SP-680
September 2010

Proceedings of the Fourth IAASS Conference

Making Safety Matter

19–21 May 2010
Huntsville, Alabama, USA

Sponsored by
Canadia Space Agency (CSA ASC)
Centre National d'Etudes Spatiales (CNES)
European Space Agency (ESA)
National Aeronautics and Space Administration (NASA)
Japan Aerospace Exploration Agency (JAXA)

European Space Agency
Agence spatiale européenne

Publication	Proceedings of the Fourth IAASS Conference 'Making Safety Matter' Huntsville, Alabama, USA (ESA SP-680, September 2010)
Edited by	H. Lacoste-Francis ESA Communications
Published and distributed by	ESA Communications ESTEC, Noordwijk, The Netherlands
Price	EUR 70
ISBN *ISSN*	978-92-9221-244-5 1609-042X
Copyright	© 2010 European Space Agency

SAFETY CRITICAL SYSTEMS CERTIFICATION: TOOL QUALIFICATION FOR HARDWARE AND SOFTWARE

Andrew J. Kornecki(1), Joseph Voelmle(2), Janusz Zalewski(2)

(1)Embry-Riddle Aeronautical University, Daytona Beach, FL 32114, USA, kornecka@erau.edu
(2)Florida Gulf Coast University, Fort Myers, FL 33965, USA, zalewski@fgcu.edu

ABSTRACT

The paper discusses issues related to certification of safety critical systems, focusing on qualification of tools for software and hardware development in avionics. The authors discuss general issues related to certification, with particular emphasis on qualification of automatic tools for software and hardware development according to DO-178B and DO-254 standards. Industry views are outlined and results of experiments on tool quality assessments are discussed.

1. INTRODUCTION

The term “certification” in computing is typically associated with certifying product, process, or personnel. Product and process certification are the most challenging in developing software for real-time safety critical systems, such as flight control and traffic control, unmanned aerial vehicles (UAV’s), road vehicles, railway interchanges, nuclear facilities, medical equipment and implanted devices, etc. These are systems that operate under strict timing requirements and may cause significant damage or loss of life, if not operating properly. Therefore, the society has to protect itself, and governments and engineering societies initiated establishing standards and guidelines for computer system developers to follow them in designing such systems in regulated industries.

Researchers and practitioners have recognized the need of conducting certification processes for such systems and a number of international workshops have been organized in the last few years to discuss related issues. The topic got particular attention since 2004, where the first major workshop has been conducted, sponsored by the National Research Council, followed by at least one each year:

- National Research Council Workshop on Software Certification and Dependability, Washington, DC, April 19-20, 2004.
- FAA and Embry-Riddle Software Tool Forum, Daytona Beach, Fla., May 18-19, 2004.
- ACM SIGSOFT/SIGART Workshop on Software Certification Management, Long Beach, Calif., November 8, 2005.
- IEEE Workshop on Innovative Techniques for Certification of Embedded Systems, San Jose, Calif., April 4, 2006.
- McMaster University International Workshop on Software Certification, Hamilton, Ont., Canada, August 26-27, 2006.
- International Council of the Aeronautical Sciences Workshop on UAV-Airworthiness, Certification and Access to the Airspace, Seville, Spain, September 24, 2007.
- ETAPS Workshop on Certification of Safety-Critical Software Controlled Systems, Budapest, Hungary, March 29, 2008.
- 2nd ETAPS Workshop on Certification of Safety-Critical Software Controlled Systems, York, UK, March 29, 2009.

The topics discussed at these workshops are extremely broad and range from low level compiler code verification to high level risk assessment. Therefore, in this paper the authors chose to focus on one particular, although important issue, of qualification of automatic tools for software and hardware development in avionics.

Most airborne systems (e.g., flight controls, avionics, or engine control) are typical examples of safety-critical, real-time systems. These systems are extremely software intensive and continue to become more complex. They often operate in environments with diverse ranges of temperature, humidity, air pressure, vibration and movement, and are subject to aging, maintenance and weather conditions. Typical characteristics required of such systems are reliability, fault tolerance, and deterministic timing behavior. The software and hardware for such systems is developed using a variety of tools that must address these issues. Appropriate tools must be selected to meet the needs of a specific project. The quality of the tool and the assurance of tool output are critical for the target system certification.

The Federal Aviation Administration has focused on identifying potential safety issues in the assessment and qualification of tools used in developing software as well as custom micro-coded complex electronic hardware components for the aircraft. Software is typically implemented on dedicated microprocessor boards. Electronic components are programmable logic devices (PLD), application specific integrated circuits (ASIC) and similar circuits used as components of programmable electronic hardware. The process of PLD programming is accomplished either by use of an external dedicated device programmer or on the circuit board via in-system programming (ISP).

The focus of the paper is to show an impact of modern complex software tools used both for software and hardware development, on creation and acceptance of certified systems in a regulated industries. While concentrating on aviation with its associated certification requirements based on the need for safe and reliable systems, many of the addressed issues might be extended to other domains with mission-critical and safety-critical constraints (such as aerospace, nuclear, medical, automotive, financial, chemical, and military).

The rest of the paper is structured as follows. Section 2 gives the general background on the subject of tools qualification, Section 3 discusses the qualification in the context of certification, Sections 4 and 5 present industry views on tool qualification, and Section 6 outlines experiments conducted by the authors. Section 7 presents conclusions.

2. TOOLS BACKGROUND

Software tools for embedded system development, including that of complex electronic hardware (CEH), are used for two different reasons: (a) development of software that runs on the processors included in the system, and (b) creation of the system hardware.

RTCA DO-178B [1], defines a software tool as: "*A computer program used to help develop, test, analyze, produce or modify another program or its documentation. Examples are an automated design tool, a compiler, test tools and modification tools.*"

The glossary of RTCA DO-254 [2] defines tool subcategories as follows: "*Design Tools - Tools whose output is part of hardware design and thus can introduce errors. For example, an ASIC router or a tool that creates a board or chip layout based on a schematic or other detailed requirement.*"

Modern software development tools have direct and growing impact on the effective and efficient development of complex, safety-critical, real-time avionics systems and consequently on the safety of the flying public. The developed avionics system software must be shown to comply with airworthiness requirements, which include functional, quality of service, and safety requirements. The development processes can be extremely complex and provide opportunities to automate the collection and documentation of evidence that the system requirements are met and that the development processes do not compromise the software and system requirements.

The need for qualified development tools and related proof of quality for the developed software increased with proliferation of model-based development (MBD) and automated code generation (ACG). The existing FAA software guidelines with regard to development tool qualification state that the tool must meet the same objectives and meet the same software assurance level as the resulting avionics software of the certified system. These guidelines do not consider the differences between development environments and the application environments of the airborne software. A major focus of the RTCA Special Committee SC205 convened to update DO-178B is to alleviate these deficiencies.

Although the business case cannot compromise the safety case, guidance to take advantage of evolving development tool capabilities may also address the business case of software development tool qualification. This could transform the growing cost and quality concerns into savings and enhanced quality and safety.

The commercial software development tools market is rather volatile and confusing to the buyer. Tool vendors, not being familiar with the FAA guidance, may claim that a tool is certifiable. It is the airborne system that is certified using appropriate guidance for both hardware (DO-254) and software (DO-178B) components. Tools can only be qualified, meaning that they can be then used to create certified artifacts without verifying their output. The tools produce artifacts in a variety of formats frequently requiring manual and error-prone translation to pass the intermediate data between the tools. The software and hardware developers face problems in an attempt to create a consistent description of the system properties. Majority of general-purpose CASE tools were created without understanding or considering the processes required by the regulated industries (such as DO-178B), practically preventing tool qualification under the current guidelines.

Several attempts have been made to create a uniform tool environment. Almost each time a new tool is released, claims are made about how its features will allow easy interface with other tools. The reality does not match such idealized picture leaving the need for plenty of gluing between the tools artifacts, i.e., in-house work by the developer to get the tools to work with one another. The elements of industry that develop software intensive systems for aviation are particularly sensitive to these issues, as the products need to be highly reliable and meet certification requirements. The goal of automation (i.e., using tools) is to develop high quality software more efficiently. By definition, a tool that has been qualified eliminates, reduces, or automates a process in the software development effort without the need that its output be verified in that development environment.

On the hardware side, likewise, by definition, a qualified design tool eliminates or reduces activities in the hardware development effort. The goal of automation, accomplished by using tools, is to develop high quality products more efficiently.

Existing guidelines defined by the FAA through advisory circular AC 20-152 [3] leave room for interpretation. It should be noted that tool qualification is only one part of the overall DO-254 and DO-178B certification process. Different qualification requirements are placed on tools depending on defined by the safety assessment Design Assurance Levels (DAL). However, regardless of the DAL, the tool qualification is not required if the tool outputs are independently assessed (DO-254) or verified (DO-178B).

In general, for the hardware certification independent assessment of the tool output will occur by analyzing any simulations that are run, any system debugging that occurs and normal verification and validation of the designed system to assure that the system meets the design requirements. Section 11.4.3 of DO-254 provides guidance on the need for independent assessment of the tool outputs: "*Using such a design tool without independent assessment of the tool's output or establishing relevant history is discouraged, as it may prove to be a task as challenging as the development of the hardware for which the tool is proposed to be used.*"

Using a tool without independent assessment of the tool outputs to assure design correctness is discouraged. The guidance also notes that the tool qualification process will be challenging and may be more difficult than the hardware design. Given that tool qualification is officially discouraged and known to be a challenging task, it is surprising to note the number of tool qualifications that occur each year.

To identify the state of the industry and current views on the software development, hardware design and verification tools market, additional questions need to be addressed, related to the current industrial practice.

3. QUALIFICATION VS. CERTIFICATION

Further explanation of the purpose and the need of tool qualification can be found in section 12.2 of DO-178B:

"The objective of the Tool Qualification is to ensure that the tool provides confidence at least equivalent to that of the process(es) eliminated, reduced or automated."

"A tool may be qualified only for use on a specific system …Use of the tool for other systems may need further qualification."

"Only those functions that are used to eliminate, reduce, or automate software life cycle process activities, and whose outputs are not verified, need be qualified."

Considering the terminology used in airborne systems development, certification and qualification are different. One certifies the system while the other may qualify a tool. Certification declares that the system or product containing the target software and/or hardware meets assurance objectives to be used in an aircraft or avionics application (according to DO-178B or DO-254, respectively). On the other hand, qualification is used to ensure that a life cycle process automated by use of the tool will result in higher or equal quality output as if the process had been performed manually. A qualification is defined only for a specific task in a specific project.

3.1 Software Aspects of Certification

The software development process consists of a series of translations between various artifacts, leading ultimately to the executable code. The goal is to accurately implement the systems requirements allocated to software without introducing faults or errors. Accurate implementation of a system assumes that the system requirements themselves are accurate and have been validated (i.e., the system requirements should be complete, correct, consistent, traceable, verifiable, and unambiguous).

The concept of building a software-intensive system by developing the structural and behavioral models of the system software is a leading theme in contemporary literature and practice. By subsequent analysis of these models, the developers can get assurance of their appropriate behavior and correct functionality; thus, provide a credible base for the final system implementation. This approach also alleviates the issue of less-than-perfect requirements, since the analysis of the models may lead to the discovery of missing, incomplete, confusing, contradicting, or incorrect requirements.

Software life cycle artifacts range from textual representation of requirements, to graphical models of the system and software structure and behavior, to algorithms represented as graphics or mathematical and logic formulas, to textual code representation, to binary version of the executable code. In the past, the translation between various artifacts was done manually. The translation relied solely on a developer's skills and ingenuity but introduced human error.

A variety of tools have been developed to assist software developers in these translation tasks. In the past, compilers and interpreters replaced manual translation of algorithmic source code into machine code. The linkers and loaders replaced manual entering of a series of zeros and ones by translating the machine code into the target executable code. The design tools with code generation capability are replacing manual writing of source code based on design algorithms, in some scenarios and certain types of systems (e.g. well-defined control systems).

Patterns are continuously being developed that expand the type and domain application of these algorithms. Experience, combined with verification activities (manual and automated), has given developers confidence that the source code is translated into its equivalent binary image. One challenge is to accept the notion that it is practical to trust similar translation on a higher level of development hierarchy: from design constructs to the source code. In an aviation environment, another challenge is to demonstrate to the certification authorities why a translator tool can be trusted in lieu of completely verifying the translator's output.

Software engineering tools, commonly known as CASE tools, provide assistance in the development of software and systems. A software engineering tool is defined as a computer program used to help develop, test, analyze, or maintain another computer program or its documentation. The current state of the art is exemplified by a variety of tools, which often support more than one process of the software development life cycle. If properly designed and used, software tools may eliminate or reduce the errors that are often introduced in software life cycle data. On the contrary, an inferior defect or improperly used tool may result in a faulty end product with potential significant impact on target system reliability and safety.

3.2 Hardware Aspects of Certification

In hardware development, logical design may be accomplished in three ways: (a) by creating a schematic diagram with a graphical computer-aided design (CAD) tool, (b) by using a text-based system to describe a design in a hardware design language (HDL), or (c) by the combination of graphical and textual methods. The initial logic entry, however it is performed, is usually not optimized. Because the initial design entry might not be optimized, dedicated algorithms are used to optimize the circuits. Once the circuits are optimized, additional algorithms are used to analyze the resulting logic equations for the purpose of synthesizing the circuit to fit the design into the PLD.

Simulation is used to verify correct operation of the circuit, often requiring the user to modify the initial design entry to correct errors. When a design can be successfully simulated to verify the correctness of its simulated behavior, it is loaded into a programming unit and used to configure the PLD. It is critical to note that after the original design entry step and any required design entry corrections performed manually by the designer, all steps are performed automatically by software tools.

The more complex programmable hardware components become the more complex and sophisticated the tools supporting development and verification of the design must be. For complex devices that can accommodate large designs, a mixture of design entry methods for different modules of a complete circuit can be used. For example, some module designs might be described using a low-level circuit description language, others might be described graphically using a symbolic schematic capture tool, while others might be described using a full-featured HDL such as VHDL or Verilog. These languages operate using variables and hardware signals in addition to sequential constructs, including a variety of concurrency constructs that specify parallel implementation reflecting the nature of digital circuits. The necessary software for these tasks is supplied either by the hardware manufacturer or a dedicated third party tool vendor.

For FPGAs, additional tools are required to support the increased complexity of the integrated circuits (IC's). The device fitting step includes: (a) mapping from basic logic gates into the FPGA logic blocks, (b) placement to select specific FPGA blocks to use, and (c) a router to allocate the wire segments to interconnect the logic blocks. With this added complexity, the tool might require a fairly long period of time (often more than several hours) to complete the design.

Software tools are critical for the implementation of CEH circuits and devices. To design any modern device, one must use a suite of sophisticated tools including (at a minimum) simulation, synthesis, and place-and-route. Such tools are typically made available by an entity external to the developer. Simulation is supported by accessible and cost-effective tools; however, place-and-route tools are tightly connected to the specific hardware silicon architecture and vendor. In the middle of this hardware development cycle is logic synthesis. The front-end of the logic synthesis problem is very complex and not specific to any silicon architecture, while the back-end stages of synthesis are architecture specific. A sophisticated technology for parsing, elaborating, and inferring conceptual logic design from code written in a hardware description language, such as VHDL, Verilog, or SystemC, facilitates both the creation of the desired digital logic circuit design and the eventual mapping into architecture-specific physical layout.

4. INDUSTRY VIEWS - SOFTWARE

To understand the industrial practice in tool use, development and qualification, an industry survey was developed with the cooperation of the FAA and NASA Langley Research Center. It was distributed in May 2002 at the FAA Software Conference, Dallas/Ft.Worth, TX, with a follow-up survey sent in the Fall of 2002. The next survey was conducted in May 2004 at the Software Tools Forum, Daytona Beach, FL, with an e-mail follow-up for the issues prioritization. The subsequent follow-up was performed in the Fall of 2004.

The 2002 paper survey collected at the FAA Software Conference included 28 responses. A much broader follow-up survey was sent in the Fall of 2002 to over 700 professionals on the FAA Software Professionals' mailing list. The survey resulted in only 14 additional responses. Such low response rate, less than 2%, has been attributed to the developers' limited experience with development tool qualification. The majority of respondents represented avionics and engine control software companies (74%) and the FAA personnel (14%). Eighty-two percent of the surveys included some information about development tools. The results of follow-up were combined with the original paper survey responses to provide the initial industry feedback.

In addition to the FAA personnel, the survey was answered by industry representatives from: Airbus, Aeronautics Corporation, Boeing, Goodrich, Green Hills, Honeywell, Patmos, Raytheon, Sikorsky, UTRC, and Verocel. Sample answers are summarized below:

- Criteria for development tool qualification are too stringent and cost prohibitive.
- The tool software is different from the resulting airborne target software, and it is used in different environment and mode of operation.
- Guidance for COTS development tool regarding their qualification and use in the DO-178B compliant development process is missing.
- There is evident need for separating the tool functionality from the platform on which the tool is running, since the platform typically is not certifiable.
- For a complex multifunctional tool, qualification is limited to selected functionality feature.
- Flexibility for tool qualification and a partial credit for some objectives would help to alleviate the stringency of the qualification process.
- Qualification of the development tool changes the subsequent verification steps and needs to be reflected in the guidelines.
- With development tools supporting Automatic Code Generation, the issue is what can be understood as the code and related objectives on code reviews.
- The MBD approach modifies the lifecycle by introducing executable specification and model checking and validation.
- There is a slightly fuzzy boundary between the requirements and design (at the early stages of requirements development, some design decisions are made due to the specific model construction).
- There is misunderstanding of the source code definition, considering notations used to express the requirements and design.
- Structural coverage and the methods for analyzing models need to be redefined.
- Guidelines for model reviews and standards for model validation need to be established.
- Reuse credit for the development tool software is too difficult to obtain.
- Formal qualification approval document following an independent tool qualification outside the certification project might help to clarify the issues (a separate TSO?).
- In such a case, a specific list of documents required for an independent development tool qualification credit needs to be identified.
- Tool upgrades clearly impact the qualification status and requalification guidelines are needed.
- Using third party qualification packages may be confusing for applicant and integrator.
- Issues of certification versus qualification and concept of qualifiable tools are sometimes misinterpreted.
- Specific qualification process for ACG technology would be needed.
- Thorough analysis of the generated code is not practical (can compilers be trusted?).
- Separate guidelines would be useful for development tool qualification (i.e., a document separate from DO-178B).
- No clear guidelines for using nonqualified development tools.
- A concern has been raised that independence may be weakened by pervasive use of development tool possibly leading to common mode errors. The proprietary nature of lessons learned concerning tool use makes it difficult for another applicant to depend on previous successes by other applicants.

The low response rate allows three interpretations to be made: (1) the representative sample of airborne software developers and project managers have not considered qualification of development tools in their work; (2) development tool qualification, in light of the current interpretation of the DO-178B, is not preferable and is rather a rarely used option; and (3) applicants are not willing to disclose information about the used development tools, since they provide a significant advantage and the information is treated as proprietary.

In addition, many respondents may have chosen not to answer the questionnaire due to their use of in-house tools. Such software is an integral component of the certification package and thus tool qualification is a internal project activity. However: (1) such tools have only little re-use impact since they are known only to a limited group within the applicant organization, (2) they are not maintained properly due to high cost and lack of dedicated resources within the applicant organization, (3) their validation and verification is limited to the small group of users, and (4) information about the tool is not available outside a small group of insiders.

5. INDUSTRY VIEWS - HARDWARE

Another survey was conducted to collect data on experiences and opinions concerning the use, development and qualification of programmable logic tools as applied to the design or verification of complex electronic hardware (FPGA, PAL, GAL, PLA, ASIC, or SoC) according to DO-254 standard. The questionnaire has been sent out, targeted towards individuals who have experience with developing or using such tools, or experience with qualifying such tools. The purpose was to gather industry and certifying authority feedback on assessment and qualification of CEH programmable logic tools.

The questionnaire was distributed first during the 2007 FAA SW&CEH Conference in New Orleans, LA, attended by over 200 participants. A special session dedicated to the CEH was attended by 54 individuals, representing industry and government organizations interested in the CEH and the application of DO-254. In addition to distributing and collecting paper copies of the questionnaire at the conference, a follow-up mailing was distributed to over 150 individuals engaged in the development of aviation software and hardware. The questionnaire was also distributed internally within several companies engaged in the design of programmable logic devices.

As a follow-up, surveys were distributed at the Programmable Logic User Group meeting in Clearwater, FL, on November 15, 2007. An external survey was posted on the web followed with an additional 266 mailings requesting response. Additionally, the link to the web-survey has been placed on the DO-254 Users Group website (http://www.do254.com). Despite the above efforts, the total number of completed responses was below 30, hardly justifying validity of statistical results. This has been a rather disappointing outcome. However, the collected results provided several interesting observations.

The majority of respondents work for avionics or engine control developers with nearly all having a bachelor or master level technical background (over 70% in electronics). Nearly all respondents had more than three years of experience, and more than half of them more than twelve years of experience.

Over 65% of the respondents' roles relevant to the CEH tools were use of the tools for development and verification of systems. A quarter of them were either managing project or acting as designated engineering representative (DER). Only one respondent had experience with actual development of tools. The respondents' primary interest was divided evenly between verification, development, hardware and concept/architecture.

The types of devices used include in order of popularity: FPGA, CPLD, ASIC, PAL, PLA and EPLD. Half of the respondents used Actel and Xilinx as the hardware vendors, with Lattice, Cypress, Quick Logic, Altera and Atmel sharing the other half. Half of the respondents used tools from Mentor Graphics and Synplify, another quarter used Synopsys, Aldec and Cadence, and the remaining quarter used other tools.

The most important criteria for the selection of tools for use in DO-254 projects, was deemed the availability of documentation, ease of qualification, previous tool use, and host platform, followed by the quality of support, tool functionality, tool vendor reputation, and previous use on airborne project. Selection of a tool for a project was based either on a limited familiarization with the demo version or on an extensive review and test in nearly equal shares. The approach of reviewing and testing the tool by training personnel and using the trial period on a smaller project seems to be prevalent.

Only a fraction (14%) of the respondents had actually experienced the effort of qualifying programmable logic tools. Over 60% stated that the quality of the guidelines and the ease of finding required information have been considered sufficient or appropriate, while the increase in workload was deemed negligible or moderate by nearly 80%. An interesting observation is that over 60% of respondents considered safety improvement as marginal to moderate and about 30% as significant. Similarly, the question about errors found in the tools may be a source for concern: no errors (11%), few and minor errors (50%), significant and numerous (17%). Despite all this, the satisfaction level regarding programmable logic tools was positive and more than nine out of ten respondents marked their satisfaction level as 4 (on a scale 1 to 5).

6. TOOL EXPERIMENTS

To explore the practical issues related to application of software tools for both software and hardware development, several experiments were conducted in the 2005-09 timeframe. The main objective of the experiments was to develop a framework for tool evaluation and collect data, which would provide an insight into the process of assessing tool quality.

6.1 Software

For software development tools two phase experiments were conducted. The preliminary experiment set the evaluation baseline, while the controlled experiment allowed us to collect experimental data.

The controlled experiment's objective was a more detailed evaluation of the selected six software design tools with automatic code generation capability. The sample included six tools from both structural (object-oriented) and functional (block-oriented) categories.

The developers were graduate software engineering students familiar with software development methodologies, software processes, and real-time design concepts. Developers shared the initial training and the final reporting. However, each of them developed the model and implemented code as an individual assignment.

The experiment consisted of two models. The first model, a simple hair dryer simulator, was to be used during the learning phase of the experiment to facilitate the learning and constitute a capstone for familiarization with the methodology, tool, and the operating environment. The second system, a simple microwave oven software simulator, was used for the actual design and data collection.

Each developer was required to keep track of engineering observations during the course of the experiment to evaluate strengths and weaknesses of the tool, the process used, and other related elements. Developers were also required to record the time spent during each process, to evaluate the effort required to develop a system while using the tool. The method employed in the initial experiment of decomposing the design models into their basic components was again used in this experiment. Additionally, the developers filled two questionnaires. The first questionnaire addressed the documentation, manuals, and support of the tool under evaluation. The second focused on the code generation capabilities of the tool.

The results of the controlled experiment included data on product size and developers' effort, subjective assessment of the tool documentation, functionality, ease of use, and general observations from the experiment.

Figure 1 illustrates some of the results of the experiment, showing the tool usability assessment measured as effort in hours. Developers' effort seemed to be related to the paradigm used. For the functional block-oriented tools the effort is, on average, less than the effort for the object-oriented tools. It is important to remember that the tools automatically generated the code, with little or no manual coding by the developers. More information of the results of experiments is included elsewhere [4].

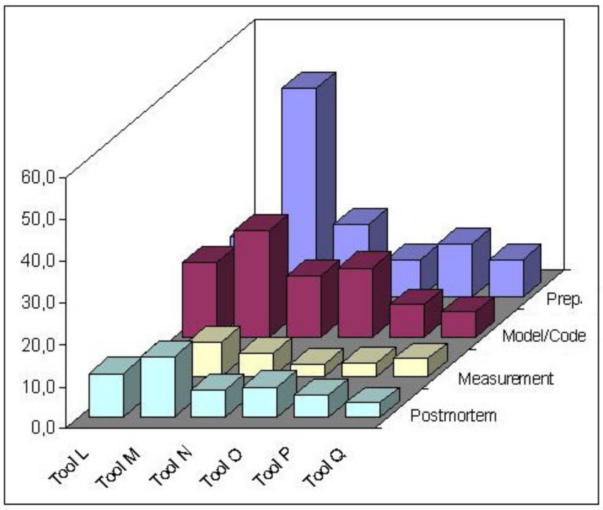

Figure 1. Usability measured as effort assessment [4]

6.2 Hardware

With experiences in tool assessment for software development, experiments on tools for hardware development were more focused on two issues: (1) establishing respective metrics and measures, and (2) testing procedures for the tools outcomes (designs developed). Since results of the latter were published elsewhere [5], herewith we only discuss the issue of measurements.

Tools from three different vendors, the market leaders in FPGA development, were used in the assessment, for a simple project to develop an FGPA based device. The quality assessment criteria were based on a previous project for software development tool assessment [5], and included: functionality, usability and efficiency. The way they were measured is illustrated in Figure 2.

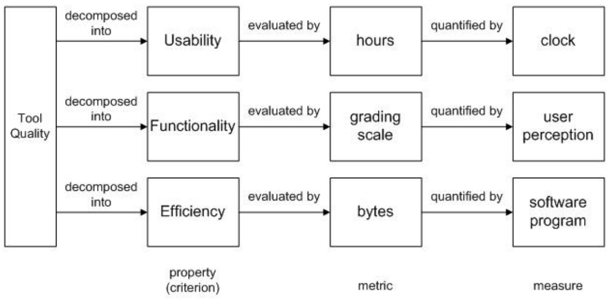

Figure 2. Criteria used for quality evaluation

Figure 3 compares the usability of all tools. The preparation for all three tools was essentially the same. The main differences among the tools were observed during code development. Since the code for the FPGA device was written during the learning phase for one of the software tools, more time was spent for it on writing the VHDL code and debugging it. One of the tools was consistently judged least usable due to its hardly accessible documentation. More information on this project is included in the FAA report [6].

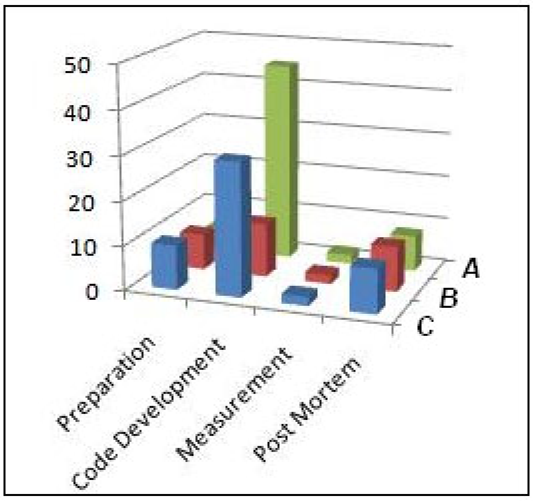

Figure 3. Comparison of usability of hardware tools

In essence, the tool quality assessment experiments, as simple as they might look, provided a significant amount of data for prospective use in defining and conducting future tool qualification processes.

7. CONCLUSION

Problems of hardware and software certification in safety critical systems, especially in avionics and aerospace, are extremely complex and difficult. The number of publications addressing respective issues, both from the academic, government and industry perspective, is growing significantly every year. Over the last 5-6 years, the authors conducted a thorough study of one specific aspect of certification: the qualification of automatic programming tools for the development of software and hardware in avionics. The results of the studies [7-8] collectively indicate that to minimize risks involved with the use of the tools in safety critical applications, the tools need to be qualified by respective independent organizations. However, methods and techniques for tool qualification have still to be developed.

8. ACKNOWLEDGEMENTS

This work was supported in part by the Aviation Airworthiness Center of Excellence under contracts DTFA0301-C-00048 and DTFACT-07-C-00010 sponsored by the Federal Aviation Administration (FAA). It was also supported in part by the National Aeronautics and Space Administration (NASA) through the University of Central Florida's Florida Space Grant Consortium. Findings contained herein are not necessarily those endorsed by the FAA or NASA.

9. REFERENCES

1. DO-178B and EUROCAE ED-12B (2001). *Software Considerations in Airborne Systems and Equipment Certification*, RTCA Inc., Washington, DC, 2001.
2. DO-254 (2000). *Design Assurance Guidance for Airborne Electronic Hardware*, RTCA Inc., Washington, DC, April 2000.
3. AC 20-152 (2005). Advisory Circular on Document RTCA/DO-254 - Design Assurance Guidance for Airborne Electronic Hardware. FAA, July 2005.
4. Kornecki A., & Zalewski J. (2005). Experimental Evaluation of Software Development Tools for Safety-Critical Real-Time Systems, *Innovations in Systems and Software Engineering – A NASA Journal*, Vol. 1, No. 2, pp. 176-188, Sept. 2005.
5. Butka, B., Zalewski, J., & Kornecki, A. (2009). Issues in Toool Qualification for Safety-Critical Hardware: What Formal Approaches Can and Cannot Do?, *Proc. SAFECOMP2009, 28th International Conference on Computer Safety, Reliability, and Security*, Hamburg, Germany, September 14-18, 2009, Springer-Verlag, Heidelberg, 2009, pp. 201-214.
6. Butka B., A. Kornecki, J. Zalewski (2010). *Qualification of Tools for Airborne Electronic Hardware*, Report DOT/FAA/AR-10/xx, FAA, Washington, DC, 2010 (in preparation).
7. Kornecki, A. & Zalewski, J. (2009). Certification of Software for Real-Time Safety-Critical Systems: State of the Art, *Innovations in Systems and Software Engineering – A NASA Journal*, Vol. 5, No. 2, pp. 149-161, June 2009.
8. Kornecki, A. & Zalewski, J. (2010). Hardware Certification for Real-Time Safety-Critical Systems: State of the Art, *Annual Reviews in Control*, Vol. 34, No. 1, pp. 163-174, 2010.