Criteria for Software Tools Evaluation in the Development of Safety-Critical Real-Time Systems [1]

Andrew J. Kornecki
Embry-Riddle Aeronautical University
Daytona Beach, FL 32114-3900, USA

Janusz Zalewski
Florida Gulf Coast University
Ft. Myers, FL 33965-6565, USA

Abstract

The paper presents various views of the criteria to be selected for evaluation of software tools used in the development of safety-critical real-time systems. It focuses on the avionics application area and derives a model for tool evaluation based on the process described in RTCA/DO-178B guidelines. The taxonomy of the tools is presented and four views of the criteria are applied to it, with an ultimate purpose to provide guidelines for the tool certification process. Data are presented from an initial experiment serving as a test-bed supporting the tool assessment methodology.

1 Introduction

This work is a part of a larger project whose purpose is to develop the criteria and procedures for evaluating software development tools used in safety-critical real-time systems, with an ultimate goal to provide guidelines for the tool certification process. Software development tools with automatic code generation features are more and more extensively employed in the real-time software design process with safety implications. Since the development tools participate in this process, their

[1] Work done in part for the Aviation Airworthiness Center of Excellence under contract DTFA0301C00048, sponsored by the Federal Aviation Administration (FAA). Findings contained herein are not necessarily those of the FAA. Part of this work has been supported by Florida Space Grant Consortium.

quality affects directly and indirectly the quality of the target software and, therefore, the overall system safety, so there is a compelling need to evaluate them with respect to safety.

We concentrate on avionics as the specific application area and place the software development process in the context of a widely accepted standard for airborne software, DO-178B [1]. Thus far, little work has been done on software tool evaluation for safety critical systems [2,3]. Our previous research included building tool taxonomy and a model of the process for tool evaluation [4], and surveying the industry and constructing an experiment for an avionics case study [5]. In this paper, we are discussing the extensive set of criteria for tool evaluation and present an initial experiment for tool evaluation using one of the criteria.

2 Tool Taxonomy

Developing the tool taxonomy, that is, categorizing the tools to define the scope of the project, we focused only on the Design Process. We left out tools related to the Requirements Phase and relevant to the Testing and Verification. In this respect, 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).

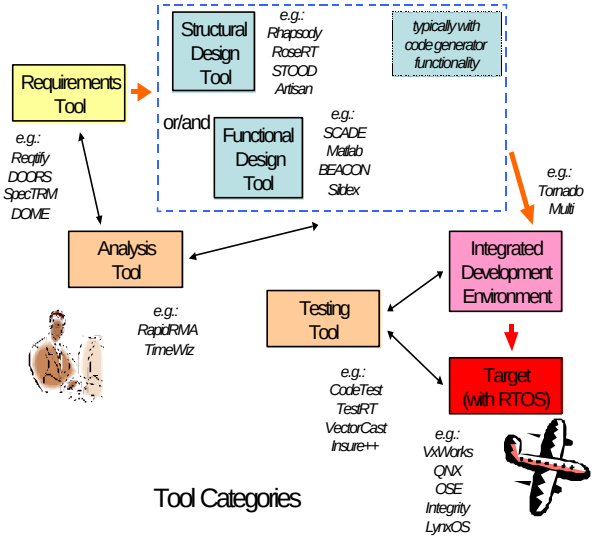

Fig. 1. Model of real-time software development process and its impact on tool use.

Categories of tools based on this model define the scope of the evaluation. The focus 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.

3 Criteria for Tool Evaluation

Four groups of criteria have been distinguished, based on four different views of the evaluation process: taxonomy view, project view, qualification view, and behavioral view. In this short paper we are only able to describe each view very briefly.

3.1 Taxonomy View

Three groups of tool evaluation criteria, called attributes, can be derived:

- The Functional Attributes related to "what" the user wants from the system.
- The Quality Attributes related to "how" the system fulfils the function regarding such criteria like dependability, performance, security, etc.
- The Business Attributes describe the quality aspects of the software without considering the functionalities.

The analysis of these attributes is based on defining for each attribute:

- Concerns, that are properties that cannot be measured directly, but they do affect the functionality, quality, or business aspects.
- Factors, that are the software-oriented characteristics of a concern. Usually several factors characterize a concern.
- Methods, concerned with metric evaluation and associated measurements. This is how to address the concerns and factors in the above definition.

The taxonomy table, which is created for a design tool under investigation identifies two areas of tool functionality: transformation of tool external graphic representation into internal format, and code generation. The identified concerns to be evaluated include, among others: determinism, robustness, and traceability.

3.2 Project View

Due to the potentially large number of software tools one can choose from, a simplified screening process is first described to eliminate those tool candidates that are ill-fitting for the project at hand. This view focuses on the aspect of software design tools that provide means to express the software component in a form of design artifacts and their subsequent translation into the source code. For the tools of such functionality the following characteristics need to be identified: programming language, complete vs. partial code generation, real-time features, safety (ability of tool to include safety-specific elements like watchdogs, redundant paths, retransmission, value checking, etc.), others, such as self documentation, learning focus, communication methods, platform, analysis capabilities, lifecycle integration, vendor support, longevity.

Each of the above characteristics can be assessed and assigned a value on a pre-defined scale. Additionally, as a separate activity, each of the characteristics must be assigned a weight. The weight depends on the specifics of the application

project for which the tool is used. The total of weighted values is a measure of the tool applicability to the specific project.

3.3 Qualification View

The tool evaluation table is constructed in the format of a Quality Function Deployment (QFD) matrix. Its purpose is to help those stakeholders who are involved in the selection, development, or certification of development tool(s), in identifying the capability of the tool(s) to fulfil the objectives in eliminating, reducing, and automating some aspect of the DO-178B process. The stakeholders can be categorized as users, vendors, and regulatory agencies.

The proposed tool evaluation table is divided into concerns and the objectives related to specific concern in the matrix rows (as documented in DO-178B). Data from this table can then be applied judiciously to the measurement information model along with other external quantitative measures (LOC, efficiency, etc.) to come up with more quantitative assessment about the tool. Concerns covered in the tool evaluation matrix include: traceability, determinism, robustness, correctness, and conformance to standards.

3.4 Behavioral View

To evaluate the tool in the operational use, we need to perform several steps: adopt a model of a typical application, develop a model of taking measurements, collect and analyze results of development. The first two steps are crucial for building the theoretical models of the measurements process for tool evaluation. They are based on the representation of the software architecture as presented in the ARINC 653 [6]. In this layered model of handling events by a real-time computer, the architecture is composed of application tasks residing at the highest level and interacting only between each other and with an operating system.

One needs to establish the parameters of this architecture, which would be strictly related to the capabilities of the tool in the process of developing the architecture. Two views of such parameters are possible, static and dynamic: addressing not only issues during building the model but also those which arise when model is run within a tool (on the host). Consequently, in two kinds of criteria are included:

- Endogenous criteria – those for which data can be collected on the tool itself in development of the architectural model, but independent of the model, and
- Exogenous criteria – such for which data can be collected on the behavior of the architectural model itself, to shed light on the dynamic properties of the model (as opposed to the software product running on a target).

A recent draft IEEE standard on “CASE Tool Interconnections – Reference Model for Specifying Software Behavior” provides an array of endogenous criteria to describe various aspects of software behavior [7]. Exogenous criteria are discussed in [8].

4 Experiments

The experiment objective was an initial evaluation of the selected software design tools with automatic code generation capability. The selected sample included four tools from both: structural (object-oriented) and functional (block-oriented) categories. Tool A was object-oriented and tools B, C and D were block-oriented. Four developers were assigned an identical problem statement to develop a real-time program to be implemented on VxWorks target. The project has been defined as a flight data collection from Opal-RT TestFlight simulator with a simplistic processing (averaging, time-stamping) and displaying results on the terminal (Fig. 2).

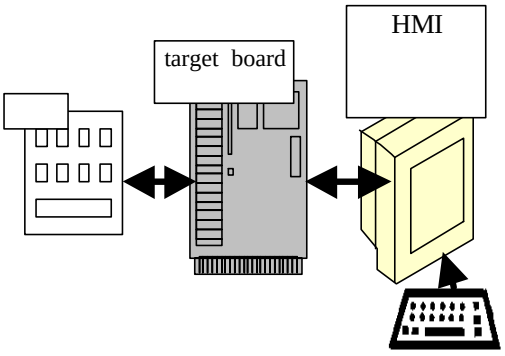

Fig. 2. The test-bed for the case study implementation

The software would capture data packets of parameter values transmitted from a flight simulator subsequently computing and displaying a moving average of the selected parameters with appropriate timestamp. A process script to follow was created to assist the developers. The following four top-level tasks were elaborated in terms of entry and exit conditions and the activities to be performed: (1) Project Preparation/Tool Familiarization, (2) Model Creation and Code Generation, (3) Measurement, (4) Postmortem.

The aggregate results are shown in Tables 1 and 2. The developers used to apply the Personal Software Process (PSP) underestimated the preparation phase effort by about 35%. The average planned time was 58 hours versus the actual of 78 hours. On the other hand, the developers planned in average about 72 hours to be dedicated to the design and coding phase. An actual average for this phase was below 39 hours. Automatic code generation reduced the development time in the order of 46%. The average code size was about 1.8 KLOC. The average total time spent of the project was 147 hrs, resulting in efficiency of over 12 LOC/hr. The learning curve is high and results may be slightly biased (as part of the modeling time was actually spent on learning tool). It is interesting to note that despite long learning curve the total project development was also completed in time. Automatic code generation reduced the planned total development time in average over 12%.

Table 1: Tool Preliminary Experiment – Effort Analysis (in hours)

	Tool A	~590	Tool B	~4,450	Tool C	~500	Tool D	~1,820
	plan	*actual*	*plan*	*actual*	*plan*	*actual*	*plan*	*actual*
Preparation	61.0	69.5	54.0	86.2	72.0	60.0	45.0	98.0
Model/Code	75.0	57.5	90.0	43.5	42.0	32.5	80.0	21.0
Measurement	24.0	5.5	24.0	4.0	16.0	18.5	41.0	2.0
Postmortem	20.0	41.0	12.0	37.0	8.0	10.0	12.0	3.0
TOTAL	180.0	173.5	180.0	170.7	138.0	121.0	178.0	124.0

Table 2. Tool Preliminary Experiment – Average Results (in hours)

	Aver	1,840	
	plan	*actual*	% change
Preparation	58.00	78.43	35.22
Model/Code	71.75	38.63	-46.17
Measurement	26.25	7.50	-71.43
Postmortem	13.00	22.75	75.00
TOTAL	169.00	147.30	-12.84
LOC/hr			12.492

To assess the requirements-design-code traceability the software requirements were matched against the design model components. Subsequently, the basic components of the created model were compared to the code sections (objects, function blocks) generated by the tool. Any component that did not map directly to a section of code is then checked against the generated code to identify any code the might cover it. Also the code is analyzed to identify any parts that did not relate to any model component, and their purpose is recorded. The analysis shows that the traceability between design and the generated code is very much tool dependent.

5 Conclusion

This initial experiment was an excellent experience for learning about software development tools, the infrastructure, and prepare base for a more advanced controlled experiment. Among the lessons learned it was established that the process scripts are a crucial element of the process and need to be carefully maintained. There will be more uniform data logs to allow for easier analysis of results. There will also be more quantified data for more tool-to-tool comparison. A qualitative (questionnaire) as well as quantitative (time and code) data will be collected. With this new knowledge, more and better-organized data can be compiled from the controlled experiment.

Acknowledgements

Graduate students of Master of Software Engineering program at the Embry-Riddle Aeronautical University: Kimberley Hall, Darryl Hearn, Herman Lau, Takashi Osako, Sameer Lakha, and Aber AbuRahma are gratefully acknowledged for contributing to this research.

References

1. RTCA, Software Considerations in Airborne Systems and Equipment Certification, Report RTCA/DO-178B, Washington, DC, 1992
2. Ihme T. et al., Developing Application Frameworks for Mission-Critical Software: Using Space Application, Research Notes 1933, Technical Research Centre of Finland, Espoo, Finland, 1998
3. Wichmann B., Guidance for the Adoption of Tools for Use in Safety Related Software Development, Draft Report, British Computer Society, London, 1999
4. Kornecki A., J. Zalewski, Design Tool Assessment for Safety-Critical Software Development, Proc. SEW'04 28th NASA/IEEE Software Engineering Workshop, Greenbelt, MD, December 2-4, 2003
5. Kornecki A., J. Zalewski, Assessment of Software Development Tools for Safety-Critical Real-Time Systems, Proc. PDS2003 IFAC Workshop on Programmable Devices and Systems, Ostrava, Czech Republic, February 11-13, 2003, pp. 2-7
6. ARINC Inc., Avionics Application Software Standard Interface, ARINC Specification 653, Baltimore, MD, 1997
7. IEEE Draft Std. P1175.3/D4.3 CASE Tool Interconnections – Reference Model for Specifying Software Behavior, IEEE, New York, 2003
8. Zalewski J., Software Dynamics – A New Measure of performance for Real-Time Software, Proc. SEW'04 28th NASA/IEEE Software Engineering Workshop, Greenbelt, MD, December 2-4, 2003