Innovations Syst Softw Eng (2005)
DOI 10.1007/s11334-005-0013-1

ORIGINAL PAPER

Andrew J. Kornecki · Janusz Zalewski

Experimental evaluation of software development tools for safety-critical real-time systems

Received: 16 April 2005 / Accepted: 1 June 2005 / Published online: 29 July 2005

Abstract Since the early years of computing, programmers, systems analysts, and software engineers have sought ways to improve development process efficiency. Software development tools are programs that help developers create other programs and automate mundane operations while bringing the level of abstraction closer to the application engineer. In practice, software development tools have been in wide use among safety-critical system developers. Typical application areas include space, aviation, automotive, nuclear, railroad, medical, and military. While their use is widespread in safety-critical systems, the tools do not always assure the safe behavior of their respective products. This study examines the assumptions, practices, and criteria for assessing software development tools for building safety-critical real-time systems. Experiments were designed for an avionics testbed and conducted on six industry-strength tools to assess their functionality, usability, efficiency, and traceability. The results shed some light on possible improvements in the tool evaluation process that can lead to potential tool qualification for safety-critical real-time systems.

Keywords Safety-critical systems · Software safety · CASE tools · Tool qualification

A.J. Kornecki
Dept. of Computer and Software Engineering
Embry Riddle Aeronautical University
600 Clyde Morris Blvd.
Daytona Beach, FL 32114, USA
Tel.: +1-368-2266678
Fax: +1-368-2266678
E-mail: kornecka@erau.edu
http://faculty.erau.edu/korn

J. Zalewski (✉)
Dept. of Computer Science
Florida Gulf Coast University
10501 FGCU Blvd.
Fort Myers, FL 33965, USA
Tel.: +1-239-5907317
Fax: +1-239-5907330
E-mail: zalewski@fgcu.edu
http://www.fgcu.edu/zalewski

1 Introduction

The objective of this paper is to study the need, state of practice, and issues in the assessment of software tools used in the development of safety-critical real-time systems, in view of recommendations for the potential tool qualification process. Although development tools are used in a variety of application domains, we concentrate on aviation systems as a representative application area. Construction of airborne and ground-based systems in the U.S. is subject to procedures outlined in various guidelines issued by the Federal Aviation Administration (FAA).

FAA document DO-178B [1] is a primary guideline for software development in certified airborne systems, but it is often referred to in other contexts related to software processes, especially for safety-critical systems [2]. Tool qualification is a supplementary process that applicants may elect to follow in the course of airborne system certification. The purpose and need for qualification is described in Sect. 12.2 of DO-178B:

> "The objective of the tool qualification process 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."

Tool qualifications are part of a type certificate, supplemental type certificate, or technical standard order approval. In the case of projects using qualified tools, two required documents—"Plan for Software Aspects of Certification (PSAC)" and "Software Accomplishment Summary (SAS)" —of the original certification project must include clear and specific references to the documents "Tool Qualification Plan" and "Tool Accomplishment Summary." The documents "Separate Tool Operational Requirements," "Tool Verification Records," and "Tool Qualification Development Data" need to be available for certification authority review. These requirements are described in Chap. 9 of FAA Notice N8110.49 [3].

The main concern regarding airborne software that may impact the development tools qualification is that the selected tool may not insert errors into the software it produces. Obviously, the tool must guarantee the chain of correctness defined by DO-178B. The qualification process must provide a means to show that the tool is predictable (i.e., deterministic). An advantage of using qualified tools is the reduction (or even elimination) of not only a time-consuming translation process but also the effort required for verification of the output product. Another concern is that in most modern applications, the software development tool must support the implementation of large projects by supporting multiuser access, configuration management, etc. This study explores the current practice of tool evaluation and development of ways to improve the evaluation process.

The paper is structured as follows. The main assumptions are presented in Sect. 2, followed by a presentation of industrial practices in Sect. 3 and a discussion of evaluation criteria in Sect. 4. Experiments and their results are presented in Sect. 5, followed by a conclusion in Sect. 6.

2 Assumptions

Three basic issues need to be considered initially in a study of any technical artifact: the subject, the context in which this subject is used, and the research method. Our subject is a set of software development tools, the processes in which they are immersed define the context, and the method describes the way in which they are evaluated.

The primary question regarding the subject concerns the identification of specific tools. The document DO-178B defines four primary processes of software development: requirements, design, coding, and integration. By adopting this sequence, we can represent the practice of using tools during the development process as follows: a requirements tool, followed by a design tool, typically with code generation capability, an integrated development environment, and the target with a real-time operating system. Analysis and testing tools also play a significant role, as illustrated in Fig. 1.

The above model covers most, if not all, of the known software development schemes, for example:

(1) High-level structural design tools (e.g., Rose RT, Rhapsody, RT Professional Studio, STOOD), based on discrete models, are used to develop software architecture in a specific graphical notation (e.g., UML). Such tools can be used directly to generate a source code framework (in C/C++, Java, Ada) for specific targets.
(2) For smaller and simpler systems the control algorithms are usually developed with continuous models (differential equations) using function/block-oriented tools (e.g., Beacon, MatrixX, Matlab/Simulink, Scade, Sildex). These tools also can generate complete source code (in C/C++, Ada), which can be directly tested.
(3) In all approaches mentioned above there is a need to maintain consistent requirements; therefore, a requirements analysis tool is typically used (e.g., DOORS, Reqtify).
(4) The code development requires thorough analysis and testing, typically supported by appropriate automatic tools (e.g., RapidRMA, TimeWiz, CodeTest, Insure++, Test-RT).

To define clearly the scope of this study, we chose *software design tools*, represented in a dashed box in Fig. 1, as the subject and focal point of the research.

A model of the tool evaluation process is necessary for developing the evaluation criteria. The framework for this process, based on the context of tool use, is shown in Fig. 2. The central part of this model is the *macroevaluation* based on the use of the tool during the design phase. However, much information on tool quality can be derived from the development of the tool itself, considered as a *metaevaluation*: evaluating the process to develop a tool. The tool vendor can provide the data for evaluation of this stage. In addition to the *macro-* and *metaevaluation*, the product developed with a particular tool can be included in the evaluation. This is called *microevaluation*, and it focuses on the level lower than the tool itself. Such a product evaluation can be based both on static code analysis and code execution. Consequently, to have the entire picture of the tool's quality, we need to do the evaluation at three different levels.

Considering the category of tools we want to evaluate, as well as the model of the evaluation process, another decision needs to be made regarding the research method: What general view of tool evaluation would one like to pursue? The four views we have identified are presented in Fig. 3, where evaluation is a companion process of development:

- Qualification view, which looks at the qualification of a development tool by applying standards and guidelines accepted and used by industry.
- Project view, which considers how well a software tool fits into the developer's organization and the specific project's cycle.
- Behavioral view, which relies on the observations and quantitative measurements of the tool's behavior during the design process.
- Taxonomy view, which concentrates on three groups of indicators (functional, related to what the user wants from the system; quality, related to how well the system fulfills its function; business, considering the rationale for using the tool) and respective measurement methods.

Since the project view is extremely broad, the behavioral view is not well developed yet, and the qualification view is limited to the very imperfect existing standards, we chose the taxonomy view as the most promising method of tool evaluation. Adopting this view requires distinguishing the most representative metrics for tool evaluation and developing their respective measures to conduct quantitative assessment [4]. The process of selecting metrics was preceded by an industry survey, which is described in the next section.

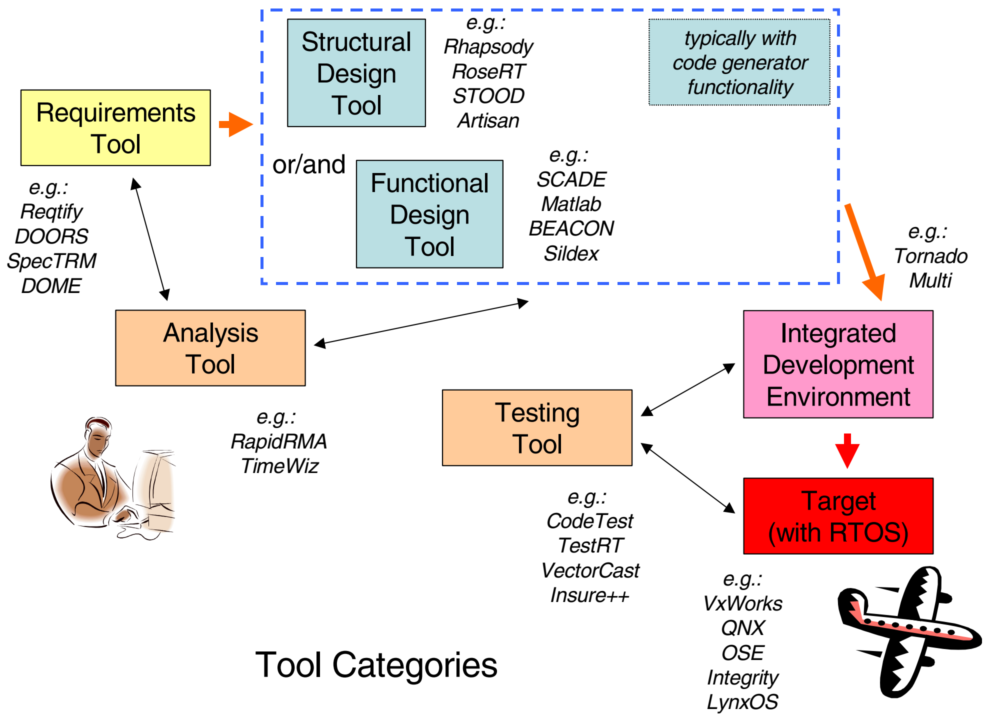
Structural Design Tool
e.g.: Rhapsody RoseRT STOOD Artisan
typically with code generator functionality
Requirements Tool
or/and
Functional Design Tool
e.g.: SCADE Matlab BEACON Sildex
e.g.: Reqtify DOORS SpecTRM DOME
e.g.: Tornado Multi
Analysis Tool
Integrated Development Environment
Testing Tool
e.g.: RapidRMA TimeWiz
Target (with RTOS)
e.g.: CodeTest TestRT VectorCast Insure++
e.g.: VxWorks QNX OSE Integrity LynxOS
Tool Categories

Fig. 1 Software tool categorization

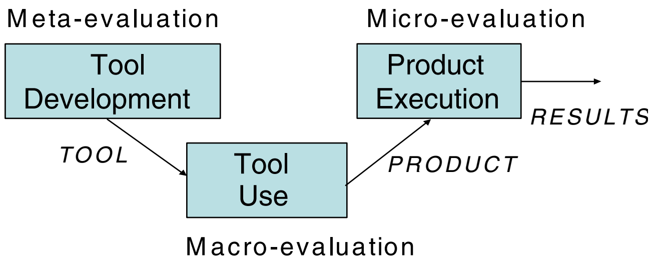

Fig. 2 Model of the tool evaluation process

3 Industry view

The objective of this step was to investigate the current state of industrial practice in tool evaluation by conducting a survey of stakeholders and other interested parties using design tools in the avionics industry. A sample of 28 respondents representing developers of avionics and engine control software (74%) and FAA personnel (14%) had been surveyed at the FAA National Software Conference, in Dallas, TX, in May 2002.

A followup via e-mail to the FAA Software Certification mailing list resulted in a rather unimpressive outcome of only 14 responses from over 500 individuals. Such organizations as Airbus, Astronautics Corporation, Boeing, Goodrich, Green Hills, Patmos, Honeywell, Raytheon, Sikorski, UTRC, and Verocel were represented. Despite a rather limited sample, the survey provided some important feedback on the evaluation and selection of software development tools, from various perspectives: applicant/developer, manager, certifying authority, and tool vendor. Figure 4 gives some insight into the tool selection process as it stands in the industry. It is interesting to note that a significant fraction of respondents selects tools based only on limited testing, review of the tool documentation, or the data as provided by the tool vendor.

Figure 5, in turn, shows the main criteria used in the selection process. From the industry perspective, the functionality and cost of a tool are the major factors in making the tool selection. Almost all of the participants cited functionality as the primary factor in their evaluation criteria. Product cost and the benefit to the company resulting from task automation were also cited by most of the participants. The third major factor was compatibility with the development platform. There seems to be a consensus that the development platform is typically selected in advance and thus cannot be influenced by new tools. It is, rather, tools that have to adapt to this stable development environment.

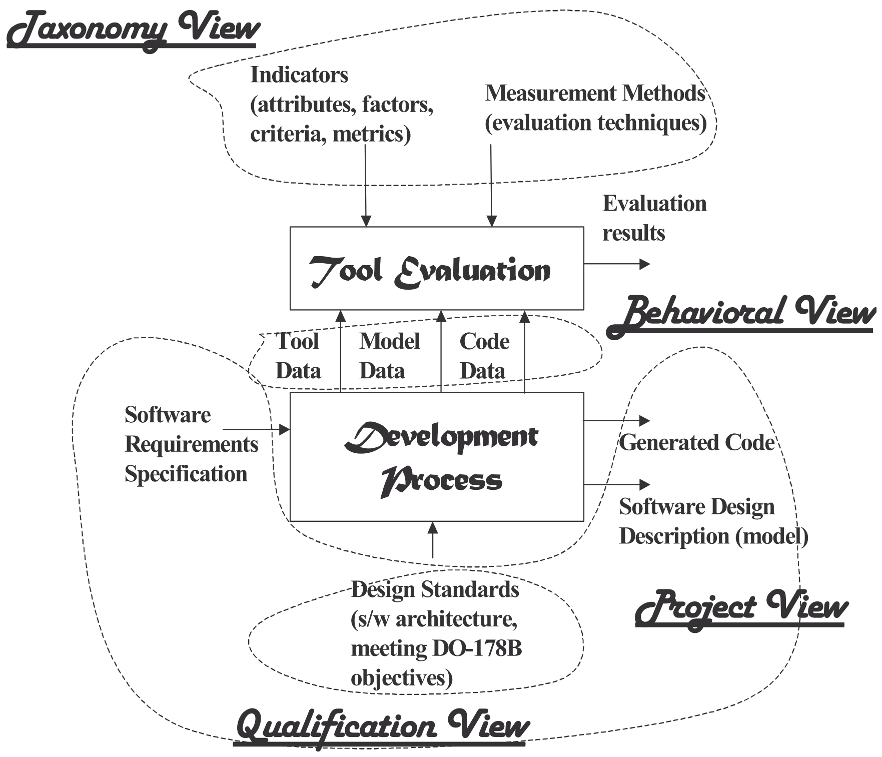
Taxonomy View
Indicators (attributes, factors, criteria, metrics)
Measurement Methods (evaluation techniques)
Evaluation results
Tool Evaluation
Behavioral View
Tool Data
Model Data
Code Data
Software Requirements Specification
Development Process
Generated Code
Software Design Description (model)
Design Standards (s/w architecture, meeting DO-178B objectives)
Project View
Qualification View

Fig. 3 Four views of tool evaluation

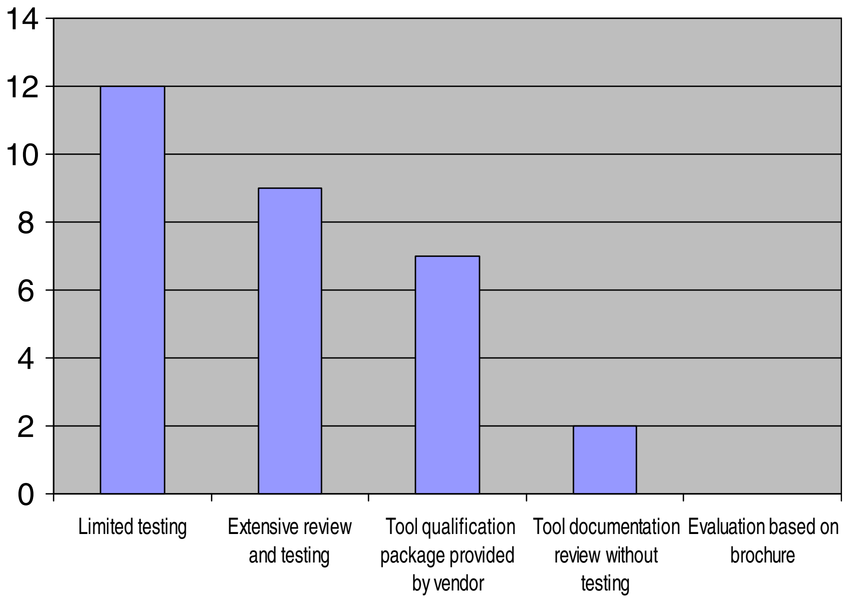

Fig. 4 Industry survey on tool selection practices

Such issues as cost, obsolescence, compatibility with existing development environments, and inadequate documentation were listed as sources of problems. Tool vendors typically do not adhere to the level of effort required for DO-178B compliance and occasionally present ungrounded claims about tool functionality and performance. Typically, tools developed in a research environment do not scale up well. An often-cited problem was inadequate training and understanding of development tools. There is some discouragement about rigor of tool qualification and a justified perception of extensive cost of qualification. Good vendor support is required to facilitate qualification of COTS

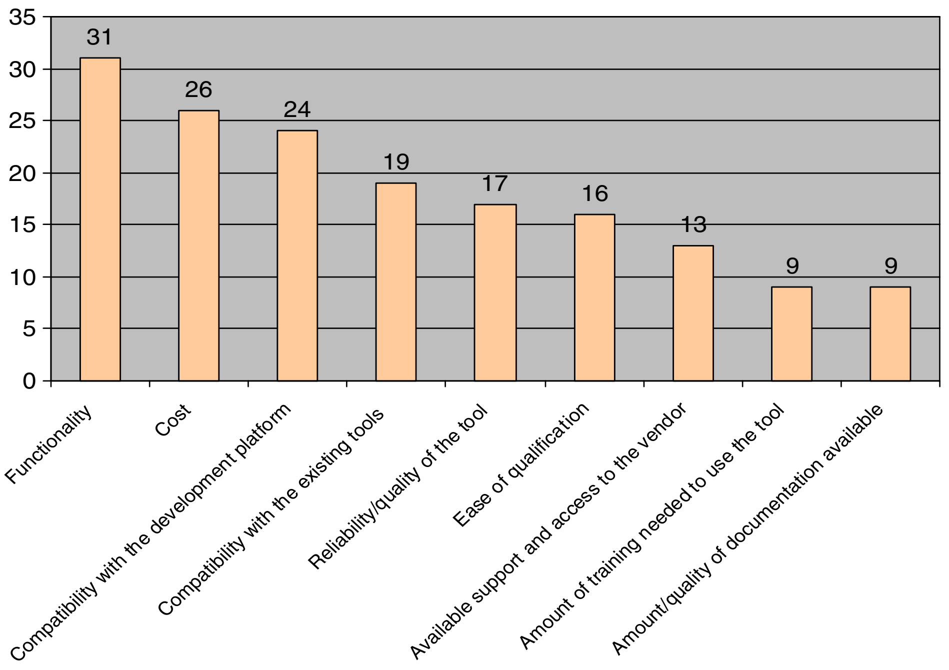

Fig. 5 Factors considered by industry when selecting development tools

development tools. The use of "alternate means" allowed in DO-178B was quoted as a potential approach to obtaining qualification credit.

4 Tool evaluation criteria

The industry survey provided some essential information on potential criteria used for tool evaluation. However, it did not give sufficient indication of what criteria would be the most important or worthwhile to use when considering a subsequent tool qualification process. This may be due to the fact that DO-178B [1] itself is not very specific on this issue. Various sections of DO-178B, however, provide some insight by referring to certain aspects of software product evaluation, for example:

– Traceability and verifiability (5.2.2)
– Consistency (5.2.2)
– Detecting modes of failure (5.2.2)
– Monitoring control flow and data flow (5.2.2)
– Complexity (5.2.2)
– Modifiability (5.2.3)
– Compliance with system requirements (6.3.2)
– Accuracy and consistency (6.3.2)

These criteria focus on the airborne system target software, and do not apply directly to software tool evaluation. However, they can be used to evaluate the tools assuming appropriate access to the tool source code and tool development documentation.

There are a few other standards and guidelines, which define criteria for software evaluation in general. The two related ISO/IEC standards [5,6] and a compatible IEEE standard [7] are very specific about software tool and software product evaluation criteria. They list six such characteristics:

• Functionality, comprising a set of attributes that bears on the existence of specific functions.
• Reliability, defined as a set of attributes that bears on the capability of software to maintain its level of performance under stated conditions for a stated period of time.
• Usability, a set of attributes that bears on the effort needed for use of the software.
• Efficiency, a set of attributes that bears on the relationship between the level of performance of software and the amount of resources used.
• Maintainability, related to a set of attributes that bears on the effort needed to make specific modifications.
• Portability, understood as a set of attributes that bears on the ability of software to be transferred from one environment to another.

Each of the above characteristics is additionally described in terms of lower-level attributes, called subcharacteristics, which are not specific, however, to safety-critical systems.

The FAA Technical Center defines an extensive set of quality factors for evaluating avionics application source code [8], from accuracy to usability, including most of the factors from the ISO/IEC standards. The report also recommends a general approach for defining metrics based on these factors, and for each factor the report points to other factors that can have an influence. These factors are identified as influencing the quality of source code and as such are not necessarily applicable to tool assessment.

Several previous tool evaluation studies give some insight into tool evaluation criteria. VTT Technical Research Center of Finland conducted research [9] to compare software development tools, evaluate them, and assess their support of the emerging object-oriented technologies for safety-critical real-time systems development. The proposed set of criteria includes a set of low-level criteria, called attributes. Each attribute, in turn, is assessed by a set of evaluation questions, 155 total for all attributes. Individual attributes are then evaluated as a percentage of positive answers, according to a certain formula. Finally, each of the six main characteristics is assessed based on the values of individual attributes, and the overall quality of the tool is assessed based on the following attributes (the weighing factors in parentheses):

- Ease of use, which involves tailoring, helpfulness, predictability, error handling, and system interface (17%).
- Power, related to tool understanding, tool leverage, tool state, and performance (10%).
- Robustness, involving consistency of operation, evolution, and fault tolerance (10%).
- Functionality, regarding correctness and methodological support (30%).
- Ease of insertion, pertaining to the learning curve and software engineering environment (13%).
- Quality of support, concerning tool history, maintenance, user's group and feedback, installation, training, and documentation (20%).

A British study [10] discusses the tools used in software development activities for safety-related systems, from risk assessment to maintenance. Tools are split into two categories, those automating a previously manually implemented technique and those introducing new concepts and techniques. The document elaborates on several questions and points of concern. The following criteria are recommended for tool evaluation:

- Ease of validation of the tool result.
- Software techniques used to develop the tool.
- Software techniques used in the tool.
- Quality system of the tool developers.
- Previous use of the tool in similar safety-related projects.

The role of each criterion is then discussed at some length. For example, ease of validation of the tool result makes sense when considered from the point of view of determinism. The idea is that the output of a tool should not be too complex to be examined or to have its validity demonstrated. The way to demonstrate validity is to show a perfect match with the original requirements. This approach would require limiting the complexity of tool output and splitting intermediary artifacts in the development of the final product, resulting in more development steps.

Results of the industry survey and review of the literature are summarized in Table 1, which includes a combined list of main tool evaluation criteria adopted from the discussed sources. On this basis, the following rationale has been developed for selecting specific evaluation criteria for the experimental study.

First of all, only technical criteria were considered suitable, as opposed to business criteria, such as cost, vendor support, ease of qualification, etc., which were eliminated up front and not even included in the table. Secondly, only criteria relevant to the design process were included, as opposed to those spanning the entire development cycle, such as maintainability, modifiability, portability, reusability, etc. In this respect, particularly important are the criteria that capture tool characteristics during the design process, as a part of the chain of processes illustrated in Fig. 1. Two specific criteria from the list are particularly relevant in this regard: *efficiency* of the generated code, which allows for conducting forward evaluation regarding the quality of code, and *traceability*, which allows backward evaluation regarding the tool's ability to maintain the right requirements. In addition, it is necessary to evaluate the tool during its operation from the perspective of the functions it provides and its ease of use. Two criteria that seem to best capture this operational tool use are *functionality* and *usability*.

It is important to note that two essential tool evaluation criteria, *reliability* and *robustness*, were not used in the experimental study for the following reasons. Reviewing the criteria analysis in Table 1, it is clear that tool *reliability* is one of the most widely considered. However, currently accepted reliability measures are based on statistical data, and collecting them, even for a single tool, would require a lengthy study, much beyond the time frame of our project. Therefore, we decided to leave it for future research. Similar reasoning stands behind eliminating *robustness* as one of the leading tool evaluation criteria. To evaluate tool robustness properly, one needs to apply a wide range of input data to the tool, which was not possible in this research due to resource limitations. Additionally, it has been noted that some sources, such as [1] and [8], are highly correlated, which weakens the significance of a specific criterion, if we base the selection on its frequency of use. Therefore, *consistency* was not considered in the experiments either.

This leaves us with four essential criteria marked in the table by asterisks. Assuming these criteria are direct metrics [11], the following specific measures to evaluate them were defined and used in the experiments:

- Usability measured as development effort (in hours).
- Functionality measured via the questionnaire (on a 0–5 point scale).
- Efficiency measured as code size (in LOC).
- Traceability measured by manual tracking (in number of defects).

5 Experiments

5.1 Experimental testbed and preliminary experiment

To take measurements on tools for the criteria defined in the previous section, an experimental testbed was built including the following components (Fig. 6):

Table 1 Selection of criteria for software development tool evaluation

Sample criteria	Industry survey	DO-178B Std	ISO/IEC Std	FAA guide	VTT study	BCS study
Consistency[a]		Yes		Yes		
Efficiency*			Yes		Power	
Functionality*	Yes		Yes		Yes	
Maintainability[a]			Yes	Yes		
Modifiability[a]		Yes		Yes		
Portability[a]			Yes	Yes		
Reliability[a]	Yes	Yes	Yes	Yes		
Robustness[a]		Yes			Yes	
Traceability*		Yes				Ease of validation
Usability*			Yes	Yes	Ease of use	
Other	Compatibility	Accuracy complexity	–	Many more	–	–

*Criteria selected
[a]Criteria eliminated

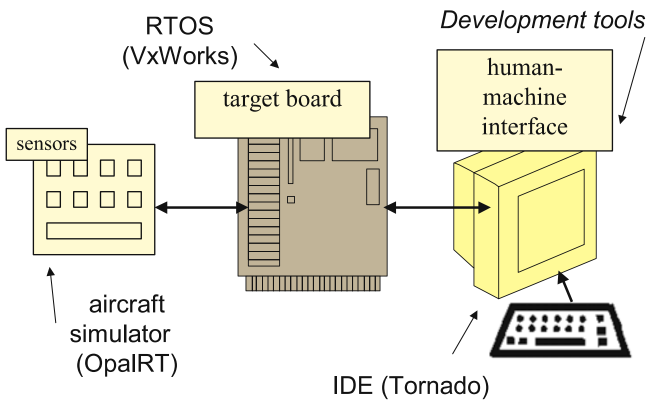

Fig. 6 Experimental testbed.

- A generic avionics application used as a model for software development (with user interface, data collection, processing, and display), for which specific design tools were used, running on a workstation;
- Hardware platform, with a standard real-time kernel, that serves as a target system for the code generated from the evaluated design tools; and
- A flight simulator, with the ability to deliver signals to the hardware platform and receive control signals from it, to make results of the research verifiable.

The evaluation experiments were conducted in two steps. First, a preliminary experiment was designed with a focus on learning and exploring capabilities of software tools used in the process of developing and implementing a safety-critical real-time project. The objective was to keep the system at the minimal complexity while concentrating on the collection of data and engineering observations that might indicate the software tool's quality. The developers collected the design artifacts from the tools' outputs (such as graphical model, automatically generated source code, and documentation) and focused on observations about the tool use. Data related to the traceability from requirements to design to code were also collected. These observations constituted the basis for conducting controlled experiments in the next step.

For the preliminary experiment, selected tool samples included four tools from both structural (object-oriented) and functional (block-oriented) categories. The project was defined as one of flight data collection from flight simulator with simple processing (averaging, timestamping) and displaying results on a terminal. Four developers were assigned an identical problem statement to use the tool for developing a program to be implemented on a real-time target in the experimental testbed.

The software was supposed to capture data packets of parameter values transmitted from the flight simulator and subsequently compute and display a moving average of selected parameters. The user interface would consist of a predefined menu of options to select 3 of over 20 parameters to be captured and the frequency of the moving average computation. The parameter values and averages would be displayed with a timestamp. Using a different tool (A, B, C, or D), each developer implemented three types of requirements: (1) timing requirement, (2) system requirements, and (3) external interface requirements.

The evaluation criterion measured in this experiment was tool usability. The data collected involved the development effort (in hours) divided into four categories: preparation, modeling and code generation, measurements, and postmortem (including report writing). A summary of results is presented in Fig. 7. Details of the software requirements and actual experimental results are discussed in [12]. The preparation phase included familiarization with the tool and the design methodology, which justified the high numbers in this category. Modeling and code generation accounted for lower values, but these varied greatly among the four selected tools.

The developers used to apply the Personal Software Process (PSP) [13] underestimated the preparation phase effort. The average planned time was 58 h vs. the actual time of 84 h. On the other hand, the developers planned for, on average, over 71 h to be dedicated to the design and coding phase. The actual average for this phase was 39 h. Automatic code generation reduced the development time on the order of 40%. The average code size was about 1.8 KLOC. The average total time spent on the project was 153 h, resulting in an efficiency of nearly 12 LOC/h. The learning curve is high, and results

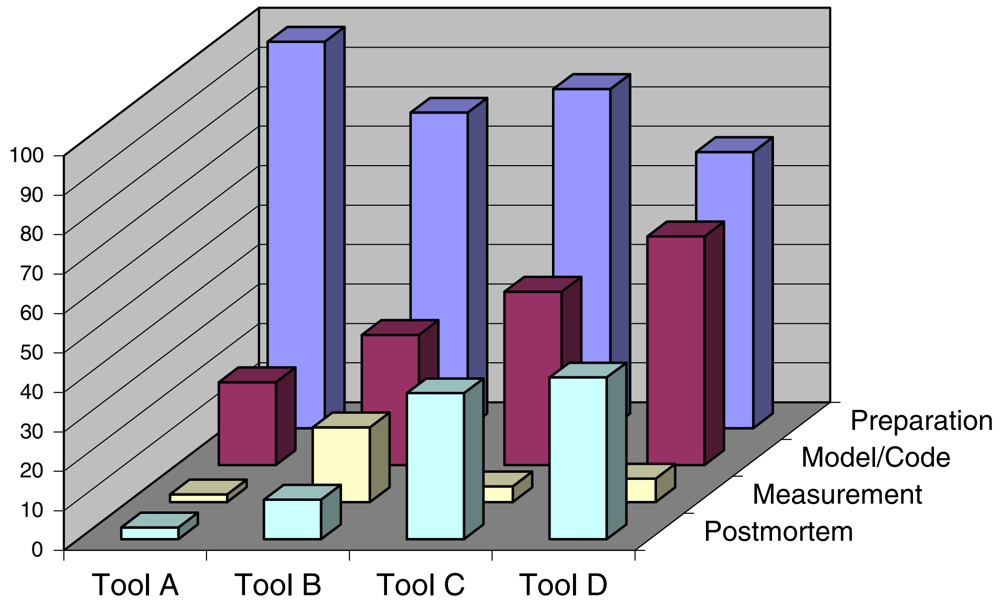

Fig. 7 Preliminary experiment: usability measured as effort assessment (in hours)

may be biased since part of the modeling time was actually spent on learning.

5.2 Controlled experiment results

The objective of the controlled experiment was a more detailed evaluation of real-time software design tools with automatic code generation capability, using the four criteria described in Sect. 4. The selected tools included three from the structural (object-oriented) category (labeled here L, M, and N) and three from the functional (block-oriented) category (labeled O, P, and Q, where tool O actually crosses the boundary of two categories).

Fourteen developers assigned to the project were graduate software engineering students familiar with software development methodologies, software processes, and real-time design concepts. Each of the six tools was assigned to a team of two or three developers who shared the initial training and the final reporting. However, each team developed the model and implemented code as an individual assignment.

To reduce the bias identified in the preliminary experiment, the second experiment was designed in two phases, each consisting of developing a separate model of embedded software. The first model, a simple hair dryer simulator, was used during the first phase of this experiment to facilitate the learning and constituted a capstone for familiarization with the methodology, tool, and operating environment. The activities included reading documentation and materials about modeling methodology, experimenting with tool demos, running tutorials, etc. The second system, a simple microwave oven controller software simulator, was used in the actual design and data collection phase.

Requirements for the first model, hair dryer simulator, are presented in Table 2, and requirements specifications for the second model, simple microwave oven simulator, are shown in Table 3.

As in the preliminary experiments, the following four top-level tasks were elaborated in terms of entry and exit conditions and the activities to be performed:

1) Project preparation and tool familiarization.
2) Model development and code generation.
3) Measurements and data collection (effort, code size, traceability, questionnaire).
4) Postmortem.

Each developer was given a process script with specific tasks to perform, the summary of which is presented below.

(1) Preparation
  (a) Creation of PSP estimates of time and code size for project.
  (b) Selection of tool and familiarization with project requirements.
  (c) Learning to use the tool and identifying available resources that can be used during development.
  (d) Development of demonstration hair dryer model as a learning aid.
(2) Model creation and code generation
  (a) Microwave oven model creation according to specified requirements.
  (b) Manual verification that all requirements were covered in the model; if the tool provided verification capabilities, they were to be used.
  (c) The code generation capabilities of the tool were then to be used to generate C code for the model.
(3) Measurement (e.g., for traceability)

Table 2 Requirements for hair dryer simulator

1. The system shall allow user to select motor speed (off, low, or high).
2. The system shall apply power to motor depending on selected speed setting.
3. The system shall cycle the heater (30 s on and 30 s off) when in low- and high-speed modes.
4. The system display shall show the selected speed, heater status, and countdown time when the heater is on.

Table 3 Requirements for microwave oven simulator

1. The oven shall allow user to set the cooking time in minutes and seconds (from default 00:00 to 59:59).
2. The oven shall allow user to set the power level (in the range of default 1 to 5).
3. The start of cooking shall initiate on an explicit user request.
4. When the cooking starts, the oven shall turn on the light and the rotisserie motor for the specified time period.
5. When the cooking starts, the oven shall cycle the microwave emitter on and off: a power level of 5 means that the emitter is on all the time, a power level of 1 means that the emitter is on only one fifth of the time.
6. The oven shall display the remaining time of the cooking and the power level.
7. When the time period expires, the audible sound shall be generated and the light, motor, and emitter shall be turned off.
8. The oven shall turn on the emitter and the motor only when the door is closed.
9. The oven shall turn on the light always when the door is open.
10. The oven shall allow the user to reset at any time (to the default values).

   (a) Decomposition of design model, which was then analyzed for traceability.
   (b) Decomposition and traceability to model.
   (c) Decomposition and traceability to requirements.
   (d) Identification of code that did not have a representation in the model.

(4) Postmortem
   (a) Completion and analysis of PSP data.
   (b) Assessment of tool's conformance to traceability.
   (c) Compilation of each developer's individual data into a joint report summarizing their findings.

Quantitative data were collected for each of the four selected criteria (metrics), with the following corresponding measures:

1. Efficiency was measured as code size (number of lines of code, or LOC, generated by the tool from the user-designed software model).
2. Usability was measured as effort (time spent in the experiment for each process phase and the overall time spent by each developer).
3. Functionality was measured as a given developer's subjective assessment (on a scale of 0 to 5) via a questionnaire with questions grouped in the following four categories: tutorial, user manuals, readability, and flexibility.
4. Traceability was given a qualitative assessment via manual tracking of code to the model and requirements, as explained in tasks (3a)–(3d) above.

5.2.1 Efficiency

Results of efficiency measurements via the size of generated code for all six tools are presented in Fig. 8. The measured code size varies greatly among the tools. Four of the tools generated the code of reasonable size on the order of less than one KLOC. The two others generated a significantly larger source code for the same problem.

5.2.2 Usability

The other major criterion used for evaluation, tool usability, was measured as effort (in hours) spent on each of the four tasks listed above. Results collected for six selected tools are shown in Fig. 9. Using a two-phase approach reduced the preparation phase and ultimately reduced the modeling phase as well. Different tools clearly require varying levels of effort reaching nearly a three-to-one ratio.

5.2.3 Functionality

Figure 10 presents results of a functionality evaluation for all six tools based on a questionnaire with scores from 0 to 5. The components of functionality were extracted from the surveys soliciting developers' feedback on tool flexibility (i.e., ease of model modification, manipulations with menus, choice of various notations, and constructs to represent the design), readability (clear understanding of a design model and the process of code creation), tutorial, and user and reference manuals. Across the evaluated tools, the overall ratings were rather low. This indicates that despite the advertised capabilities of these tools, the available resources for developers to make effective use of the tools are insufficient or of marginal quality. All the developers confirmed this in their feedback. It happened to be a particular problem during the preparation phase, where the need for supporting materials was the greatest. It should be noted, however, that the experiment was carried out in an academic environment and the developers were not exposed to extensive vendor-supported training.

5.2.4 Traceability

Another indicator used, traceability, was assessed only qualitatively. The developers' activities focused on the fundamental characteristics of a tool to accurately translate the requirements into design models and then into the target code and back to the requirements. Due to the relative simplicity of

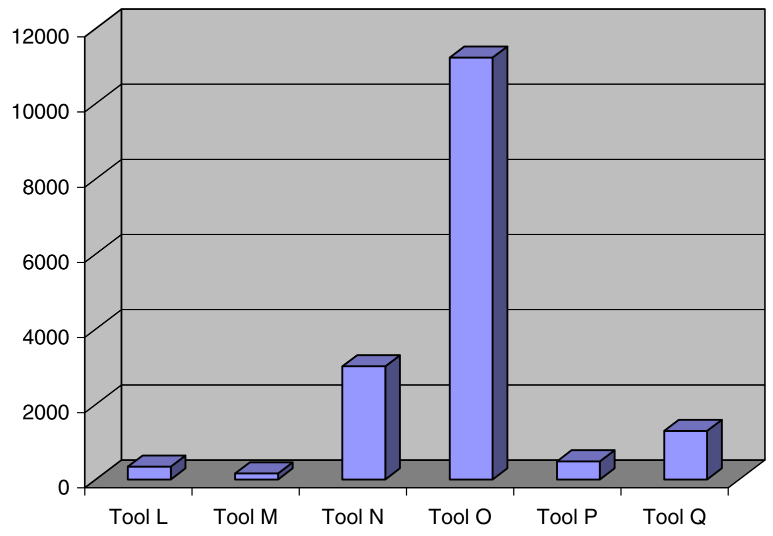

Fig. 8 Efficiency measured as the size of code generated (in LOC)

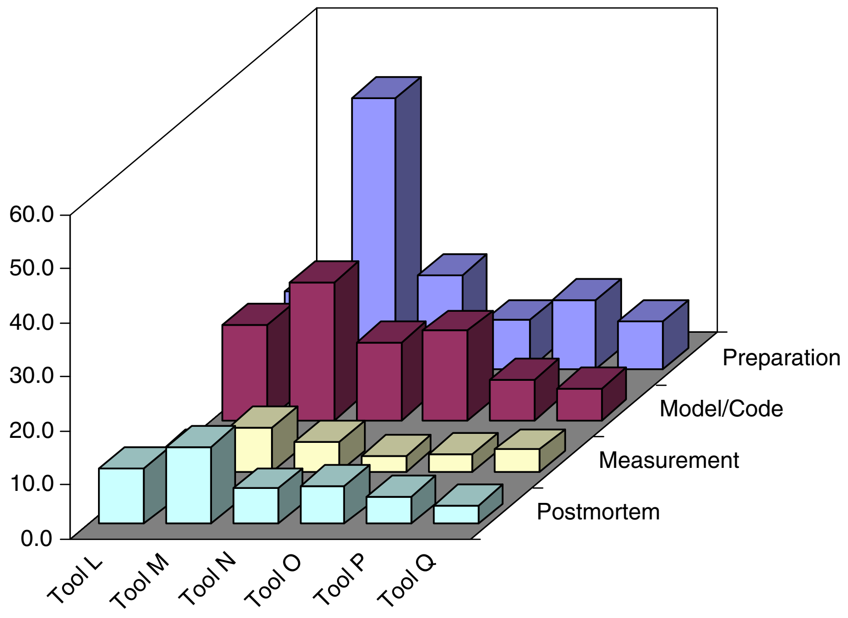

Fig. 9 Usability measured as effort assessment (in hours)

the project, all software requirements were traced to specific model components. The created model components were in turn mapped to the appropriate code segments generated by the tool (objects, methods, or function blocks) that represented them. Any component that did not map directly to a section of code was then checked against the generated code to identify any code that might cover it. The code was analyzed to identify any part that did not relate to a specific model component, and if possible its purpose was recorded to identify the reason for any nontraceable function/code. With this approach the relationship between the requirements, design, and code was established. Several tools generated sections of code with the framework run-time control that could not be directly traced to the model component. The analysis showed that traceability in the requirements–design–code chain is very much tool dependent. An example of traceability analysis for one of the tools is shown in the appendix.

One of the challenges faced in this experiment was the use of tools based on object-oriented notations and methods for development of a simple reactive system. The translation of object-oriented methods and techniques to generate C code proved to be a challenge for most developers. Most of them felt that important aspects of the system being developed, such as timing constraints, were not properly captured or were simply "lost in the translation." This also proved to be a hindrance in the learning process, as the focus was already on the problem at hand and the task then became fitting the tool into the problem solution.

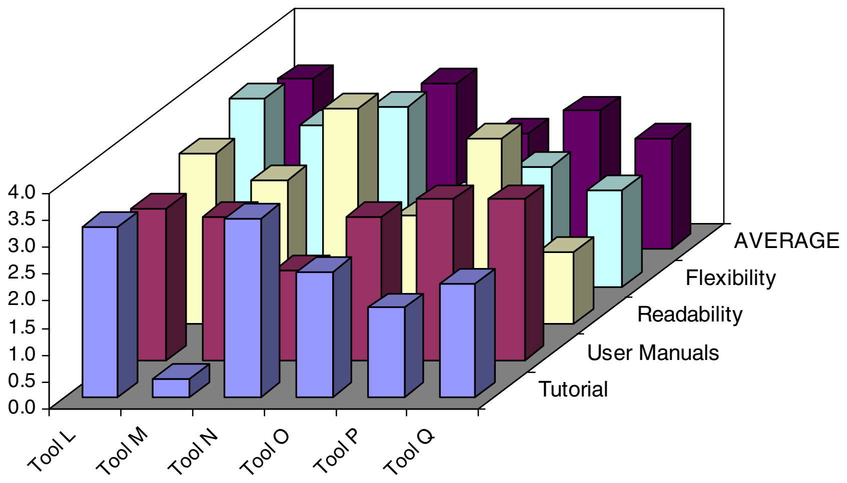

Fig. 10 Functionality measured on tool and auto code generation questionnaire score

6 Conclusion

The objective of this study was to address some of the problems in evaluating software development tools used in safety-critical real-time systems and potential application of the findings to the tool qualification process. Such tools, supporting model-driven development, are the fastest growing category of development tools. They are extremely popular across the wide range of application areas, including the aviation industry. Our work focused on evaluating design tools with code generation capability.

Applying the taxonomy view for evaluation purposes, based on the analysis of industrial practice and earlier research studies, four criteria and their respective measures were established: *functionality*, *usability*, *efficiency*, and *traceability*. Experiments conducted for a wide array of industry-strength tools on a specially designed testbed proved that, for this or other sets of criteria, a well-designed assessment process can provide significant insight into a tool's quality and help make decisions on how well the tool meets qualification criteria. In particular, the process of tool assessment can be characterized by the following four activities:

- Application of a domain-specific benchmark problem and platform on which the tool will be evaluated;
- Identification of specific criteria (metrics such as usability, traceability, etc.) against which the tool will be evaluated;
- Development of a measurement method to evaluate each criterion;
- Collection and analysis of results.

In general, the assessment process poses several challenges. In this study, the tool evaluation experiments used rather simple projects of embedded software development. Therefore, the approach may not scale up very well because of the necessity to spend significantly more resources on actual experiment preparation and data collection. In particular, traceability assessment in both experiments relied on manually tracing the sections of code that fulfill a particular requirement and evaluating the expressiveness and clarity in the structure and logic of the code. It was possible to do so in the experimental project because of the relatively few software requirements. In commercial product development, such activity may be too time consuming for practical purposes. The same observation is valid for other important criteria, which were not used in this study, such as reliability or robustness. They may contribute to a significantly more insightful evaluation of a given tool; however, the collection of respective experimental data may not be practical.

The need for conducting more detailed evaluation experiments and collection of data on the indicators defined in the taxonomy will be more pressing with the increasing number of newly emerging safety-critical real-time applications. As confirmed in a recently held tools forum [14], further research is needed on:

- Integrating the current tool evaluation criteria into a coherent set of metrics.
- Developing measurement methods (evaluation techniques) to apply these metrics to tool evaluation.
- Conducting further experiments with practical tool evaluation according to these methods.
- Proposing a tool qualification methodology considering both the process of using a tool in software development to collect observations on its use and procedures for creating the process required to qualify tools.

Appendix: Sample traceability checks for tool A

Model item	Code check
Module: I/O unit	√
Attributes	
int numParam	√
typedef double valArray [maxParam]	√

Model item	Code check
typedef int array [maxParam]	✓
array parameterArray	✓
array frequencyArray	✓
valArray valuesArray	✓
int counter [3]	✓
double total [3]	✓
string names [45]	✓
string unit [45]	✓
Functions	
void input()	✓
void display (in char**str)	✓
void request_paramID (in array parameter Array)	✓
unsigned short request_num()	✓
void request_avgFreq (in array frequencyArray, in array parameterArray)	✓
void getPackects (in array paramArray, in array freqArray) in string names[], in int numParam, in string units[])	✓
void getPackects (in array paramArray, in array freqArray, in string names[], in int numParam, in string units[])	✓
Module: Control unit	✓
Attributes	
array parameters	✓
array frequencies	✓
int numberOfParameters	✓
Functions	
char* timestamp()	✓
void set_paramID (in array parameterArray)	✓
void set_num (in unsigned short param)	✓
void set_avgFreq (in array frequencyArray)	✓
double get_avgdata (in array paramArray, in int tempParamFreq, in int*counter, in double*total, in int I, in string names[], in string units[], in int index)	✓
Module: Average calculator	✓
Functions	
double calc_avg (in double *dTotal, in int*iCounter)	✓

Acknowledgements The presented work was supported in part by the Aviation Airworthiness Center of Excellence (AACE) under contract DTFA0301C00048 sponsored by the Federal Aviation Administration (FAA). Findings contained herein are not necessarily those of the FAA. Acknowledgement is due to students from the Embry Riddle Master of Software Engineering program who contributed to experiments and data collection. Additional support was provided by the Florida Space Grant Consortium under Grant No. UCF01-E000029751.

References

1. RTCA (1992) Software considerations in aiborne systems and equipment certification. Report RTCA/DO-178B, Washington, DC
2. Zalewski J, Ehrenberger W, Saglietti F, Gorski J, Kornecki A (2003) Safety of computer control systems: challenges and results in software development. Annu Rev Control 27(1):23–37
3. U.S. Department of Transportation (2003) Qualification of software tools using RTCA/DO178-B. Federal Aviation Administration, Software approval guidelines, Chap 9, Notice N8110.91, March 2003
4. Kornecki A, Zalewski J (2004) Criteria for software tools evaluation in the development of safety-critical real-time systems. In: Proceedings of the PSAM-7/ESREL'04 European conference on safety and reliability, Berlin, Germany, 14–18 June 2004
5. International Standards Organization (1991) ISO/IEC 9126–1991: Information technology – software product evaluation – quality characteristics and guidelines for their use. ISO, Geneva, 15 December 1991
6. International Standards Organization (1995) ISO/IEC 14102–1995:Information technology – guideline for the evaluation and selection of case tools. ISO, Geneva, 15 November 1995
7. IEEE (1993) Standard 1209–1992: Recommended practice for the evaluation and selection of case tools. IEEE, New York, February 1993
8. US Department of Transportation (1991) Federal Aviation Administration, Software quality metrics, Report DOT/FAA CT-91/1
9. Ihme T, Kumara P, Suihkonen K, Holsti N, Paakko M (1998) Developing application frameworks for mission-critical software: using space applications as an example. Research Notes 1933, Technical Research Centre of Finland, Espoo
10. Wichmann B (1999) Guidance for the adoption of tools for use in safety related software development. Report, British Computer Society, March 1999
11. IEEE (1998) Standard 1061–1998: Software quality metrics methodology. IEEE, New York, December 1998
12. Kornecki A, Zalewski J (2005) Process-based experiment for design tool assessment in real-time safety-critical software development. In: Proceedings of the SEW-29 NASA/IEEE 29th workshop on software engineering, Greenbelt, MD, 6–7 April 2005. IEEE Computer Society Press, Los Alamitos, CA
13. Humphrey W (1997) Introduction to the personal software process. Addison-Wesley, Reading, MA
14. Federal Aviation Administration (2004) Software tools forum, Daytona Beach, Fl, 18–19 May 2004. `http://www.erau.edu/db/campus/software-toolsforum.html`

About the Authors

Dr. Andrew J. Kornecki is a professor in the Department of Computer and Software Engineering at Embry Riddle Aeronautical University, Daytona Beach, FL. He has over 20 years of research and teaching experience in areas of real-time computer systems. He has contributed to research on intelligent simulation training systems, safety-critical software systems and served as a visiting researcher with the FAA. He has been conducting industrial training on real-time safety-critical software in medical and aviation industries and for the FAA Certification Services. Recently he has been engaged in work on certification issues and assessment of development tools for real-time safety-critical systems.

Dr. Janusz Zalewski is a professor of computer science at Florida Gulf Coast University in Fort Myers, FL. Before taking a university position, he worked for various nuclear research institutions, including the Data Acquisition Group of Superconducting Super Collider and Computer Safety and Reliability Center of Lawrence Livermore National Laboratory. He also worked on projects and consulted for a number of private companies, including Lockheed Martin, Harris, and Boeing. He served as chairman of IFIP Working Group 5.4 on Industrial Software Quality and of an IFAC TC on the safety of computer control systems. His major research interests include safety-related real-time computer systems.