

# I. General Context and Theories

*He who loves practice without theory is like the sailor who boards ship  
without a rudder and compass and never knows where he may cast.*

Leonardo da Vinci

This part shall provide the fundamental understanding of most core concepts involved in the construct of ideas leading to this thesis and its results. Consequently, the following chapters provide an overview over the major research fields having influence on the outcome of this thesis. If employed, *requirements traceability* can be seen as a crosscutting concern of all development activities. Correspondingly these chapters strive a considerable set of very different general research disciplines.

Stepping into any research topic of considerable depth often implies a steep entry curve for any reader being non-expert of the research domain. One of the problems is that topics are often manifold interconnected making it difficult to find a good start. The author has tried to flatten the entry curve by starting with chapters with lower entry barriers. These are the chapters that are more independent of the other chapters. With the understanding and argumentation collected in the first more independent chapters, the further chapters build on the previous chapters and then have lower entry barriers.

In this thesis, the model concept is an essential foundation, since different types of models are referred to in different theories. Correspondingly, this part starts with a general discussion on the model concept and related terms needed at later discussions (ch. I.1). This is followed in ch. I.2 by a general discussion about developing embedded systems in general. A certain category of embedded systems, called *safety-critical embedded systems*, demand special concerns about quality, because malfunctions in these systems can involve significant fatal consequences. Correspondingly, special standards for development processes (ch. I.7) have evolved to ensure quality of the developed systems. One central demand are especially rigid demands for *requirements traceability*. As results of this thesis arose in the context of companies involved in the automotive domain, a special ch. I.2.3, discusses specific peculiarities of the automotive domain. Even though the concepts of the developed R2A tool in principle can be applied to any development project, some of the features provide special help in embedded projects of the automotive domain. This is, e.g., the case for the special improvements of *supplier management* (see ch. III.23.1), as the automotive domain is a domain with very extensive and deep chains of suppliers.

Ch. I.3 and ch. I.4 then provide general introductions into the theories of software engineering and systems engineering. Both theories' concepts are an integral part of current development process standards such as the quality standards applied for *safety-critical embedded systems* (ch. I.7).

In ch. I.5, current *requirements engineering and management (REM)* theory is discussed. The *traceability* concept is a nascent of this theory. Correspondingly, the sub ch. I.5.7 also discusses the *traceability* concept in the context of *REM*-theory and explains concepts needed in the following chapters of this part. An extensive discussion of the *traceability* concept is then performed in ch. II.10 of part II.

Concerning the transition from requirements to design, the author considers this an especially difficult *traceability* problem, because this transition is a transition from the *problem space* description (requirements) to the *solution space* description (design) implying a considerable semantic gap between both. Therefore, this thesis lies a special focus on this topic. Ch. I.6 outlines design with its concepts and theories that are important to understand the problems of *traceability* concerning this transition. Instead of concentrating on a specific modeling paradigm or method related to software or systems engineering, this chapter rather tries to outline several general theories about design that describe the role of design and how design emerges from designers' thinking.

After the previous chapters have outlined fundamental concepts of different general theories building the theoretical groundwork of this thesis, ch. I.7 describes process standards to be fulfilled by organizations developing *safety-critical* embedded systems. Due to its extent and complexity, the outlined process standards cannot be described in full depth. Instead, after a general overview is provided, the engineering processes concerning requirements and design with their *traceability* demands are described in detail. In this way, the author derives important demands, which the tool-approach described in part III must fulfill in order to conform to the standards.

Last but not least, ch. I.8 refers to findings from practice of embedded engineering that should be kept in mind considering a practice-oriented solution for *traceability* in the context of design.

# I.1 The *Model* Concept

*We can only make a model of a fact in the world we live in,  
i.e. the model must be essentially related to the world we live in  
and what's more, independently of whether it's true or false.*

Ludwig Wittgenstein (\*)

“Models are a fundamental concept of our world's handling. All scientists and engineers use and create models to prove universal evidences for and to find more detailed information on their speculations. Often models mark intermediate step on the road to new artifacts as bridges, cars and mobile telephones. In Software Engineering the importance of models is even higher, because they not even represent the intermediate steps, but the endpoints of our work: a specification but also a program is a model” [LL07; p.3 (\*)].

Stachowiak [St73] found several general properties that models have in common with each other (the following statements are taken from [LL07; p.5-6] and [BR07b; p.4]):

- A *purpose* (or purposes),
- A reference to the original, also called *mapping characteristic*<sup>5</sup> [LL07; p.5],
- Abstraction of certain qualities of the original, also called *shortening characteristic*<sup>6</sup>: A diversity of relationships can exist between model and original emerging by the model's usage purposes [BR07b; p.4],
- A *pragmatic characteristic*: “Under certain conditions or problems, models can supplement the original” [LL07; p.6 (\*)];

Fig. 1-1 shows the connections between original and its model according to [LL07; p.6] and [St73]. Together three kinds of properties can be distinguished:

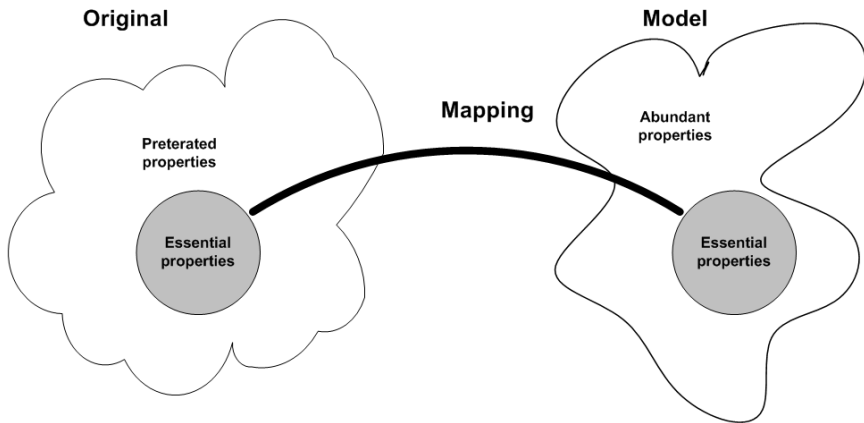
- *Essential properties* (also called non-neglected) are the properties of the original considered in the model.
- *Preterated properties* (also called neglected) are properties of the original not considered in the model.
- *Abundant properties* are properties in the model, not present in the original. These properties emerge from the nature of the model<sup>7</sup> (Simon [Si06; p.113] calls this the implicit logic of the sign system).

---

<sup>5</sup> In German: Abbildungsmerkmal

<sup>6</sup> In German: Verkürzungsmerkmal

<sup>7</sup> Considering the photo of a person, preterated properties of the person would be its weight, name, type, whereas the quality of the photo paper or the photo's format would be abundant properties (cf. [LL07; p.6]).



**Figure 1-1** Properties of original and model [LL07; p.6 (\*)]

These properties distinctions lead to two fundamental problems that should always be considered when working with models:

1. Due to the *preterated properties*, “models are always a 'simplification, a kind of idealization' of the aspects to be modeled. ... We choose for our model these characteristics of the reality that we consider essential for our purpose. In complex situations ... this act of already distinguishing the essential from the non-essential must be at least partially an act of judgment, often of political or cultural judgments. And this act must then necessarily base on the intuitive thinking model of the model constructor” [We76; p.202 (\*)].
2. On the other hand, *abundant model properties* can lead to erroneous conclusions about the original. “The implicit logic of the sign system resp. symbols, representations, languages, texts, formulas, etc., are in general different to the represented phenomena or items; If both are mixed up, the danger arises that peculiarities of the observation method (resp. the observers) and its results are considered instead of the observed fact” [Si06; 113 (\*)].

Generally, two different model types exist according to [LL07; p.5] (also cf. [St73], [Mo04; p.64f]):

- *Descriptive models* describe already existing connections or systems.
- *Prescriptive models* are manuals for the construction of, e.g., systems.

In the context described here, both types of models occur. Thus, e.g., a SW documentation is a *descriptive model*, whereas a model as basis for model based code generation represents a *prescriptive* one. Following these interpretations, a

*SW design model* can be first a *prescriptive model* determining the structure of the code to develop. After coding has been finished, however the model would become *descriptive*. Later in ch. I.7, it is shown that a similar connection may exist in the area of *process models* and that users of *process models* should be aware of possible misinterpretations sparked by an inadvertent transformation of *descriptive process models* into *prescriptive process models*.

Due to these possible interpretation ambiguities where the real character of a model is not absolutely clear, Schefe [Sch99; p.132] asks for abandoning meaning from the model concept in *software engineering* except its clear meaning emerges from the usage context [Sch99; p.134] (see also [Mo04; p.65]). In fact, as the discussion in ch. I.7.3.1 shows, these dangers of interpretation and unconscious shift of meaning can happen.

The main purpose of a model is the communication of ideas and concepts [Mo04; p.171]. Correspondingly, attention must be paid for conclusiveness of the modeled ideas. In this context, it seems legitimate to speak of a certain aesthetics models should have [Kr95; p.43]. Ch. I.6.1.2 again discusses model esthetics in connection with *SW architectures*.

Concerning system and software development, models have some special characteristics. In more complex development processes, at least two kinds of models must be considered ([De04], [Br07a]):

- A model<sup>8</sup> for the targeted system.
- A model for the development project's processes.

This thesis deals with both kinds. In the context of design (but also a bit in *requirement engineering*) the first mentioned model kind is essential. When process standards as SPICE (see ch. I.7) or process related concepts such as *traceability* are discussed, the second kind is equally essential.

Often, strict *formal* semantics are also observed as an obstacle to designers ([Sch83], [HA06a]). As further discussed in ch. I.6.2.3 and ch. II.9.4.2, this is especially the case in earlier phases of design, or when designers encounter significantly complex situations where no solution covering all aspects can be found at once. In this context, some designers (cf. [AMR06], [Kr95; p.49], [Go99], [Go95]) emphasize that especially sketching is important because it produces ambiguity, a widening of the problem scope and general uncertainty about the final solution as nourishment for designers' creativity to bring up new solution ideas (see ch. I.6.2.3).

---

<sup>8</sup> In most practice, not one model but several models exist. This is the case, because different models with different semantics are often employed at different levels of abstraction. Perhaps it is better to say that it should be the goal to have a model of the system.

## I.2 Embedded Systems Development

*Grey, dear friend, is all theory and green is the golden tree of life.*  
J. W. v. Goethe (\*)

Most of the topics and interrelations discussed in this thesis are not really limited to the embedded systems development market, but the special conditions of the embedded area force a much stronger need for employing some of the later described concepts and techniques. Therefore, before beginning with other more specific topics a short introduction into this very complex field shall be given.

### I.2.1 Definition and Context

*Embedded systems* – or better *embedded control units* (ECUs) – are computer based systems embedded into a bigger surrounding technical (total) system (automobiles, airplanes, power plants, consumer electronics etc.) often also referred as the context of an ECU. In most cases, ECUs perform complex control, regulation, observation and data processing activities on physical-mechanical components with decisive impact on functionality and performance of the complete system (cf. [Sch05], [Ge05; p.5]).

ECUs itself mostly work very integrated into the complete system so that users are usually not really aware of the ECUs itself, but the bigger processes or technical components are somehow controlled by humans [Ge05; p.5]. Nonetheless due to its broad range of employment from very small systems as RFID<sup>9</sup> chips to normal day-life devices (CD-players or washing machines) to high technology devices (air planes or computer tomographs), over 90 percent of electronic components are embedded systems. This means that of 8.3 billion produced processors in 2002, 8.15 billion were used for embedded systems whereas only 150 millions of processors were part of ordinary computers [Sch05; p.2]. Due to the diversity of usages for embedded systems, the embedded market is still one of the fastest growing markets [Sch05; p.2].

### I.2.2 Characteristics

The fact of being embedded in a higher technical system leads to a set of characteristics different to ordinary computers [Sch05; p.3ff], [Ge05; p.5f].

---

<sup>9</sup> Radio Frequency Identification

An ECU's primary source of interaction is not humans but the surrounding processes or technical components. Humans indirectly influence ECUs by controlling the processes and devices they are integrated, but, primarily, ECUs retrieve input by sensors and perform output by actuators integrated into the surrounding system. Accordingly to the special purposes ECUs often fulfill, the ECUs in most cases have specialized hardware (HW) specifically designed for efficiently fulfilling their purposes.

Since the surrounding system mostly is an electronic, physical-mechanical, chemical or biological device or process, developing ECUs has a strong need for interdisciplinary development efforts such as *systems engineering* discussed in ch. I.4.

Ordinary computer systems can be described as *interactive* systems. This means, the computer system actively determines the interaction process with the environment. Whenever an *interactive* system needs input for further processing the system prompts the user for input and proactively synchronizes with the environment.

ECUs on the contrary react more on the settings and changes of the environment. They are therefore called *reactive* systems. This difference has significant influence on their behavioral determinism. Interactive systems can be more seen as non-deterministic (e.g., interactive systems decide on their own how to schedule different tasks), whereas ECUs have well defined input and reaction relations with mostly strict temporal interdependencies derived from the needs of their surroundings. Three implications can be deduced from this fact:

- At first, Scholz emphasizes that “the different characteristics of both system types must be considered when adequate techniques, methods or tools are developed” [Sch05; p.4 (\*)].
- Secondly, *SW designs* of *reactive systems* can heavily rely on the very well defined and researched concept of *state machines*. Since *state machines* are deterministic and have a complete *formal* semantics (other to, e.g., the semantics of activity diagrams in UML), they can be properly used for *formal requirements specification*, their early simulation, verification and complete code generation providing very positive effects on complexity handling [Ma08a; p.19] (see also [MB05]).
- Unfortunately, the temporal interdependencies force ECUs to obey timing limits. In this context, ECUs are often referred as *real time* systems. *Real time systems* can be distinct between systems that must obey their timing rules at any time (so called hard real time) and systems that should obey their timing rules as good as possible with exceptions allowed (so called soft real time) [Do04; p.3], [Sch05; p.4].

Another not yet explicitly mentioned demand for ECUs is their functional correctness. Different to programs running on ordinary computer systems, errors in already delivered ECUs cannot be easily fixed by users installing updates. Instead, expensive product recalls are necessary to fix those problems.

In many application contexts, such as medical equipment, space, aviation, nuclear power plants, production lines or automotive, system malfunctions and other defects can cause more severe consequences such as threats to life or physical condition. Such systems are called *safety-critical*. Constructing *safety-critical* systems demands enforced efforts on avoiding or at least diminishing the probability of malfunctions, other defects, or fatal consequences. Two factors are the central means to achieve this goal:

1. Explicit consideration in the design of these systems (e.g., providing redundant system parts).
2. Employing development processes ensuring high quality of the resulting system.

Concerning the first point, it is to say that this thesis speaks about design, but more on a higher *meta-level* and therefore point one will not be directly<sup>10</sup> in the focus of this thesis. The second point, however, is directly addressed in this thesis, as *requirements traceability* is seen as an important foundation to achieve those high quality development processes.

A fundamental principle of these processes is that their potential to ensure high quality outcomes must be controlled in an objective way. This is achieved by a set of standards such as the ISO 15504<sup>11</sup> (SPICE) defining necessary characteristics that development processes for *safety-critical* systems must fulfill. Correspondingly, the solution proposed here must obey the criteria demanded by those process standards. Ch. I.7 provides a description of these standards with the demanded criteria that are important to this thesis.

Differently to normal PC applications, ECUs are designed for a specific purpose. To optimize costs, the principle of HW/SW Co-design<sup>12</sup> is used, where HW and SW are designed in parallel with high interdependencies between each other to only fulfill its specific purpose. Especially for applications with high volumes, the so called mass market, the costs per part are decisive. Therefore

---

<sup>10</sup> Indirectly it well touches this issue in the sense that design for *safety-critical* issues involves decisions to be taken that impose significant consequences on the design outcome. As communication and documentation of decisions and their consequences is one of the special concerns of this thesis, this topic is indirectly connected and this connection is show in part III as real-world example of decision-making in embedded projects.

<sup>11</sup> Software Process Improvement Capability dEtermination (SPICE).

<sup>12</sup> For more information on this topic cf. [ME01].



extreme optimization of HW costs has highest priority often leading to highly specialized SW. This kind of SW has to deal with very tight resource restrictions leading to a significantly higher complexity to be handled in the SW development activities.

### I.2.3 Embedded Development in the Automotive Domain<sup>13</sup>

*Technical complexity of electronics and software in the automotive industry is similar complex as avionics and aerospace. Today, cars are the mass production product with the strongest cross-linking of separate computers at the smallest space. Meanwhile, more than 90 % of all functions are realized with support of software. The quality of a car is substantially determined through the quality of electronics and software. For this reason, software quality has become a central competitive factor.*  
[HDH+06; p.267-268 (\*)]

“Modern premium automobiles contain by now up to 100 ECUs, with increasing tendency accompanied by approx. 3 kilometres of cable and approx. 2000 plug connectors. In these ECUs, SW with more than 600 000 lines of code regulates numerous functions and their cooperation. ... In this way, the value creation changes significantly in Automotive construction. 90% of the innovation in cars are driven by electronic components, thereof 80% software“ [Sch05; p.12f (\*)].

At present as in the near future, the proportion of software (SW) and SW-based ECUs in everyday products increases exponentially [Br06], (also cf. [CFG+05], [KCF+04], [HDH+06; p.267]) and this increase is accompanied by a growth of development complexity. Correspondingly, developing these SW-based ECUs meanwhile has a central strategic importance for the automotive industry.

The automotive domain has some special conditions imposing special challenges for embedded systems engineering. Generally, the following special challenges can be identified playing significant key-roles in automotive embedded development (cf. [Br06], [Gr05], [KM06], [SZ06; p.20], [Sch05; p.5]):

1. *Safety-criticality*: As mentioned in the chapter before, cars involve several *safety-related* issues. These issues must be significantly addresses as described in the chapter above.
2. *Costs*: As cars are mass-market products with high unit volumes, costs play a decisive role. Thus, proportional manufacturing costs dominate the price. In this way, ECUs' costs are also under strong pressure. The proportional manu-

---

<sup>13</sup> Parts of this chapter base on [TWT+08].

facturing costs of ECUs are mainly dominated by HW costs. This leads to highly cost-optimized HW with minimal HW resources concerning memory calculation power, and other components. Correspondingly, software must often be fitted to handle these, often leading to higher complexity and unnatural solutions in the software design [SZ06; p.20], [Sch05; p.5].

3. *Quality*: Buying a car involves significant costs for the customer. In consequence, cars are intended for long product life-cycles of about 25 years [SZ06; p.20]. Correspondingly, cars must provide a high overall quality, especially if they are premium products.
4. *Hard or at least weak timing restrictions*<sup>14</sup>: Reasons can be physical requirements for exact timing (e.g., when controlling motor injection), extremely cost optimized HW where strong resource restrictions lead to strong demands for timing; or *safety-related* issues (e.g., exact timing of inflating airbags during crash situations).
5. *Strong cross-linking of ECU systems*: Increasing cross-linking of vehicle functional features leads to increasing cross-linking of ECUs<sup>15</sup>. Such features are typically realized by a collaboration of several ECUs, leading to higher interdependencies between ECUs. ECUs in automotive development are usually an integrated system consisting of HW, SW and mechanical components [MHD+07; p.91]. In most cases not one ECU handles a certain function in a car, but several ECUs in interplay with each other realize a certain function. Thus, the different ECUs can communicate with each other using communication protocols such as Controller Area Network (CAN), Local Interconnect Network (LIN), Media Oriented System Transport (MOST) or Flexray. In summary, the interconnected ECUs can be seen as distributed systems with distributed control logic and changing control hierarchies [Ge05; p.5].
6. *High demands on inter-organizational collaboration*: The development of a strongly cross-linked car system can only take place in collaboration with the car manufacturers (Original Equipment Manufacturers (OEM)) and heterogeneous chains of suppliers.
7. *High numbers of variants*: Today, the buyer of a car has the choice between hundreds of options being partly connected to each other (e.g., different motors can be combined with different gearboxes) [SZ06; p.9]. As a plus, cars

---

<sup>14</sup> Mostly, not all timing restrictions of hard real time systems are strict. Some functions may also have weaker or even no timing restrictions.

<sup>15</sup> A typical scenario might look like this: A car crash triggers crash sensors which activate several airbag ECUs and a crash management ECU (CM-ECU). The CM-ECU sends an 'Unlock\_Doors' signal to all door ECUs, requests the position from the Global Positioning System-ECU and sends an automatic emergency call via a Universal Mobile Telecommunications System-ECU to local rescue organizations.

Tool-Based Requirement Traceability between  
Requirement and Design Artifacts

Turban, B.

2013, XXVII, 439 p. 60 illus., Softcover

ISBN: 978-3-8348-2473-8