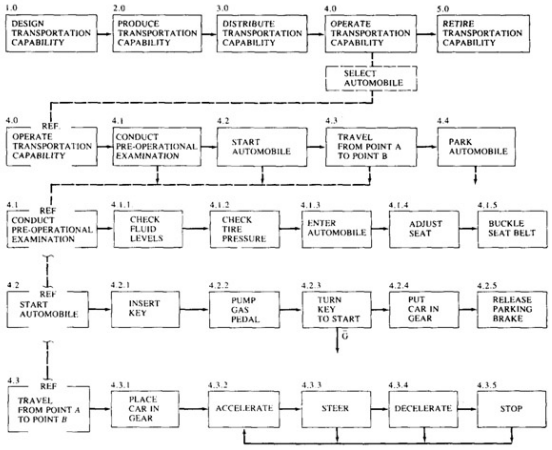

Figure 2.1: An example of a Functional Flow Diagram for an automobile.

2.1.2 Time Line Analysis

The FFD shows the sequential relationship between functions. But it does not show the actual duration of these functions. **Time Line Analysis** (TLA) does provide this functionality. When doing this analysis, we represent functions by blocks. The width of the block then denotes the duration of the function. An example of a TLA, for the European Robotic Arm (ERA), is shown in figure 2.2.

2.1.3 Functional Breakdown Structure

Quite often, a function isn't as easy to understand as you might want. In that case, a **Functional Breakdown Structure** (FBS) comes in handy.

In an FBS, a function is split up into several sub-functions. All of these sub-functions must be performed, to perform the actual function. (The FBS is therefore an 'and-tree'.) For example, let's examine the FBS shown in figure 2.3. Function A will only be done correctly, if the functions B, C, D and E are all performed correctly.

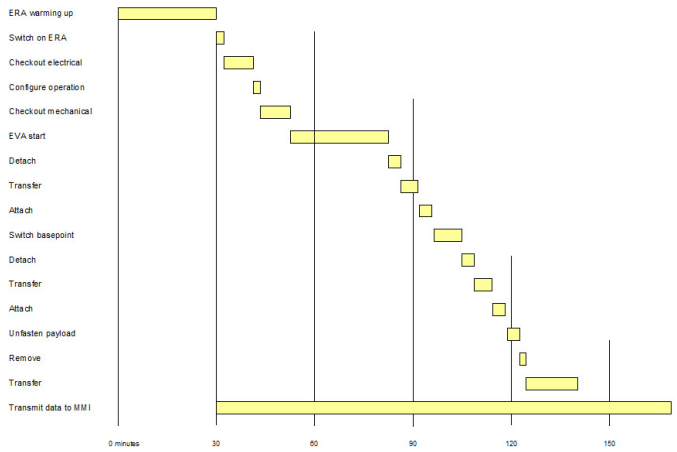

Figure 2.2: An example of a Time Line Analysis for the European Robotic Arm.

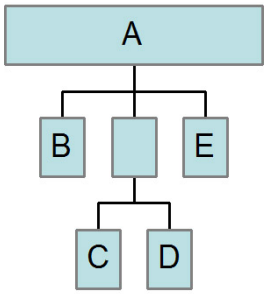

Figure 2.3: General form of the Functional Breakdown Structure.

2.1.4 Relations between functions

Let's suppose we have N functions. There are often relationships between functions. One way to display these relations, is by using an N^2 **diagram** (also known as an **interface diagram**). The general form of an N^2 diagram is shown in figure 2.4.

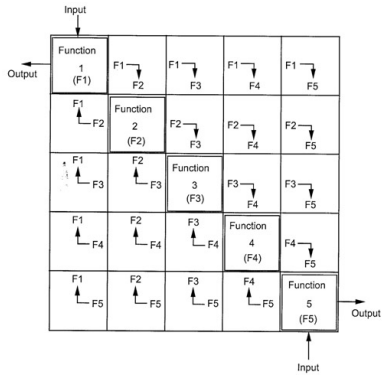

Figure 2.4: General form of the N^2 diagram.

An N^2 diagram is, in fact, an $N \times N$ table. On the diagonal, we place the functions. We use the remaining fields to display the relationships between functions. In those fields, we indicate which functions send what data (as input/output) to what other functions. By convention, input is displayed vertically and output is display horizontally.

You may wonder, how does this displaying work? Well, let's suppose function 1 sends data to function 2. In this case, we write this down in the second column of the first row. If, however, function 1 does not send anything to function 2, the corresponding cell remains empty.

2.2 Requirements

2.2.1 What are requirements?

Let's suppose we're designing a system. The **requirements** state what our system should do. When designing the system, we should keep these requirements in mind a lot.

We can sort requirements, based on their importance. **Key requirements** are requirements having an importance that is above average. There could also be requirements which have a very strong influence on the project. Such requirements are often called **driving requirements**. Finally, **killer requirements** are requirements driving the project to an unacceptable extent. In other words, they are virtually impossible to reach.

Requirements can be about a lot of things. For example, they can be about...

- Functions
- Configurations
- Interfaces
- Looks
- Environmental influences
- Quality
- Operation
- Support

- Verification

2.2.2 Generating requirements

How do we get requirements? The requirements are usually set by the customers. They are derived from the the POS and/or the MNS. Generating requirements is generally a difficult thing. There are therefore tools that can help you with it. The **Requirements Discovery Tree** (RDT) is one of them. An example of such a tree can be seen in figure 2.5.

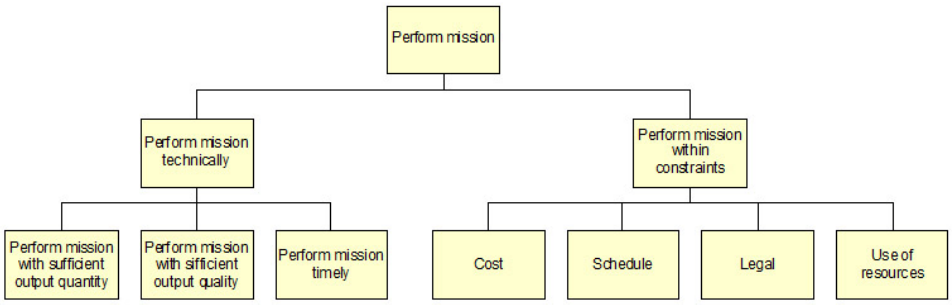

Figure 2.5: A general example of a Requirements Discovery Tree.

To generate an RDT, we start with the main requirement. This requirement is split up into separate sub-requirements. Each of these sub-requirements is then again split up. And this continues, until the requirement tree has sufficient detail. In this way, the whole Requirements Discovery Tree is generated.

2.2.3 The Requirements Traceability Matrix

When generating the requirements, we have seen that requirements often originate from other requirements. In fact, requirements usually have parent and children requirements. The relation between these requirements is summarized in a **Requirements Traceability Matrix** (RTM). By the way, the RTM looks quite a lot like the Requirements Discovery Tree.

The RTM is quite handy to check the requirement structure. If a certain requirement has no parent, then something has gone wrong. It could, on the other hand, also occur that a rather general requirement does not have any children. In this case, further requirements need to be added.

2.2.4 Valuing requirements

Let's suppose we have a list of requirements. Of course, some requirements are more important than others. To determine which requirements are more important than others, we can use a **Quality Function Deployment** (QFD) diagram. The general shape of a QFD diagram is shown in figure 2.6. A QFD diagram is often also called a **house of quality**, due to its house-like shape.

Let's suppose we're generating the QFD diagram for the requirements of a wooden desk. To generate it, we have to take certain steps.

1. First we write down the **product attributes**. These are the properties of the product, as seen by the customer. (The looks of the desk, the ease in use, the strength, etcetera.)
2. We then indicate the importance of the product attributes, according to customer. This goes on a scale from 1 to 5. (The customer could, for example, want a very efficient desk. Then the ease in use would get a 5, while the looks would only get a 2.)

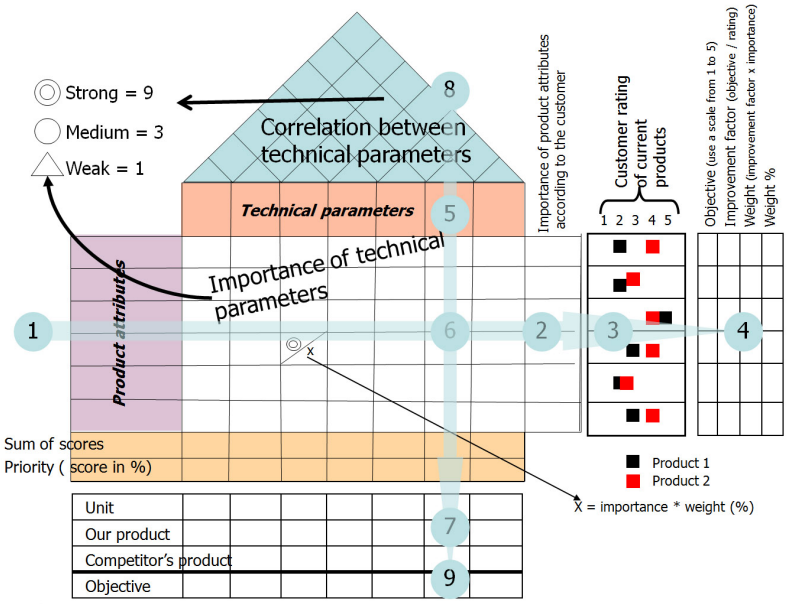

Figure 2.6: The general shape of the Quality Function Deployment diagram.

3. We will be comparing our product design to other similar products. To do this, we need to find out what our customer thinks of those other products. We do this for all product attributes. (So, for example, we have to know what our customer thinks of the looks, the ease in use and the strength of that brand new IKEA desk.)

4. We continue by setting the objectives for our product. This is again on a scale from 1 to 5. (How good do we want our desk to look? How strong should it be?) Once we have done this, we can also fill in the remaining columns on the right of the QFD. This eventually gives us the **weight** of the product attributes.

5. We then start to write down the **technical parameters**. These are the parameters, as seen by the designers. (Examples are the thickness of the wooden planks, the number of supporting beams, the wood type, the paint quality, etcetera.)

6. It is time to fill in the middle part of the QFD. Here, we insert the effect of the technical parameters on the product attributes. This can be either strong (9), medium (3), low (1) or non-existing (0). (For example, the thickness of the wooden planks effects the strength a lot (9), but it effects the looks only a bit (3 or 1). Also, the quality of the paint effects the looks quite a bit (9 or 3), but it does not effect the strength at all (0).) Once we know the effects, we multiply them by the weight (determined at step 4) to find the **importance**.

7. The hardest part is now over. To fill in the fields of the bottom rows, we simply need to add up all the values of the columns above them.

8. We can also indicate the **correlation** between the technical parameters. They can effect each other in a strong way (9), a medium way (3) a light way (1) or not at all (0). We do this for every pair of technical attributes. (For example, the number of supporting beams in the desk effects the thickness of the wooden planks. More supports means that the planks can be lighter.)

9. Finally, based on the results that were found, we give actual values to the technical parameters. (We can, for example, decide that we want wooden planks of exactly 1 centimeter thick.)

2.3 Resource Management

2.3.1 Technical Resource Budget

Let's suppose we're designing a system, like an aircraft. This design has several **technical resources**, like mass, capacity, availability, and so on. The set of all these parameters, which determine the success/failure of the project, is called the **Technical Resource Budget** (TRB).

During the design process, the technical resources change. However, they usually change in a bad way. (The mass usually increases during design.) It is our task to make sure they don't increase too much. (We should not consume too many technical resources.) The process of doing this is called **Technical Performance Measurement** (TPM). The technical resources involved in this are therefore also called **TPM parameters**.

2.3.2 Applying Technical Performance Measurement

How does Technical Performance Measurement work? To apply TPM, we need to set four values.

- First we set a **specification value** for our TPM parameters. This is the maximum our parameter may be. For example, we can say that the mass of our aircraft should be at most 100 tons.
- The next step is to construct a **target value**. This is the target we are strifing for. It is slightly below the specification value. For example, it could turn out that (during aircraft design) the mass often turns out to be 25% higher than planned. (This percentage is called the **contingency**.) In this case the target value for the mass will thus be 80 tons. (If the mass now rises by 25%, we're still not above the specification level.)
- The third step is to find the **actual value**. For example, it could be that (according to our current design) our satellite will weigh 88 tons.
- Finally, we determine the **current value**. It is equal to the actual value, plus the contingency. In our example, we would have a current value of 110 tons (being 25% above 88 tons).

When the current value is bigger than the specification value, there is a problem. There are three things that can be done. We could either improve our design, change the specification value, or change the uncertainty in our estimates (the contingency).

3. Design generation and validation

How do we generate the design of a system? And how is it validated? The procedures for that are discussed in this chapter.

3.1 Designing

3.1.1 The Design Option Tree

Let's suppose we're designing some system. How do we do that? Well, the first step is to start generating options. In what ways can we design our system?

A very handy tool for this, is the **Design Option Tree** (DOT). In such a tree, we start with a certain task our system needs to be able to perform. We then look at the ways in which this task can be performed. Every time, we split things up even more. It is important to note that the DOT is an 'or-tree'. Only one of the available options can be selected. An example of a DOT for registering images without delay can be seen in figure 3.1.

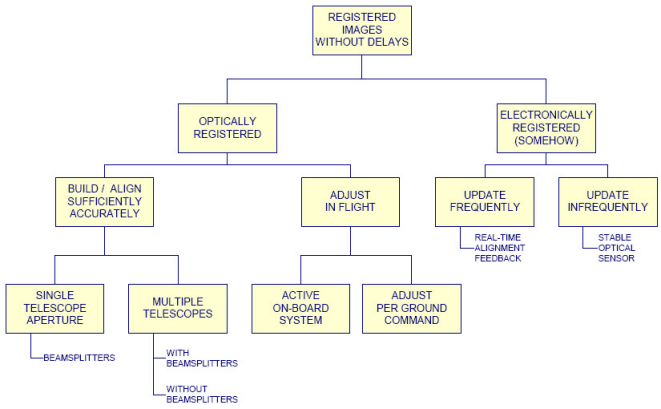

Figure 3.1: An example of a Design Option Tree for registering images without delay.

3.1.2 Choosing between designs

The DOT gives us a set of options. The next step is to choose the best option. This is a difficult process. There are therefore many ways to approach it.

A first thing we could do is eliminate concepts that are obviously not feasible. In this way, we can reduce the number of design options by quite a bit already.

The next step is to choose a **decision method**. We can distinguish two types of decision methods. In **ordinal (qualitative) methods**, the options are only ranked from best to worst, for every criterion. No individual score is given to them. So this is relatively easy. On the other hand, in **cardinal (quantitative) methods**, the options are given a score. This requires a bit more work.

3.1.3 Ordinal methods

Let's suppose we have a set of design options. Also suppose that, for every design criterion, we have a ranking of options, from best to worse. How do we now select the best option? There are, again, multiple

methods for this.

In the **Majority Rule**, we compare two alternatives. If, for a certain design option, alternative A is better than alternative B, then alternative A gets a point. Otherwise alternative B gets a point. We continue to do this for every design option. Eventually, the alternative with the most points wins.

The **Copeland Rule** is similar. However, we now compare an alternative with all other alternatives. Eventually, the alternative with the most points wins.

The **Rank-Sum Rule** is a more simple method. We simply add the ranking of all the criterions. So if an alternative is second for criterion 1 and third for criterion 2, then its score is 5. The alternative with the lowest score now wins.

Finally, in the **Lexicographical Rule**, we first rank the criterions on their importance. We then only look at the most important criterion. The alternative that is best for this criterion wins. (In case of a tie, we look at the second most important criterion, and so on.)

3.1.4 Cardinal methods

To apply Cardinal methods, we first need to rate our alternatives. This must be done for every criterion. (So that will give us a lot of numbers.) Once this is done, how do we select the best alternative?

The most-often used cardinal method is the **Weighted Objectives Method** (WOM). To use this method, we must also give importance rankings for the criterions. Once that is done, we should find the score for the alternatives. This is done by multiplying the criterion importance by the rating score. The individual criterion scores are then added up. When doing this, a WOM table, as shown in figure 3.2, is often helpful. Eventually, the criterion with the best score wins.

	Criterion 1		*Criterion 2*		……		*Criterion n*		*Maximum score*
Weighting factors	10		7		…….		4		yyyy
	Raw score	Wtd score	Raw score	Wtd score	Raw score	Wtd score	Raw score	Wtd score	Solution score
Alternative solution 1	5	50	8	56			6	24	xxxx
Alternative solution 2	…..	…..	…..	…..	…..	…..	…..	…..	…..
……	…..	…..	…..	…..	…..	…..	…..	…..	…..
Alternative solution m	…..	…..	…..	…..	…..	…..	…..	…..	…..

Figure 3.2: The general form of a Weighted Objectives Method table.

3.2 Design verification

3.2.1 Verification methods

So now we've actually gotten a design for our system. This design should of course match the requirements. But we do have to be sure of this. The process of making sure the design meets the requirements is called **verification**.

There are several methods of verification (so-called **verification methods**). Examples are, in increasing level of complexity, ...

- **Review of design:** Inspecting the design documentation, to make sure that the product satisfies the requirements.
- **Inspection of the product:** By inspecting the product, ensuring that it meets the requirements.
- **Analysis:** Using (mathematical) analysis techniques to check the properties of the product. (The FEM method is often used.)
- **Testing:** Subjecting the product to actual tests. For example, by using a prototype.

3.2.2 The verification level

Next to the verification method, also the **verification level** matters. Verification is usually performed on a level as low as possible. In **low level tests**, as few elements as possible are involved. This means that tests can be done early, and aren't very complicated. (For example, you often don't need to consider the satellite structure if you want to test its on-board computer.)

However, sometimes low-level tests are not possible. This is when elements start influencing each other. In this case **high level tests** may be necessary. (For example, when doing an automobile crash test, a nearly complete car is used. This is mostly because every part (having weight) influences the test.)

3.2.3 Test types

There are incredibly many ways to actually test a product. To list a few...

- **Functional tests:** Testing whether the product does the right things, and thus has the correct functionality.
- **Integration tests:** Testing whether all parts (mechanical/electrical/software) fit together in the correct way.
- **Structural tests:** Checking whether the structure meets the demands. By doing this, also the analysis tools (like FEM) and the workmanship are tested.
- **Aerodynamic tests:** Checking whether the aerodynamic properties of the product are sufficient. It also checks whether the mathematical model used during design was accurate. (Especially important for aircraft applications.)
- **Thermal tests:** Checking whether the thermal system works in a sufficient way. (Especially important for space applications.)

3.2.4 Aircraft design verification

When designing an aircraft, there are a lot of regulations it should comply with. Checking whether the aircraft design actually meets these regulations is called **compliance finding**.

The process of compliance finding can be quite elaborate. Several tests need to be done to show compliance with the airworthiness requirements. When this is done, the authorities can issue a **type certificate**. This certificate shows that the type of aircraft could be airworthy.

3.3 Design for production

3.3.1 What is design for production?

Let's suppose we're designing an aircraft. This aircraft eventually needs to enter production. With **production**, we mean all activities needed to turn a design into an actual product. During the design, we could make sure that production becomes relatively easy. By doing this, we **design for production**.

3.3.2 Recurring and non-recurring processes

A lot of processes are necessary to produce an aircraft. We can make a rather import distinction between recurring and non-recurring processes. **Non-recurring processes** are processes that are executed only once for every product type. (Examples include creating the design, programming machines and acquiring tools.) On the other hand, **recurring processes** are processes repeated for every product of a specific type. (Examples are manufacturing of parts, assembly of parts, and testing of the product.)

3.3.3 Designing the production process

To produce an aircraft, we need a production process. This production process of course also needs to be designed. The designing a production process consists of five phases. These are...

- **Concept generation:** Creating the general concept of the production process.
- **Feasibility check:** Checking whether the general concept is feasible/realistic.
- **Process definition:** Actually define the production process in detail.
- **Full scale development:** Set up the actual production process, according to the process definition.
- **Process validation:** Checking whether the production process actually meets the requirements and creates the right products.

When designing the production process, there are several things we need to pay attention to. The five most important points are...

- **Quality:** Is the quality sufficient? (Remember, we don't have to have the best quality for our product. A sufficient quality is good enough, and costs less.)
- **Time:** Is everything produced in time?
- **Money:** Is production as cheap as possible? (Is the amount of waste material minimized? Is the amount of elapse time minimized? Are transportation costs minimized? Etcetera.)
- **Volume:** Is the current production volume adequate to fulfill the demand? (Not too small, but also not too big?)
- **Law:** Does the production process obey safety, health, environmental and airworthiness regulations?

When all these points have received sufficient attention, then the production process is well designed.

3.3.4 Lean manufacturing

There is a philosophy in the world of production called **lean manufacturing**. According to this philosophy, production should be customer-focused, knowledge-driven, eliminating waste, creating value, dynamic and continuous.

The main focus lies on eliminating waste. According to lean manufacturing, **waste** is anything that uses resources, but does not add real value to the product or service. There are eight important kinds of waste, being

1. Overproduction
2. Waiting time
3. Work in progress (WIP) or inventory
4. Processing waste
5. Transportation
6. Movement or motion
7. Rework
8. Underutilizing people

4. System risks, reliability and costs

Systems always have risks attached to them. How do we deal with risks? How reliable are systems? And what do systems cost anyway? That's what we'll look at in this chapter.

4.1 Risk Management

4.1.1 What is Risk Management?

When designing a system, unexpected things can always pop up. That's the risk of designing. In fact, the **risk** is defined as the measure of uncertainty, when attaining a goal. Risk is always present. So it needs to be dealt with. The process of managing risk is (surprisingly) called **Risk Management** (RM).

Risk Management has two main branches. First, there is **Project Risk Management** (PRM). This branch deals with technical risks: anything having to do with development. The second branch is **Environmental Risk Management**. This concerns the management of environmental health and safety risks.

4.1.2 Determining risks

When performing risk management, we should first identify the risks. But once that is done, how do we know which risks we should reduce?

To see that, we need to look at two things. We need to examine the **likelihood of occurrence** and the **consequence of the event**. Multiplying these to things gives us the risk. (For example, if a lightning strike has a 20% chance of occurring in a year, and would cost ten thousand euros, then the risk in a year is two thousand euros.)

When risks are too high, two things can be done. Either the likelihood of occurrence can be decreased, or the consequence of the event can be decreased. (Of course we could also do both.)

4.1.3 The Risk Map

A way to visualize risks, is by using a Risk Map. One axis of the Risk Map denotes the likelihood of occurrence. The other axis indicates the consequence of the event. An empty risk map can be seen in figure 4.1.

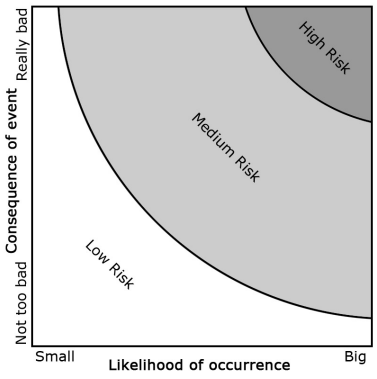

Figure 4.1: An empty risk map.

An event is represented by a point on the risk map. The seriousness (the risk) of the event can then be read from the diagram. The darker the region it is in, the more serious the risk is.

4.2 Reliability, maintainability and availability

4.2.1 The reliability function

In this part, we will examine reliability. The **reliability** R is the probability that a system will perform in a satisfactory manner, for a given period of time. The reliability of a system usually depends on time. It is therefore described by the **reliability function** $R(t)$.

From the reliability function, we can derive several other functions. The **failure distribution** $F(t)$ (also known as the **unreliability function**) is given by $F(t) = 1 - R(t)$. This failure distribution is then, in turn, related to the **failure density function** $f(t)$, according to

$$F(t) = \int_0^t f(\tau)d\tau. \tag{4.2.1}$$

Finally, the **hazard function** $r(t)$ (also known as the **failure rate**) is defined as

$$r(t) = \frac{\text{Failure Density}}{\text{Reliability}} = \frac{f(t)}{R(t)} = \frac{f(t)}{1 - \int_0^t f(\tau)d\tau}. \tag{4.2.2}$$

4.2.2 A probability refreshment

We saw that reliability is defined as a probability. So let's quickly refresh our knowledge of the world of probabilities. We define $P(A)$ as the probability that event A occurs. We also define $P(A|B)$ as the probability that event A occurs, given that event B occurs. There are several rules and definitions concerning this probability.

Two events A and B are **independent** if $P(B)$ does not depend on whether A occurs or not. (In an equation, $P(B|A) = P(B)$.)

The **conditional theorem** states that $P(A \cap B) = P(A)P(B|A) = P(B)P(A|B)$. For two independent events A and B, we can simplify this to $P(A \cap B) = P(A)P(B)$. This relation is also known as the **multiplication theorem**.

Two events A and B are **mutually exclusive** if they can not occur at the same time. So if A occurs, then B does not occur. (In an equation, $P(A \cap B) = 0$.)

The **addition theorem** states that $P(A \cup B) = P(A) + P(B) - P(A \cap B)$. If A and B are mutually exclusive, this can be simplified to $P(A \cup B) = P(A) + P(B)$. So to find the probability that either of two mutually exclusive events A and B occurs, we can simply add up their probabilities $P(A)$ and $P(B)$.

Finally there is Bayes' rule, stating that

$$P(A|B) = \frac{P(B|A)P(A)}{\sum_{i=1}^{n} P(B|A_i)P(A_i)}. \tag{4.2.3}$$

4.2.3 Failure types and distributions

Reliability is all about predicting failures. This is difficult, because there are many types of failures. (Think of fatigue, structural overload, electrical overload, wear, etcetera.) And every failure mode often has its own type of failure distribution.

There are many types of failure distributions. The type of failure distribution mainly depends on the failure rate. If there is, for example, a constant failure rate $r(t) = \lambda$, then we will get a **negative exponential distribution**. This distribution is described by

$$f(t) = \frac{1}{\theta} e^{-t/\theta}, \tag{4.2.4}$$

where $\theta = 1/\lambda$ is the **Mean Time Between Failures** (MTBF). From this, we can derive the reliability function. It is given by

$$R(t) = e^{-t/\theta} = e^{-\lambda t}. \tag{4.2.5}$$

Negative exponential distributions often occur when considering failures caused by a random event, like bad weather, bird strikes, etcetera.

Of course not all failure distributions have a constant failure rate. Many types of failures have an increasing failure rate as time progresses. (Older products are more likely to fail.) An increasing failure rate can be described by a **normal distribution**. The average of the normal distribution then indicates the average life time of the product.

Some failure types have a decreasing failure rate. (An example is the failure of humans. In the first few years, humans are relatively likely to fail. But if they live through their first years, they'll quite likely survive the next 50 years too.) Such failure rates can be described by a **Weibull distribution** or a **gamma distribution**.

4.2.4 Reliability of systems

Let's suppose we have a system consisting of n components. Also suppose that the system is build up, such that, if one component fails, then the whole system fails. (The components are said to be in **series**.) The reliability of the system can now be found, by multiplying all individual reliabilities. In an equation,

$$R_{system}(t) = \prod_{i=1}^{n} R_i(t). \tag{4.2.6}$$

We can also build up the system a bit differently. This time all components have to fail, for the system to fail. (The components are said to be in **parallel**). We now have

$$R_{system}(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t)). \tag{4.2.7}$$

4.2.5 Fault Tree Analysis

As stated before, there are many ways in which a system can fail. To investigate those ways, a **Fault Tree** can come in handy. A fault tree describes what events are necessary to cause a certain type of failure. An example of a Fault Tree is shown in figure 4.2.

When examining failures, it is important to know the minimum cut sets. A **minimum cut set** is defined as a minimum set of events leading to failure. For example, for the Fault Tree in figure 4.2, events F and H would cause failure. The set (F, H) is thus a minimum cut set. The set (E, C) would also cause failure. This set is, however, not a minimum cut set. (Note the word 'minimum'.) This is because event C is not necessary to cause failure. (Event E by itself already ensures failure.)

4.2.6 Maintainability

The **maintainability** M of a system is defined as the ease, accuracy, safety, and economy in the performance of maintenance actions. There are two important kinds of maintenance. There is **preventive**

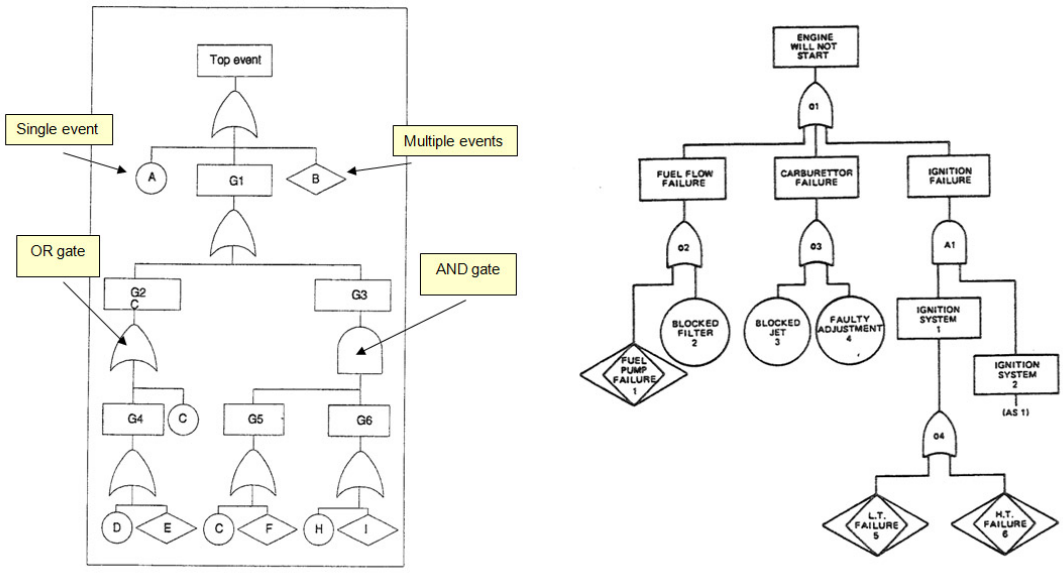

Figure 4.2: The general Fault Tree form (left) and an example applied to engine failure (right).

maintenance, aimed to prevent failure. (Inspection and periodic replacement of parts are part of preventive maintenance.) There is also **corrective maintenance**. The goal of this is to patch up the system after a failure has occurred. (Repairs are thus part of corrective maintenance.)

There are several parameters with which the maintainability can be expressed. The most important ones are...

- **Mean Time Till Failure** (MTTF): Average time until the system fails.
- **Mean Time To Repair** (MTTR): Average time needed to repair/restore the system.
- **Mean Preventive Maintenance Time** (MPMT): Average time needed to perform preventive maintenance.
- **Mean Time To Maintain** (MTTM): Average time needed to perform both preventive and corrective maintenance.
- **Mean Down Time** (MDT): The MTTM plus delays (for example, due to logistic reasons).
- **Mean Time Between Maintenances** (MTBM): The average time that is between two maintenance sessions.

4.2.7 Availability

The **availability** A is the probability that a system will be available when required for use. If we only consider the effects of failures and ignore maintenance, then we are examining the **inherent availability** A_i. It is given by

$$A_i = \frac{MTTF}{MTTF + MTTR}. \tag{4.2.8}$$

If we, however, don't consider failure, but only maintenance, then we are dealing with the **achieved availability** A_a. This is given by

$$A_a = \frac{MTBM}{MTBM + MTTM}. \tag{4.2.9}$$

4.3 Life Cycle Costs

4.3.1 Cost types

A system of course has costs. All the costs that are made in the life of a system/product are called the **Life Cycle Costs** (LCC).

The LCC can be split up into two important categories: Recurring and non-recurring costs. **Non-recurring costs** are costs that only occur once for every product type. (Examples include design costs, machines/tool costs and facility costs.) **Recurring costs** are costs that are present for every product of a specific type. (Examples are material costs, labour costs and energy costs.)

4.3.2 The Cost Breakdown Structure

There can be a lot of different types of costs, in the life of a system. One way to display these costs, is by using a **Cost Breakdown Structure** (CBS). The CBS is a tree, in which the Life Cycle Costs are split up into categories. An example of a CBS for an aircraft can be seen in figure 4.3.

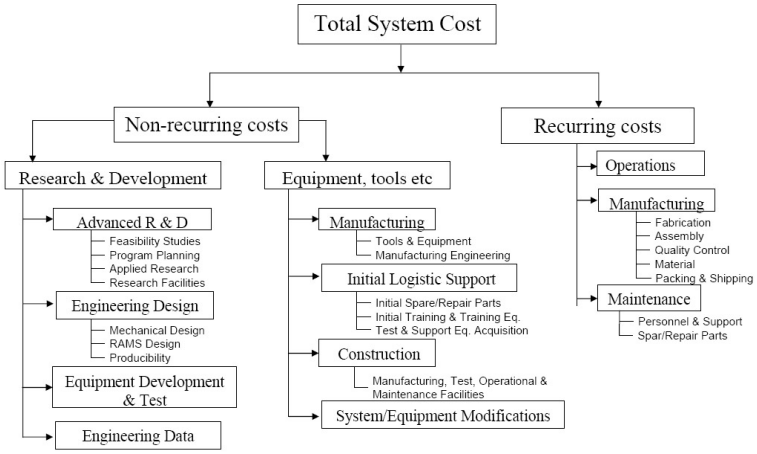

Figure 4.3: A Cost Breakdown Structure for an aircraft.

After the CBS has been created, numbers often need to be attached to categories. For this, **Cost Estimating Relations** (CER's) are often used. These relations are usually derived from experiences from earlier projects.

5. Systems Engineering practices

Now it's time to look at some practices which Systems Engineers face daily. How do we make documentation of projects? How do we apply concurrent engineering to projects? And how do we manage projects in general?

5.1 Documentation

5.1.1 Contents of documentation

Let's suppose we're creating a design, a product, or anything similar. It is our task to also create documentation. **Documentation** is a compilation of documents, giving information about the design/product. The documentation should include things like the requirements, the specifications, the planning, and so on.

You may wonder, what should we include in our documentation? The basic elements of documentation are...

- Relevance of the documentation/applicability (for example serial numbers)
- References used to build it
- References needed to use it
- Author/preparing entity (can also be a program)
- Checked by ...
- Released by ...
- Version (draft number)
- Date of preparation
- Amendments

Often computer programs are used to manage the documentation. If this is the case, we're one step closer to Knowledge Based Engineering. In fact, **Knowledge Based Engineering** (KBE) is defined as the use of advanced software techniques to capture and reuse product and process knowledge.

5.1.2 Structuring documentation

There should of course be a certain structure in our documentation. This is where Configuration Items come in handy. **Configuration Items** (CI's) are entities that are worthwhile to be controlled separately. A CI has a charecteristic...

- **Fit:** The physical interface of the CI to other items. (If the CI changes, other items may need to change as well.)
- **Form:** The (geometrical) shape of the CI.
- **Function:** The actions the CI is designed to perform.
- **Flow:** The origin of the CI. (Where the part comes from.)

5.1.3 The Object-Oriented design paradigm

The **Object-Oriented** (OO) design paradigm is a way of thinking about problems. It can also be used to structure documentation in a nice way. In the OO paradigm, we use **objects** to describe entities. These objects both incorporate the **attributes** of the item, as well as its **behaviour**.

We can classify multiple objects into a **class**. This is called **classification**. (For example, both the car that's parked in front of my house and the car that's just passing by are objects of the class **car**.) An object is said to be an **instance** of its class. Each instance of the class has the same attributes. (For example, all cars have an attribute 'brand'.) However, each instance does have different values for their attributes. (For example, car A can have as brand 'BMW', while car B has as brand 'Kia'.)

Inheritance is the sharing of attributes and operations among classes. It is based on a hierarchical relationship. (For example, both a car and an airplane are vehicles. And both the car, the airplane and the vehicle have an attribute 'weight'.) **Polymorphism** means that the same operation may behave differently on different classes. (For example, both a car and an airplane have an operation called 'open door'. However, this operation is quite different for the two vehicle types.)

5.2 Concurrent Engineering

5.2.1 What is Concurrent Engineering?

Concurrent Engineering (CE) is defined as the concurrent running of separate phases during the product definition trajectory. In other words, multiple things are done at the same time. Concurrent Engineering thus employes as much **parallel processing** as possible.

When applying Concurrent Engineering, it is important to have correct and complete **information**. If this is not available, parts might not fit during assembly. There should also be effective **communication**. Or the customer might wind up with a product he didn't ask for. The **responsibilities** should also be clear and well-documented.

5.2.2 The downsides of Concurrent Engineering

Concurrent Engineering can save a lot of time. But it is not without downsides. There are risks attachted to it. In CE, you start on one process, while not having finished the process before it. (Partial results are often used.) This may lead to necessary rework.

To reduce these risks, good communication is necessary. Also, often use is made of **Design/Build Teams** (DBT). These teams both participate in the design and the build phase of the project.

5.2.3 The Integrated Product Model

Managing information is very important, when applying CE. When using an **Integrated Product Model**, all information is located in one single source. This has several advantages. One of them is that the information can be controlled by a **Product Data Management** (PDM) system. This system guards the data and manages modifications applied to it.

Inside the PDM system, information often has a **status**. This status gives information on the stage the information is in. Is it a concept? Or is it already a final version? Can it be used for later parts in the project? The PDM system also keeps track of who needs to be notified, when information changes.

5.3 Project Management

5.3.1 What is Project Management?

Let's suppose we're working on a project. (A **project** is defined as a coherent set of activities meant to realize a defined outcome.) This project needs to be managed. This task is called Project Management.

In fact, **Project Management** (PM) is defined as a formal management discipline whereby projects are planned and executed according to a systematic, repeatable, and scaleable process. Project Management is an important task. The quality of Project Management can make or break a project.

Project Management consists of three important phases. Before the project has even started, it should already be **defined**. After this, the actual **project planning** is set up. The third phase is when the project is running. It then needs to be **controlled**. A general overview of the Project Management tasks can be seen in figure 5.1. In the next few paragraphs, we'll go a bit more into depth concerning those tasks.

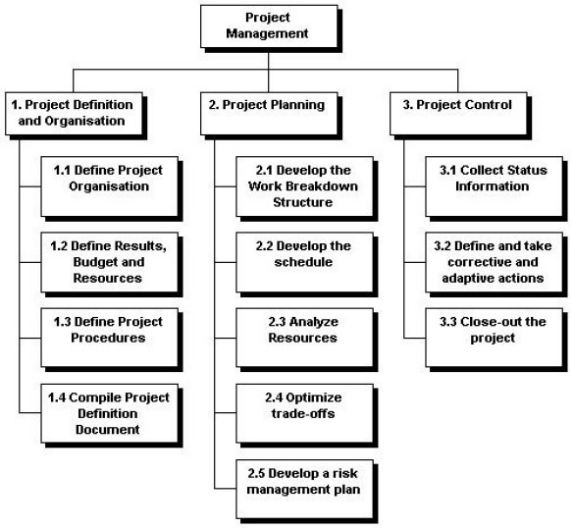

Figure 5.1: An overview of everything that is part of Project Management.

5.3.2 Project definition

The first thing to do, when managing a project, is to define what the project is. This consists of three sub-steps.

- We need to define the **project organization**. It should be clear which person has what responsibilities.
- We define **results, budgets & resources**. This is the step in which the POS is defined. It should be known when the project should be finished, and what resources are allocated to it.
- We also define **project procedures**. How do team members communicate? How often do we have team meetings? What do we do if people are late? How do we solve issues in the team? Questions like that should be answered.

Once these three subjects are defined properly, the **Project Definition Document** (PDD) can be compiled. The project is then defined.

5.3.3 Project planning

The second phase of project management is project planning. In this step, it is decided what processes take place at what times. There are five important parts in this planning.

- First we need to know what **work** needs to be done. For this, Work Flow Diagrams and Work Breakdown Structures are used.
- Once we know what needs to be done, we can set up a **schedule**. What team member is good at what task? And what tasks can be done simultaneously? (Think of Concurrent Engineering.)
- We then analyze the **resources**. (With resources we also mean non-material resources, like manpower.) Are there resources that are overutilized or underutilized? In this step, **Gantt charts** can be used. Although we won't go into detail on that.
- We **optimize trade-offs** between the project parameters (results, budget, resources). Will the POS be satisfied? Can we satisfy it in a more efficient way? What other parts of the planning can be optimized?
- We develop a **Risk Management Plan** (RMP). What risks are there? How can we minimize the probability of occurrence? And how do we keep the damage as low as possible, if something bad does occur?

When these five steps have been performed, the project planning is complete.

5.3.4 Project control

During the project, the Project Manager will not sit still. It is his task to inform key participants of the progress made. He/She should also realign the project, when things tend to go wrong. The tasks of the Project Manager can be summarized in three groups.

- The Project Manager has to **collect status information**. He should determine what information is monitored, how this is done, and how often it is done. Important subjects to monitor are the schedule status, the open issues and the risks that are present.
- The Project Manager should also take **corrective and adaptive actions**, when necessary. It is his task to determine which actions need to be taken, and how these are communicated to the team and the stakeholders.
- Finally, the Project Manager should **close out the project**, once it has been completed. What can be learned from the past project? What could have been improved? And has all the paperwork been finished? Those are questions that need to be answered.

When a Project Manager performs all these tasks succesfully, the project has a high likelihood of succeeding. However, if the tasks are not performed satisfactory, the project will most likely fail.