SYSTEMS DESIGN AND ENGINEERING
- LUBRICATING MULTIDISCIPLINARY DEVELOPMENT PROJECTS

Dr. Ir. G. Maarten Bonnema
Ir. Karel T. Veenvliet
Dr. Ir. Jan F. Broenink
Preface

This book has a history of several years. It is based on the systems engineering lectures for Industrial Engineering Students since 2004. Adding to that the knowledge of years of teaching systems engineering to Civil Engineering and Electrical Engineering Students, a nice multidisciplinary mix is created.

This book has been on the mind of the first author for a long time, however it would not have been written without the support and knowledge of the other authors and the initiative of Prof. Rikus Eising.

Enschede January 2012
Dr.Ir. G. Maarten Bonnema
Ir. Karel T. Veenvliet
Dr.Ir. Jan F. Broenink
## Contents

<table>
<thead>
<tr>
<th>1 Introduction</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Development of Modern Systems</td>
<td>9</td>
</tr>
<tr>
<td>1.2 Systems Engineering: the Approach in this Book</td>
<td>10</td>
</tr>
<tr>
<td>1.3 Designing a Solar Race Car</td>
<td>11</td>
</tr>
<tr>
<td>1.4 Notes on How to Use this Book</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2 The Systems Engineering Process</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 The Essence of the Systems Engineering Process</td>
<td>16</td>
</tr>
<tr>
<td>2.2 A Practical Implementation of Systems Engineering</td>
<td>18</td>
</tr>
<tr>
<td>2.2.1 System Level</td>
<td>20</td>
</tr>
<tr>
<td>2.2.2 Sub-system Level</td>
<td>23</td>
</tr>
<tr>
<td>2.2.3 Integration</td>
<td>25</td>
</tr>
<tr>
<td>2.2.4 Validation</td>
<td>26</td>
</tr>
<tr>
<td>2.3 The Role of the Systems Engineer</td>
<td>28</td>
</tr>
<tr>
<td>2.4 A Short History of Systems Engineering</td>
<td>29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 Systems Thinking Tracks</th>
<th>33</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Introduction</td>
<td>33</td>
</tr>
<tr>
<td>3.2 Dynamic Thinking</td>
<td>35</td>
</tr>
<tr>
<td>3.3 Feedback Thinking</td>
<td>37</td>
</tr>
<tr>
<td>3.4 Specific-Generic Thinking</td>
<td>39</td>
</tr>
<tr>
<td>3.5 Operational Thinking</td>
<td>40</td>
</tr>
<tr>
<td>3.6 Scales Thinking</td>
<td>41</td>
</tr>
<tr>
<td>3.7 Scientific Thinking</td>
<td>43</td>
</tr>
<tr>
<td>3.8 Decomposition–Composition Thinking</td>
<td>44</td>
</tr>
<tr>
<td>3.9 Hierarchical Thinking</td>
<td>45</td>
</tr>
</tbody>
</table>
## CONTENTS

3.10 Project Thinking .................................................. 47
3.11 Life-cycle Thinking ............................................... 47
   3.11.1 Product Life-cycle Thinking .............................. 48
   3.11.2 Resource Life-cycle Thinking ............................. 49
   3.11.3 Project Life-cycle Thinking ............................... 49
3.12 Safety Thinking .................................................. 51
3.13 Risk Thinking .................................................... 51
3.14 Wrap up .......................................................... 52

4 System Design Tools ................................................. 53
   4.1 Introduction .................................................... 53
   4.2 9-windows diagram .............................................. 54
   4.3 Context Diagram ............................................... 55
   4.4 Scenario’s ...................................................... 56
   4.5 Functional Modeling and Analysis ............................ 58
      4.5.1 Function trees ............................................. 59
      4.5.2 Functional Block Diagrams ............................... 60
      4.5.3 State-Transition Diagrams ................................. 61
      4.5.4 Influence Diagrams ...................................... 62
   4.6 N²-diagram ...................................................... 64
   4.7 Architectures .................................................... 65
   4.8 System budgets ............................................... 67
   4.9 FunKey Architecting ............................................ 69
      4.9.1 Coupling matrix to budgets ............................. 71
      4.9.2 FunKey as tracking tool ................................ 72
   4.10 A3 Architecture Overviews ................................... 72
   4.11 Failure Mode and Effect Analysis ............................ 75
      4.11.1 Introduction .............................................. 75
      4.11.2 FMEA-team ................................................ 76
      4.11.3 FMEA Form ............................................... 76
      4.11.4 FMEA procedure ........................................... 78
   4.12 Risk Management ................................................ 81
      4.12.1 Decision tree .............................................. 83
      4.12.2 Risk register .............................................. 87
   4.13 Documentation and Reviewing .................................. 88
      4.13.1 General Documentation Guidelines ....................... 88
      4.13.2 Document Contents ....................................... 90
The field of designing multidisciplinary systems is presented with a few examples. It is shown that in addition to the well-known engineering fields, another discipline is needed: systems engineering. It is introduced, together with the way we treat it in this book. Also, we will give an example that is used throughout the book.

1.1 Development of Modern Systems

A digital single-lens reflex camera (DSLR camera), a medical imaging device like an Magneto Resonance Imaging-scanner, a storm surge barrier, a modern passenger car and commercial aircraft are products that contain many parts and have to perform many functions. All parts and functions have to work together and work in an environment with users. Even more so, some of these products have a long life in which the environment may change dramatically.

Looking inside the products, we see that there are mechanical and electronic parts, software components, styled parts, and there is behaviour that is designed for good interaction with the users. Also, the products have to perform well from a financial point of view, whether that is because they are bought from hard-saved money of a consumer, or paid for by tax money. All these aspects are dealt with mostly by specialists: electronic, mechanic, software engineers, user interface designers, product stylers, financial experts etc. Nowadays the effect is that these products, systems as we will call them in this book, are developed by a large team—often even several teams—of people with diverse backgrounds. Communicating and working towards a common goal is not easy in such a setting. Systems Engineering is a discipline that aims at coping with this by focussing on the whole.
1.2 Systems Engineering: the Approach in This Book

Systems engineering is an established discipline for which quite a few good books exist [Blanchard and Fabrycky, 2011; Maier and Rechtin, 2000; Sage and Armstrong jr, 2000, to name a few]. Designing systems is a less documented competence. In this book we treat a combination of the two. Although we will not go into the finest details of systems engineering, we do provide a firm basis so that the designing process can take place as smoothly and efficiently as possible. It bases on three pillars:

- **The Systems Engineering Process**: how is the process of bringing systems to being organised? This is what is generally called systems engineering.
- **Systems Thinking Tracks**: a number of ways of thinking about the system that is being designed, its context, its user, its past and its future are given. These thinking tracks stimulate the creativity of the system designer, and force him or her\(^1\) to think in- and outside any limits set by the system requirements, defined in The Systems Engineering Process.
- **System Design Tools**: a number of practical tools that help in both The Systems Engineering Process and Systems Thinking Tracks.

We separate the three, as we believe often the treatment of systems engineering is needlessly complicated by documentation processes, standards and formalities, which are in fact only part of support tools. The SE process in itself is quite simple and logical. The way of thinking in systems design, as described in chapter, is a competence that is widely applicable and leads to systems that fit their purpose, not only the requirements.

Throughout the book, there will be cross-references from the systems engineering process to the thinking tracks and the tools and vice versa. These will be marked in the margin. When, for instance using the N\(^2\)-diagram can be useful, it will be marked like is shown in the margin here.

\(^{\text{[4.6]}}\) N\(^2\)-diagram

As for the subtitle “lubricating multidisciplinary development projects”, lubrication is more than making sure things run smoothly. Of course that by itself is already quite an accomplishment, and the systems engineer in cooperation with the project manager will have to take care of that. However, a

\(^1\)In the remainder of this book we will refer to the system designer as him. This is by no means intended as a preference. It is merely to avoid using him/her, he/she in the text.
Figure 1.1: The World Solar Challenge across the Australian Continent (picture from the Veolia World Solar Challenge organization [http://www.worldsolarchallenge.org/])

lubricant also cools, reduces noise, prevents rust and seals. These can be seen as metaphors for the systems engineer’s practice as well:

**cooling**: prevent the project from getting overheated by balancing the development load over people and time;

**noise reduction** in the sense of unwanted sounds: are the things said correct? Is that what we deliver verified? Or are we merely building upon assumptions?

**rust prevention** as in avoiding decay of knowledge: are the results, arguments and criteria documented? Is the code documented?

**sealing**: ensuring all parts fit together tightly. If not, the systems engineer acts as the sealing so that nothing leaks without noticing: no energy, no information, no material.

### 1.3 Designing a Solar Race Car

The book is aimed at being practically applicable. One way this is done is by being concise and to the point. Also, practical applications are used. As a running example, we use the design of a solar race car that can compete in
1. Introduction

Table 1.1: Characteristics of the World Solar Challenge.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Value</th>
<th>Unit</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total length</td>
<td>3010</td>
<td>km</td>
<td></td>
</tr>
<tr>
<td>Number of race days</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duration of a race day</td>
<td>8:00 – 17:00</td>
<td>h</td>
<td>a little margin allowed to find a camping spot</td>
</tr>
<tr>
<td>Maximum speed</td>
<td>130</td>
<td>km/h</td>
<td>Northern Territory</td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>km/h</td>
<td>South Australia</td>
</tr>
</tbody>
</table>

Project characteristics

| Total Budget | 1 M€  |
| Development time | 14 months |
| Team size      | 18 students |

The World Solar Challenge ([http://www.worldsolarchallenge.org/](http://www.worldsolarchallenge.org/)), a race across the Australian continent, figure 1.1. This book is written by faculty members from the University of Twente. Student teams from that University have competed in the Challenge in 2005, 2007, 2009 and 2011. Such a development project is complicated enough to explain most of the principles in this book, yet it is not that difficult that a lot of subject matter information is needed before the problem can be understood. In this section a short introduction to the problem is given. The main characteristics of the race, and of the project to develop the race car are given in table 1.1.

The solar racer must receive its energy from the sun. The regulations have developed over the years. In 2011 it is allowed to have either a 3m² high-performance GaAs solar panel, or a 6m² silicon panel. The difference in performance comes from the efficiency in conversion from the incident solar power into electricity. Where GaAs-panels can have an efficiency above 30%, silicon panels will not exceed that value (up to 25%). In addition to the solar panel, a small battery is allowed to accommodate for periods with low solar irradiation. At the start of the race, this battery may be fully charged.

In addition to the development of the car, as the project in Twente is fully carried out by students and sponsored by mainly regional organizations, organizing funding, support and publicity play an important role. Of course, as the race-start is a clear date, delay in the project is not an option.
1.4 Notes on How to Use this Book

This book assumes a general knowledge in engineering and design although the discipline is not very relevant. It is not necessary to read the book from cover to cover, it is highly recommended to read chapter 2 thoroughly as it forms the basis for all systems engineering processes.

Chapter 3 can be read per section, or when specific issues arise in a project. Nevertheless it is also worth reading completely. Chapter 4 can be used as a handbook or reference manual: looking up the quick start guide for a specific tool when the need arises.
This chapter introduces the systems engineering process: the way of working in developing large and complex systems. By investigating the problem, and dividing it into smaller pieces, the work is made more manageable. The importance of identifying the interfaces between the pieces, and of how to put the pieces together is treated. Also, the role of the systems engineer is discussed briefly.

Systems Engineering is an accepted approach for designing large systems. In most military development projects it is compulsory to apply strict systems engineering guidelines as defined in guidelines like DOD 5000.02 [Blanchard and Fabrycky, 2011]. In the Netherlands, a cooperation between government, public transport, construction companies and engineering firms, has led to a guideline for systems engineering (www.leidraadse.nl). Using this guideline is not compulsory, though.

In this chapter, we will present the systems engineering process concisely. One mere chapter is not enough to show all aspects of all possible situations. But as presented in the introduction, that is not the purpose of this book. Yet, the essence of the systems engineering process in a design situation can be given in sufficient detail for an interested engineer to understand the principles of systems engineering. It is expected that with the basis given in this chapter, one can start working in a systematic way that fits the systems engineering process, and apply the principles that will be presented in the remainder of the book.

If one wants to become an expert in systems engineering, and seek certification through for instance INCOSE’s CSEP program (www.incose.org), then further study using for instance [Blanchard and Fabrycky, 2011] [INCOSE SEH Working Group, 2008] will be essential.
The chapter ends with a look at the role of the systems engineer, and a brief history of the systems engineering discipline.

2.1 The Essence of the Systems Engineering Process

A system is “a set of interrelated components functioning together toward some common objective(s) or purpose(s)” [Blanchard and Fabrycky 2011, p.17]. There are dependencies between the components in the set, and between the system in its entirety and individual components. This means that although a component can be regarded separate from the system, its full potential and its full functioning and behavior will only show in the system. Analogously, the system will only show its full potential and behavior when it is fully composed by its components and in use in a context. Moreover, the system might show behavior that was not intended or expected, or even thought possible at design time.

Figure 2.1 shows the goal of system design. A system under design (S.U.D. or SUD) is designed to deal with an identified issue. This issue can be a market opportunity, a space mission, a traffic congestion etc. However, the issue is not an isolated item. It is part of a context. There are—often many—stakeholders involved. And even when everything is inventoried, there may be things left out: the rest of the world. The system under design has to fit all these. The systems engineering process is developed to investigate the identified issue, the context and the stakeholders, and to develop a suitable solution. It is the system designer’s responsibility to deliberately wander outside these inventoried issues and context to ensure proper fit between the SUD and the rest of the world. The thinking tracks and some of the tools presented in chapters 3 and 4 are particularly suited for this wandering.

As the systems we are dealing with here typically consist of many components, developing is done by large to very large teams. (Please note that the principles in this book are equally applicable to the development of smaller products.) Starting from an identified issue, the development process deals with increasingly more, yet smaller details. The number of these details can be mind-bogglingly large, as is shown in Figure 2.2. This pyramid shows how, starting from an identified issue with a few characteristics, the number of details explodes. A further complication is to put all these individual parts and lines of software code together—integrate them—to create a coherently working system. As a warning, that part of the development cycle is in general greatly
2.1 The Essence of the Systems Engineering Process

Figure 2.1: The System Under Design (S.U.D.) has to fit the identified issue, the context, the stakeholders and the rest of the world.

Figure 2.2: The Muller Pyramid, adapted from Muller [2007]. It shows how for a complex system like an ASML wafer scanner, a Philips Healthcare MRI scanner, or a passenger aircraft, the number of issues explodes when going from the initial customer need to engineering details.
underestimated in required time and effort.

In the next section we will explore a practical way to deal with this explosion in the number of details, and with the putting-together.

### 2.2 A Practical Implementation of Systems Engineering

System Engineering is, as figure 2.1 shows, all about finding a solution to an identified issue, while considering the context, the stakeholders and the rest of the world. In fact, this is what happens at different levels of detail, see figure 2.3 defining the issue, then finding an appropriate solution and accepting it before proceeding. Many engineers think in solutions. They even investigate the problem using a solution. With the loop in figure 2.3 in mind this is not a problem. However, when the distinction between problem domain and solution domain is not made, jumping to conclusions, and picking the first solution that comes to mind become real dangers. Every engineer, not only system engineers need to be continuously aware whether they are thinking in the problem domain or in the solution domain. And they need to switch often between the two. The thinking tracks in chapter 3 can serve as aids in this.

The solution defined from progressing the loop in figure 2.3 one to many times, describes the context for the next level of detail. It does mean, that when the level of detail increases, the number of parallel processes increases. In a design process this is often seen as parallel sub-projects, where each of the sub-projects develops a sub-system. This is visualized in figure 2.4. This figure shows how systems engineering repeats the same development process, shown in figure 2.3, on different layers:

1. System Level
   - Customer Wish $\rightarrow$ Requirements $\rightarrow$ Design
2. Sub-system Level
   - Requirements $\rightarrow$ Design
3. Assembly Level (depending on the field of application, this can be called differently, e.g. the module level)
   - Requirements $\rightarrow$ Design
4. Component Level
   - Requirements $\rightarrow$ Design
5. etc.

In the next sub-sections, we will treat this process in further detail.
2.2 A Practical Implementation of Systems Engineering

Investigating the problem

Defining the solution

Input from customer, stakeholders, experts etc.

Next development step

Figure 2.3: The systems engineering process as a loop of exploring the issue, and defining a solution. The right-hand (downward) arrow is actually designing the solution. The left-hand (upward) arrow depicts the verification: does the solution resolve the issue sufficiently and without (too much) negative effects?

Customer Wish

System Requirements

System Design

SubSystem Req.

SubSystem Design

Assembly Req.

Assembly Design

Figure 2.4: The Systems Engineering Process. In each of the arrows in this diagram, one can see the loop from figure 2.3
2. The Systems Engineering Process

2.2.1 System Level

The process starts with a customer wish. In industrial design, the exact customer wish is mostly not clearly defined. Sometimes there is a market opportunity, or a gap in the product portfolio of a company. Sometimes it is merely an idea. Then, the idea has to be elaborated before the full development process can start. In [Eger et al., 2010, in Dutch, an English version is in preparation], this creative goal finding process is dealt with. Also, the customer need not be one person. It may be a group, or a target group that is defined by a marketing agency. In such cases, the marketing department of the company expresses the customer wish.

It is wise to discern between the customer of the development company, and the end user(s). The system designer has to take interests of both into account, and balance them.

In engineering projects, the customer wish is often expressed in the form of an initial or preliminary design, or in the form of a system requirement specification. The task of the system engineering, system designer, system architect, lead designer, or how he is called in the organization, is to investigate the customer wish further, and to translate that wish into a system requirement specification (often simply called system requirements, or system specifications). This involves trade-offs between what the technology can offer, and what risks are involved. What the end-user wants, and what he likes to pay, what the customer is willing to invest, and what the supplier can deliver, so that:

on the one hand, the customer:

- gets a product in time,
- for a reasonable price,
- with a reasonable change of success in the market.

while on the other hand, the supplier:

- can build the product,
- with sufficient margin,
- with acceptable risks in the development process.

1 also called system requirement specification or program of requirements, or similar terms.
The end-users will make the success in the market, if the product has a desirable combination of features (=functionality) for a competitive price at that time. So the end-user’s interests should be largely in line with those of the customer, if dealt with properly.

For this, it will be necessary to not simply translate the customer’s wish(es) into measurable quantities. A deeper understanding of what is needed and what is possible is necessary. Most of the thinking tracks in chapter 3 are essential for this. Further, the system designer will have to elaborate on what the customer in essence expects from the product. At the same time the state of the art technology should be known and understood. What break-through is imminent? What can be put to use in the time-frame of this development project? Some preliminary investigation into the system design might be necessary to come to a complete and acceptable set of system requirements. These system requirements have to be reviewed and accepted by both the customer (Meaning: "If you build this, I will buy it, and pay this price for it."), and the builder ("Yes, I believe, we can build this for this price and in this time-frame."). System Requirements, System Design and all other blocks (with the exception of Customer Wish) are mostly written in documents. Section 4.13 contains detailed contents of the documents that are treated here.

Up to here, we have looked at the left-most column in figure 2.4 The top two blocks –Customer Wish and System Requirements– correspond to the the system level in the pyramid in figure 2.2 The system design process becomes really interesting in the block System Design, and in the transition to the second column. This corresponds with the intermediate level in the pyramid in figure 2.2. As one can see, the number of issues increases significantly to a number that cannot be overseen easily. Also, the number of disciplines and the deepness of the knowledge required from those disciplines is increasing as very diverse knowledge is required:

- what is technically feasible in the time frame before product launch?
- what is producible (e.g. can we find the right production equipment)?
- what can our development team handle (e.g. can we hire enough developers?)
2. The Systems Engineering Process

Figure 2.5: Subsystems and interfaces: all interaction between the subsystems takes place via identified and specified interfaces (IFx). Note that within a subsystem, a further splitting up can take place. Then again interfaces have to be defined.

- what will the impact on our corporate image be?
- will it actually solve the identified issue?

The thinking tracks in chapter 3 are intended to aid in resolving these (and other) questions.

From the system requirements, a top-level design, or system design is created. Here, the total system is divided in its main subsystems and the interfaces between these subsystems are described. A subsystem here is defined as a system that is largely independent from the other subsystems, and it performs a set of functions, as described in the system design. From the definition of a system in section 2.1 it should be clear that complete independence of one subsystem from the others is impossible. If it were independent, there would be no system. All cooperation between the subsystems is made explicit by the interfaces, see figure 2.5 what

- information,
- energy,
- materials,
- persons,
- etc.

is transferred between two subsystems? For the systems engineering process to run smoothly, there should be no hidden transfer of any of those. All inter-
2.2 A Practical Implementation of Systems Engineering

action takes place only via identified interfaces. Be aware that investigating the interfaces is the core business of a system designer. As noted by Robert Halligan, a dyed-in-the-wool system designer and trainer:

"There are two kind of system engineers: those who look at the interfaces, and amateurs."

While a designer is mainly interested in delivering his component or subsystem, the system designer should be focused on the things that bind these components and subsystems together: the interfaces. Therefore, the system design document specifies the performance of the subsystems, and describes the interfaces. Depending on the type of the interfaces, this can be done using CAD drawings (to ensure proper mechanical fit), signal definitions, and software class diagrams, etc. But also conformity to standards and testing procedures should be specified as these define interfaces with the rest of the world (see figure 2.1). The main concept choices and the design of long-lead items are also covered by the System Design. A more detailed description is found in the document structure in §4.13.

2.2.2 Sub-system Level

Moving to the second columns marks a change in the development project. Up to here, it was a systems-only project, involving a relatively small team. Now, in most cases, with starting the definition of the subsystem’s requirements the project teams for developing these subsystems also start. The lead designer of such a team studies and interprets the system design for his subsystem, and distills the requirements for his subsystem from it. This results in a set of subsystem requirements in much the same way as the system requirement is an interpretation of the customer wish. And like there has to be agreement between the customer and the supplier about the system requirement, there has to be agreement about the subsystem requirement between the system engineer and the subsystem lead designer (and a few more people, as will be dealt with in §4.13).

2 Founder and owner of PPI international [http://www.ppi-int.com/](http://www.ppi-int.com/)
Figure 2.6: The integration process that accompanies the development process in figure 2.4. The arrows shown are placeholders for the cyclic process in figure 2.7.

In the same way as was done on system level, from the subsystem requirements, a subsystem design is created: in what way can we define an implemented system that meets the requirements. On the subsystem level, there will be more implementation information like what hardware will be used, what software, etc. In §4.13, more information about the contents of the subsystem design document is given.

From here, the process becomes less and less systems design and engineering as it will become less and less multidisciplinary engineering. Of course this depends on the scale of the initial project. It may be necessary to repeat these steps at some more levels before the intermediate level in figure 2.2 is left. So, while figure 2.4 shows three levels: the system level, the subsystem level and the assembly level, there can be four, five, six or even much more levels. As the process is defined generically, using hierarchical thinking and the 9-window diagram, one can find the appropriate level for the system engineer(s) to stop leading the development process and start monitoring it.
2.2.3 Integration

Figures 2.3 and 2.4 suggest that developing is done by “divide and rule”. By splitting up the initial problem into smaller, more manageable chunks, and by having the system engineers watching the interfaces, there will be a well-functioning system in the end. This is not necessarily so, as already suggested in §2.1. When the assemblies become available, they have to be put together to become subsystems. The subsystems then have to be put together to finally become the requested system. Putting it all together and turning the power switch will almost certainly not work.

Figure 2.6 shows how integration takes place. In this case we show it starting at the assembly level, but it generally starts at component or part level. The assembly is built and verified. Although figure 2.6 shows this as a sequential process, it is in practice a series of loops as shown in figure 2.7.

The verification part in figure 2.6 and 2.7 is to ensure that the individual components/assemblies/subsystems are in agreement with their requirements. Thus prior to being admitted to the next step in the integration process, verification (by means of testing, simulation, etc.) of the individual components/assemblies/subsystems has to be done (see §2.2.4). The outcome of the verification can be:
• success,
• fail,
• minor deviation that needs fixing, but is no barrier for further integration.

Whatever the outcome, it has to be documented carefully! Engineers tend to trust their work (of course) and thus expect success of the verification; they make their planning accordingly. However, in practice many issues occur during the verification and further integration process. Therefore, significant amount of time has to be reserved for integration and verification! According to [Muller 2011a], there are different types of problems that will be encountered in a typical order, when integrating a system or subsystem:

1. the (sub)system does not build/cannot be made/it does not fit;
2. the (sub)system does not function;
3. interface errors;
4. the (sub)system is too slow;
5. the (sub)system does not meet main performance parameter(s);
6. reliability is not met.

Clearly, issues 1-2 are on the lower level in figure 2.7 while the issues 4-6 are on the upper level in figure 2.7. The interface errors (issue 3) refers to both levels, even link them.

The way to organize the integration process is to start working with hard-/software set-ups as soon as possible. Even when the set-ups contain mock-ups and simulated components, the learning done is very important, and will result in saved time later in the process when the real material is there.

2.2.4 Validation

A very important aspect of the system engineering process is to verify a proposed solution before proceeding with further development. There are several verification methods applicable:

• reviews;
• experiments, for instance by creating functional models;
• comparison to existing products;
• analyses and/or simulations;

Where verification is a general term for checking performance against requirements using any of the methods listed above, the ultimate goal is to deliver a system to the customer, and have it accepted by him as the system that solves
his problem. In other words, the SUD has to create value for the customer. Therefore, we will use the term validation for a specific form of verification, namely verification against the customer requirements, wishes and needs, and investigating in what way the system can generate value. This, by the way is more than the system he may have requested (in the requirements). In systems engineering literature, and legal literature, the goal is to deliver the system that was requested and agreed upon. While that will always remain formally the maximum achievable, the goal of the system designer should be to deliver the system that solves the problem and generates value. Note that a possible outcome of any verification and validation step can be a new development cycle, as depicted in figures 2.3 and 2.7.

The accepted system is a system that conforms to a test\(^3\) that is agreed upon between the customer and system builder. There should be close correspondence between the system requirements and these test specifications. The Vee-model shows this for three levels of the system decomposition, see figure 2.8. This model that is often referred to in systems engineering ([Blanchard and Fabrycky, 2011]), shows that while the systems requirements are compiled, there has to be a testing procedure devised to show the specifications are met.

\[^3\text{customer acceptance test}\]
when the full system is built. Thus these testing and verification procedures have to be compiled in conjunction with defining the requirements, not when the development process is already well underway. Also, the requirements should be testable.

Also on the next level, the subsystem level, the subsystems have to be verified according to their respective subsystem requirements; and so further. Note therefore the corresponding colors between figures 2.3 and 2.8.

As for the definition of testing procedures: If a requirement cannot be verified in a testing, measurement, or other evaluation method its value as a requirement should be seriously questioned.

2.3 The Role of the Systems Engineer

The text above is as fact-based and neutral as possible. In addition, it is useful to look at the role of the systems engineer in this process and the required skills.

The blue parts in figures 2.4 and 2.6 show very visible work and output of the systems engineer. In the design phase, it is the systems engineer’s responsibility to develop the customer’s wish into the requirements, and further into a realizable system design as described in §2.2.1. In the integration and validation phase, the systems engineer is involved as knowledge holder: as the designer of the system, he is most aware of the system’s composition, and can thus deal with issues regarding interaction between components and assemblies. Also, when subsystems, or the system, is verified against the requirements, the systems engineer should be involved to make sure the verification is done so that meaningful results for the customer (and end-users) are gathered.

Does this mean that the systems engineer has not much to do in between these activities? Quite the contrary. In between, the systems engineer is busy safeguarding the concept, and monitoring the design process. This is not a very exact activity, but requires many skills. A cup of coffee and a good set of questions is a very good starting point for monitoring. In particular, the systems engineer should pay attention to:

- is the concept still maintained?
- will the requirements be met?
- are the interfaces not violated, or circumvented (that is a particularly nasty thing when it happens)?
identify and solve problems that occur, together with the designers involved,
- continuously monitor the balance in the system.

A systems engineer should be technically well educated in a broad set of disciplines. In general there is one home-field, with several adjacent fields where the systems engineer is comfortable. In addition to that he should communicate easily and frequently with engineers and non-engineers alike.

A final remark is that the systems engineer should learn to live with incomplete and uncertain information. Using intuition and experience, he has to make decisions in the absence of certainty.

### 2.4 A Short History of Systems Engineering

Effective systems engineering starts by mutual consultation and agreement between disciplines about the difference between defining what must be done and how well it must be done to determine what should be, what can be and what is. To come to this effective systems engineering approach two elements should be added to a project that are traditional not present [Arunski et al., 1999]:

- A disciplined focus on the
  - end product
  - its enabling products, and
  - its internal and external operational environment (i.e. a System View)
- A consistent vision of stakeholders' expectations independent of daily project demands (i.e., the System's Purpose)

The dilemma described above is from all decades. There is no particular date linked to the origin of Systems Engineering. From the past the heritage of systems engineering can be considered in complex projects and specific ways of working. Already, Noah's Ark was built to a system specification. Other examples from history are shown in table 2.1.

The main turning point in the maturity of Systems Engineering as a distinct engineering discipline arose from the increased and complex demands in the fields of military, missile and aerospace during and after the second World War. Solving complex problems in very short terms with high quality and low costs needed advanced technologies, new approaches in planning, technical collaboration and management processes. New revolutionary advances
Table 2.1: Major milestones in history seen from a systems engineers viewpoint.

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000BC</td>
<td>Water distribution systems in Mesopotamia</td>
</tr>
<tr>
<td>3300BC</td>
<td>Irrigation systems in Egypt</td>
</tr>
<tr>
<td>400BC</td>
<td>Development of urban systems like Athens, Greece</td>
</tr>
<tr>
<td>300BC</td>
<td>Construction of Roman highway systems</td>
</tr>
<tr>
<td>1808–1825</td>
<td>Design and build of water transportation systems like Erie canal</td>
</tr>
<tr>
<td>1877</td>
<td>Development of telephone systems</td>
</tr>
<tr>
<td>1880</td>
<td>Electrical power distribution systems</td>
</tr>
<tr>
<td>1937</td>
<td>Analyzing air defence system by British multi-disciplined team</td>
</tr>
<tr>
<td>1939–1945</td>
<td>Supporting Nike missile development by Bell labs</td>
</tr>
<tr>
<td>1948–1967</td>
<td>RAND corporation developed operational research and systems analysis approaches</td>
</tr>
<tr>
<td>1951–1980</td>
<td>SAGE air defence system defined and managed by MIT</td>
</tr>
<tr>
<td>1954–1964</td>
<td>ATLAS intercontinental ballistic missile program managed by Ramo-Woodridge corp</td>
</tr>
<tr>
<td>1964</td>
<td>Nine universities offered systems engineering programs</td>
</tr>
<tr>
<td>1966</td>
<td>USAF systems engineering handbook 375-5 appeared</td>
</tr>
<tr>
<td>1979</td>
<td>U.S. Army Field Manual 770-78, systems engineering appeared</td>
</tr>
<tr>
<td>1983, 1990</td>
<td>Defense systems management college and systems engineering management guides</td>
</tr>
<tr>
<td>1995</td>
<td>NASA systems engineering handbook appeared</td>
</tr>
<tr>
<td>1998</td>
<td>International Counsel On Systems Engineering (INCOSE) Systems Engineerin...</td>
</tr>
<tr>
<td>2001</td>
<td>Federal Highway Administration’s (FHWA) elaborated handbook about “System Engineering for Intelligent Transportation Systems”</td>
</tr>
</tbody>
</table>
about transportation and application of materials, information and energy were needed that could only be developed by thinking out of the box and over the boundaries of existing disciplines like electrical, mechanical, civil, chemical engineering and physics and mathematics [Kossiakoff and Sweet, 2003]. The cooperation between the systems engineering approach with traditional engineering approaches became more-and-more mature, strengthened by the military demand and space and aircraft engineering during the Cold War of the 1950s, 1960s and 1970s. A major driver in the development of systems engineering and other engineering approaches became the “information age”. Development of the digital computer in combination with advanced software technology gave a boost in supporting the problem solving process of engineering teams in simulating the performances of complex systems as a whole in relation to the necessary human interactions before the systems were build. New and innovative products, services and processes arose in a very short period and in this flow of technology and management modern systems engineering grew to higher capability maturity levels up to now.

From this flow the relations of systems engineering to its origins can be understood in terms of three basic factors (all cited from [Kossiakoff and Sweet, 2003]):

1. Advancing technology, which provides opportunities for increasing system capabilities, but introduces development risks that require systems engineering management;

2. Competition, whose various forms require seeking superior (and more advanced) system solutions through the use of system-level trade-offs among alternative approaches;

3. Specialization, which requires the partitioning of the system into building blocks corresponding to specific product types that can be designed and built by specialists, and strict management of their interfaces and interactions.
To achieve the optimal fit between the system under design (SUD), the identified issue, the context, stakeholders and the rest of the world, a systems designer has to think along several lines. In this chapter a number of these lines will be elaborated. This is based on, among others, Boardman et al. [2009], Martin [2004], Muller [2004a], Richmond [1993] and experience from the authors.

3.1 Introduction

This chapter shows another side of systems engineering. It connects more to the design part of the title of this book. As design is not finding the answer to a closed problem where there is only one solution, we have to look for inspiration on the one hand and constantly evaluate possible solutions on the other hand (see figure 2.3). In literature several approaches are presented. Some are more in the line of creativity [De Carvalho and Back, 2000; Eger et al., 2010; Gelb, 1998], others use lines of reasoning or systems thinking tracks [Muller, 2011b, 2004a; Richmond, 1993]. Finally, there are descriptions of how to look at systems [Boardman et al., 2009; Martin, 2004]. Combining these with experience from the authors, we show in this chapter a series of systems thinking tracks that can be used as:

- checklist;
- means to avoid mental inertia by looking at the problem from different viewpoints, and
- creativity starter.

It is important to realize that the procedure in chapter 2 is quite precise and results in a description of the system to be designed that will fit the identified issue, the context and the stakeholders. However, how it will fit the, –sometimes even largely– unknown rest of the world is not treated. The thinking tracks
Figure 3.1: The System Under Design (S.U.D.), the identified issue, the context, the stakeholders and the rest of the world and one of many thinking routes (in red) for the system designer.

given here will deliberately pull the thinking of the system designer out of the identified issue–SUD–context–stakeholders path, as shown by the more or less chaotic red path in figure 3.1.

There is a contrast with chapter 2 where a process was described that can be followed. Here, the softer side –as opposed to formal– of systems engineering is shown. This less formal side is in the opinion of the authors not less important. The use of examples and metaphors illustrates this.

An inherent problem in design is shown in figure 3.2. At the start of the design process, the outcome is unknown, while the freedom to make decisions is infinite. In that situation, the architecture of the system has to be defined (see §4.7), which shapes the overall system. This is a typical situation that describes the core of the system designer’s job: making important and far-reaching decisions, based on little and uncertain information. The information that is there, should be made as accessible as possible. The thinking tracks presented here, and most of the tools in chapter 4 are meant for that. Yet, one should be aware that to remove most of the uncertainty, the whole design process has to be completed. This, of course, is not feasible. Therefore, the system designer has to sample the whole design field, using the thinking

---

1Sections 3.2 to 3.7 inspired by and generalised from Richmond [1993]
3.2 Dynamic Thinking

The sort of systems designed by system designers and system engineers are not monolithic invariant systems. They are mostly systems that interact with the environment and are often highly dynamic\footnote{Definition of dynamic: marked by usually continuous and productive activity or change (Merriam-Webster online)}. Looking at the system from a dynamic perspective is thus essential:

- How does the system change over time?
- How does the environment change over time?
- When a change in input/output occurs, what are the effects?

Figure 3.2: In the early phase of development, many decisions have to be made, and technologies have to be committed to. Yet, a significant uncertainty (or: lack of knowledge) exists at that point of time. The catch is that also the ease of change decreases rapidly in the progressing of the project. (Based on, and adapted from, Blanchard and Fabrycky [2011].)
Simple tools are to plot the input and outputs over time. The use of different time-scales is essential, depending on the type of system from seconds to years.

As an example: the solar racer from the Twente Solar Team. Here interesting time scales are:

- seconds: how does the car react to vibrations due to small unbalances and road surface damages?
- minutes: how does the weather change, and what is the effect of wind gusts? How to react to a puncture?
- hours: driver behavior/feeding and short-term strategy;
- days: overall strategy and race planning, including foraging, camp planning vehicle Manning etc.
- weeks: project planning and manufacturing, communication strategy.
- months: financial planning, motivation, training and overall project planning.

Modeling, for instance the impact of a puncture on the total racing time can be done for velocity as function of time, see figure 3.3(a). From this, conclusions can be made about the time lost. However, what consequences this time lost has about position in the race is hard. Therefore an alternate model, or merely another representation of data already present in the model, may be needed, namely distance as function of time; figure 3.3(b). Now, the trajectory of a competing car that has no puncture can be easily incorporated, see the blue line. From these two diagrams design consequences can be derived: changing a tyre has to be done fast, but also accelerating after a puncture should be done fast. Maybe a device to store electricity close to the motor can help here? Another modeling switch that has to be made often in dynamic systems is that between time-domain and frequency-domain. The two models in figure 3.3 are both time-domain models. In particular when the time period of interest decreases it becomes more interesting to switch to frequency-domain models and diagrams like Bode-plots. Many systems react differently to slow disturbances than to fast disturbances.

Fortunately, modern modeling and simulation packages can easily output different diagrams like the two in figure 3.3; also they can output frequency responses and transfer functions; see §4.14.
3.3 Feedback Thinking

Closely related to the fact that systems have time-related behavior, is the question whether there is feedback. The principle of feedback is that of comparing the output of a system with the desired output, and adjusting the available controls so that the requested output is achieved. Figure 3.4 shows a basic feedback (control) system: something that has to be controlled, traditionally called the **plant**, creates an output. The output is measured by a measurement system. The result is compared to the desired output, which is given

![Diagram of feedback system](image)

**Figure 3.4: Basic Feedback (control) system. The “plant” is the thing to be controlled.**
as an input to the complete feedback control system. The error, $\varepsilon$ is used in a controller to create control signal inputs for the plant as to create the desired output.

There is a large body of knowledge on feedback control systems that will not be repeated here (see for instance [van Amerongen, 2010] for an introduction). When designing systems it is important, though, to look for feedback mechanisms. Are there ways to compare the output of the “plant” with the desired output? And is the desired output known? Are there ways to control the plant in case the output is not in line with the desired output? That means that the behavior of the plant has to be known to some extent.

Please note that for the controller several options are available. It can be very fine and gentle tuning, or it can be discrete (ON versus OFF) controlling, and everything in between.

In feedback thinking, the following questions need to be answered:

1. What is the process to be controlled (the plant in figure 3.4)?
2. What is the quantity (or are the quantities) to be monitored (the output in figure 3.4)?
3. What is the desired value for this quantity (these quantities)? E.g. what are the requirements?
4. How to measure this quantity (these quantities)? Is there a sufficiently accurate measurement system feasible? Is the response time of the measurement system sufficiently short?
5. Is the plant controllable? E.g. are there ways to control the plant, so that the output of the plant changes predictably?
6. Can a controller be devised? For this, study of control engineering may be necessary [van Amerongen, 2010; Bolton, 2003; Onwubolu, 2005].

Modeling and simulation is a useful –often even necessary– tool for these kind of problems.

An interesting example of a system without feedback was the stimulation of eco-friendly cars in the Netherlands. In 2007 (effective 2008) a new tax program was introduced, where both buying and leasing “green” cars resulted in tax benefits. It bases on the labels A to G for CO$_2$ exhaust.

The program has been such a huge success that the sales of eco-friendly cars has increased per year more than double than it did in the three years before the program started. However, there was no feedback in the sense that:

1. the absolute or relative number of eco-friendly cars were monitored,
2. or the amount of money invested this way,
3.4 Specific-Generic Thinking

3. or the CO₂-reduction.
4. or the relative performance of the most eco-friendly cars relative to the average car.

This has resulted in the mentioned large sales of eco-friendly cars on the one side. However, on the other side the investment of money (in fact reduction in tax-income) is much larger than originally budgetted. Therefore adjustments are needed. As of 2012 the norm for eco-friendliness will be gradually tightened. This is in fact introducing a soft feedback in the 4th sense.

Feedback thinking does not apply to the SUD alone, but also to the project undertaken to create the system. It is a valuable way of looking at the project organization. By giving frequent feedback about the current status of affairs within the planning (how much delay, money spent), the context (e.g. the company status in terms of financial status, people status, plans etc.), and the rest of the world (e.g. the market), people will understand their position in the big picture of the whole project, and that their contribution is essential. Tools like Lean Manufacturing, or Knowledge Based Production are very suitable for this on the project level [Mascitelli 2011]; they will not be treated in this book, though.

3.4 Specific-Generic Thinking

When discussing how to undertake a business trip to another part of the country many motorists use the argument that “public transport is always delayed, and crowded”. The crowded part may be true during rush hour, but the always delayed part is often based on one or two incidents from the past. If these specific situations are then used as generic facts for every transportation situation, wrong decisions are imminent.

Specific-Generic thinking is about the scale of the problem and solution. Is it worth devising an automated system to level a painting on a wall, when doing it by hand once a month will do? In politics (but not only in politics) this often occurs: the minister is called to answer questions about an incident, while 99.9% of all situations are handled very well. Then the parliament is so shocked that a new committee is installed to investigate and organize this exception, and all 99.9% normal cases. This results in increased numbers of

---

3 A recent study confirmed this overestimation of public transport traveling time by motorists, [van Exel 2011]

[van Exel 2011]
government employees and increased government spending. For which the minister is called to the parliament again.

There is a link with Scientific Thinking (§3.7) in the sense that one measurement cannot be used to proof something. Yet, there is an inherent problem here in design, as illustrated in figure 3.2. In the early phases of the design project, everything is unknown. Nearly all decisions have to be made based on models, estimations and even imagination. Measurements can only serve as a “reality check” for the order of magnitude as they are performed on systems that may behave very differently. As the project progresses, more knowledge becomes available, so less imagination is needed and the measurements that can be done are more relevant for the system under design.

In this track it is necessary to determine the scale of magnitude of the problem, and the estimated investment for the proposed solution. The moment numbers are introduced in the discussion –even if they are off by a factor of two– will already filter out many “shooting a mosquito with a canon” situations.

3.5 **Operational Thinking**

In operational thinking it is investigated how something is done in the real world. This is best explained using an example. Using our solar race example, as shown in table 1.1 a race day starts at 8 o’clock: from that moment it is allowed for the car to start driving. Operational thinking is then analyzing what is necessary to come to this point: what operations and procedures have to be undertaken to get the car (and all other vehicles) up and running safely?

Issues to consider for this specific case are (among others):

- waking up, making and eating breakfast;
- aligning the solar panel with the sun the moment the sun rises;
- starting up the solar car’s systems;
- technical check of the solar car;
- updating all model parameters (weather, competitors, etc.);
- sending press updates;
- packing the cars and setting up the convoy;
- taking down the tents and cleaning the area;
- health and safety checks;

A similar list can be made for all stops and for setting up the camp at the end of the day. For other systems, other lists have to be made. As this thinking
track relates to how the actual system works, it is not possible to make a
generic list that works for all systems. However, there are a few guidelines
that can be applied:

- Make a story/storyboard/scenario of the whole process.
- Include the startup and shutdown phases, or even make specific stories
  for these phases. In many cases these are very complicated, whereas the
  normal operation phases are relatively simple.
- Then, investigate non-normal behavior: emergency stops, failures etc.

Makes these stories in a form that they are presentable to experts, and with
people that have experience with operating comparable systems. Present to
and discuss with them, or even better: involve the operators in this process.
There is a wealth of practical information available with the operators. Engi-
neers sometimes have a fear going to the operators. Overcome that fear.

3.6 Scales Thinking

Engineers are educated to work with exact and verifiable data. As mentioned
before, in design, uncertainty plays a large role: the problem is not exactly
clear (although it may be clearly written down and reviewed, there may be a
mismatch with the actual problem, see figure 3.1), nor is the outcome known
at the beginning. In design, options are created that are compared and choices
are made based on arguments and data. However, arguments and data may
contradict. It is a rare situation that one option outperforms the others on
all aspects. Scales Thinking is about finding the nuances in arguments, and
about avoiding creating opposing camps.

Computer simulations are regularly based on linear or linearized equa-
tions. In reality, systems behave only linearly on a part of the spectrum.
Even more so, when a technology is stretched, often a step (a paradigm shift)
has to be made. One example is the telephone. Originally, the connections
were made by hand, by a telephone operator. With the increasing number
of connections this became impossible, so a paradigm shift was needed: au-
tomated telephone switches. At first these were electro-mechanical switches.
Nowadays, these are completely digital. The shifts from manual to electro-
mechanical, from electro-mechanical to fully electrical, and from electrical to

\footnote{If that is the case, one should severely question the seriousness of the other options and the
effort spent in creating these alternatives.}
digital each mean a change in relation between for instance cost and performance; or performance and energy consumption; or the space and number of lines they can serve. These shifts cannot be easily modeled and beyond linearized models. In this case one can see that a technology has limited scale of application. The same holds for measurement systems, actuators, and even managers!

For each shift, new scales of measures hold. Also, new limitations and opportunities exist. In the case of a new, unknown technology, these limitations and opportunities have to be scouted with care. When it involves treating humans, or when the food chain is affected, rigorous measures have to be taken. An example from history where this has not been done, is the (accidental) introduction of rabbits in Australia. Because of the lack of natural enemies, the rabbits could spread fast, and become a plague. Even to this day, controlling the rabbit population in Australia is an issue.

Another way of using scales thinking is the difference between yes/no or black/white and shades of gray. For instance, many people read the date on a carton of milk as the absolute date after which it cannot be consumed. However, it is a statistical prediction based on the whole process of producing and transporting the milk. This includes the average time the consumer leaves the milk outside the fridge after purchase. When someone only takes the carton out of the fridge to pour a glass of milk, and he has the fridge at 4°C, then the milk is well drinkable one or two days after the date on the carton. However, when someone left the milk outside the fridge on a table in the sun for two hours, the milk is sour maybe two days before that date. So, an initially thought clear division may be less clear when analyzed to more detail.

This way of thinking can be put to use for the system thinker in two ways. A stuck argument, that is in two (or three) camps, can be brought to life by analyzing the gray scales between the two camps. On the other hand, when there is too much nuance (or uncertainty), one can find the edge of pain by increasing the limits further in a thought experiment (SE: system engineer, E: expert):

SE: How much power will you need for this function?
E: I have no idea.
SE: Oh. Well, can you make an estimate?
E: No, really, I don’t know.
SE: Hmm, shall I put, 500W then?
E: No, that is way too much.
SE: Right, is it more like 50W?
E: Well, 50W may be a bit too little.
SE: What power do you think then. Or can you give a range?
E: It will be somewhere between 75 and 100W.

Of course, such an experiment does not give an exact number, but at least there is a clearer value than there was before. In subsequent improvements at later stages, the actual number can be found. Of course, additional measurements or other experiments can be useful too. This relates to the next thinking track: \textbf{Scientific Thinking}

3.7 \textit{Scientific Thinking}

Maybe the last part of the previous section gave another impression, but in systems engineering and systems thinking, it is important to use scientific principles:

1. formulate a theory;
2. formulate a hypothesis, based on the theory;
3. create a model;
4. verify the hypothesis using the model, by means of:
   - experiments;
   - simulations;
   - consulting literature;
   - consulting experts;
5. adjust the theory if necessary.

The verification part is \textit{essential} in this process. The model, can consist of, or be based on, many different things. Of course, a computer model can be used, but also a previous version of the system, or a comparable system can be taken as a model. One can build a \textit{functional model} that performs similarly, but which will not be used as a production type. One can use experts’ opinions as model (then use several experts), etc. Make sure the results are verifiable, and take into account that measurements have limited accuracy. Increasing the number of measurements improves the accuracy (with a factor of $1/\sqrt{n}$ for $n$ independent measurements). However, a systematic error cannot be reduced by repeated measurements. Also, consider that, as said in §3.6, measurement
systems also have scales of application. At the edges of their applicable range, deviations from the expected behavior may occur. Consulting the calibration data, or making calibration tables, can be useful.

As for more information on scientific research in the field of systems engineering, where case studies are often used, the reader is referred to [Martin and Davidz, 2007].

3.8 Decomposition–Composition Thinking

In many cases, systems engineering is presented as a way to determine how the system can be decomposed in subsystems, assemblies, parts, components etc. What often is left untreated in education is how can all these be integrated into a well-functioning system. And how can, in the whole process, the integrity be safeguarded? Decomposition–Composition Thinking may be the thinking track that has to be practiced most frequently by the system engineer to deal with this.

Top-level functions have to be allocated to subsystems, which often creates interfaces between these subsystems. In this process, one can step to other hierarchical levels to investigate the functions that are needed to accomplish the top-level function.

Decomposition–Composition Thinking is on the one hand a formal, logic, process, of determining what happens when a function is split: what interface(s) are created? This formal process and logic needs to be accompanied by good and thorough documentation, possibly supported by a computer tool that can check consistency and traceability. This divide and rule strategy is paramount in organizing the development process. On the other hand, considering the nature of most designers and the management culture in organizations, in this way the big picture gets easily lost. A less formal view by the systems engineer/system designer is then essential. He has to ask questions like:

- How do we put this together?
- How do we check whether it will fit, before shipping the parts, and start building on-site?

5The German language uses the word Schnittstelle for interface. Literally translated, it means: “cutting place”.

6This is where some people think “system engineering” only pays off. A tool that can be considered is IBM Rational DOORS.
• Is there an easy way to see whether a module is finished?
• Can each module be tested before integrating into the system? If not, what test setup is needed?

Managing and monitoring the interfaces between parts, and between the electronics, mechanics, optics, and software is crucial in this matter.

The system designer has to be aware of the fact that the total system is not on most designers’ mind. It should be worrying the system designer very much; in fact, it should be his main worry. The thinking tracks presented earlier help, however, making mental and concrete pictures of the total system is essential. The tools mentioned in the margin are of particular use, and most tools in chapter 4 may be helpful. Involvement of the designers in making these pictures/diagrams/models helps in avoiding problems. A simple way is to let the electronics designer draw the subsystem from his point of view, including the interfaces with mechanics and software. Then let the mechanical engineer do the same, and then the software engineer. In most cases there will be discrepancies. If identified early, they are easily solved. Important advise is to let them draw, not leave it to describing in words.

Questions to ask the designers:
• If you start detailing the design, what information do you need from software/electronics/mechanics/optics/…?
• …and in what format do you expect that information?
• …and what kind of connectors/fasteners do you use?
• If you have the first prototype, what hard- and software do you need to debug, test and validate it?

In particular, look for dead-lock situations where developers have to wait for each others’ information, or need each others’ deliverables.

3.9 Hierarchical Thinking

The systems engineering process treated in §2.2 shows a hierarchical organization7 from system to subsystem to assembly etcetera. This structure is also how many systems are organized. But this organization does not necessarily dictates the way the system’s control functions. In hierarchical thinking, the system designer has to reason about ranking authority, facilities and priorities of the system’s parts.

7hierarchy: a graded or ranked series (Merriam Webster online)
As an example, for a house-heating system, several options can be designed:

1. Every room has a temperature sensor (or even several) and a room unit where the desired temperature can be set. Both the set temperature $t_s$ and actual temperature $t_a$ are sent to a central control unit where all values from all rooms are gathered, and the heater is controlled. A set of valves is controlled so that each room gets the exact amount of heat that is needed.
   
   *central control, central actuation, central generation*.

2. Measuring the same as above. But now, there is a central heater that provides a constant heat flow. The room unit directly controls a valve in the room to control the heat flow into the room.
   
   *decentral control, decentral actuation, central generation*

3. Measuring as above. Now, there is a heater in the room, that is directly controlled by the room unit.
   
   *decentral control, decentral actuation, decentral generation*

And of course, more variants can be devised, for instance when the municipalities are generating the heat flow. As an exercise, add two or three variants, and list advantages and disadvantages of each of them, including the variants mentioned above.

As seen in the example, the hierarchical reasoning can be done for control, but also for generation and actuation. In a traditional car, there is only one engine. An electrical car can easily have an electrical motor in each wheel, thus providing four wheel drive. Central control reduces the number of signals, but increases the number of power lines, or power shafts. Localized control does the opposite. In computing there is the trend of cloud computing: storage of data and even programs in the cloud, this means on servers that can be accessed from anywhere via the Internet. The computer can be small and light (both in weight and processing power).

A final note on hierarchical thinking is that the system designer should be aware of hierarchies in organizations; both the company structure, and the project structure. There is a line of responsibility similar to the line of command in an army. Dutch society is very egalitarian and most decisions are made based on convincing and consensus. However, most cultures use a more formal way of decision making, with more use of hierarchy. In any case, a system designer often has no formal power. Yet, a good system designer will gain –a lot of– respect over the years. This will put him in a position of
great influence, also bringing great responsibility. Combined with the fact that a system designer often works with uncertain information can be a complex situation.

### 3.10 Project Thinking

This book does concentrate on the development of the System (see figure 2.1), but it should be noted that a system does not come to being without an organization that creates it: the project structure. Most of what is said about the system under design is valid for the project that creates the SUD, therefore most thinking tracks above can –and should– be applied to the project structure as well. There is an intricate relation between the project structure and the architecture of the system [Gulatti and Eppinger 1996]. This relation may originally (when the system was not too complex) be directed from the system architecture to the organization; once in place, the organization can influence or even dictate the system architecture.

Too much hierarchical layers in a project structure can hamper communication just as too many interface layers can hamper quick responses in a real-time control system. Depending on the size and required agility of the development, the organization structure of the project should be relatively flat. It can be argued that a moderate level of chaos can result in very agile development teams, although it requires a special kind of developers, and in particular system developers.

### 3.11 Life-cycle Thinking

Life-cycle thinking can be understood in three very distinctive, but equally important, ways:

1. the life-cycle from idea-design-production-deployment-use-retirement, and
2. the life-cycle as in resource use and environmental impact, and
3. the life-cycle of the project that is put in place to develop, build and sustain the product.

Each of these will be treated in separate subsections. The first, we will call the product life cycle; the second will be called the resource life cycle, the third is the project life cycle.
3. Systems Thinking Tracks

(a) The inside of the car, exposing the monocoque structure (detail of a photo by Gijs Versteeg)

(b) The mockup used to perform tests while the monocoque was being designed and manufactured

Figure 3.5: The 2011 solarteam Twente car "21 Connect" using a monocoque structure.

3.11.1 Product Life-cycle Thinking

Any product goes through different phases in its life. Although the exact names can differ, mostly the following are recognized:

- Need identification and/or idea generation,
- Design,
- Production,
- Distribution,
- Use,
- Retirement.

For mass-production goods, large emphasis has to be put on the production and distribution. For one-off products, like space systems, distribution is less critical, but then the way the system is put into use is critical. For civil engineering, the design and production phase are very critical; in particular the impact of the production phase has to be considered: creating an extra traffic lane for a highway may not impact normal traffic during rush hour.

Product-Life-cycle-Thinking can be characterized by evaluating the impact of a design decision on all phases in the product life-cycle, much like standing in a paternoster lift that moves along different floors of a building. The floors resemble the phases in the product life-cycle, the person in the paternoster resembles the system designer evaluating a design decision.

In the design of the 2011 Twente solar car, it was decided to use a monocoque construction, see figure 3.5(a) as this provides a light and stiff construction (advantages in the use phase). However, in the design and construction phases, consequences are that the monocoque frame cannot be designed be-
3.11 Life-cycle Thinking

Before the aerodynamic shell design is finished, the monocoque frame is the first part needed in the construction phase as all other parts are connected to it. A solution was found in making a second frame, a simple welded steel frame, that would act as a test setup for most other parts, see figure 3.5(b). Some additional work time was needed as most construction work had to be done twice. The increased freedom in the planning more than counter-weighted this small disadvantage, as also many things were learned during this first build, that were avoided in the final build. Referring to the TRIZ toolset (Appendix A), the principles 10, Prior action; 13, Do it in reverse; and 22, Convert harm into benefit have been applied.

Often the consideration of all product life-cycle phases results in trade-offs. For instance between making a product more durable (and thus more expensive), or implementing more frequent maintenance at a lower cost. This may result in negotiations with the customer.

3.11.2 Resource Life-cycle Thinking

Producing products uses resources. Resources in the form of human involvement, energy use, material use, etc. At present, there is an increasing awareness of the impact this has on the sustainability of the way of the western life [MacKay, 2008, also an excellent example of modeling]. Taking into account the resource use, and whether it has irreversible consequences becomes essential in the system design process. LifeCycle Analysis (LCA) is an often used tool. Cradle to Cradle is an approach to ensure optimal use of resources.

The systems engineer, though not a specialist in this area, should be aware of the basic principles of LCA, as for instance outlined in [Baumann and Tillman, 2004], and Cradle to Cradle [Braungart and McDonough, 2009], and think about the resource use. Again, trade-offs will occur between using more resources in the development and production phase versus using more resources in the actual use phase.

3.11.3 Project Life-cycle Thinking

The project also has a life-cycle just like the SUD. In order to create the SUD, a project has to be designed, created, used, maintained and retired. Typically, a design project starts small, with a team that investigates the need and creates a document stating the customer wish and the system specifications. This team is typically formed from the marketing and systems engineering staff,
if needed supported by others. The next phase in the project life cycle is, see figure 2.4 when the system design is created. This requires a larger team, and thus more facilities.

Even more organization is required in the next phase, when the split into subsystems is made. Then each subsystem gets its own staffing, and the number of people involved increases rapidly. The need for communication and documentation increases rapidly, and this needs to be anticipated by project management and the systems engineer(s) in the early phase.

After the SUD has been designed, it will be produced, distributed/put to use, it has to be maintained, and has to be retired one day. This part of the life cycle is sometimes called sustain. This has to be supported. Depending on the number of systems, and the context of use, the team is small or large. For railway systems the maintenance team is very large, and even the part of the total life cycle cost spent on maintenance is significant. For instance in railway material, maintenance cost over the life-span (30 years) is twice the purchase cost [van Dongen, 2011].

Also, the type of people involved changes. In the design phase, people with a creative mind are needed, while in the sustain part, more practical people are needed. Note that this causes a problem, as information about situations that occur during the use phase, that are dealt with by the sustain-people may never reach the designers. Thus these situations may show up in the next generation in the same manner. Regular contact between “sustain” and “design” is therefore recommended.

In most civil engineering projects (the Design-Bid-Build structure), the sustain part is often left to a totally different organization than the design and built phase(s). This complicates things, as design choices that ease building may have a very negative impact on maintenance. Current developments in the direction of Design and Construct or Design-Build-Finance and Maintenance (or even -and Operation) type of contracts are expected to alleviate this.

In any case, from a project life cycle perspective it is wise to mark the important mile-stones in the whole life cycle. Having a party at times can help to improve cohesion in the project team, increase flow of information between different parts of the organization and increase motivation.
3.12 Safety Thinking

Product safety is paramount. There are many regulations where products and systems have to adhere to before they can be declared as safe. Yet, during the development process, safety in all situations has to be ensured. With the constant pressure to save money, safety measures and procedures, or added components to ensure safety may not be wanted. However, the consequences of ignoring or neglecting safety can be enormous. One recent example is the BP Deepwater Horizon disaster in 2010 [Wikipedia].

The systems engineer has to reason about how the product can be used, and whether that is safe; in what ways the product can be used unsafely. Note that depending on the type of user (persons with specific training and responsibilities, or general public) the safety measures, or handling non-normal behavior, can take a large amount of the development effort.

It is often wise to build in points of failure, to avoid failure in awkward or dangerous parts of the system. Examples are circuit breakers in electrical systems. These are the –designed– weakest links in the circuit to prevent overheating or worse in other parts of the circuits. A mechanical example is a breaking pin (like a wooden pole) to connect the plough to the tractor. When the plough hits a rock, the pin breaks and the rest of the machinery remains in tact.

3.13 Risk Thinking

Designing innovative products and systems comes with risks. Avoiding risks will automatically lead to dull and non-discerning products; and thus to less profit or even loss. It is therefore a matter of managing the risks in the project such that their impact can be controlled and contained. Nevertheless, there is always a gambling component involved. Everyone involved in the development process must be aware of any risks that may develop, and the organization must have a mechanism in place to handle the risks identified.

Handling these risks can be either by:

avoiding finding an alternative solution where the risk does not occur.
containing analyzing the risk thoroughly, then going the way, but taking each step very carefully, and making sure that everything is done to contain the risk from increasing and causing a chain of negative effects.
taking taking the risk. This may be necessary when there is no alternative.
delegating the risk is taken, but the consequences are made someone else’s responsibility. This can be a good option when the development project is a large joint venture. Sometimes even the risk can be put to the end-user (read the end-user-license agreement that accompanies software).

In all cases time is an important factor. The earlier a risk is identified, the more time there is to identify options and alternatives. So, risk identification and analysis should be done as early as possible in the project.

### 3.14 Wrap up

The thinking tracks presented above try to summarize years of experience and results from theory and scientific literature in different engineering disciplines. Some advice may look obvious or too general. Nevertheless, during a design project wandering along these thinking tracks will result in new insights, and therefore increased knowledge about the SUD. This, in turn will yield a better end result and/or avoided mistakes and/or reduced risks.

In the next chapter more concrete tools will be presented that can help in wandering these tracks.
In this chapter a multitude of practical tools will be introduced that help in supporting the thinking tracks, and in implementing the systems engineering process.

4.1 Introduction

After the Systems Engineering Process in chapter 2 and the Thinking Tracks in chapter 3, it is now time to fill the Systems Engineers’ toolbox. In this chapter we will introduce several tools that can be used at various places in the SE process, or to support the thinking tracks. Some tools are suitable throughout the process, others are more to solve problems as they occur. In the previous chapters reference was made to tools to be used at specific points in the process or the thinking tracks. Here introduction to the tools can be found. As a general introduction, a few terms need to be introduced and/or defined.

**function** a specific or discrete action (or series of actions) that is necessary to achieve a given objective [Blanchard and Fabrycky [2011], p.100]. Functions and thinking in functions are important in systems engineering and systems design. The functionality of a system determines to a large extent the very reason for existence of the system –it is often the name of the thing. A coffee-maker makes coffee, a computer computes, a music-player plays music etc. This also illustrates the way to describe a function: as a verb and a substantive. The verb describe the action, the substantive describe the object where the action acts upon.

**stakeholder** anyone using the system, or involved in the development process.

**utility** or **infrastructure** functions that provide service to other functions in the system, without direct benefit to the customer.
4. System Design Tools

The 9-window tool, figure \ref{fig:9-window-diagram}, originates from the theory of inventive problem solving, known as TRIZ \cite{Altshuller1997, Salamatov1999} (see appendix \ref{appendix-a} for a short general introduction on TRIZ). The tool is simple to use and puts the system to be developed in its temporal and hierarchical context. It gives the opportunity to describe the previous situation, the present situation and the envisioned future situation on the level of the system that is developed. By also looking at the lower hierarchical level, the implications on that level are shown. Even more interesting is to look at the higher hierarchical level (the supersystem-level, or the system of systems level). The 9-window diagram forces one to think about it, and to discuss the consequences in the multidisciplinary team.

Another way of using the tool is in supporting scenario based design (see \S\ref{section:scenario-based-design}). Then several versions for the middle and right columns are devised. These can be discussed and evaluated based on criteria that resemble the customer wish. The best version is selected and elaborated into a system design.

\begin{figure}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
 & past & future \\
\hline
higher & & \\
\hline
SUD, Now! & & \\
\hline
lower & & \\
\hline
\end{tabular}
\caption{(a) The general 9-window diagram}
\end{figure}

\begin{figure}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
 & past & future \\
\hline
higher & & \\
\hline
student room & living room (small apartment) & large living room \\
\hline
transistor radio & HiFi system & high-end system \\
\hline
earphones & cd-player, amplifier, speakers, cables & cd-player, pre&power-amps, speakers, cables \\
\hline
\end{tabular}
\caption{(b) An example of using the 9-window diagram}
\end{figure}

\begin{figure}[h]
\centering
\begin{tabular}{|c|c|}
\hline
 & \\
\hline
\end{tabular}
\caption{Figure 4.1: The 9-window diagram. Horizontal is the temporal development, vertical is the hierarchical development. In the center window is the System Under Design as it is developed now.}
\end{figure}

\subsection{9-windows diagram}

The 9-window tool, figure \ref{fig:9-window-diagram}, originates from the theory of inventive problem solving, known as TRIZ \cite{Altshuller1997, Salamatov1999} (see appendix \ref{appendix-a} for a short general introduction on TRIZ). The tool is simple to use and puts the system to be developed in its temporal and hierarchical context. It gives the opportunity to describe the previous situation, the present situation and the envisioned future situation on the level of the system that is developed. By also looking at the lower hierarchical level, the implications on that level are shown. Even more interesting is to look at the higher hierarchical level (the supersystem-level, or the system of systems level). The 9-window diagram forces one to think about it, and to discuss the consequences in the multidisciplinary team.

Another way of using the tool is in supporting scenario based design (see \S\ref{section:scenario-based-design}). Then several versions for the middle and right columns are devised. These can be discussed and evaluated based on criteria that resemble the customer wish. The best version is selected and elaborated into a system design.
4.3 Context Diagram

The tool is particularly suited for discussions with very diverse stakeholders like people from marketing, senior managers, system architects and specialists. Also note that variations can be made where one of the two axes are changed. So instead of the combination time-hierarchy, one can make a 9-window diagram of time-customer, or time-market.

Applied to the case of developing the solar racer, one version of the 9-window diagram can look like figure 4.2. Here the 2007 and 2009 cases are used in the left and middle columns. The 2011 case is seen as a future case, as it was for the 2011 team when they started. Of particular interest is the fact that sponsor and supplier satisfaction is important in the supersystem row. Apparently it is in the next team’s interest to keep these stakeholders satisfied before, during and after the race, in order for the next team to have a good basis for a new development project.

4.3 Context Diagram

The context diagram shows the system in its context. In its simplest form it looks like figure 4.3 with the system under design in the center and the stakeholders around it. The stakeholders can then be named and detailed with who they are and what they require of the system.
4. System Design Tools

The context diagram can be extended by showing other systems around the SUD. When simple pictures are taken to represent these other systems, a rich and very communicative model of the context is created. An example, still at a very high level, is shown in figure 4.4, where the solar team Twente is taken as an example. It shows the team in its context.

From the context diagram one can determine the interest each of the stakeholders has. This is important when the requirements specification is not yet defined. It helps in balancing the customer wish with other stakeholders’ requirements (also see figure 2.1).

4.4 Scenario’s

In [Muller 2004a; Wojtkowski and Wojtkowski 2002] the usage of story telling in systems engineering is treated. In other places, it is called using scenario’s. This is a textual description of the use of the system to be designed. As an example the following story can be given on the use of a coffee cup:

On his busy day at work, Piedro needs his first cup of coffee. So he picks up his cup from the desk and heads towards the coffee machine. On his way he sees that the cup is not entirely clean. It has been a while since he took it home to give it a good wash and there is no way to properly wash up his cup at the office.

Figure 4.3: Simple context diagram showing the stakeholders around a system under design (SUD).
4.4 Scenario’s

Well, it is not that dirty that he cannot drink from the cup, so when he arrives at the coffee machine he places his cup in the appropriate opening, presses the series of buttons (there is no coin needed as the company provides for the coffee) and gets his extra strong coffee, with only little sugar, and no milk. Unfortunately the cup was not placed exactly right so some coffee is spilled. Making the handle dirty. Now he burns his fingers because he has to hold the cup itself, not the handle.

The coffee is too hot to drink right off the machine, so Piedro walks back to his office, puts the cup on the desk and resumes his work. When he thinks about his coffee again, it is a bit cold but Piedro drinks it anyway. He needs the caffeine!

On Friday when he has left the building, Piedro remembers that again, he forgot to take his cup home to give it a good wash. Then he smiles because he realizes that this way he cannot forget it to take it back to the office on Monday morning.

This story has been written only thinking about the use of the coffee cup and context knowledge about working at an office. The form of a story provides for a rich and easy to interpret description that makes discussions among colleagues easy. Before a story is analysed it should have been discussed among the designers, and whenever possible with the end-user and other stakeholders. Discussions with the latter two are possible because the use of jargon should be kept to a minimum.
When analysing the story many functions can be seen in the usage of the cup and its direct surroundings:

- signal the need for coffee/caffeine;
- move to and from the coffee machine;
- clean the cup;
- take the cup home to give it a wash;
- position the cup in the coffee machine;
- order the type of coffee;
- contain the coffee
- isolate the coffee from the hand/fingers;
- signal the temperature;
- drink the coffee;
- take the cup to work.

And based on this story, a state transition diagram, like the one in figure 4.7 can be derived.

Another story about the coffee cup describes its entire life-cycle, from its design to its removal. In that case the functions that relate to the production, distribution, use and removal can be found. For a coffee cup this will be relatively simple. For more complex systems like cars, aircraft or wafer steppers, this can be a complicated story. In particular the production will pose many problems that can be found early in the design process by creating and analysing these stories. Thus for each product several stories can and should be written.

From a story as given above, software engineering tends to derive use cases. These are short descriptions of the usage of the software system. In [Daniels and Bahill, 2004], the application of use cases as a complement to the “shall”-type of requirements common in systems engineering, is treated.

4.5 Functional Modeling and Analysis

As said, functions play an important role in systems design and engineering. Therefore modelling the functions from the earliest possible stage in the project is essential. Many different formalisms have been developed. In this book we will treat only a few of them that are used the most, or that are very useful in a multidisciplinary context:

- Function trees/mindmaps;
- Influence Diagrams;
4.5 Functional Modeling and Analysis

4.5.1 Function trees

When starting to design a new system or even when planning an upgrade to an existing system it is useful to create an inventory of what the system has to do. Starting from the highest level, one can make a tree of functions, like is shown in figure 4.5. Mindmapping software can be very useful for this. There are commercial programs, but the open source package freemind (http://freemind.sourceforge.net/) is good and can be used on several platforms.

By making a mindmap, one can subsequently create hierarchical levels that can later be used to define subsystems. Freemind can hide lower branches so that discussions can focus on part of the tree. Also, complete branches can be cut into a new file that can be used for further, lower level, elaboration. This way, hierarchical thinking is supported easily.

- Functional Block Diagrams (FBD);
- State-Transition Diagrams (STD);
4.5.2 Functional Block Diagrams

The functional block diagram (FBD) is possibly the most widely used model of the systems engineer after a requirement specification. It is a series of blocks that show the functions the system has to perform sequentially or in parallel. A FBD can be made on any hierarchical or life-cycle level, and even for an entire project. So there are FBD’s on life-cycle functions, showing a satellite’s life, from development, via launch, mission to re-entry and landing, and FBD’s that span only seconds or micro-seconds when they describe a series of events that form a computer interface or protocol.

An example of a simple FBD is shown in figure 4.6. This shows that functions can be sequential and parallel. But also utility functions can be modeled. They are not directly contributing to the output of the system. However, they are needed to ensure proper operation of the other functions. By the way, figure 3.4 is also a functional block diagram; one with feedback.

The FBD is most suited when there is one main track, and time is running from left to right. There are possibilities to show parallel functions (see figure 4.6), and even choices (by incorporating “OR”-items). Also, hierarchy
4.5 Functional Modeling and Analysis

Figure 4.7: A state-transition diagram. Here, the STD for a coffee-machine is shown. It can be seen that the machine detects when a cup is placed, and then asks for what kind of beverage is desired. So, the STD is a design tool.

can be shown, as it is possible to work out one or more functions in one or more separate diagrams. A FBD is limited, though, when it is unknown in what sequence the functions will be used, for instance when there is user interaction, or non-predictable interaction with other systems. In that case the state-transition diagram is better suited.

4.5.3 State-Transition Diagrams

The state-transition diagram (STD) is a tool to investigate and design behaviour of a system. For this, states and transitions are defined:
**state** a mode of operation or being.

**transition** passing from one state to another state.

It should be noted that in a state several —even many— functions can be performed in parallel or sequentially. Even more so, one state may be analyzed further and found to consist of several other states.

In a STD, the states are modeled as boxes, and the transitions are shown as arrows between the boxes. For every transition, the condition is given when it occurs. Figure 4.7 shows a STD for a coffee machine for offices. It is the outcome of discussions, as it is already decided that the machine detects a cup that is placed. A next step can be, for instance, to further elaborate on the “ask beverage” and “self check & heat water” states. Also, the error handling has to be worked out further.

Note that every state needs to have at least one transition coming in, and one going out (except for weapon systems, etc.). The STD can reveal possible dead-lock or one-way situations. It is easy to make a simulation based on such a STD, to check whether the behavior is logical and understandable for the target user.

### 4.5.4 Influence Diagrams

There are two type of influence diagrams: a pictorial investigation of influences and a more logical approach.

A pictorial influence diagram shown in Figure 4.8. Such a diagram consists of a pictorial representation of the system under consideration (like an iconic diagram, a sketch, block diagram) with all effects influencing the system indicated with text and/or symbols. The influences in Figure 4.8 are indicated with either ellipses or clouds. An ellipse is used when the place of influence can be clearly indicated, a cloud in other cases.

Figure 4.8 shows a wafer stepper or scanner. Here overlay is one of the key performance-indicators. It describes the accuracy between the lateral position of two layers of an integrated circuit. Accurate overlay can only be achieved by carefully selecting working principles, and checking every decision against its possible influence on overlay. Note that overlay does not map onto one or a few specific functions of the wafer stepper; it is a characteristic of the system as a whole. In figure 4.8 an inventory of the mechanisms that contribute to overlay are shown (also see Muller[2004b]). The diagram was made by the author in order to create an overlay-budget (§4.8) on the one hand, and to
Figure 4.8: Concept of a wafer scanner: items influencing overlay are shown as clouds and ellipses in a schematic representation of a wafer scanner.
raise awareness among the designers about the importance of designing with overlay in mind on the other hand.

The logical influence diagram models how variables and effects relate to each other. One can say that such a influence diagram is a visualisation of (a chain of) formulas. An example is shown in figure 4.9 where the effects that influence the cruise speed of a solar racer is investigated.

Such a diagram is created starting from the effect to be investigated. Then, the model is improved by subsequently asking the question: what variables and parameters do I need to calculate this effect?

4.6 $N^2$-diagram

The $N^2$ diagram is a useful tool in different stages of the architecting process with respect to interface management:

- to inventory all interfaces between functions,
- to compare architectures regarding the number and type of interfaces,
- when the system is developed further to monitor the interfaces.

By its nature the $N^2$ diagram aids in working through all possible interfaces, minimizing the chance of omitting one.
Table 4.1: An $N^2$ diagram. F1 to F6 represent functions 1 to 6. a is an output of F2 and input to F5; b is output from F6, and input to F4.

\[
\begin{array}{cccc}
F1 & F2 & a & \\
 & F3 & & \\
 & F4 & & \\
 & & F5 & \\
 & & b & F6
\end{array}
\]

The $N^2$ diagram consists of:
- a square matrix (thus its name) with the functions on the diagonal.
- the outputs of a function are listed on the row of that function.
- the inputs to a function are listed in the column of that function.

This means that if one starts reading from a diagonal cell clockwise, one reads the function, its output and the destination of that signal. It is often useful to expand the matrix with a row/column for the user and a row/column for the environment.

This functional $N^2$ diagram is useful for analysing the required interfaces. Once this is done, the functions will be allocated to subsystems. Then another $N^2$ diagram is made: the modular $N^2$ diagram. Here, the subsystems are put on the diagonal, and the interfaces between the subsystems are represented, see table 4.2. According to an heuristic from [Maier and Rechtin, 2000], a sensible architecture has low complexity between the subsystems, and high complexity inside the subsystems. This is seen in the modular $N^2$ diagram as only a few filled off-diagonal cells.

Note that $N^2$ diagrams can be made hierarchically. So, one top-level diagram that represents the system interfaces. Then, for each subsystem the interfaces inside this subsystem is worked out in a separate $N^2$ diagram etc.

4.7 Architectures

In creating large and/or complex systems the term architecture is often used. This term that originates from building, can have many meanings, and defining it is not an easy task, as is described in [Maier and Rechtin, 2000]. For this text, we will use the following definition from [Bonnema, 2008], which is
Table 4.2: N² diagram for a teletext to xml converter that is used to fill the electronic program guide (EPG) of a personal video recorder. The cells on the diagonal show the main components. Note that there are no interfaces below the diagonal.

<table>
<thead>
<tr>
<th>Environment</th>
<th>channelIDs.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receive teletext and filter unwanted info</td>
<td>EPG_data.txt</td>
</tr>
<tr>
<td>Convert time, program name and add station info</td>
<td>EPG.xml</td>
</tr>
</tbody>
</table>

PVR app

based upon \cite{Gulatti and Eppinger 1996}:

**System architecture** defines the parts constituting a system and allocates the system’s functions and performance over its parts, its user, its super system and the environment in order to meet system requirements.

In other words, the architecture defines how the SUD is split into smaller pieces to ease the development, but at the same time, it also defines how the SUD is positioned in the environment, and how it interacts with the user(s), as already shown in figure 2.1.

Note that any system of some complexity has an architecture, but it may not be created consciously, or documented explicitly. It may, for instance, be in the head of only one person. It is in general needed to document the architecture. This requires several representations in different formats. Nearly all of the tools in this chapter are useful as a representation for one or more aspects of the architecture. The combination of all different representations form the architecture description. (Note, though, that it is not always required to use all representations; a wise selection has to be made.)

The architecture has to address the following issues:

- how is the SUD divided into smaller, more manageable parts: the sub-systems?
- what are the interfaces between the subsystems?
- how are the functions allocated over the (sub)systems?
• how is the SUD positioned in the environment and the rest of the world?
• what are the interfaces with the environment?
• how are the development paths between the subsystems (de-)coupled?
• can parts be reused (for instance from previously developed systems, or from systems developed by others)?
• what is done in hardware or software?
• what are the long-lead items (see page 91)?

With careful architecting, for instance as described in §4.9, technical and business opportunities can be identified. For a given function, one can ask whether that should be done by the system, the user, the environment, or a service provider. And then, whether profit can be generated from that. As an example, selling roasted coffee beans will result in less margin than selling portions of sealed, ground espresso coffee. This is also the razor-blade marketing philosophy. Here the razor-holders are cheap or even free. Profit is made on the blades. Although they are not very expensive, they are sold in large quantities and thus a modest margin a piece adds up to large sums for the company selling them.

4.8 System budgets

The word budget makes most people think of money. A budget is “a quantity (as of energy or water) involved in, available for, or assignable to a particular situation”\(^1\). In finance, a budget also shows the breakdown of the budget over the diverse expenditures. The same holds for a system budget: it states the available quantity, and the way it is divided over the expenditures. In systems design, it can be used for many more than dollars or euro’s. System budgets are used to allocate parts of the system requirements to subsystems, assemblies, components and parts. Requirements that can be divided using budgets are:

- mass;
- power consumption;
- (floor) space;
- waste production;
- production time;
- service time;
- (position) accuracy;
- productivity;
- etc.

\(^1\)2011 Merriam-Webster online
Table 4.3: A mass budget for a solar racer. Note that the structure is real, but the numbers are modified somewhat for confidential reasons.

<table>
<thead>
<tr>
<th>Item</th>
<th>mass [kg]</th>
<th>remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL powertrain</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>EL telemetrie</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>EL solarpanel</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>monocoque</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>suspension</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>wheels</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>steering</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>braking</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>detailing</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>driver</td>
<td>82</td>
<td>80kg by regulations, plus harness and cushion</td>
</tr>
<tr>
<td>margin</td>
<td>5</td>
<td>managed by SE</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>260</strong></td>
<td>including driver</td>
</tr>
<tr>
<td><strong>Excluding</strong></td>
<td><strong>180</strong></td>
<td>excluding driver</td>
</tr>
</tbody>
</table>

In fact, the largest part of the system requirements can be—and should be—divided over the subsystems using budgets. Some requirements like conformity to standards and safety regulations cannot.

The budget should be made early in the development process, and should be the development goal for each subsystem. The basis for the budget is a combination of:

- modeling;
- experience with previous systems;
- experiments;
- extrapolation of past results;
- *guesstimation*\(^2\).

Based on these, a first version that may only be one or two levels deep, is made by the systems engineer. Through interviews with experts and stake-

\(^2\)A guesstimate may be a first rough approximation pending a more accurate estimate, or it may be an educated guess at something for which no better information will become available. Source: [http://en.wikipedia.org/wiki/Guesstimate](http://en.wikipedia.org/wiki/Guesstimate)
holders, refinements are made. Possible conflicts can be identified and should be solved through further modeling and analysis and/or discussions with the corresponding stakeholders. The result is a budget that is accepted by all stakeholders. It can be useful to represent it in a tree-form that corresponds with the system breakdown in subsystems.

As shown in figure 3.2, in the early phase when the first budgets are made, there is quite some uncertainty. When more information becomes available during the development process adjustments to the budget may be necessary. There can be transfers from one item to another, or adjustments to the total budget. Therefore a margin should be included in the budget. This can be done implicitly or explicitly:

**explicit margin** an explicit margin is like is shown in table 4.3 where there is a separate budget item “margin”. Every subsystem has to stick to his own budget, but in case there is an overrun, this margin can be applied for. The systems engineer manages the margin.

**implicit margin** in case of an implicit margin, there is no margin in the budget accounted for. However, the systems engineer knows about some extra available budget. In case of a subsystem in trouble, there will be quite some fuzz, but then the total budget will be increased.

Both solutions have advantages and disadvantages that have mostly to do with the culture in the organization. An implicit margin appears to be more strict. However, once it is known that there is some room, the assumed strictness is gone.

Managing the budget during the design process can be done in a graphical manner as shown in figure 4.10: plotting the value based on the current design knowledge against time together with the design target in graph.

### 4.9 FunKey Architecting

In FunKey Architecting [Bonnema 2008, 2011], a combination is made between the functions a system has to perform (§4.5) and the performance that has to be achieved (§4.8) in order to balance the budgets. At the same time by making explicit links between the functions and the stakeholder’s benefit through the use of key drivers, the contribution of every engineer is made explicit.

In addition to the earlier defined functions and budgets, we thus need to define the term “key driver”: 
Figure 4.10: Tracking a mass budget. The black line shows the target value. The red line shows the present value at different moments in time, based on the design at these moments. At the beginning parts are missing leading to a very low mass. Later the target is exceeded.

**key driver** a generalized requirement representing the main interest of a stakeholder.

Examples of key drivers are *total race time* and *media exposure* for a solar racer, *cost per passenger per mile* and *turn around time* for a passenger aircraft. For a system, the goal should be to define five to ten key drivers that each should be preferably quantifiable. The set of key drivers should span the set of stakeholders. On the one hand the key drivers show the desired (required) performance. On the other hand they express the engineering achievements. In that way, the key drivers act as interface between the customer and builder of the SUD.

Next step then is to determine which system function contributes to which key driver. For this, we propose to use a *coupling matrix C* shown in table 4.4. Here, a cross (×) indicates the function contributes—either positively or negatively—to the key driver. At first, this is a mere qualitative inventory. However, in a later stage, a quantitative analysis can follow. But before that, the qualitative analysis can already reveal a lot with a little more effort, and even provide clues for system improvement using TRIZ. One of the tools in TRIZ is the contradiction matrix in combination with the 40 principles, as
Table 4.4: The FunKey coupling matrix $C$ used to investigate contributions of functions to key drivers ($kd_i$).

<table>
<thead>
<tr>
<th></th>
<th>$kd_1$</th>
<th>$kd_2$</th>
<th>...</th>
<th>$kd_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>function $f_1$</td>
<td></td>
<td>×</td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>function $f_2$</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>function $f_n$</td>
<td>×</td>
<td>×</td>
<td></td>
<td>×</td>
</tr>
</tbody>
</table>

outlined in appendix A.1. In [Bonnema 2008; Ivashkov and Souchkov 2004] it is investigated which principles are most appropriate in improving a certain technical parameter (see §A.1.1). This can be directly used once the coupling matrix is populated.

If, for example, $kd_2$ is throughput, and function $f_2$ is measuring, we have to relate throughput to one of the technical parameters as defined in TRIZ, see the first columns in table A.3. This is not a very formal or strict mapping. The numbers given on the row of that parameter present the innovative principles that are most likely to be successful. In this case throughput relates to productivity: technical parameter #39. The innovative principles are #10: Prior Action and #35: Transformation of Properties, as given in table A.2. Prior Action can be implemented as measuring before the actual process takes place, instead of during production. Although not feasible in all cases, it is a valid way of improving throughput. Transformation of Properties may work in some cases, but is not seen feasible in this general example.

4.9.1 Coupling matrix to budgets

The coupling matrix $C$ is a good basis for further detailing into budgets. The crosses in table 4.4 can be further analyzed and replaced by numbers. These numbers can be percentages of the total available mass/time/money/memory etc. in order to balance the total design. Another approach is to fill in known values from existing systems or the present state of the technology and compare the total with the performance to be achieved. The tension is then made visible. Also, the playing field of where the improvement should be found is clearly shown.
4.9.2 FunKey as tracking tool

Similarly to the use of budgets shown in figure 4.10, the coupling matrix $C$, once filled with numbers, can be used for tracking the design process. By doubling each key driver column into an aim and an actual column, the development is tracked based on numbers. In [Bonnema, 2008] the use of Symmetrical Triangular Fuzzy Numbers (STFNs) is proposed to make uncertainty visible.

For more in depth information on FunKey architecting, the reader is referred to [Bonnema, 2008] and [Bonnema, 2011].

4.10 A3 Architecture Overviews

Systems engineers produce architectures. These are mostly represented in diagrams and documents. Although standards for architectures are available, the most used tools by architects are word processors, spreadsheet programs and presentation programs. The diagrams and tables are put together into an architecture document with –mostly– a lot of text. These documents are not often consulted after they are produced, reviewed and accepted as asking the architect for the required information is faster than looking up the information.

A recent development [Borches and Bonnema, 2010; Borches Juzgado, 2010] is to present architectures in a compact and visual format so that the architecture information is readily available. The philosophy is to make the representation as simple as possible, but not simpler. This is achieved by strictly limiting the available space to both sides of an A3 sheet of paper. One side shows models (figure 4.11(b)), the other shows a textual summary and explanation (figure 4.11(a)).

The title of the A3AO shown on both sides is very important as it describes, defines and limits the scope of the document. It should contain the SUD and the specific aspect that is modelled and described. If this is not done, it will become impossible to stay within the bounds of one A3-sheet.

---

3 paraphrasing Einstein
Figure 4.11: A3 Architecture overview of a litter collecting robot.
The model side of the A3 architecture overview (A3AO for short) comprises three main views:

- a functional view, showing the functionality of the system of interest, (shown at the left in figure 4.11(b))
- a physical view, showing how the functionality is divided over the system’s components, and showing the physical components (shown at the bottom right in figure 4.11(b)), and
- a quantification view that shows the main performance numbers, much like is done in the system budgets (§4.8) and FunKey (§4.9) (shown at the top right in figure 4.11(b)).

Additional models can be used to support these views. These are shown in the middle section of figure 4.11(b).

The textual side (figure 4.11(a)) shows, each in its own box (from top to bottom, then left to right):

- definitions and abbreviations
- an introduction: what is this A3AO about,
- the related system concerns: what should be taken into account when creating the architecture. These follow for example from several of the system thinking tracks presented in chapter 3. The system concerns generally are repeated over several A3AO’s.
- the top level view showing the main partitioning. In this case how is the functionality of cleaning litter divided over the robot, the operator and the remote controller.
- additional information for the functional view on the model side.
- additional information for the physical view
- information for parameters and requirements, this relates to the quantification view.
- logistics information about document owner, creation date, contributors etc.
- design strategies, assumptions and known issues.
- roadmap for further development.
- references to more information and experts.

\footnote{Note that the physical view for a software system shows the way how modules and classes are defined}
4.11 Failure Mode and Effect Analysis

4.11.1 Introduction

A Failure Mode and Effect Analysis (FMEA) study identifies and prevents system, design and process failures in a systematic way before they occur. To do so FMEA emphasizes on failure prevention, enhancing safety and increasing customer satisfaction.

“Failure modes” means to understand the ways, or modes, in which something might fail in fulfilling functions and in particular the failures that affect the customer satisfaction and company image. “Effects analysis” means studying the consequences of those failures, like severity, cause mechanisms,
occurrences. So it is important to first prioritize according to how serious the consequences are, how frequently they occur and how easily they can be detected. If the failure modes are determined and accompanying effects are analyzed, actions can be taken to eliminate or reduce failure modes, starting with the highest-priority ones.

Because of the systematic way of working, the FMEA-study brings in current knowledge and actions about the risks of failures in products and processes captured as transparent and traceable documentation. Many companies apply this information for continuous improvement of products and processes.

FMEA can be used during design as well as later for control, before and during ongoing operation of the process. The most ideal situation is to apply an FMEA-study at the earliest stage of a project and continuously throughout the life-cycle. The sooner failures are detected the less expensive to address them and the less costly to eliminate or reduce them (see figure 3.2). Nowadays there are several types of FMEAs, as outlined in Appendix B.

4.11.2 FMEA-team

To elaborate a FMEA-study a team approach is necessary due to the necessity of an integral assessment of failures. The team should be led by a responsible design or manufacturing engineer familiar with FMEA. Team members can be design engineers, process engineers, manufacturing engineers, suppliers, logistic engineers, customers, operators, persons responsible for maintenance, etc. Prior to assembling the team the responsible FMEA-engineer can arrange a meeting with two or three key engineers, responsible for design, quality and testing. In advance they can determine the scope of the FMEA-study, elicit background reference material from previous FMEA-studies and the context of the project, creating and updating function block diagrams of the system, product and/or process. Further on they identify the team members, prepare the agenda, schedule, milestones and at the end a first overview of main item functions, failure modes and their effects is made.

4.11.3 FMEA Form

As an aid for performing an FMEA, a form is used, as shown in table 4.5. It consists of four parts each with several columns. The first part identifies the failure modes and their effects (columns 1–4); the second part the causes of
### Table 4.5: The Failure Mode and Effect Analysis form.

<table>
<thead>
<tr>
<th>Process or Product:</th>
<th>Prepared by:</th>
<th>Page:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process owner:</td>
<td>FMEA date:</td>
<td>Rev:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **What is the process step or input?**
- **In what ways can the process step or input fail?**
- **What is the impact on the output variables once it fails?**
- **How severe is the effect to the customer?**
- **What causes the key input to go wrong?**
- **How often does cause or FM occur?**
- **What are existing controls and procedures that prevent the cause or FM?**
- **How well can you detect the cause or FM?**
- **What are the actions for reducing occurrence or improving detection?**
- **Who is responsible for the action?**
- **Note the actions taken. Include dates of completion.**

**Identify failure modes and effects**
*Identify causes and controls*  
**Prio**

**Determine and assess actions**
Table 4.6: Severity ratings for an FMEA. These can be used as a general guideline. Precise agreement has to be created for a specific project.

<table>
<thead>
<tr>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>safety issue or regulatory violation without warning</td>
</tr>
<tr>
<td>9</td>
<td>safety issue or regulatory violation with warning</td>
</tr>
<tr>
<td>8</td>
<td>main function is lost or seriously degraded</td>
</tr>
<tr>
<td>7</td>
<td>subfunction is reduced and customer is impacted</td>
</tr>
<tr>
<td>6</td>
<td>subfunction is lost or seriously degraded</td>
</tr>
<tr>
<td>5</td>
<td>subfunction is reduced and customer is impacted</td>
</tr>
<tr>
<td>4</td>
<td>loss of function or appearance such that most customers would return product or stop using service</td>
</tr>
<tr>
<td>3</td>
<td>loss of function or appearance that is noticed by customers but would not result in return or loss of service</td>
</tr>
<tr>
<td>2</td>
<td>loss of function or appearance that is unlikely to be noticed by customers and would not result in return or loss of service</td>
</tr>
<tr>
<td>1</td>
<td>little or no impact</td>
</tr>
</tbody>
</table>

the failure modes and controls (columns 5–8); the third part the priorization (column 9) and the last part (columns 10–16) determines and assesses the actions. Before the columns can be filled for each issue in the procedure outlined below, the header should be filled: All team members and the revision date must be listed (see §4.13.1). Product names and numbers (ID's) must also be detailed in the FMEA header.

4.11.4 FMEA procedure

The FMEA procedure consists of 10 steps. The numbers in brackets [] indicate the column(s) of the FMEA form.

1. List the key process steps [1]. Every FMEA should detail the assumptions and ratings used. This information may come from the highest ranked items of the previously developed functional block diagram. It describes what the system or component is designed to do and the environment in which the system should operate. For example: The storm-surge barrier must close the tide river within 4 hours after measurement
of the critical height of an increasing sea-level and must open within 3
hours after measurement of the critical height of an descending sea-level
(as specified in functional performance specification –include reference
to source document–).

2. List the potential failure mode for each process step [2]. Potential failure
is the way in which a system, product or process could potentially fail to
meet the specified objectives. The main question should be: “How could
this system, product or process fail to meet each customer requirement
or company image?” Consider potential failure modes under conditions like: operating (dusty and dirty or wet and dry), usage (above
or below life-cycle, severe circumstances), incorrect service (wrong part,
backwards, omitted, hard to assemble). For the storm surge barrier, one
can think of that it does not move after measurement signal, or that
measurement of the water level is higher than the actual height.

3. List the effects of this failure mode [3]. If the failure mode occurs what
does this mean to the image of the company and the satisfaction of the
customer. The main question is: “If the failure occurs then what are the
consequences to whom?” So describe the effects in terms of the exposure
to whom is concerns. The most important notifications are the impact on
safety or noncompliance to law and regulations. Be aware that there is
no single customer but several stakeholders who have their own specific
requirements. For the barrier, consequences of a barrier that does not
close are that water flows over the dykes into the harbor facilities.

4. List the severity [4]. Rate how severe this effect is with 1 being not severe
at all and 10 being extremely severe, also see table 4.6. Ensure the team
understands and agrees to the scale before you start. Severity means
the assessment of the seriousness of the effect(s) of the potential failure
mode on the next component, subsystem, or customer if it occurs and
from the perspective of the team members. Severity applies to effects (for
failure modes with multiple effects the highest rating should be selected
as severity for the failure mode). Water flowing over the dykes - severity
10, Ships that cannot enter the harbour - severity 5

5. List causes of failure [5] and occurrence [6]. Identify the causes of the
failure mode/effect and rank them in the occurrence column, similar to
the way was done for the effects. This time, as the name implies, we are
scoring how likely this cause will occur. So, 1 means it is highly unlikely
to ever occur and 10 means we expect it to happen all the time, see table
4. System Design Tools

4.7(a) Every imaginable failure cause or mechanism should be listed as concisely and completely as possible just to take the most effective action. In the storm surge case, incorrect position of level indicators gets occurrence value 5.

6. List current system, product or process control [7] and detection code [8]. Identify the control procedure or mechanism in place to detect the issue. This involves the activities that will assure the design adequacy (see 2.2.4) and measurement or detection systems for the failure cause/mechanism under consideration. These controls will detect cause and subsequent failure mode prior to production and operation, and or will prevent the cause from occurring. Three types of control can be distinguished:
   (a) Prevention from occurring
   (b) Detect cause mechanism
   (c) Detect the failure mode

One can for instance review the specifications thoroughly, or measure the condition of the surge barriers bearings on regular interfaces. The detection scale is larger for items that are unlikely to be detected with 10 being the least likely, see table 4.7(b). The greater the probability of detection the lower the rating. The water level sensor of the barrier is checked daily and so the detection rating for “Water level” is 1.

7. List the Risk Priority Number (RPN) [9]. The RPN is a multiplication of severity, occurrence, and detection ratings. This is the key number that will be used to identify where the team should focus first. RPN ranges from 1 -1000. The team must make efforts to reduce higher RPN’s through corrective action. General guideline is that when the RPN is over 100 action is required.

8. List recommended actions [10]. Sort by descending RPN numbers and identify the most critical issues. The team must decide where to focus first. Only design revisions can bring about reduction in the severity ranking. Examples of recommended actions are: reliability testing, revise test plan, revise design.


10. List the Action results [12-16]. Once actions have been completed, re-score the occurrence and detection. In most cases we will not change the severity score unless the customer decides this is not an important...
4.12 Risk Management

Risk is a very generally used term, however its meaning is not often clearly defined. For development projects, it is necessary to consider two aspects of risks $R$, namely the chance of its occurrence $P_r$ and the severity of the

---

Table 4.7: Example of occurrence and detection definitions for an FMEA.

<table>
<thead>
<tr>
<th>value</th>
<th>description</th>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1 in 2</td>
<td>10</td>
<td>uncertain</td>
</tr>
<tr>
<td>9</td>
<td>1 in 10</td>
<td>9</td>
<td>very remote chance</td>
</tr>
<tr>
<td>8</td>
<td>1 in 50</td>
<td>8</td>
<td>remote chance</td>
</tr>
<tr>
<td>7</td>
<td>1 in 250</td>
<td>7</td>
<td>very low chance</td>
</tr>
<tr>
<td>6</td>
<td>1 in 1 000</td>
<td>6</td>
<td>low chance</td>
</tr>
<tr>
<td>5</td>
<td>1 in 5 000</td>
<td>5</td>
<td>moderate chance</td>
</tr>
<tr>
<td>4</td>
<td>1 in 10 000</td>
<td>4</td>
<td>moderately high chance</td>
</tr>
<tr>
<td>3</td>
<td>1 in 50 000</td>
<td>3</td>
<td>high chance</td>
</tr>
<tr>
<td>2</td>
<td>1 in 250 000</td>
<td>2</td>
<td>very high chance</td>
</tr>
<tr>
<td>1</td>
<td>1 in 1 Million</td>
<td>1</td>
<td>almost certainty</td>
</tr>
</tbody>
</table>
consequences $C$. More formally defined:

$$R = Pr \times C$$

Where $R, Pr, C$ are all on a relative $0 - 1$ scale. For chance, such a relative scale is easily grasped, for consequences, it is more difficult. One can for instance define 1 as complete loss of the total project budget. Essential is to make a clear definition and communicate it across the project.

This does mean that an event with only a small chance of occurring, but catastrophic consequences has a high risk, just like an event with high chance of occurring with moderate consequences, see figure 4.13. If in a project multiple risks will occur, then the overall project risk can be calculated by focusing on the chance on success, instead of on the chance of the chance of the failure:

$$P_{success} = (1 - Pr_1) \cdot (1 - Pr_2) \ldots (1 - Pr_n)$$

And thus:

$$\Rightarrow Pr_{total} = 1 - P_{success}$$

Of course, although this looks like exact analyses, the numbers representing the risks are based on estimates. Therefore, risk analysis should be mostly considered a qualitative affair.
For risk management – this is keeping the risks in the project within acceptable limits – we discuss the following tools in separate subsections:

- decision tree, to quantify risks when making a decision;
- risk register, to track the number and magnitude of risks in a project.

### 4.12.1 Decision tree

In its simplest form (figure 4.14), a decision tree maps out the expected decisions to be made, with the alternatives as branches. As example, figure 4.14 shows three alternatives that can be taken: upgrading an existing system; creating an entirely new design, or buy a system in an alliance with another organization. For each of the alternatives, there are three possible outcomes, once a prototype is available:

1. the performance is according to the requirements;
2. the performance is not ok, but a minor redesign will suffice;
3. the performance is not ok and a major redesign is required.

Of course, for other projects other courses may exist.

A second use of the decision tree is to quantify each of the courses; that is, to analyse the probability of each of the alternatives to occur, and to quantify the time and money involved in each of the tracks. This is shown in figure 4.15. Probabilities branching from one point have to add to unity. For the first branch the meaning of the probabilities can be read as how large is the track expected to be at all feasible. This can be considering for instance suitable partners for an alliance (third alternative), experience with the present system A (first alternative), etc. From these estimates, based on past experience, interviews, preliminary modeling, the time and cost of each of the branches can be calculated as shown in figure 4.16.
Figure 4.16: Each of the alternatives can now be assigned a total running time and cost by summing the time and cost along the track of the alternative.
Table 4.8: Evaluating the decision tree quantitatively. The columns Prob, Time and Cost are derived directly from the decision tree in figure 4.16. Prob (probability by multiplying the probabilities along a route for a variant, Time and Cost for adding the time and cost along a route, respectively. The columns $Time_w$ and $Cost_w$ are the weighted time and cost, respectively, calculated by multiplying the time and cost by the probability. The expected values are the summed time and cost over all variants.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Upgrade A</td>
<td>none</td>
<td>0.15</td>
<td>14</td>
<td>1050</td>
<td>2.1</td>
<td>157.5</td>
</tr>
<tr>
<td></td>
<td>minor</td>
<td>0.07</td>
<td>17</td>
<td>1170</td>
<td>1.19</td>
<td>81.9</td>
</tr>
<tr>
<td></td>
<td>major</td>
<td>0.28</td>
<td>24</td>
<td>1850</td>
<td>6.72</td>
<td>518.</td>
</tr>
<tr>
<td>New design</td>
<td>none</td>
<td>0.24</td>
<td>22</td>
<td>1550</td>
<td>5.28</td>
<td>372.</td>
</tr>
<tr>
<td></td>
<td>minor</td>
<td>0.042</td>
<td>26</td>
<td>1710</td>
<td>1.092</td>
<td>71.8</td>
</tr>
<tr>
<td></td>
<td>major</td>
<td>0.018</td>
<td>30</td>
<td>2350</td>
<td>0.54</td>
<td>42.3</td>
</tr>
<tr>
<td>Buy/alliance</td>
<td>none</td>
<td>0.1</td>
<td>10</td>
<td>2550</td>
<td>1</td>
<td>255.</td>
</tr>
<tr>
<td></td>
<td>minor</td>
<td>0.04</td>
<td>14</td>
<td>2700</td>
<td>0.56</td>
<td>108.</td>
</tr>
<tr>
<td></td>
<td>major</td>
<td>0.06</td>
<td>16</td>
<td>3050</td>
<td>0.96</td>
<td>183.</td>
</tr>
<tr>
<td><strong>Expected values</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>19.4</strong></td>
<td><strong>1789.5</strong></td>
</tr>
</tbody>
</table>

The decision tree is particularly useful when a calculation of a project has to be made. Referring to figure 4.16, one can see the minimum and maximum cost and running time: $1050 \leq C \leq 3050[k€]$ and $10 \leq T \leq 30[\text{months}]$, respectively. Depending on whether the calculation has to be done safe on the timing or the financial side, the choice can be made either to bid for 30 months, or for a 3M€ offering; possibly with a profit and/or margin added. However, other issues come into play as well: how important is this project for the continuity of the organization? Is there some prestige to gain, is the customer a long term relation? Then it may be worthwhile to bid with more risk at a lower price. Here, the probabilities can be used.

Every alternative in the decision tree can be assigned a probability, as shown in figure 4.16. Multiplying the probabilities along a route (so, for example the top route from NOW, to upgrade existing A [0.5], performance OK [0.3], resulting in 0.15) gives the estimated probability of this variant occurring, see table 4.8. Even a mean expected value can be made. This value should be considered with caution, as it bases on probabilistics. Only when
Table 4.9: Risk Register as risk inventory and management tool. Ow is Owner; dates are shown in the short ISO notation (YYMMD). The T/s (=trend/status) column uses the following symbols: o: stable, +: increasing, –: reducing. For the strategy type: bac means backup scenario, mon is monitoring, sim is simulating.

<table>
<thead>
<tr>
<th>ID</th>
<th>Identified risk</th>
<th>Ow</th>
<th>Dates</th>
<th>Cost</th>
<th>P</th>
<th>Cost,</th>
<th>T/s</th>
<th>Strategies description</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Panel defect</td>
<td>F</td>
<td>110xxx</td>
<td>9000</td>
<td>15</td>
<td>1350</td>
<td>o</td>
<td>bac</td>
<td>9000</td>
</tr>
<tr>
<td>2</td>
<td>Scrutineering</td>
<td>B</td>
<td>111010</td>
<td>3000</td>
<td>12</td>
<td>360</td>
<td>o</td>
<td>sim</td>
<td>150</td>
</tr>
<tr>
<td>3</td>
<td>Extra media costs</td>
<td>A</td>
<td>110ppp</td>
<td>9000</td>
<td>25</td>
<td>2250</td>
<td>+</td>
<td>mon</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>21000</td>
<td></td>
<td>3960</td>
<td></td>
<td></td>
<td>10150</td>
</tr>
</tbody>
</table>

several projects a year are treated this way, one can use this to make a bid. In other cases, an analysis like this does reveal margins and opportunities. A table like the one in table 4.8 summarizes the minima, maxima, means and expected value and is a good basis for decision making.

4.12.2 Risk register

Where the decision tree is useful in the project planning phase, the risk register that is treated in this section, is useful while the project is underway. Some of the elements in the decision tree reappear in the risk register, some new elements are added.

For each identified risk the following is listed; most are self explanatory, if not, a short description is given:

1. ID
2. Short description of the identified risk
3. Owner
4. Start date
5. Relief date (what is the last moment that action has to be taken)
6. Cost impact (the total cost when the risk becomes reality in full)
7. Probability
8. Weighted cost (this is, as in §4.12.1 Cost impact × Probability)
9. Status/trend (status describes the actual situation. If the risk relates to for instance mass or performance, one can indicate “too high, reducing”)
10. Strategy, including action plan and cost of action plan. NB: the cost of
the action plan has to be explicitly compared to the weighted and total cost of the identified risk. Also the total cost of the action plan and total risk costs have to be compared.

A risk register should be regularly updated and part of the regular project meeting. If the project meeting takes place on a weekly basis, the risk register is updated and discussed weekly. Table 4.9 shows a risk register used in the solar team project.

4.13 Documentation and Reviewing

In many books on systems engineering, documentation plays a central role. In this book, we only spend one section to this subject. Nevertheless, every systems engineer should be aware of the fact that thinking, brainstorming, drawing, discussing and commenting is easy, but putting it on paper (or in digital form) and getting such a document accepted by the relevant stakeholders is sometimes difficult. In the first place because just by writing down, inconsistencies are revealed (as put by Eising: “check whether it can withstand writing down”). Also, a document requires commitment: “I/we stand for what is written here.”

Another purpose of documentation is decoupling of the development processes. This was already mentioned in §2.2 when creating the system design. Thus the documentation structure is the reflection of the process, and therefore—as a reflection on a meta level— the entangling of process and documentation in many systems engineering texts is easily understood.

In this section, first general documentation rules are given (§4.13.1), then specific guidelines are given for the documents in the systems engineering process presented in §2.2 and figure 2.4 is given. The section ends with guidelines for reviewing documents (§4.13.3).

4.13.1 General Documentation Guidelines

Every document should be traceable. This means that always on the first page(s), the following has to be clearly indicated:

- title and short summary
- date
- version
- author (name and role in organization)
4.13 Documentation and Reviewing

- owner (name and role in organization)
- status, this includes:
  - draft/concept/accepted
  - change control: yes/no. If yes, who is the change control authority.
- documents on which this document is based
- distribution list

Avoid duplicating information in a single document. This is a general rule in information technology. Instead, state the information on one place and use references in other places.

Be consistent in the use of terms and units, and use them explicitly. A glossary in the document, or even a global glossary in the project or organization may be needed—a systems engineering task.

It is wise for an organization to have a template so that documents are created easily and uniformly, and so that authors are guided in filling in these details. An author of a document is namely an engineer, not a writer. Note that such a template is not optimized for beauty but for effectivity and efficiency—though beauty should not be ignored—.

The organization also has to define archiving and naming conventions. There are numerous ways and philosophies, this book is not the place to elaborate on them. In fact, the particular convention chosen is of less importance than the fact that if one is chosen it has to be clearly communicated and all people involved have to stick to it. Two remarks are made here:

- a convenient date format is the ISO format: YYYYMMDD as it allows for sorting.
- make sure that archiving is done so that regular backups are made.

Depending on the way the documents are created and updated, it can be wise to use a computerized versioning system. There are several available, like Apache SVN, which can also be used for version control of software code generation.

Returning to the remark that engineers are not authors: notwithstanding that fact, documents have to be readable and use proper language. Also, diagrams should be of good quality. The general rule should be: the document is written for the reader(s), not for the author.
4.13.2 Document Contents

In this section, we will give a short description of the contents of a requirement document, a design document etc. The consequences of the system engineering process as outlined in §2 are that there is an alternating process of interpretation of the need resulting into requirements and interpretation of the requirements resulting into a design (see figures 2.3 and 2.4). Thus, apart from the customer wish, there is an alternating sequence of requirements – design – requirements etc. Therefore we will treat these two types of documents in separate subsections below.

Requirements Document

A requirements document like a system requirement specification (SRS) or a subsystem/element requirement specification (ERS) describes what performance level the (sub-)system has to achieve, without describing the solution (describing the what, without the how). It is, as said above, an interpretation of the need such that:

- on the one hand the customer can verify that what will be delivered is according to his wishes;
- while on the other hand the builder knows what the goal of his development effort is.

A checklist for a requirement specification is given in table 4.10.

The requirements specification is written by the team that has to create the (sub-)system. So the system requirements are written by the system designers, based on the customer wish; the subsystem requirement requirements are written by the subsystem design team, based on the system design specification. This is needed to ensure that the design loop (fig. 2.3) is closed. The requirements are interpreted by the persons that have to create the solution. The review process (§4.13.3) is an essential part of closing the loop as well.

Design Document

The design document like a system design specification (SDS) or subsystem/element design specification (EDS) describes the solution that is chosen to achieve the performance (so the how).

---

5this in contrast with a design document
A requirement specification document . . .

- treats all life cycle phases of the system (see §3.11);
- treats the performance of the system, referring to the functions;
- differentiates between must haves and likes;
- indicates functions that are needed, but are not part of the system under design (derived functions);
- can be used to test the system under design, thus measurable criteria are required;
- is not aimed at one solution (in other words: the solution is not contained in the requirements specification);
- is feasible and supported with tests, experiments, experience etc.

On the system level it is at least an identification of the subsystems with the definition of the interfaces between them. Also, the main system concept has to be described and supported. How this is done, depends on the industry and company. It can be an artist’s impression, or a very quantitative description, or a combination. Also, the items—the so called long-lead items—that have very long development or production times have to be identified and detailed, so that their production can start as soon as possible. This, however has the consequence that their design cannot be changed.

On subsystem and lower levels the documents become more and more specific. Also, there will be more mono-disciplinary documents like mechanical CAD documents, electrical schematics, simulation codes, etc. that are part of the documentation for the subsystem. Nevertheless, the different separate documents have to answer to the subsystem requirements, and this has to be checked, both by the author of the design document, and in a review (see §4.13.3).

Other Documents

While the requirements document and design document have a close relation to the system design process as presented in chapter 2, and more specifically in figure 2.4, during the course of the system design process more documents...
are produced:

**Investigation reports** to present the results of diverse investigations like:
- possible concepts;
- performance of competitor’s products;
- possible causes of a problem;
- areas for improvement of the current system(s);
- design and results of experiments;
- etc.

**Test protocols** that define a test procedure. These are closely related to the requirements documents in the sense that the combined test protocols have to show that the requirements are met (note the link already made in figure 2.8). Important test protocols are the acceptance tests that are conducted with representatives of the customer just before or just after delivery, and that have to be run before payment is completed. These protocols are the official validation of the system (see §2.2.4).

**Test reports** to describe the set-up of a (set of) test(s) and results. Test reports relate to, in contrast to the test protocols above, components and subsystems.

These documents can serve as supporting information for other documents, like requirements documents and design documents. If so, they should be referenced from these requirements and/or design documents.

### 4.13.3 Reviewing a Document

As figure 2.3 shows, designing is a cyclic process. Yet, as we saw above documents are created that describe requirements, designs, experiments etc. A document is the condensation of the process shown in figure 2.3, and can be used for building a next step in the system design process (figure 2.4). This requires the document to be checked, accepted and stable. The checking and accepting is done through reviewing.

A classical review is done by people sitting in a room and going through the document page by page. The people involved are:
- the author,
- the client (the one using the unit, this does not have to be the end customer, but can be someone higher up in the hierarchical chain, see figures 2.2 and 2.4),
- the builder (e.g. the one who has to realise the unit),
• the supervisor of the builder,
• one or more experts,
• representative of (sub)systems that interface with the unit,
• if needed: a facilitator and/or a representative from the systems engineering department.

The experts can be for instance people who have designed or built a comparable unit before, or who have written the requirements. Nowadays reviews can be done online in an electronic fashion using tracking systems. However, this does eliminate the interaction between the people, and removes triggers that exist when people are together in a room.

What should be checked? Reviews should be seen as on the one hand feedback in the design process, as shown in figure 4.17, on the other hand as a communication aspect of creating mutual agreement on language, performance and in case of design documents: solutions.

Therefore the following should be checked:
• consistency:
  – in terms and language within the document and with documents...
higher in the chain,
  – in performance (so compliance with (sub)system requirements and budgets),
  – of interface specifications.

• compliance:
  – with company values and mission,
  – with system concept.

• consequences:
  – for other (sub)systems on the same hierarchical level,
  – for subsystems lower in hierarchy,
  – for project planning
  – for reuse of (sub)systems and components.

The result of a review can be that the document is accepted, that it is accepted with some modifications, or that it remains a concept. In case of acceptance, the document is put under chance control and cannot be altered without notification and consequences, as others have based their work upon it. If changes are needed, a new review with the same committee is required.

4.14 Modeling and Simulation

A symbolic model in the form of equations or statements can be simulated to see how the model behaves. Dynamic models of engineering systems (i.e. describing the behaviour of the model over time) mostly consist of differential equations (ODEs), possibly extended with algebraic constraint equations. The generic form is:

\[ \dot{x} = f(x, u, t) \] \hspace{1cm} (4.1)
\[ y = g(x, u, t) \] \hspace{1cm} (4.2)

where equation \[\text{4.1}\] represents the set of differential equations, with \(x\) as so-called state variables, and \(u\) the input variables; equation \[\text{4.2}\] represents the output equations, they are needed to compute other variables in the system \((y)\), which are not necessary to compute the state variables \(x\). The differential equations may be implicit, implying some algebraic constraint equations are present, hence the term DAE (Differential and Algebraic Equations); equation \[\text{4.1}\] becomes:
These models can be simulated on a computer using standard numerical integration methods, which are basically an approximation of the mathematical integration functionality. The independent variable time will be discretized, i.e. only at discrete points $t_k$ the model is computed (otherwise we would have an infinite amount of time points during a simulation). Further, the numerical integration formulas are approximations of the theoretical integration process. This implies that different numerical integration methods exist, each having their own class of models for which they are suitable in terms of accuracy and simulation speed. The choice of the best usable numerical integration method can be tricky, although for normal models, a contemporary simulation tool, like Simulink or 20-sim, provides suitable integration methods.

During a simulation, the result of the model computation (equation 4.1) is used to compute the value of the states $x$ on the next time, indicated as $x_{k+1}$. After assigning the initial values to the states, successive values of $x_k$ and $y_k$ are calculated successively by calculating the model equations and applying

\[
\dot{x} = f(x, \dot{x}, u, t) \quad (4.3)
\]
the numerical integration method alternatively (right-hand side loop in figure 4.18), until the specified end time is reached. Note that most numerical integration methods need calculation of the model equations (4.1) as part of the numerical integration process (the left-hand side loop in figure 4.18).

Once the simulation is done, or often already during the simulation, the results can be visualized in many ways. Graphs and plots are one way, but modern computer tools include animators that can show a simplified version of the product in motion. Often a combination of the mechanical CAD tool and the simulation program can deliver a rich visualization.

### 4.15 Question Generator

When the project is through the initial phase (the system column in figure 2.4), and the team of people involved has increased, the role of the systems engineer changes, as discussed in §2.3: he becomes more of a watchdog and knowledge keeper. But also he has to probe the decisions made and the impact they have on the system as a whole. For that a combination of walking around with a cup of coffee and asking questions is useful. The questions should not be random, but directed to the system function (both normal operation and exceptions). There are several ways to generate questions.

In [Muller, 2004a], a WWHWWW (why, what, how, who, when, where) question generator is proposed to sample the design space. The generator uses this simple rule:

How about the <characteristic> of the <component> when performing <function>?

Based on the background of the systems engineer, there are many issues to ask detailed questions about. These can relate to mechanical construction issues, or electronic matters. But also physics that may limit the operation of the SUD.

Another way to generate questions is to use the systems requirements and take these to investigate whether the requirements will be met. However, be aware that asking an engineer: "Will you meet the requirements?", will before the deadline always result in the answer "Yes". So, a little more detail should be put in the question, using the system budget (§4.8), for instance.
In many cases, there are open issues in the design; things that have not been detailed, yet. These provide ample base for generating questions.

In all cases, the systems engineer should be sensitive to not behave like the nosy wise guy, but more like an equal who likes to think with the engineer about possible issues and solutions. In the same way the engineer should not treat the systems engineer as the person who has to solve all problems. Therefore the coffee cup is a useful prop to carry while asking questions.

van Amerongen, J.: 2010; Dynamical Systems for Creative Technology; Controlab Products B.V., Enschede.


http://books.google.nl/books?id=3QdsQgAACAAJ

http://www.pearsoned.co.uk/bookshop/detail.asp?item=100000000374477

http://dx.doi.org/10.1109/ICSMC.2009.5346211


Bonnema, G. M.: 2008; FunKey Architecting - An Integrated Approach to System Architecting Using Functions, Key Drivers and System Budgets; Ph.D.-thesis; University of Twente.
http://purl.org/utwente/58868


De Carvalho, M. A. and N. Back: 2000; *Cross-Fertilization Between TRIZ and the Systematic Approach to Product Planning and Conceptual Design*; in TRIZ-CON2000; Altshuller Institute for TRIZ studies, Nashua, NH.


Eising, F.: 2007; *Als je het begrijpt kun je het veranderen*; Inaugural address (in Dutch).

van Exel, N. J. A.: 2011; Behavioural Economic Perspectives on Inertia in Travel Decision Making; PhDThesis; Vrije Universiteit Amsterdam. [http://www.jobvanexel.nl/vanExel_full%20manuscript.pdf](http://www.jobvanexel.nl/vanExel_full%20manuscript.pdf)


Muller, G.: 2011a; System Integration How-To. 
http://www.gaudisite.nl/SystemIntegrationHowToPaper.pdf

http://books.google.com/books?id=MDxSSAAACAAJ

http://www.gaudisite.nl/Thesis.html

Muller, G. J.: 2004b; The Waferstepper Challenge: Innovation and Reliability despite Complexity. 

Muller, G. J.: 2007; Design objectives and design understandability; Last accessed on 13 May 2011. 


http://dx.doi.org/10.1002/sdr.4260090203


Wikipedia; Deepwater Horizon oil spill; Wikipedia, retrieved 20120130. 
http://en.wikipedia.org/wiki/Deepwater_Horizon_oil_spill

http://www.res-systemica.org
APPENDICES
A.1 Short Introduction to TRIZ

Altshuller with his team, has investigated hundreds of thousands of patents looking for patterns in the inventions. This resulted in a method (or better: a set of tools) that can be used to solve various technical problems, hence the name Theory of Inventive Problem Solving (in Russian, this results in the acronym TRIZ). The common pattern in all of TRIZ is, see Figure A.1 that a specific problem is translated into a generalized problem. TRIZ then provides principles and patterns for finding a generalized solution. Final step is then to translate this generalized solution to the specific case at hand.

One of the mostly used tools of TRIZ is the list of innovative principles (IPs) combined with the contradiction matrix. Altshuller found that many patents resulted from solving a contradiction. This contradiction is of the form: when \(<property\ A>\) improves, \(<property\ B>\) deteriorates. The properties in the specific problem can be connected to 39 parameters of a technical system [Altshuller, 1997]. Examples of these parameters are "weight of moving object"; "speed"; and "reliability".

Based on the information in the patents researched, a limited set of ways to solve contradictions was found: the 40 innovative principles (IPs), see Table A.2. Thus there are, according to TRIZ, only a limited number of parameters (39) that constitute contradictions, and a limited number of solving principles (40). Therefore, the contradiction matrix was created (Table A.1 shows a part of this contradiction matrix; the full contradiction matrix can be found in Altshuller [1997]). Any possible contradiction of the form: when \(<technical\ parameter\ X>\) improves, \(<technical\ parameter\ Y>\) deteriorates, can be found in the contradiction matrix. The numbers mentioned in

\[\text{This appendix has been published before as appendix to Bonnema 2011}\]
Figure A.1: The TRIZ approach to design and problem solving. By taking a "detour" via a generalised problem and solution, solving problems is simplified.

the cell indexed by X and Y point to one to four promising innovative principles. E.g. the speed (parameter 9) of a machine has to be increased. However, this leads to an unwanted increase of forces (parameter 10). Then Table A.1 suggests using Innovative Principles 13, 28, 15 and 19. These can be looked up in Table A.2.

Based on the work described in Ivashkov and Souchkov [2004], the priorities of TRIZ principles were determined. This was done both for determining the most appropriate principles to improve a technical parameter (Section A.1.1) as for reducing the negative impact on a technical parameter (Section A.1.2). The former is directly based on the work in the earlier mentioned reference [Ivashkov and Souchkov 2004]; whereas the latter is analogously.

Please note that the body of knowledge in the TRIZ community is large, and expanding rapidly. For more information, a further introduction and latest findings, please refer to http://www.triz-journal.com/

A.1.1 Positive Priority Matrix, PM+

For the positive priority matrix PM+, the number of occurrences of an innovative principle, IP, on a row in the TRIZ contradiction matrix (Table A.1 and Altshuller [1997]) is counted. The principles with the highest occurrences
Table A.1: Part of the contradiction matrix used in TRIZ [Altshuller 1997]. Only the segment for the technical parameters 9–12 is shown. In the cells, the corresponding Innovative Principles are shown. The numbers refer to the principles listed in Table A.2.

<table>
<thead>
<tr>
<th>Improving</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>13, 28, 15, 19</td>
<td>6, 18, 38, 40</td>
<td>35, 15, 18, 34</td>
<td></td>
</tr>
<tr>
<td>Force</td>
<td>13, 28, 15, 12</td>
<td>18, 21, 11</td>
<td>10, 35, 40, 34</td>
<td></td>
</tr>
<tr>
<td>Tension/Pressure</td>
<td>6, 35, 36</td>
<td>36, 35, 21</td>
<td>35, 4, 15, 10</td>
<td></td>
</tr>
<tr>
<td>Shape</td>
<td>35, 15, 34, 18</td>
<td>35, 10, 37, 40</td>
<td>34, 15, 10, 14</td>
<td></td>
</tr>
</tbody>
</table>

are the favorite principles for improving the system’s performance regarding that technical parameter. In addition to the results presented in Ivashkov and Souchkov [2004], we added two columns for the second and third best principles for each parameter. The result is shown in Table A.3.

For some technical parameters the second and/or third best principles are blank. This indicates that the difference between the occurrence of the favorite and second or the second and third best principles was so large that the fitness of that IP can be doubted.

A.1.2 Negative Priority Matrix, $PM^-$

The negative priority matrix $PM^-$ is determined identically. The only difference is that the columns are analyzed, instead of the rows. The resulting negative priority matrix is shown in Table A.4. This matrix can be used to reduce the negative consequences of the system on a given technical parameter.
Table A.2: The 40 principles as identified in Altshuller [1997].

<table>
<thead>
<tr>
<th>#</th>
<th>Principle</th>
<th>#</th>
<th>Principle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Segmentation</td>
<td>21</td>
<td>Rushing through</td>
</tr>
<tr>
<td>2</td>
<td>Extraction</td>
<td>22</td>
<td>Convert harm into benefit</td>
</tr>
<tr>
<td>3</td>
<td>Local quality</td>
<td>23</td>
<td>Feedback</td>
</tr>
<tr>
<td>4</td>
<td>Asymmetry</td>
<td>24</td>
<td>Mediator</td>
</tr>
<tr>
<td>5</td>
<td>Consolidation</td>
<td>25</td>
<td>Self-service</td>
</tr>
<tr>
<td>6</td>
<td>Universality</td>
<td>26</td>
<td>Copying</td>
</tr>
<tr>
<td>7</td>
<td>Nesting (Matrioshka)</td>
<td>27</td>
<td>Dispose</td>
</tr>
<tr>
<td>8</td>
<td>Counterweight</td>
<td>28</td>
<td>Replacement of mechanical system</td>
</tr>
<tr>
<td>9</td>
<td>Prior counteraction</td>
<td>29</td>
<td>Pneumatic or hydraulic construction</td>
</tr>
<tr>
<td>10</td>
<td>Prior action</td>
<td>30</td>
<td>Flexible films or thin membranes</td>
</tr>
<tr>
<td>11</td>
<td>Cushion in advance</td>
<td>31</td>
<td>Porous materials</td>
</tr>
<tr>
<td>12</td>
<td>Equipotentiality</td>
<td>32</td>
<td>Changing the colour</td>
</tr>
<tr>
<td>13</td>
<td>Do it in reverse</td>
<td>33</td>
<td>Homogeneity</td>
</tr>
<tr>
<td>14</td>
<td>Spheroidality</td>
<td>34</td>
<td>Rejecting and regenerating parts</td>
</tr>
<tr>
<td>15</td>
<td>Dynamicity</td>
<td>35</td>
<td>Transformation of properties</td>
</tr>
<tr>
<td>16</td>
<td>Partial or excessive action</td>
<td>36</td>
<td>Phase transition</td>
</tr>
<tr>
<td>17</td>
<td>Transition into a new dimension</td>
<td>37</td>
<td>Thermal expansion</td>
</tr>
<tr>
<td>18</td>
<td>Mechanical vibration</td>
<td>38</td>
<td>Accelerated oxidation</td>
</tr>
<tr>
<td>19</td>
<td>Periodic action</td>
<td>39</td>
<td>Inert environment</td>
</tr>
<tr>
<td>20</td>
<td>Continuity of useful action</td>
<td>40</td>
<td>Composite materials</td>
</tr>
</tbody>
</table>
Table A.3: The positive priority matrix $PM^+$, found from counting the times an innovative principle is mentioned in the row of a particular technical parameter. The numbers in the columns marked “1st”, “2nd”, and “3rd” are pointers to the relevant innovative principles, see Table A.2.

<table>
<thead>
<tr>
<th>Technical Parameter</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Weight of moving object</td>
<td>35</td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>2 Weight of stationary object</td>
<td>35</td>
<td>10, 19, 28</td>
<td>1</td>
</tr>
<tr>
<td>3 Length of moving object</td>
<td>1, 29</td>
<td>15, 35</td>
<td>4</td>
</tr>
<tr>
<td>4 Length of stationary object</td>
<td>35</td>
<td>28</td>
<td>1, 10, 14, 26</td>
</tr>
<tr>
<td>5 Area of moving object</td>
<td>2, 15</td>
<td>13, 26, 30</td>
<td>4</td>
</tr>
<tr>
<td>6 Area of stationary object</td>
<td>18</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>7 Volume of moving object</td>
<td>1, 35</td>
<td>2, 10, 29</td>
<td>4, 15, 34</td>
</tr>
<tr>
<td>8 Volume of stationary object</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 Speed</td>
<td>28</td>
<td>35</td>
<td>13</td>
</tr>
<tr>
<td>10 Force (Intensity)</td>
<td>35</td>
<td>10, 18, 37</td>
<td>36</td>
</tr>
<tr>
<td>11 Stress or pressure</td>
<td>35</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>12 Shape</td>
<td>10</td>
<td></td>
<td>1, 14, 15, 32, 34</td>
</tr>
<tr>
<td>13 Stability of the object’s composition</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 Strength</td>
<td>3, 35</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>15 Duration of action of moving object</td>
<td>19</td>
<td>35</td>
<td>3, 10</td>
</tr>
<tr>
<td>16 Duration of action of stationary object</td>
<td>35</td>
<td>1, 10, 16</td>
<td>40</td>
</tr>
<tr>
<td>17 Temperature</td>
<td>35</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>18 Illumination intensity</td>
<td>19</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>19 Use of energy by moving object</td>
<td>35</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>20 Use of energy by stationary object</td>
<td>19, 35</td>
<td>18, 27</td>
<td>many</td>
</tr>
<tr>
<td>21 Power</td>
<td>35</td>
<td>19</td>
<td>2, 10</td>
</tr>
<tr>
<td>22 Loss of energy</td>
<td>7</td>
<td>35</td>
<td>2</td>
</tr>
<tr>
<td>23 Loss of substance</td>
<td>10</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>24 Loss of information</td>
<td>10</td>
<td>26, 35</td>
<td></td>
</tr>
<tr>
<td>25 Loss of time</td>
<td>10</td>
<td>35</td>
<td>18</td>
</tr>
<tr>
<td>26 Quantity of substance/the matter</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27 Reliability</td>
<td>35</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>28 Measurement accuracy</td>
<td>32</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>29 Manufacturing precision</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30 Object-affected harmful factors</td>
<td>22</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>31 Object-generated harmful factors</td>
<td>22, 35</td>
<td>2</td>
<td>1, 39</td>
</tr>
<tr>
<td>32 Ease of manufacture</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33 Ease of operation</td>
<td>1</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>34 Ease of repair</td>
<td>1</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>35 Adaptability or versatility</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36 Device complexity</td>
<td>13, 26</td>
<td>1, 28</td>
<td>2, 10</td>
</tr>
<tr>
<td>37 Difficulty of detecting and measuring</td>
<td>28</td>
<td>35</td>
<td>16, 18, 26, 27</td>
</tr>
<tr>
<td>38 Extent of automation</td>
<td>35</td>
<td>13</td>
<td>28</td>
</tr>
<tr>
<td>39 Productivity</td>
<td>10</td>
<td>35</td>
<td></td>
</tr>
</tbody>
</table>
Table A.4: The negative priority matrix $PM^-$, found from counting the times an innovative principle is mentioned in the column of a particular technical parameter. The numbers in the columns marked “1st”, “2nd”, and “3rd” are pointers to the relevant innovative principles, see Table A.2.

<table>
<thead>
<tr>
<th>Technical Parameter</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight of moving object</td>
<td>35</td>
<td>8, 15, 28, 26, 40</td>
<td></td>
</tr>
<tr>
<td>Weight of stationary object</td>
<td>35</td>
<td>26, 27</td>
<td></td>
</tr>
<tr>
<td>Length of moving object</td>
<td>1</td>
<td>15</td>
<td>14, 17, 28, 29</td>
</tr>
<tr>
<td>Length of stationary object</td>
<td>14, 26</td>
<td>1, 10, 35</td>
<td>7, 15, 28</td>
</tr>
<tr>
<td>Area of moving object</td>
<td>17</td>
<td>13, 15, 26, 10, 29</td>
<td></td>
</tr>
<tr>
<td>Area of stationary object</td>
<td>16, 18, 35, 40</td>
<td>2, 10, 17, 39, 30, 36</td>
<td></td>
</tr>
<tr>
<td>Volume of moving object</td>
<td>35</td>
<td>2, 10</td>
<td>29</td>
</tr>
<tr>
<td>Volume of stationary object</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speed</td>
<td>35</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Force (Intensity)</td>
<td>35</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Stress or pressure</td>
<td>35</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Shape</td>
<td>1, 35</td>
<td>29</td>
<td>10, 14, 15</td>
</tr>
<tr>
<td>Stability of the object’s composition</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strength</td>
<td>28</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Duration of action of moving object</td>
<td>35</td>
<td>3</td>
<td>19</td>
</tr>
<tr>
<td>Duration of action of stationary object</td>
<td>16, 10, 35</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Temperature</td>
<td>35</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>Illumination intensity</td>
<td>19</td>
<td>32</td>
<td>1, 13</td>
</tr>
<tr>
<td>Use of energy by moving object</td>
<td>35</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>Use of energy by stationary object</td>
<td>1</td>
<td>35</td>
<td>18, 19</td>
</tr>
<tr>
<td>Power</td>
<td>35</td>
<td>10, 19</td>
<td>2, 18</td>
</tr>
<tr>
<td>Loss of energy</td>
<td>35</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Loss of substance</td>
<td>10</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Loss of information</td>
<td>10</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Loss of time</td>
<td>10</td>
<td>35</td>
<td>28</td>
</tr>
<tr>
<td>Quantity of substance/the matter</td>
<td>35</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>10</td>
<td>11</td>
<td>40</td>
</tr>
<tr>
<td>Measurement accuracy</td>
<td>28</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Manufacturing precision</td>
<td>32</td>
<td>10</td>
<td>28</td>
</tr>
<tr>
<td>Object-affected harmful factors</td>
<td>22</td>
<td>35</td>
<td>2</td>
</tr>
<tr>
<td>Object-generated harmful factors</td>
<td>35</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Ease of manufacture</td>
<td>1</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Ease of operation</td>
<td>1</td>
<td>32, 35</td>
<td>2, 28</td>
</tr>
<tr>
<td>Ease of repair</td>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Adaptability or versatility</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device complexity</td>
<td>1</td>
<td>26, 10</td>
<td></td>
</tr>
<tr>
<td>Difficulty of detecting and measuring</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extent of automation</td>
<td>35</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Productivity</td>
<td>35</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The history of FMEA goes back to the early 1940’s by the U.S. military and was further developed by the aerospace and automotive industries. Up to now several formal FMEA standards have been developed for different type of industries, based on the ISO9001 standards. The following types of FMEA can be distinguished:

**B.1 System FMEA**

The SFMEA: SYSTEM-FMEA is applied in interaction of parts of a systems. Used to analyze complete systems and/or sub-systems during the concept design stage. The focus is on minimizing failure effects on the SYSTEM to provide SYSTEM objectives like, quality, reliability, robustness, maintenance, life-cycle costs.

**B.2 Design FMEA**

The DFMEA: DESIGN-FMEA is applied to the development of a product. Used to analyze a product design before it is released to the manufacturing stage. The focus is on minimizing failure effects on the PRODUCT to provide PRODUCT objectives like, quality, reliability, robustness, maintenance, life-cycle costs.
B.3 Process FMEA

The PFMEA: PROCESS-FMEA is applied to the process. Used to analyze manufacturing and/or assembly process. It focus on minimizing failure effects on the PROCESSES to provide maximize total PROCESS objectives.
agility, 47
architecture, 65
coupling matrix, 70
Cradle to Cradle, 49
customer acceptance test, 27
derived functions, 91
environment, 67
frequency-domain, 36
function, 53
goal of system design, 16
influence diagram, 63
infrastructure, 53
interfaces, 22
key driver, 70
LifeCycle Analysis, 49
likes, 91
long-lead items, 23 91
mile-stones, 50
must haves, 91
problem domain, 18
product life cycle, 47
project life cycle, 47
resource life cycle, 47
Risk Priority Number, 80
scenario, 56
solution domain, 18
stakeholder, 53
state, 61 62
story telling, 56
subsystems, 22 66
system, 16
system designer, 10
system of systems, 54
system requirement specification, 20
system requirements, 20
system specifications, 20
Systems Engineering, 9
systems engineering, 10
time-domain, 36
transition, 61 62
TRIZ, 54 105
use case, 58
utility, 53
validation, 27
verification, 26
WWHWWW, 96