ASSESSMENT OF SOFTWARE DEVELOPMENT TOOLS FOR SAFETY-CRITICAL REAL-TIME SYSTEMS*)

Andrew Kornecki1) and Janusz Zalewski2)

1) *Department of Computing, Embry-Riddle Aeronautical University*
Daytona Beach, FL 32114-3900, USA
`kornecka@erau.edu`
2) *Computer Science Program, Florida Gulf Coast University*
Ft. Myers, FL 33965-6565, USA
`zalewski@fgcu.edu`

Abstract: The paper presents guidelines on criteria and procedures for evaluating software development tools used in safety-critical real-time systems. We present, first, a view of the taxonomy of software development tools from the perspective of the development process and the development environment. The investigation concentrates on evaluating the design tools, considering their interfaces with the requirements and testing phases of the software lifecycle. Furthermore, we discuss the taxonomy of criteria for tool evaluation. The major observations are related to the differences between evaluating the tool itself (macro-evaluation), evaluating the process of producing the tool (meta-evaluation), and evaluating products developed with this tool (micro-evaluation). Building the criteria for evaluation is based on the triad: choosing an appropriate attribute (property) of the tool, describing the metric for the evaluation of this property, and applying some measure (measurement procedure) to use the metric and obtain the results of evaluation of this property. *Copyright © 2003 IFAC*

Keywords: real-time systems, software safety, safety related systems, software tools.

1. INTRODUCTION

The objective of this paper is to present the taxonomy of criteria and procedures for evaluating software development tools used in safety-critical real-time systems. Automatic development tools are more and more extensively employed in the design process. Although some research has been previously done in this area (Sherry et al. 1998), experts in this field not necessarily agree on the issue. Some say that tools do not have a direct impact on the executing code, so their role is not critical. Others see the tool at the same standard as the resulting target system.

As the tools participate in the development of safety-critical software, the evaluation of the tools should be made an intrinsic part of the development. Just like the companies developing safety-critical software employ the best professionals to participate in the design process, we need the best tools to be used in this process as well. It is also generally agreed that errors introduced to software artifacts in early stages of development are extremely costly. Thus, a process for evaluating these tools has to be created.

The ultimate goal of developing the safety-critical real-time systems is to provide evidence that the safety requirements (in addition to other requirements) have been met. Since the development tools participate in this process, their quality affects directly and indirectly the quality of the target software and, therefore, the overall system safety.

*) This work was sponsored in part by the Federal Aviation Administration (FAA). Findings contained herein are not necessarily those of the FAA.

In this paper, we concentrate on avionics, as the specific application area. For this reason, we start with a widely accepted standard for software considerations in airborne systems, DO-178B (RTCA 1992). Of the four primary processes of software development, as defined in DO-178B:

- Software Requirements Process
- Software Design Process
- Software Coding Process
- Integration Process

we make the Software Design Process a focal point of tool evaluation. With this in mind, we need to choose both the software tools relevant to this process, as well as the criteria for the evaluation of these tools. In this view, we concentrate in this paper on the taxonomy of the tools (Section 2), selection of tool evaluation criteria (Section 3), and plans for the experimental work (Section 4).

2. TOOL TAXONOMY

2.1 *Process Aspects*

The initial assumptions for developing the tool taxonomy are as follows:

1) Our focus is on Design Process, that is, we do not deal with tools related to the Requirements Phase nor do we deal with tools relevant to the Testing and Verification Phases.
2) The practice of using tools during the Design Process can be represented as the following sequence: requirements tool, followed by design tool, typically with code generation functionality, an Integrated Development Environment and the target with real-time operating system; a significant role is also played by analysis and testing tools (Fig. 1).

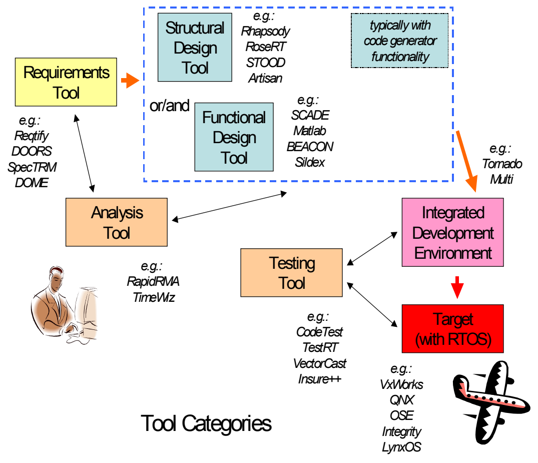
Requirements Tool
e.g.: Reqtify DOORS SpecTRM DOME
Structural Design Tool
e.g.: Rhapsody RoseRT STOOD Artisan
typically with code generator functionality
or/and
Functional Design Tool
e.g.: SCADE Matlab BEACON Sildex
e.g.: Tornado Multi
Analysis Tool
e.g.: RapidRMA TimeWiz
Testing Tool
e.g.: CodeTest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

Fig. 1. Model of real-time software development process and its impact on tool use.

This model covers most, if not all, of the known development schemes, for example:

1) A high-level structural design tool (e.g. Rose RT, Rhapsody, Artisan Studio, Esterel Studio, STOOD) is used to develop software architecture in a specific graphical notation (e.g. UML). Also, such high-level design tools can be used directly to develop real-time software and generate source code in a high level programming language (C/C++, Ada, Java) for specific targets.
2) For smaller and simpler systems the control algorithms are usually developed using function-oriented tools (e.g. SCADE, Matlab/StateFlow, BEACON, MatrixX, Sildex). These tools also can generate source code (C/C++, Ada), which can be directly tested. In either case developers use Integrated Development Environments (e.g. Tornado for VxWorks, MULTI for Integrity, etc.) to supply hand-written or generated source code for compilation for specific target machines.
3) In all approaches mentioned above there is a need to maintain consistent requirements; therefore, a requirements analysis tool (e.g. DOORS, Reqtify) is typically used.
4) The development requires thorough analysis and testing, typically supported by appropriate automatic tools (e.g. PERTS, TimeWiz, CodeTest, Insure++, TestRT).

Categories of tools based on this model define the scope of the evaluation. The focus of this research is on Software Design tools - the center of the diagram in Fig. 1. Although the tools outside the dashed-line box are important in software development, their role in the Design Process is limited and includes only interfacing to this process, from the point of view of requirements specification, analysis, and testing.

2.2 *Other Aspects*

Thus far, we discussed only one specific aspect of the use of automatic software design tools, that of their relationships with other phases of the development process. Therefore, we can call it a *process aspect* and, following the idea of a waterfall model of software development, where it naturally fits, call it a vertical view of the tool, or tool's *vertical dimension*.

There are, however, other aspects related to the way software development tools are used. One of the most important ones is the *environment aspect*. An IEEE standard on tool interconnections addresses this issue (IEEE 1992a), and distinguishes among several contexts, in which the tools are used (interfaces for the tool use):

- user context, which refers mostly to the interaction via the human-computer interface
- organization context, related to the processes, in which the tool is used
- other tools, related to interactions and data exchange with other software systems
- platform context, describing the infrastructure on which the tool is hosted.

Based on this general distinction, one can specify more accurately a variety of roles a software development tool is playing in the environment. One can thus list corresponding views of the tool, or tool's dimensions, which may be helpful in the evaluation. In this respect, a user context represents an internal view, or *internal dimension* of the tool, since the task of the user is typically to develop the graphical representation of the design and check it, for example, using tool's animation or simulation capability. These are internal tool functions.

Two other contexts, organization interface and interface to other tools refer, respectively, to the communication the tool has with other processes (projects) within the organization, as well as to communication with other software within the same project. The first type of communication is mostly static (off-line) and relies on exchanging design models with other projects. The second type is more dynamic (on-line) and relies on maintaining direct connectivity with other tools. Since it normally involves peer-to-peer interaction, a corresponding view of the tool related to communication with other tools is called its *horizontal dimension*. Similarly, since the exchange of design models between projects happens across the organization, it is called *diagonal dimension*, symbolizing crossing the project boundaries (or even organizational boundaries). The final context, the platform context, although it is an important factor in the tool environment, can be considered an *external dimension*, not critical in the quality evaluation of the tool.

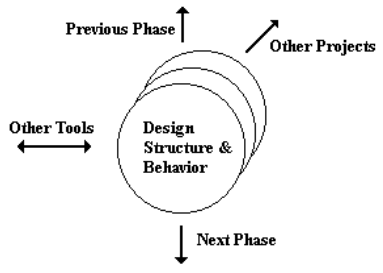

Fig. 2. Four aspects (dimensions) of the tool use.

Consequently, four different aspects of the design tool use can be illustrated (Fig. 2), referring to four specific dimensions of the tool, with respect to its role in the process and the environment:

- *vertical dimension*, related to the process aspects of the tool use, specifically to its support for the next and previous phases of the development process (see Sec. 2.1)
- *internal dimension*, related to user aspects of the tool (developing the model of the design and simulating its operation and performance)
- *horizontal dimension*, related to the environment aspects in a view of on-line communication with other tools, for instance, via TCP/IP protocol, and
- *diagonal dimension*, referring to the ways of exchanging design models with other processes or projects, for example, importing a non-UML model into a UML-based tool.

3. TAXONOMY OF CRITERIA FOR TOOL EVALUATION

3.1 Basic Model of Tool Evaluation

Once we have agreed upon the categories of tools to evaluate, and discovered various aspects of tool interfacing with the process and the environment, we need to derive a model of the tool evaluation process, necessary for developing the evaluation criteria. The framework for this process is shown in Fig. 3.

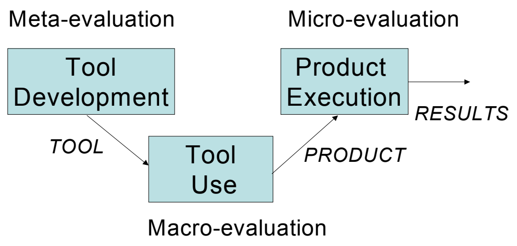

Fig. 3. Model of the tool evaluation process.

The central part of this model is the *macro-evaluation* based on the use of the tool during the Design Process. However, this is not and should not be the only basis for evaluating the tool. A lot of information on tool quality can be usually derived from the development of the tool itself. Therefore, the model includes tool development as a separate phase. This can be considered a *meta-evaluation*: evaluating the method and the process to develop a tool. The data for evaluation of this phase can be requested from and provided by the tool vendor.

In addition to the *macro-evaluation* and the *meta-evaluation*, the product developed with a particular tool should be included in the evaluation. A good product can provide data on the tool quality, so does a poor product. Evaluating a product is called *micro-evaluation*, since it focuses on the level lower than the tool itself. Such product evaluation can be based both on code analysis and on code execution.

Correspondingly to this three-level evaluation model, three categories of tool evaluation criteria have to be defined: those related to the tool development, the tool itself, and the product developed with this tool. In other words, to have the entire picture of the tool, we need to do three types of evaluation, not just one limited strictly to the tool itself.

3.2 The Criteria Considerations

Evaluating Software Products and Tools. Evaluating quality of the tool is different from evaluating quality of the product. Product quality is evaluated for its compliance with the requirements. For the tool, requirements are typically unknown. There are several basic documents, which define criteria for software evaluation, in general (ISO/IEC 1991). The ISO/IEC standard is very specific about the software evaluation criteria. It lists six such characteristics:

- *functionality*, comprising a set of attributes that bear on the existence of specific functions
- *reliability*, defined as a set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time
- *usability*, set of attributes that bear on the effort needed for use of the software
- *efficiency*, a set of attributes that bear on the relationship between the level of performance of software and the amount of resources used
- *maintainability*, related to a set of attributes that bear on the effort needed to make specific modifications, and
- *portability*, understood as a set of attributes that bear 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 sub-characteristics. The validity of this approach has been positively tested in (Abel and Rout 1993). It is not, however, tool specific. One report offers a particularly good view on the criteria for evaluating software tools, an ISO/IEC guide for evaluation and selection of CASE tools (ISE/IEC 1995). The approach presented there is compatible with the above mentioned ISO/IEC standard (ISO/IEC 1991), as well as with an earlier IEEE Std 1209 (IEEE 1992b), addressing the same subject. This guide also advocates evaluating tools with respect to specific characteristics understood as evaluation categories. They can be further divided into attributes that may be assigned values during the evaluation process, based on some accepted metrics.

An earlier document on defining criteria for software tool evaluation (Firth et al. 1987) has been used in (Ihme et al. 1998a, 1998b) to evaluate several tools applied to the development of mission critical software. Each criterion includes a set of low-level criteria, or attributes:

- *ease of use*, which involves tailoring, helpfulness, predictability, error handling, system interface
- *power,* related to tool understanding, tool leverage, tool state, performance
- *robustness,* involving consistency of operation, evolution, fault tolerance, instrumentedness
- *functionality*, regarding correctness and methodological support
- *ease of insertion* pertaining to learnability and software engineering environment
- *quality of support* concerning tool history, maintenance, user's group and feedback, installation, training, documentation.

Each attribute, in turn, involves 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 weighing factors:

- ease of use, 17%
- power, 10%
- robustness, 10%
- functionality, 30%
- ease of insertion, 13%
- quality of support, 20%.

On this basis, the tool is assessed globally as meeting the set of criteria, according to an additional scale.

There is very little done and published work on evaluating tools specifically for safety-critical real-time systems. One particular report deserves attention, since it attempts to attack the problem directly from the point of view of safety related software development (Wichmann 1999). The report suggests focusing on those tools, which involve the highest risk associated with their use. These are normally the tools having direct influence on the safety system, such as compilers, but also design tools that generate code for safety related target systems. Respective concerns related to tool use are grouped into three categories:

- technical concerns related to commercial tools
- commercially related COTS concerns, and
- difficulties associated with in-house tools development and support.

After a discussion of potential approaches to the use of tools in safety related software development, several recommendations are given in terms of questions addressing:

- general issues
- those to be posed to tool suppliers
- actions recommended for users selecting tools
- actions recommended for tool users having adopted a tool, and
- long-term recommendations for industry and the profession.

Building the Quality Evaluation Criteria. Based on discussion in previous sections, the evaluation of software tools used in designing safety-critical real-time software should involve:

- a list of important criteria
- a list of factors (attributes) to evaluate a specific criterion from the list above, and
- procedures to evaluate (measure) values of the above mentioned attributes.

In other words the taxonomy has to be developed that:

- identifies the criteria, which are representing quality of the tool in the opinion of its users and evaluators (one has to remember that the tool is used for safety related software, not just general-purpose software)
- for each criterion, establish specific attributes and their levels on a certain scale, which would best characterize the criterion and allow assigning a value for each of the attributes
- for each attribute, recommend a procedure, equivalent to some kind of a measurement process, which would lead to associating a

numerical value with this attribute, in a process as objective as possible.

As described above, it is a typical problem of software quality measurement. Several reports exist, which propose various, usually consistent with each other, approaches to this problem (IEEE 1988, FAA 1991, Barbacci et al. 1995, IEEE 1998). In a view of identifying software quality attributes, assigning them scale of values, and establishing procedures for assessing these values, the following definitions from (IEEE 1998) would apply:

- *Attribute* - a measurable physical or abstract property of an entity.
- *Metric* (Software Quality Metric) - a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality.
- *Measure* - a way to ascertain or appraise value by comparing it to a norm (to apply a metric).

To illustrate this discussion by an example, measuring properties of software can be compared to measuring physical quantities according to the same scheme. One example is *time*:

- attribute (or physical property): time
- metric: second
- measure: any device that incorporates the procedure to calculate time (e.g. clock, stopwatch, chronometer).

The key issue in this scheme is a definition of a metric. In case of *time*, a second is *"the duration of 9 192 631 770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the caesium 133 atom."* Metrics definition is, however, typically the most difficult part of any software project.

In terms of the tool properties, one must identify when evaluating the tool, there are several views to consider. For example, according to the tool aspects from Section 2.2, the following sample attributes can be assessed:

- *correctness and performance*, in the internal dimension
- *traceability and testability*, in the vertical dimension
- *connectivity and interoperability*, in the horizontal dimension
- *reusability and portability*, in the diagonal dimension.

On the other hand, following the ISO/IEC standards (ISO/IEC 1991; ISO/IEC 1995), one can focus on the model discussed in Section 3.2, involving the following attributes: functionality, reliability, usability, efficiency, maintainability, and portability.

4. EXPERIMENTAL WORK

Once we have defined the criteria and procedures to evaluate the tool, we are essentially facing the problem how, within the process defined in DO-178B, to conduct an experiment of software development applied to a simple airborne application, so one could evaluate the role of the design tool and associated processes, to verify the approach suggested above. For this reason, a testbed has been defined and implemented, for the measurements to take place. Two aspects of such testbed are equally important:

1) A sample model of an avionics application.
2) Equipment and software for host and target platforms.

We define a standard application that could be used as a typical benchmark for further studies on tool evaluation. Specifically, it has to take into account a number of sensors that gather various kinds of in-flight data, as well as a typical human interface including simplified pilot controls and a number of displays, typically present in a cockpit, to track flight parameters. At least one such application has been described, known as the Generic Avionics Software Specification (Locke et al. 1990), shown in Fig. 4.

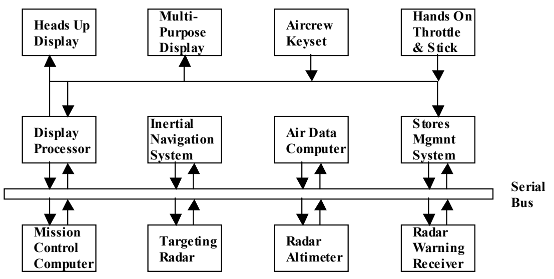

Fig. 4. Generic avionics architecture block diagram; adopted from (Locke et al. 1990).

The diagram shows a number of processors responsible for various types of data gathering from sensors, such as:

- air data computer, collecting barometric altimeter plus static and dynamic pressure data
- inertial navigation system, providing aircraft position and velocity, plus attitude and heading
- radar altimeter, providing measured height above the ground
- radar warning receiver, that warns the aircrew of hostile radar energy
- targeting radar, providing range and angle data of sufficient accuracy to track moving targets.

The generic system presented in Fig. 4 represents most of the functions of a real avionics application. For the purpose of this project, it is reduced to include only the most important aspects of human interfacing and data collection from sensors. A reasonable way to proceed is to eliminate parts of the system not relevant to civil aviation. Thus, only INS and Air Data Computer with Radar remain as the primary sensor equipment, with one of the displays and the simplistic interface for controlling aircraft.

The primary assumption regarding the subsequent choice of the target platform is to choose standard hardware and software to make experiments as repeatable as possible. Regarding the hardware, due to their widespread use in industry, VME and cPCI bus architectures with either PowerPC or Pentium processors are being considered. The ultimate choice depends on the previous experience with this type of hardware and compatibility with the supporting real-time kernel selected for use.

Choosing appropriate equipment to generate sensor information is another critical part of the project, since it may obscure the results, if not done properly. The idea is to avoid developing hardware or software, and rely on off-the-shelf solutions. Our choice is to employ a flight simulator, which can produce signals deliverable to an avionics bus, with a possibility to receive control signals developed by the equipment hooked to the bus. All commercial solutions we considered were more optimal than developing the system in-house.

5. CONCLUSION

This paper presented guidelines on criteria and procedures for evaluating software development tools for use in safety-critical real-time systems. We proposed, first, the starting point for the research on tool evaluation and outlined a view of the taxonomy of software development tools from the perspective of the development process and the development environment. The main suggestion is to concentrate on evaluating the design tools, considering their interfaces with the requirements, analysis, and testing phases of the software lifecycle.

Furthermore, we discussed the taxonomy of criteria for tool evaluation. The major observations are related to the distinction between evaluating the tool (evaluation proper), evaluating the tool development (meta-evaluation), and evaluating product developed with this tool (micro-evaluation). Building the criteria for evaluation should be based on the triad:

- choosing an appropriate attribute of the tool
- describing the metric for its evaluation, and
- applying some measure (measurement procedure) to use this metric and obtain the result of evaluation of this attribute.

It is essential to verify findings on an experimental testbed, which employs the following elements:

- a generic avionics application used as a model for software development
- standard bus-based hardware platform, with a standard real-time kernel, that will play a role of a target system for the code generated from the evaluated design tools, and
- a flight simulator, which has the ability to deliver signals to the hardware platform and receive control signals from it, to make the results of the research verifiable.

ACKNOWLEDGEMENTS

The work on safety critical real-time software development tool assessment is still in progress and a close cooperation with the software and aviation industry is the critical element for success. We are grateful to our FAA contacts and numerous companies for their thoughtful discussions. Help of graduate students in the ERAU Software Engineering Program is also gratefully acknowledged.

REFERENCES

Abel D.E., T.P. Rout (1993), Defining and Specifying the Quality Attributes of Software Products, *The Australian Computer Journal*, Vol. 25, pp. 105-112.

Barbacci M.R., M.H. Klein, T.A. Longstaff, C.B. Weinstock (1995), *Quality Attributes*, Report CMU/SEI-95-TR-021, Software Engineering Institute, Pittsburgh, Penn., USA.

FAA (1991), *Software Quality Metrics*, Report DOT/FAA/CT-91/1, FAA Technical Center, Atlantic City, NJ, USA.

Firth R. et al. (1987), *A Guide to the Classification and Assessment of Software Engineering Tools*, Technical Report CMU/SEI-87-TR-10, Software Engineering Institute, Pittsburgh, Penn.,USA.

IEEE Std. 982.1 (1988), *Standard Dictionary of Measures to Produce Reliable Software*, IEEE, New York, USA.

IEEE Std 1175 (1992a), *Trial-use Standard Reference Model for Computing System Tool Interconnections*, IEEE, New York, USA.

IEEE Std 1209 (1992b), *Recommended Practice for the Evaluation and Selection of CASE Tools*, IEEE, New York, USA.

IEEE Std. 1061 (1998), *Software Quality Metrics Methodology*, IEEE, New York, USA.

Ihme T. et al. (1998a), CASE Tool Evaluation, *Proc. DASIA Conf. On Data Systems in Aerospace*, Athens, Greece, pp. 121-126.

Ihme T. et al. (1998b), *Developing Application Frameworks for Mission-Critical Software: Using Space Application*, Research Notes 1933, Technical Research Centre of Finland, Espoo, Finland.

ISO/IEC 9126 (1991), *Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for Their Use*, Geneva, Switzerland.

ISO/IEC 14102 (1995), *Information Technology – Guideline for the Evaluation and Selection of CASE Tools*, Geneva, Switzerland.

Locke C.D. at el. (1990), *Generic Avionics Software Specification*, Tech. Report CMU/SEI-90-TR-8, Pittsburgh, Penn., USA.

RTCA (1992), *Software Considerations in Aiborne Systems and Equipment Certification*, Report RTCA/DO-178B, Washington, DC, USA.

Sherry L., A. Suarez, P. Wolfe (1998), Application of CASE Tools in the Development of Commercial Avionics Software, *Proc. SMC'98 IEEE Int'l Conf. on Systems, Man, and Cybernetics*, Vol. 1, pp. 875-880.

Wichmann B. (1999), *Guidance for the Adoption of Tools for Use in Safety Related Software Development*, Draft Report, British Computer Society, London, UK.