Abstract - Incorrect requirements engineering is the source of most of the operational failures and misbehaviours of modern systems. They are often related with contributions of the applications when “accidents” occur. Proper requirements engineering is not just a matter of technical and domain knowledge, but especially a communication and formalisation activity that must be performed with extreme care and must be thoroughly review and approved by all parts. Appropriate verification, validation and correction of requirements allows the reduction of failures and unexpected situations by reducing the probability of misunderstandings and ensuring that the requirements set is complete, coherent and attainable. The requirements gathering is even more critical at a system level, namely when hardware (HW) and software (SW) are critical parts of the foreseen system. This is why we believe this practical work and the extracted lessons learnt can be of utmost importance to industrial institutions related to the system development life-cycle.

Keywords— requirements, verification, specification, complex systems verification.

I. INTRODUCTION

Requirements’ gathering is key to any project’s success. In fact, since requirements define the system and its functions, they are the project. Some “engineering culture” assumes upfront that requirements are only required for complex systems and for projects with large staffs to support the requirements management effort. However, to guarantee a common understanding of the requirements, to find defects earlier, to keep system/product definitions up to date, and to communicate changes, a formal requirements management shall be in place.

Complex systems, such as HW/SW or Application-Specific Integrated Circuits (ASIC)/Field-Programmable Gate Arrays (FPGA)/System-on-a-Chip (SoC) based systems have specific attributes that shall be guaranteed and, for this, system requirements are a major cornerstone. These systems are usually quite important in the role they play and also quite complex in terms of qualification. Verification and Validation (or qualification/certification) of such systems became one of the most important tasks to guarantee the accomplishment of the requirements.

The requirements quality/dependability plays a very important role in any development. They drive the development and shape the system, but they might introduce issues and impact the qualification/certification activities, especially in the case of safety-critical systems and more stringent domains.

This Industrial Practice intends to demonstrate the importance of near-perfect requirements engineering and generic examples of what is commonly assumed by industry.

First section, introduced the article. Section 2 provides a snapshot of existing requirements management systems. Section 3 briefly enunciates the properties of good requirements. Section 4 presents the methodology used to check the requirements. Section 5 presents some results of industrial application of the methodology. Section 6 focuses on quick and effective recommendations for solving some common problems. Finally, section 7 presents some conclusions of this work.

II. SPECIFICATIONS: WHAT ARE WE TALKING ABOUT?

Requirements and specifications are very important components in the development of any system. Requirements analysis is one of the main steps in the system development process, it is where user requirements are gathered, discussed and documented to generate the corresponding specifications. These tasks are frequently left as optional, quickly performed and left on the shelf, since engineers are keen to start designing and implementing. However, discussing requirements with the customer is essential in the development of safety-critical systems. The level of completeness and quality of the results of these
Many accidents can be traced to requirements flaws, incomplete implementation of specifications, or wrong assumptions about the requirements (interpretation). For non-safety-critical systems these flaws might not have a significant impact, but for safety-critical systems the impact can be catastrophic.

But what is the difference between requirements and specifications? A requirement is a condition needed to solve a problem or achieve an objective, while a specification is a document that specifies, in a complete, precise, verifiable manner, the requirements, design, behaviour, or other characteristics of a system, and often, the procedures for determining whether these provisions have been satisfied. So a specification is not just a set of requirements but also technical information about specific design aspects. Specifically, requirements specification is a document that specifies the requirements for a given system or component. It might include functional, performance, interface, design requirements, and development standards. This is the definition and distinction between requirements and specifications used throughout this article, although other distinctions (e.g. [6] and [7]) exist and are a bit different, we do not intend in this work to discuss definitions.

The work related to this study intended to verify specifications by analysing the requirements that compose them, and the complementary information. Verification of specifications or individual requirements is being applied commonly by industry in the same way, taking advantage of domain knowledge that contributes to distinguish between the different verifications in different industrial areas.

The System/Software engineering world today is extremely diversified. Ranging from very mature expert companies that develop multidisciplinary systems to small software/domain houses with limited or no knowledge about systems engineering, and consequently about requirements engineering. Thus, from our industrial experience we are used to deal with customers that do not use requirements at all to some others that consider requirements as the “Bible” for any task to be performed. In terms of format, we encounter several levels of textual requirements, formatted and not formatted, mixed sources (e-mails, documents, drawings, databases, persons), formally defined and specified in tools and diagrams, some of which can be converted into design and code, some others can be directly and easily mapped into subsequent phases of the system development life-cycle.

III. Specifications: The way they should be written

As we have seen, specifications encompass not only requirements but other contextual and technical information. Independently from the way they should be written (text, drawing or diagram, formal language, model-based, database, etc) one should be very careful and follow a few essential rules.

Writing good specifications can be considered an art, complementary to the science behind it. Several steps contribute to accomplish such a task with quality [1]:

- Negotiate a common understanding (to avoid ambiguities);
- Ask questions, the right questions (to define the global properties of the product);
- Make sure the right resources are involved (both domain and technical);
- Consider alternatives (explore possibilities);
- Properly define functions, attributes, constraints and limitations, priorities and expectations of the product;
- Test/Validate the requirements;
- Guarantee that the requirements are well written, a good reference is SMART requirements [2] (Specific, Measurable, Attainable, Realisable and Traceable).

We can conclude the both communication and technical skills are important for defining good requirements, although they are often written by technical personnel without the required communication skills.

A common flaw on industrial requirements is the lack of specification of what the system should not do, it is as important as what the system must do (for safety purposes mainly). These extra requirements are important to guarantee the system safety in any situation (normal or abnormal).

Moreover, the completeness property also depends on the thorough system and domain knowledge, highlighting the multidisciplinary nature of the requirements engineering discipline.

Specifications should be built using the available resources, methods and tools to ensure they are simple, precise, complete and clear. These resources include standards, guidelines, modelling languages (e.g. UML) or formal methods, but also guarantee traceability of each requirement along the project lifecycle. Traceabilities are excellent tools but there is no consensus about what should be covered by them, this
is a case by case decision on what to trace and up to what level it must be done.

IV. Specifications: How can REAL Specs be verified? (Proposed Methodology)

Section 3 globally described the way specifications should be written and some important tasks to ensure they are accurate, complete and not causing misunderstandings.

Now we proceed to present a simple way to verify the specifications. This methodology is meant to be easily applicable by any industry, and shall act as a generic checklist to guide engineers in improving the quality of the specifications.

The specifications verification activity, for complex HW/SW systems shall verify the following list of properties:

- conformance with applicable standards;
- adequacy, completeness and consistency of the set of requirements;
- selected and complete traceability of the requirements (HW/SW, System/HW, System/SW);
- correctness of the requirements versus system requirements. Ensure all state transitions, data and control flows, interfaces, and data formats are considered. Ensure that the requirements are in conformance with all applicable documents;
- consistency of requirements documentation. Ensure all requirements are documented in a detailed way. Ensure that HW and SW interactions are properly described and according to system requirements;
- completeness of the requirements. Ensure that the requirements include all assumptions and limitations, including the functional and performance properties, as well as the product quality requirements;
- readability of the requirements. Ensure a clear and consistent structure of the specifications. Ensure that the documentation is easy to read, oriented to the target audience, and that all the necessary information for understanding the requirements is described or properly referenced;
- testability of requirements. Ensure that the acceptance criterion of the requirements exists and is quantifiable. Ensure that each requirement is testable and objective. Guarantee that there is no ambiguity;

- viability of producing and architecture with the specifications. Ensure that the set of requirements is implementable (HW and/or SW);
- existence of dependability and safety requirements according to the system criticality and the environment where the system is used;

All these properties shall be checked by skilled personnel keeping in mind the SMART requirements [2] rules that must always be part of the checks performed. The previous properties shall be applied to the following list of items (Table I).

### Table I: Input Items for the verification activity

<table>
<thead>
<tr>
<th>Input item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Requirements allocated to product</td>
<td>List of the System Requirements that are specifically applicable to the product under study. Contains the stating of the purpose, functionalities and constraints of the system.</td>
</tr>
<tr>
<td>Requirements Specification</td>
<td>List of the Requirements designed for the product objectified by the lifecycle. States the purpose, the functions that shall be implemented and constraints to the design. It also contains a description of the logical model and should address the following types of requirements: - Functional, Performance, Operational, Resource, Quality, Safety and dependability, Testability, Security, Procurement, Installation, Human factor and Verification and validation requirements; - Design, implementation and test constraints;</td>
</tr>
<tr>
<td>Software-Hardware interface requirements</td>
<td>Requirements list describing interfaces between hardware devices. Specification of logical characteristics of each interface between the software and the hardware components.</td>
</tr>
<tr>
<td>Interface Control Documents</td>
<td>Specification of the external interfaces. Identification of the items participating in the interface, both the control and data flows for each interface.</td>
</tr>
</tbody>
</table>
The payload of a satellite is only functional, several types of requirements have been identified:

- Performance
- Timing
- Interfaces
- Reliability
- Redundancy

The issues found through the requirements verification activity have been reported in the form of RIDs (Review Item Discrepancies). The RIDs have been categorised according to two main factors:

- **Problem Type** – The nature of the discrepancy;
- **Severity** – The impact that it has on the system.

A total of 26 discrepancies have been identified during the requirements verification activity.

<table>
<thead>
<tr>
<th>Severity</th>
<th>#</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>9</td>
<td>34.6</td>
</tr>
<tr>
<td>Minor</td>
<td>15</td>
<td>57.7</td>
</tr>
<tr>
<td>Major</td>
<td>2</td>
<td>7.7</td>
</tr>
<tr>
<td>TOTAL</td>
<td>26</td>
<td>100</td>
</tr>
</tbody>
</table>

The same practice has already been used in other case studies, but for software only systems, not FPGA systems. A comparison between the results obtained in such distinct domains has been performed, in order to assess if one could identify any parallelisms between the development processes and any particular differences related to requirements verification. One shall note that both systems under comparison have been developed by the same type of organizations, with equivalent level of maturity and experience.

Several on-board SW module parts of a satellite payload have been used as case studies for comparison. A total of over 850 requirements have been verified. The following tables provide a general overview of the severity and discrepancy types collected.

<table>
<thead>
<tr>
<th>Type</th>
<th>#</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Consistency</td>
<td>3</td>
<td>11.5</td>
</tr>
<tr>
<td>Correctness</td>
<td>1</td>
<td>3.9</td>
</tr>
<tr>
<td>Completeness</td>
<td>12</td>
<td>46.2</td>
</tr>
<tr>
<td>Internal Consistency</td>
<td>6</td>
<td>23.1</td>
</tr>
<tr>
<td>Technical Feasibility</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Readability and Maintainability</td>
<td>4</td>
<td>15.4</td>
</tr>
<tr>
<td>TOTAL</td>
<td>26</td>
<td>100</td>
</tr>
</tbody>
</table>
While the severity distribution is similar in both situations, the problem type distribution is rather divergent. This highlights the different nature of products, the processes by which they are developed and the different industries that develop them.

The percentage of traceable requirements to the higher-level requirements (also denominated as “user requirements”) has also been compared between both case studies. While the FPGAs case study presented 31,4% of not traceable requirements, only 23,2% of the SW modules case study requirements have been considered as not traceable. This demonstrates a higher focus on traceability from the SW development processes wrt the FPGAs development process.

C. Effort Analysis

This case study has been performed with high schedule and effort limitations. The total effort required to perform the requirements verification activity was about 90 man-hours. Table VI provides several metrics related with the effort spent, providing valuable statistical data for further studies.

### Table IV: SW modules case studies number of discrepancies per severity

<table>
<thead>
<tr>
<th>Severity</th>
<th>#</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>38</td>
<td>26,0</td>
</tr>
<tr>
<td>Minor</td>
<td>70</td>
<td>47,9</td>
</tr>
<tr>
<td>Major</td>
<td>38</td>
<td>26,0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>146</td>
<td>100</td>
</tr>
</tbody>
</table>

### Table V: SW modules case studies number of discrepancies per problem type

<table>
<thead>
<tr>
<th>Type</th>
<th>#</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Consistency</td>
<td>65</td>
<td>44,5</td>
</tr>
<tr>
<td>Correctness</td>
<td>26</td>
<td>17,8</td>
</tr>
<tr>
<td>Completeness</td>
<td>37</td>
<td>25,3</td>
</tr>
<tr>
<td>Internal Consistency</td>
<td>5</td>
<td>3,4</td>
</tr>
<tr>
<td>Technical Feasibility</td>
<td>1</td>
<td>0,7</td>
</tr>
<tr>
<td>Readability and Maintainability</td>
<td>12</td>
<td>8,2</td>
</tr>
<tr>
<td>TOTAL</td>
<td>146</td>
<td>100</td>
</tr>
</tbody>
</table>

A comparison between the effort metrics of this case study and the SW case studies already referred in section 5.2 has been performed. Once again, the aim is to assess any similarities between the efforts required to perform the same requirements verification activity for different products. Table VII provides the same effort metrics presented in Table VI, but for the SW case studies already used as reference (section 5.2).

### Table VII: SW case studies effort metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements/hour</td>
<td>1.31</td>
</tr>
<tr>
<td>RIDs/hour</td>
<td>0.22</td>
</tr>
<tr>
<td>Number of pages/hour</td>
<td>0.73</td>
</tr>
</tbody>
</table>

As can be observed, the effort metrics are strongly correlated, which allows concluding that similar ratios may be used to estimate the necessary effort to perform a requirements verification activity for SW and Complex Systems such as FPGAs.

VI.  RECOMMENDATIONS FOR GOOD REQUIREMENTS ENGINEERING

As can be observed in Table II and Table III, RIDs have been identified in five of the six problem type categories defined for this analysis. The most common problems are related with “Completeness”. These discrepancies will degenerate in lack of information at implementation phase, leaving the design engineer with the responsibility to decide over functional aspects, which may result in unwanted and potentially erroneous features.

Although most of the discrepancies have been categorised as “Minor”, there is a high probability that these may become “Major” in case they are undetected until the implementation phase.

In order to mitigate issues in the requirements engineering, a list of common causes has been identified:

- Requirements are not atomic, presenting several functionalities within the same requirement;
- Requirements written using informal and ambiguous language, resulting in uncertain function implementations;
- Requirements are not unique, resulting in the repetition of the same functionality in several requirements, with considerable specification differences;
- Requirements are not specified with the same level of detail;
- Lack of requirements focusing on performance, robustness and dependability, resulting in an unpredictable system behaviour in abnormal situations;
- Requirements not traceable to higher level specification, hardening the ability to validate the end product with the initial objectives.
Several recommendations can be proposed to improve the completeness and correctness of the requirements:

- Requirements engineering training, focusing on:
  - SMART requirements;
  - Non-functional requirements importance;
- Language training (formal requirements);
- Traceability tools training;
- Standards and templates and reuse strategies;
- Modelling tools training (e.g. UML or SysML);
- Automated or semi-automated verifications (e.g. checklists, formal methods).

VII. CONCLUSIONS

The results from the analysis performed on this case study are in-line with the industry common understandings: requirements engineering is one of the main sources of issues in a system development process. The need to improve this activity is vital. Although not performed in this case study, several studies have assessed the impact that requirements issues have in later development stages. These studies conclude that, from all the issues found at testing phase, those that are traced back to requirements are the ones which require more effort to fix, since one requirement issue often translates into multiple issues at later stages. The following figure (presented in [3] and [4]) represents the relative number of defects in the different development phases and the savings obtained by early defects detection and correction.

Defects generated during the requirements and specifications stage may lead to errors in the design stage. When these errors are discovered at testing phase, engineers must revisit the requirements and specifications to fix the problem. This leads not only to more time spent in the fixes but also the possibility of other requirements and specifications being wrong (masking).

The effects of lack of quality in the requirements are transversal to any engineering project. Therefore, it is essential to improve requirements engineering and complement it with verification activities. An earlier problem detection and correction is essential to save unnecessary effort in later stages and contributes to a better quality end-product. Several studies (see [5]) demonstrate that the investment performed in requirements verification activities in earlier development stages is largely compensated by the savings obtained in defect corrections later on the development chain.

The results from this case study also allow concluding that requirements verification activities often applied in typical SW development projects may be successfully applied in HW/Complex Systems such as FPGAs, providing similar level of early issue detection and contributing to a better product specification.

ACKNOWLEDGEMENT

This work has been partially supported by the project CRITICAL Software Technology for an Evolutionary Partnership (CRITICAL-STEP, http://www.criticalstep.eu), Marie Curie Industry-Academia Partnerships and Pathways (IAPP) number 230672, within the context of the EU Seventh Framework Programme (FP7).

REFERENCES