Evaluation of Software Development Tools for High Assurance Safety Critical Systems

Andrew J. Kornecki, Kimberley Hall, Darryl Hearn, Herman Lau [1)], Janusz Zalewski [2)]
[1)] *Embry-Riddle Aeronautical University, Daytona Beach, FL 32114-3900, USA*
[2)] *Florida Gulf Coast University, Ft. Myers, FL 33965-6565, USA*
< kornecka@erau.edu >

Abstract

This interim report describes investigation of leading software development tools used in development of aviation systems. It is a part of three-year effort with the ultimate objective to provide guidelines for a potential tool qualification. The hypothesis is that the tool qualification allows the developing organizations reduce the effort required to verify the artifacts produced by the tool. The saving on costly verification process is an incentive for pursuing tool qualification.

1. Introduction

The software development tools are actively transforming the artifacts of software development phases. Obviously, tool quality and correctness may affect directly and indirectly the quality of the target software and, therefore, the overall system safety. There is an extensive body of literature related to assessment of software quality [1,2,3]. However, the assessment of the tools used for safety-critical software has not received enough attention in the literature

Since airborne software is the selected application area, a widely accepted standard for software considerations in airborne systems, DO-178B is used [4]. The DO-178B certification objectives require developers to provide assurance that the development artifacts are verified and validated. While the airborne systems are certified, the tools used for their development may be qualified. When a qualified tool is used it may eliminate and/or reduce the need for verification of a software artifact it produces.

2. Research Methodology

The research started with collection of literature and informal data on the use and evaluation of software development tools, here focusing on the needs of the software intensive safety critical real-time systems. Extensive interactions with tool vendors and industry surveys were conducted. A variety of considerations of technical and managerial nature were a base to propose development tool taxonomy with the specific concerns, factors, and evaluation methods. The analysis of available tools led to acquisition and installation of the representative tools in the laboratory. In an initial experiment, described briefly below, graduate software engineering students were assigned a specific tool to develop simple typical avionic application. The data regarding the tool use, effectiveness in terms of learning and actual development, and traceability between the requirements, design, and code were a base for more extensive follow-up experiments.

3. Industry Feedback

A two-phase industry survey provided initial data on the development tool qualification. The DO-178B qualification process was criticized as being not suited to the industry needs. The rigor of data required for the development tool qualification, makes it impractical to qualify a COTS tool.

The functionality, cost, and compatibility with the development platform are the major factors for the tool selection. In the framework of DO-178B, the savings come from elimination/reduction of review activities. The survey shows that evaluation and potential adoption of new tools is driven by economic concerns and the high up-front cost is one of the major barriers of tool qualification. While the reuse of a qualified tool may lead to savings, the past practices do not show applicants focusing mostly on the current project. A company does not want to spend money to qualify a tool so that another company could reuse it. Only recently the tool vendors start showing interest in tool qualification and reuse.

4. Tool Selection

Two approaches are eminent in industrial applications. In the "software engineering" viewpoint a high-level structural design tool (e.g. *Rose RT*) is used to develop software architecture in a specific graphical notation (e.g. UML) as a collection of sequence diagrams, class diagrams, and state diagrams. In a "control engineering" paradigm, the algorithms are developed using function or block-oriented tools based on data flows (e.g. *Scade*). The tool can simulate the system behavior and evaluate its performance. Subsequently, the tool will generate source code to be compiled and loaded onto specific target machines.

+ The presented work was supported in part by the AACE under contract DTFA0301C00048 sponsored by the FAA. Findings contained herein are not necessarily those of the FAA.

Some tools serve only as code wizards and require the developer to enter specific code components, representing the behavioral aspects of the design, in a dedicated tool window. Other tools provide full automatic code generation without the developer writing a single line of source code. Considering the availability for the experiment the following tools were selected for evaluation: *MatLab* with *Simulink* and *Real-Time Workshop* by MathWorks, *Scade* by Esterel Technologies, *Sildex* by TNI-Valiosys, and *RT Studio Professional* by Artisan Software.

5. Experiment

To meet the objectives of DO-178B, the developers of safety-critical software are required to provide arguments of traceability of the artifacts from requirements to the design to the code. If one can be assured that the tool transforms the model to source code accurately, then verification of traceability can be reduced. The evaluation experiment was limited to the tool's extent in ensuring traceability between artifacts generated from one development phase to another. In a three-tier tool evaluation scheme [5] it is only *macro-evaluation* focusing on tool use.

The experiment used a simplistic project: embedded software (implemented on target VxWorks Arcom board) to capture and process data generated by a flight simulator (Opal RT). Engineering observations and effort data were collected. One observation was about excessive amount of time required to learn the tool, and the awareness of a large number of features that have not been mastered.

The tool automatic code generation functionality allows developers to focus on the higher level of abstraction rather than engaging in a mundane coding. The developers, familiar with the Personal Software Process, 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). Despite required learning period, the total project development was also completed in time. Using automatic code generation reduced the planned total development time in average over 12%.

Number of features, viewpoints, and the complexity of a tool may be overwhelming to a novice developer. A tool may have memory leaks or even bugs that cause it to malfunction, which would not interfere with creation of the model and generation of correct code. The experiment found inadequate coverage of a tool features and constraints in documentation and tutorials. The messages produced by tools, are often unclear and cryptic.

To assess the design-code traceability 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 was analyzed to identify parts that did not relate to model components, and their purpose was recorded. Different tools have varying levels of comments in source code to assist in traceability effort. The analysis shows that the traceability between design and the generated code is very much tool dependent.

6. Summary

Software development tools play an important part in the development of safety critical software. The quality of software engineers, the methods and the tools they use affect the quality of the produced software and, therefore, the overall system safety. As a consequence, the process for evaluating development tools is highly important and needs to be created. The assessment of these tools must be an essential part of the development process.

The objective of these investigations is to collect data on use of development tools in high assurance aviation systems. The tool classification and tool evaluation taxonomy shall be a base for creation of guidelines for the future tool qualification requirements.

7. References

[1] Barbacci M.., Klein M.., Longstaff T., Weinstock C. (1995), *Quality Attributes*, Report CMU/SEI-95-TR-021, Software Engineering Institute, Pittsburgh, PA., USA

[2] Federal Aviation Administration (1991), *Software Quality Metrics*, Report DOT/FAA/CT-91/1, FAA Technical Center, Atlantic City, NJ, USA.

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

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

[5] Kornecki A., Zalewski J., (2003), *Assessment of Software Development Tools for Safety Critical Real Time Systems*, IFAC Workshop on Programmable Devices and Systems, Ostrava, Czech Republic, February 2003, pp. 2-7