Navigating the Requirements Jungle ⋆

Boris Langer[1] and Michael Tautschnig[2]

[1] Diehl Aerospace GmbH
An der Sandelmühle 13, 60439 Frankfurt, Germany
[2] Institut für Informatik
Technische Universität Darmstadt
Hochschulstr. 10, 64289 Darmstadt, Germany

Abstract Research on validation and verification of requirements specifications has thus far focused on functional properties. Yet, in embedded systems, functional requirements constitute only a small fraction of the properties that must hold to guarantee proper and safe operation of the system under design.
In this paper we try to shine some light on the kinds of requirements occurring in current embedded systems design processes. We present a set of categories together with real-life examples. For each of them, we briefly describe possible approaches towards formal modeling and automated verification of the respective properties.

1 Introduction

Control systems deployed in commercial aircrafts, emergency systems installed in nuclear power plants, remote-controlled surgery robots in hospitals, and automated braking assistants in automobiles are just a few examples of modern safety-critical systems. During the last decades, humans are increasingly poised to hand the responsibility for their lives over to electronic systems.

The growing complexity of these systems, and the fact that a failure of a single subsystem may have fatal consequences for the users, forces the industry to reconsider the underlying development process to obtain products of the required quality.

The large scale of the systems inevitably requires a great number of stakeholders. Communication of needs and constraints is thus fundamentally complex, both because of the sheer amount, and even more because of different—domain specific—vocabularies. Ambiguities in early specifications (Figure 1) then yield costly changes at later project phases.

To overcome such issues, *requirements* based approaches (cf. [1]) are used in systems development. Requirements describe properties of a system. They give specifications of varying precision of the system to be developed, and later provide means to judge whether the product meets the goals.

The IEEE Standard 830 [2] lists desirable characteristics of requirements: (a) correct, (b) unambiguous, (c) complete, (d) consistent, (e) ranked in importance and/or stability, (f) verifiable, (g) modifiable, and (h) traceable. Further, each requirement shall be uniquely identified.

⋆ Supported by DFG grant FORTAS – Formal Timing Analysis Suite for Real Time Programs (VE 455/1-1).

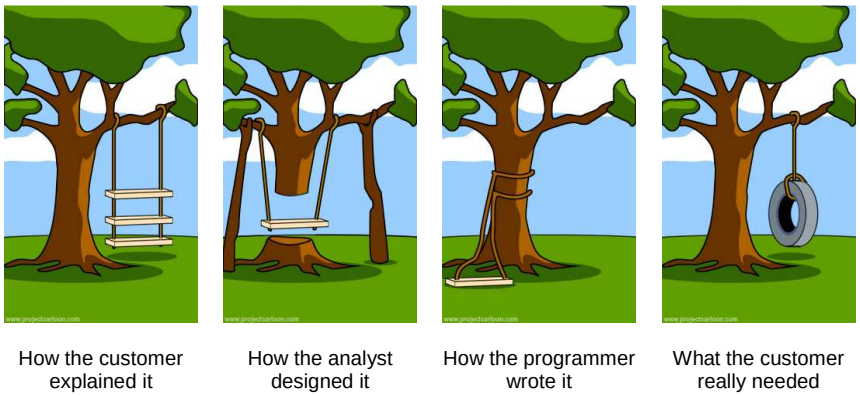

Figure 1. Well known misunderstandings in a project

Requirements Engineering in Avionics Industry

Because of the background of the authors, our view on the development process is inherently biased towards *avionics industry*. The term "avionics" is a synthesis of *aviation* and *electronics*. Starting in the early 1970s, the number of systems in an aircraft has grown in orders of magnitude. Thus it was necessary to emphasize integration of subsystems. It was primarily driven by the change from mechanical instruments to electronic instruments. This gave birth to avionics industry. Nowadays more than half of the budget of a new aircraft is spent on avionics systems.

Diehl Aerospace is one of the largest avionics suppliers in Europe. We deliver systems for Airbus and Boeing. Typical products are control systems for flaps, display systems, and cabin systems.

The avionics industry has been successful since around 30 years in developing highly reliable systems. Traveling by aircraft is safer than in any other vehicle. This might be due to a good understanding of the typical problems that can occur during a flight, but also because of the huge amount of money that is spent during the analysis phases of each newly developed program. As we expect air traffic to triple in the next 20 years, sustaining the quality and safety becomes a huge challenge for the existing infrastructure and new systems will need to be developed.

To guarantee quality and safety, regularities and guidelines controlled by official authorities must be adhered to. At systems level this is the international standard SAE-4754 [3], and for the software sub-systems DO-178B / ED-12B [4] applies. All guidelines are built on consensus of all players in industry. Most of these standards are based on older international standards like [5]. Further, many manufacturers compile internal standards based on existing ones to tailor them towards their typical projects (cf. [6]).

As we focus on software in this paper, we are primarily concerned with the objectives laid out in DO-178B / ED-12B. It delineates verification constraints to detect and report errors that may have been introduced during the software development processes. Software verification objectives are satisfied through a combination of reviews

and analysis, the development of test cases and procedures, and the subsequent execution of those test procedures.

Contribution In all such assessments, requirements specified earlier are checked. Not only the ability to assess varies largely among the set of requirements, but also the techniques used to describe the requirements are non-uniform. In this paper we we give a taxonomy of the kinds of requirements typically found in avionics system design, and take a short glance at possibilities of verifying that a requirement is met by an implementation. We therefore wade through the jungle of requirements along several paths: Requirements are first grouped according to their occurrence in the development process in Section 2.1, and then following our categorization in Section 3. We give examples from our domain to illustrate the abstract terms.

2 Requirements in the Design Process

Abstraction is a key concept in dealing with large scale system designs. Successive refinement keeps complexity local and manageable. Requirements follow this schema and occur at all levels of abstraction. For example, requirements coming from the customer are very high level and implementation independent, whereas the requirements for a specific software module are as detailed as necessary to directly derive source code based on them.

2.1 Hierarchy of Requirements

In Figure 2 the hierarchy of requirements is illustrated as a pyramid. The top level process in the aircraft development cycle includes the identification of aircraft functions and the requirements associated with these functions ([3] §5.1). As a result the purchaser technical specification (PTS) is the first document that describes the need of a new program. Each of the potential suppliers must explain to the purchaser how they intend to satisfy the requirements.

High Level Requirements Specifications given in the PTS are also known as *high level requirements* (HLR). These are then refined to more detailed requirements. It is up to the systems design department to decide which of the requirements to allocate to a software modules or to a hardware modules.

Low Level requirements The refinements of HLRs are called *low level requirements*. These describe software and hardware components in further detail. In the implementation phase, the low level requirements will be directly linked to parts of the system, i.e., source code or software architecture artifacts.

Derived Requirements The class of *derived requirements* contains all specifications that do *not* stem from any design decision and thus cannot be traced backwards the requirements hierarchy. They are, however, essentially linked towards lower level requirements. As an example, consider the use of a specific scheduler (derived requirement), which dictates the least possible sampling rate (low level requirement).

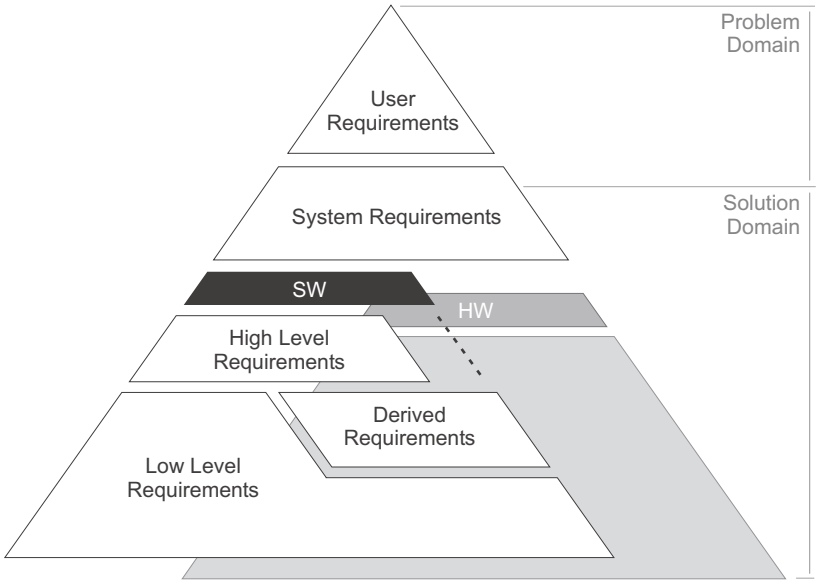

Figure 2. The requirements pyramid

2.2 Assessment of Requirements

Requirements present the *necessary* conditions for quality and safety. Achievement, and thus the *sufficient* conditions thereof, must be checked in all phases of systems design. Therefore several principal processes of assessment are linked to requirements, which are detailed next. An overview of all activities in the development process is given in Figure 3.

Validation In avionics, [3] §7 defines the process of *validation* as follows:

> Validation of requirements and specific assumptions is the process of ensuring that the specified requirements are sufficiently correct and complete so that the product will meet applicable airworthiness requirements. Validation is a combination of objective and subjective processes.

Thus, in Figure 3, the arrows labeled "validation" are drawn backwards to preceding phases. In each phase, all requirements must be validated against the requirements of the previous phases before a project can proceed.

Traceability The key to validation of requirements is *traceability* [7], i.e., the existence of links between requirements, with the exception of derived requirements, as laid out in Section 2.1. Traceability enables later validation of the entire design and guarantees correspondence of the customer's high level requirements down to implementation details. DO-178B / ED-12B [4] §5.1.3 states this as follows:

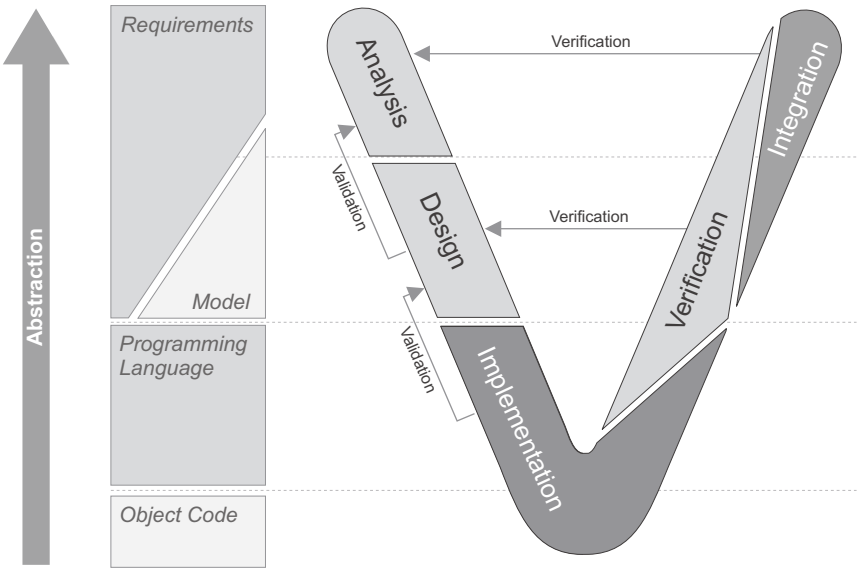

Figure 3. The V-Model

> Each system requirement allocated to software should be traceable to one or more software high-level requirements.

Verification While validation and traceability are only concerned with requirements themselves, *verification* links requirements and implementation. It is the process of assessing correctness of the implementation according to given requirements derived during analysis phase [3] §8.

Because of the complexity of requirements involved in large scale systems design, this calls for tool support (cf. [8]). While this is available using tools like Telelogic's Doors and Rhapsody, and Mathworks' MATLAB, *qualification* as required by DO-178B / ED-12B restricts the set of tools applicable in a DO-178B / ED-12B conforming process.

Qualification of a tool is needed when processes of DO-178B / ED-12B are eliminated, reduced or automated by the use of a software tool without its output being verified. Only deterministic tools may be qualified, that is, tools which produce the same output for the same input data when operating in the same environment. The tool qualification process may be applied either to a single tool or to a collection of tools.

Formal Methods Proper *formal verification* requires the use of formal methods, both in requirements specification and in the process of verification itself. Further, [3] states:

> Any attempt to justify that a complex or highly integrated system is sufficiently error free, solely by means of testing, quickly becomes impractical as the system complexity increases.

DO-178B / ED-12B §12.3.1 explicitly states that formal methods may be applied to software requirements that (a) are safety-related, (b) can be defined by discrete mathematics, and (c) involve complex behavior, such as concurrency, distributed processing, redundancy management, and synchronization.

Current considerations for the successor of DO-178B / ED-12B take further formal methods into account. This means that future avionic programs can and/or must benefit from the power of mathematical techniques for validation and verification.

Nevertheless, there is a certain gap in practical applicability. In general, the benefits of formal methods are well acknowledged in avionics industry and several ongoing projects work on the introduction of related tools. From a system engineers perspective, however, there is a hurdle of fear to be overcome. The anxiety is twofold: First, a loss of control because of the involved complexity and lack of understanding is feared. Second, costs have not been fully understood, not even investigated, yet. Therefore, acceptance of formal methods is still lacking in many areas.

3 Categorizing Requirements

In the following we try to establish a sensible taxonomy for requirements found in real avionics projects. We also list possibilities for formal modeling and verification of the respective requirements. These listings will emphasize some gaps between the often idealized academic view and industrial requirements found in our projects. The examples of requirements are taken from an assessment of six recent projects conducted by Diehl Aerospace:

- The Onboard Airport Navigation System (OANS) for the A380
- The Doors and Slides Control System (DSCS) for the A350
- The Doors Control and Management Unit (DCMU) for the A380
- The Smart MultiFunction Display for the NH90 (SMD88)
- The Loader Software for the A400M
- The Display System of for Sikorsky S76

To avoid issues with intellectual property, however, in the given examples we have replaced the original names referring to these projects by *MODULE*, *SYSTEM*, etc.

3.1 Towards a Taxonomy

Obtaining a common taxonomy has proved to be very difficult. Even in the projects analyzed there were inconsistencies and heterogeneous categories. We decided to choose the main categories proposed by [9] and consider all other categories as sub-categories that can be assigned to one main category. The main categories are:

- Functional
  - behavior of the system
  - inputs, outputs and the functions it provides to the user
- Non-Functional
  - express attributes of the system

  - attributes of the environment
  - usability
  - reliability
  - performance
  - supportability
- Design Constraints
  - impose limits on the design of the systems
  - do not affect the external behavior of the system
  - must be fulfilled to meet technical, business or contractual obligations

To contrast some current trends, we will focus on non-functional requirements and design constraints in this document. Functional properties are already well supported by several modeling toolkits. The verification then focuses on simulation-based techniques, but formal methods like model checking [10] are in use as well.

Our focus on non-functional properties is also due to the fact that non-functional requirements are more generic than application-specific functional properties. Additionally, the ratio between functional and non-functional requirements in the programs under consideration was significantly towards the non-functional requirements (approximately 70 percent).

3.2 Non-Functional Requirements

Non-functional requirements are also known as *qualities* of a system. We define *non-functional requirements* as the set of properties specifying in detail *how* to perform the intended functionality, in contrast to *functional properties* that state *what* the system under scrutiny shall do.

Safety Requirements We refer to *safety* as described in [3]:

> Safety is defined as the state in which risk is lower than the boundary risk. The boundary risk is the upper limit of the acceptable risk. It is specific for a technical process or state.

Safety requirements should be determined by conducting a functional hazard assessment consistent with the processes in [11]. Safety can be considered as the most important aspect in avionics systems. Even though closely related, the *security* aspect currently is under consideration, but has less impact. This will change due to the fact that a system that is not secure cannot be safe (due to malicious persons). We distinguish *qualitative* and *quantitative safety*, as defined by [6]:

> Qualitative Safety is the compliance with requirements associated with hazard reduction principles, (common cause failure avoidance, requirement for segregation or fail safe at item of equipment level, requirements for particular type of function, monitoring, type of failure detection, requirement associated with components, requirement of Development Assurance Level, etc.).
>
> Quantitative Safety is the compliance with requirements for occurrence rate associated to particular functioning mode or to particular mal-functioning repercussions (failure conditions at item of equipment level).

We consider the following examples as representatives of safety requirements typically occurring in our projects:

Req 1 *Neither run-time errors, nor non-deterministic constructs the consequence of which includes processor halt, nor data corruption, nor security breaches, shall remain in the software.*

Req 2 *The code in use shall assure that no undesired or unscheduled events can be generated by the software itself.*

Formal correctness according to such requirements may be shown through static analysis [12] or model checking [10]. Both approaches may be fully automated, but require specifications to be expressed using specific logics and implementations to be translated or rewritten. To make the specification accessible to the system engineer, approaches such as SALT [13] may be followed. Recently, software model checking [14,15,16] has emerged as a sub-discipline of model checking, where implementations provided as C or Java code may be checked directly. Specifications may be given within the source code, using assertions and dedicated labels. The effective applicability of model checking depends upon the expressibility and availability of requirements in the specification dialect in use. We will, in the following, refer to model checking as one possible approach in several requirement categories, but note the dependency on usable formal specifications.

At this point it should be noted that the term *safety* is used for a specific class of specifications in this community. While at an abstract level qualitative safety may be expressed using *safety properties* (as used in formal verification), this does not hold for all examples of requirements listed here, as seen below.

Further formal and semi-formal techniques include automated and interactive theorem proving [17,18], Simulation [19], and Testing [20,21]. Most notably, both simulation and testing can be performed without any explicit specifications at hand and may thus see frequent use in development processes; but conversely, guarantees of correctness are a lot harder to obtain there.

In all of the above listed approaches, scalability to large scale systems is still an issue. Furthermore, purchaser and supplier must establish a level of trust that verification has been applied [22].

Req 3 *The safety relevant items/functions should provide adequate isolation; i.e., failure of one component/sub-function shall not cause a failure of another one.*

Req 4 *It shall be demonstrated where practical that all possible combinations of input signals independent of their sequence will not lead to abnormal system operation or status indication.*

Requirements describing interaction between components may be formally modeled using interface automata [23], or in certain cases also using type- and effect systems [24]. Whereas the latter has seen frequent practical applications [25], interface automata have not been widely adopted in industry thus far.

Req 5 *Inadvertent activation of the MODULE shall be less than* 1×10^{-5}.

Req 6 *The SYSTEM equipment shall be designed to minimize the potential for human errors that would significantly reduce safety.*

Whenever requirements do not fully prohibit errors, but instead constrain error rates by certain bounds, probabilistic models [26,27] and failure mode and effect analysis (FMEA) [28] is called for. This may be combined with model checking [29,30].

Req 7 *Special care shall be taken by the supplier and the purchaser to avoid display of ambiguous and/or meaningless information and messages.*

Even though clearly a safety requirement, the lack of metrics for such a requirement make the application of formal methods impossible in Req. 7.

Reliability Requirements Reliability is the probability that an item will perform a required function under specified conditions, without failure, for a specified period of time [5].

Req 8 *The possibility of common mode faults that significantly reduce the reliability should be avoided.*

Req 9 *The reliability of monitoring functions shall be better (at least one order of magnitude) than the reliability of the corresponding monitored systems.*

The aspect of *availability* is closely tied to reliability. Availability is the "probability that an item is in a functioning state at a given point in time [6]" or the "Continuity of function [3] §5.2.1". Since some aircraft systems are required to perform a safe landing (e.g., the primary flight display in the cockpit) they have very high availability requirements. Example requirements for availability are:

Req 10 *The equipment should continue to operate correctly and continue to meet the safety requirements when subjected to several simultaneous fault conditions.*

Req 11 *Total loss of the MODULE functions shall be less than* 1×10^{-6}.

Both, reliability and availability requirements may be modeled and checked using FMEA techniques and probabilistic systems, as detailed for Req. 5 and 6. It shall be noted, however, that for software systems numbers analogous to MTBF (mean time before failure), which constitute the core of assessment in hardware systems, have not been established yet. Further, probabilistic reasoning generally is applied on abstract models, and not on an effective implementation.

Performance Requirements Performance requirements define attributes of the function or system that make it useful to the aircraft and the customer. In addition to defining the type of performance expected, performance requirements include function specifics such as accuracy, fidelity, range, resolution, speed and response time [3] §5.2.2.3. This means that performance requirements do not cover only the aspect of processing speed, but also consider operational aspects and usability. Nevertheless, the worst case execution time (WCET) is often assessed during requirement analysis and has to demonstrated on the final product.

Req 12 *If the mechanism has no strictly predictable time behavior, as e.g., main loops applying polling mechanism, additional design precautions and verification measures shall be taken to fulfill the real time requirements of the system.*

It should be noted that in real-time systems *time* likely affect proper function, and thus must also be considered to be a *functional requirement* in some cases:

Req 13 *The maximum response time between MODULE commands available on NETWORK receiver and availability on NETWORK transmitter shall not exceed 50ms.*

Req 14 *The MODULE shall draw the reference format in a maximum time of 20 ms.*

Execution time analysis has traditionally been based on informal testing of the system under scrutiny on the effective target platform or using a simulator of the platform [31,32]. Formalizations of testing based approaches are presented in [33]. Safe upper bounds on WCET may be computed using static analysis [34].

Examples of performance requirements not related to execution time are:

Req 15 *The SYSTEM shall ensure that A/C position accuracy is not degraded by more than 0.5 m.*

Req 16 *The MODULE shall process only a single operation at a time.*

Again, probabilistic models and (probabilistic) model checking may be applied. As above, the translation of Req. 15 and 16 to specifications usable in model checking may be difficult and is very specific to the techniques used in the implementation. If it is applicable, however, the specifications used in model checking may become part of the software model. This would cater for support within the software engineering process, which is essential in cases where timing behavior affects proper function (see above).

Physical and Installation Requirements We consider two sub-categories of physical requirements: *environmental* and *equipment specific*. The former deal with the location and surroundings of the avionics system, which is located in a special room called *avionics bay*. This room has an air conditioning system available that produces an optimal climate for electronic components. In case the air conditioning system fails, the electronic equipment has to perform the functionalities for a certain amount of time without degrading. Another problem arises when the air conditioning is powered down and the aircraft is parked in a hot location. Thus temperatures in the avionics bay can climb up to $80°C$. Even in this situation the equipment has to perform its tasks after power up without problems.

One of the issues coming up recently in avionics is the *single event upset* (SEU). With a higher integration of electronic circuits the probability of a neutron hitting a memory cell has increased. Especially for aircrafts flying at high altitudes this has become a real issue. To guarantee safe operations some measures have to be taken.

Req 17 *The hardware and software implementation solutions shall consider the possibility of atmospheric radiation effect. E.g., SEU (single event upset) and MBU (multiple*

bit upset, specifically MBUs leading to single word multiple upset) due to particle environment (radiations as for example: neutrons, protons, heavy ions, etc.) at high flight altitude (see also ABD0100.1.2 §4, also applicable for MBU).

Both at software and hardware level (using models of the processing units), model checking and other formal methods listed for safety requirements may be applied. Specifications, however, will be highly involved and must be tailored towards each implementation. A more generic method would thus be desirable.

Req 18 *Each equipment of the SYSTEM shall be compliant with the conditions specified in document RTCA/DO-160 §13 "Fungus Resistance" with category depending on the component's installation location.*

Equipment specific requirements deal with all physical onboard pieces. Each piece of equipment adds up to the total weight of an aircraft. The airlines demand an efficient fleet and therefore the aircraft manufacturers try to minimize weight and power consumption on every single component.

Req 19 *The maximum weight of the complete SYSTEM shall be less than 35 kg.*

Req 20 *The maximum power consumption of the system/equipment shall be 80 VA.*

Mathematical modeling of Req. 18–20 may be based on computer aided design (CAD) tools, possibly with specific annotations. Statistic analysis of the modules in each design then yield the desired numbers. This technique is referred to as *computer aided engineering* (CAE) and tool support is widely available.

Maintainability Requirements In [5] *maintainability* is defined as follows:

> Maintainability is the capability of the [...] product to be modified. Modifications may include corrections, improvements or adaptions of the software to changes in environment, and in requirements and functional specifications.

This category includes scheduled and unscheduled maintenance requirements, and any links to specific safety related functions. Factors such as the percent of failure detection or the percent of fault isolation may also be important. Provisions for external test equipment signals and connections should be defined in these requirements [3] §5.2.2.5.

Req 21 *The supplier shall comply with ABD0100.1.14 and GRESS module 1.8 for Obsolescence Management requirements.*

Req 22 *Fault tolerance principles or components intrinsic reliability shall be adopted where appropriate to achieve operational reliability targets and minimizes line maintenance work.*

Req 23 *SYSTEM software in-field loading shall not exceed 15 minutes.*

At best, annotated CAD models or module lists may be used to check such requirements. Maintainability, however, essentially involves business processes and thus would require formal models of the development processes as well.

3.3 Design constraints

Design constraints can be considered as special non-functional requirements. In fact, these requirements restrict the designer in choosing their architecture. Although requirements should be implementation independent, these special requirements are often used to enforce a certain design to conform with other developments. Design constraints are usually found in the lower level requirements specifications.

Architectural Requirements Architectural requirements cope with the structure of a software module, and the way components are tied together. In the last years, the architecture of a system has become more and more important. Since most software systems are part of a larger system, interface design becomes crucial, and the chosen structure must be easily integrated into the next level of abstraction.

Req 24 *Software modularity shall be considered in order to improve the efficiency of future function evolutions.*

Req 25 *A modular programming style with clear predefined module interfaces shall be introduced in the operational software in order to layer the whole software package in a hardware dependent part and in a hardware independent part.*

Req 26 *Functional independent software parts shall be segregated in different software modules.*

Req 27 *The SYSTEM shall be able to support slight modifications with a minimum impact on the software and without needing a new architecture definition.*

Req 28 *The SYSTEM function software shall be designed to be highly re-usable and to optimize hardware/software independence. In particular, the software design shall ensure:*

- *Independence of the SYSTEM functions related to the aircraft environment (such as HMI or I/O functions).*
- *Independence of the different SYSTEM software functions between themselves to ensure efficient future evolutions of these functions.*
- *Independence of the SYSTEM software related to the hardware (for portability on PC host unit).*

Req 29 *The breakdown of the software shall be the same as the one used by the purchaser to produce the application detailed specification. To achieve this objective a procedure shall be mutually defined so that the purchaser specification integrates the supplier's wishes.*

From a formal point of view, Req. 24–29 describe syntactic properties of the implementation. Checking such specification thus is tied to the languages and modeling formalism used in the implementation. Further, metrics to measure progress and fulfillment of the requirements must be defined (cf. [35]).

Development Requirements This class of requirements is also referred to as *coding guidelines*. Most important are those requirements that restrict the usage of dynamic memory allocation. Almost all of the projects under analysis contained one of these requirements.

Req 30 *The use of pointers is allowed provided the supplier applies specific coding standard rules and review check lists to restrict and manage its use. These rules shall be agreed upon by the supplier and the purchaser, shall forbid the dynamic memory allocation, and shall be applied for the new software and C++ reused software.*

Req 31 *Features with dynamic run-time behavior shall be avoided. No dynamic objects shall be created or destroyed during run-time.*

Req 32 *The MODULE shall not use dynamic memory allocation. All the memory shall be allocated at startup. The update by copy of the pre-allocated memory is allowed but the boundaries shall be checked.*

Other development requirements are:

Req 33 *The policy for the intended use of IEEE floating point computation shall be described in the Plan for Software Aspects of Certification (PSAC) and detailed in Software Design and Code Standards.*

Req 34 *The instrumented code shall only be used for demonstration of structural coverage, timing behavior, etc. Subsequently, the target executable object code shall be compiled and linked from the non-instrumented source code. Requirements based testing shall be repeated at the same level of testing and documented for the non-instrumented software package to demonstrate equivalence of functional and runtime-behavior for both instrumented and non-instrumented code.*

Req 35 *For validation purposes, the equipment shall allow to simulate internal failure.*

An analysis of approaches towards formal verification yields three groups here: (i) Static properties of source code. Here, syntactic checks as proposed for architectural requirements apply. (ii) Dynamic properties of the implementation. Model checking and static analysis may be used, as suggested for Req. 1. (iii) Business process related. Formal methods at software level do not apply.

HW/SW Interface Requirements

Req 36 *If the status returned by a MODULE register access function call is not RESULT_OK, then an application error shall be raised.*

This reachability property is best modeled and verified using model checking tools. Even though it may involved interaction with hardware, abstract models enable checking of the combined system.

Req 37 *The software design shall not compromise the hardware failure tolerance.*

The lack of metrics makes the formal analysis of such a requirement infeasible. Establishing appropriate metrics would be highly desirable, however, and enable probabilistic modeling and analysis of such requirements.

Req 38 *If the supplier uses the cache memory of a processor, they shall demonstrate as part of the verification plan the deterministic behavior of their solution.*

Req 39 *Usage of special software dependent resources (e.g., usage of CPU-registers for special purposes or cache memory) shall be justified and mentioned within the Software Accomplishment Summary.*

Req. 38 and 39 involve parts of the business process and thus cannot be formally checked at the implementation level.

4 Conclusions

Requirements form the basis of all systems developments processes in avionics industry. The large scale systems, however, yield a vast amount of requirements that must be managed and communicated. Based on our ongoing projects, we have presented a taxonomy to categorize the occurring requirements.

Focusing on non-functional properties and design constraints, we have given a set of examples of effectively occurring requirements and tried to elaborate formal means of verifying the respective properties.

The list of requirements and possibly applicable formal methods emphasizes the gap between an idealized mathematical model and practical applicability in an industrial context. While we do acknowledge the progress in fundamental research, we also hope that our work stimulates the development of tools that can be applied in our industrial context to further improve quality and safety in airborne traffic.

References

1. Nuseibeh, B., Easterbrook, S.: Requirements engineering: A roadmap. In Finkelstein, A.C.W., ed.: The Future of Software Engineering. Companion volume to ICSE. (2000)
2. IEEE New York, NY, USA: IEEE Recommended Practice for Software Requirements Specifications. (June 1998)
3. Society of Automotive Engineers, Inc. Warrendale, PA, USA: SAE ARP 4754, Certification Considerations For Highly-Integrated Or Complex Aircraft Systems. (November 1996)
4. RTCA Inc. / EUROCAE: DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification. (December 1992)
5. International Organization for Standardization: ISO/IEC 9126-1:2001, Software engineering – Product quality – Part 1: Quality model. (2001)
6. Airbus Industries Blagnac Cedex, France: Equipment – Design – General Requirements For Suppliers. (December 1996)
7. Eide, P.L.H.: Quantification and Traceability of Requirements. Technical report, NTNU Norwegian University of Science and Technology (2005)
8. Kornecki, A.J., Hall, K., Hearn, D., Lau, H., Zalewsi, J.: Evaluation of software development tools for high assurance safety critical systems. In: HASE. (2004)

9. Leffingwell, D., Widrig, D.: Managing Software Requirements. Addison-Wesley (2003)
10. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press (1999)
11. Society of Automotive Engineers, Inc. Warrendale, PA, USA: SAE ARP 4754, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment. (December 1996)
12. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. 2^{nd} edn. Springer (2005)
13. Bauer, A., Leucker, M., Streit, J.: SALT—structured assertion language for temporal logic. In: ICFEM. (2006) 757–775
14. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Software Verification with BLAST. In: SPIN. (2003) 235–239
15. Clarke, E.M., Kroening, D., Lerda, F.: A Tool for Checking ANSI-C Programs. In: TACAS. (2004) 168–176
16. Khurshid, S., Pasareanu, C.S., Visser, W.: Generalized Symbolic Execution for Model Checking and Testing. In: TACAS. (2003) 553–568
17. Robinson, J.A., Voronkov, A., eds.: Handbook of Automated Reasoning (in 2 volumes). Elsevier and MIT Press (2001)
18. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL—A Proof Assistant for Higher-Order Logic. Springer-Verlag (2002)
19. Wall, A., Andersson, J., Norström, C.: Probabilistic simulation-based analysis of complex real-times systems. In: ISORC. (2003)
20. Tretmans, J., Brinksma, E.: TorX: Automated model-based tesing. In: ECMDSE. (2003)
21. Holzer, A., Schallhart, C., Tautschnig, M., Veith, H.: FShell: Systematic Test Case Generation for Dynamic Analysis and Measurement. In: CAV. (2008) 209–213
22. Chaki, S., Schallhart, C., Veith, H.: Verification Across Intellectual Property Boundaries. In: CAV. (2007) 82–94
23. Luca de Alfaro and Thomas A. Henzinger: Interface Automata. In: FSE. (2001) 109–120
24. Pierce, B.C.: Types and programming languages. MIT Press (2002)
25. Kühnel, C., Bauer, A., Tautschnig, M.: Compatibility and reuse in component-based systems via type and unit inference. In: SEAA. (2007) 101–108
26. Vesely, W.E., et al.: Fault tree handbook. Technical Report NUREG-0492, Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, Washington, DC (1981)
27. Kemeny, J.G., Snell, J.L.: Finite Markov Chains. Van Nostrand Reinhold, New York (1960)
28. Stamatis, D.H.: Failure Mode and Effect Analysis: FMEA from Theory to Execution. 2^{nd} edn. ASQ Quality Press (2003)
29. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: FOCS. (1985) 327–338
30. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 2.0: A tool for probabilistic model checking. In: QEST. (2004) 322–323
31. Kirner, R., Lang, R., Freiberger, G., Puschner, P.: Fully automatic worst-case execution time analysis for Matlab/Simulink models. In: ECTRS. (2002) 31–40
32. Wang, Z., Haberl, W., Kugele, S., Tautschnig, M.: Automatic Generation of SystemC Models from Component-based Designs for Early Design Validation and Performance Analysis. In: WOSP. (2008)
33. Kirner, R., Veith, H.: Formal timing analysis suite for real-time programs. Technical Report 58, Technische Universität Wien, Vienna, Austria (2005)
34. Ferdinand, C., Heckmann, R., Langenbach, M., Martin, F., Schmidt, M., Theiling, H., Thesing, S., Wilhelm, R.: Reliable and precise WCET determination for a real-life processor. In: EMSOFT. (2001) 469–485
35. Lakos, J.: Large Scale C++ Software Design. Addison-Wesley (1996)