Process-Based Experiment for Design Tool Assessment in Real-Time Safety-Critical Software Development[*)]

Andrew J. Kornecki
Dept. of Computer & Software Engineering
Embry-Riddle Aeronautical University
Daytona Beach, FL 32114-3900, USA
kornecka@erau.edu

Janusz Zalewski
Computer Science Department
Florida Gulf Coast University
Fort Myers, FL 33965-6565, USA
zalewski@fgcu.edu

Abstract

The paper presents selected experimental results of evaluating six real-time software development tools for use in safety-critical systems. The experiments were designed to collect data, such as project effort, code size, functionality, documentation, traceability, etc., in four stages: preparation, model and code development, measurements, and post-mortem. Preliminary experiments were conducted to enable fully controlled experiments for the development of well defined but simple real-time systems. The results give the base for successful determination of tool quality and making preliminary conclusions on potential tool qualification.

1. Introduction

Development tools play vital role in the construction of software-intensive airborne and land-based systems. Developers use these tools to improve productivity and accelerate production and certification processes. Tool performance and quality may directly and indirectly affect the quality of the resulting target software, with significant impact on the overall system safety. The number and type of software development tools available in the commercial market is very dynamic with an array of tool vendors offering complex tools of an apparently similar functionality, but with often diverse characteristics and based on different design philosophy. Additionally, many companies developing avionics software have been using in-house created tools.

The general purpose of this research in the long term is to identify the assessment criteria and methods that would allow both developers and certifying authorities to evaluate specific safety-critical real-time software development tools from the system and software safety perspective.

The research focus is on the assessment of the development tools supporting the design and implementation phase of the software lifecycle, which starts with creation of the design model and ends with generating the source code. The specific objective of this study was to conduct an experiment to shed some light on the usefulness of assessing the tool with one particular criterion in a process-based model. The airborne software intensive systems are going through scrutiny of certification guided by the RTCA DO-178B standard [1]. The issue of tools supporting software development is addressed in these guidelines. However, the interpretation is still wide open since the guidelines were conceived well before explosion of the new software development methodologies and the introduction of most popular Model-Based Development paradigm. Model-Based Software Development

The paper is structured as follows. First, we concentrate on outlining the basic assumptions for conducting the experiments (Section 2). Next we discuss the preliminary experiment (Section 3), which is followed by a discussion of the controlled experiment (Section 4), and discussion of the results in conclusion (Section 5).

2. Basic Assumptions

Several assumptions have to be made initially to conduct a full-scale experiment. They are concerning primarily: the subject of experiments (software tool), the parameter(s) to be evaluated, and the methodology to conduct evaluations.

First of all, tools from a specific category have to be selected, possibly homogeneous, that is, with identical or very similar functionality. In this regard, the previous studies [2,3] provided enough material for selection of software design tools with code generation capability. In software development for avionics systems, two categories of tools have been used, those based on software engineering paradigm and those based on control engineering paradigm of software development. Examples of both categories include:

[*)] Supported in part by the Aviation Airworthiness Center of Excellence (AACE) under contract DTFA-0301C00048 sponsored by the Federal Aviation Administration (FAA). Findings contained herein are not necessarily those of the FAA.

- In the *Software Engineering* paradigm (structural, object-oriented):
  - *Real-Time Studio* from Artisan
  - *Rhapsody* from iLogix
  - *Rose RT* from Rational/IBM
  - *Stood* from TNI-Valiosys
  - *Tau* from Telelogic
- In the *Control Engineering* paradigm (functional, block-oriented):
  - *MatLab* (*Simulink*, *Stateflow, Real Time Workshop)* from MathWorks
  - *Scade* from Esterel Technologies
  - *Sildex* from TNI-Valiosys
  - *Beacon* from Applied Dynamics.

Due to confidentially and legal concerns, details and names of tools used in experiments are not identified. The purpose of this research is to develop general evaluation criteria – not to promote or criticize any specific tool. Also, as a result of volatility of the tool market some of the tools may not be available at the time of this writing.

The second assumption was the selection of specific characteristics of the tool, based on the use of an appropriate evaluation view for tool assessment. Some of the views we considered are presented in Figure 1. The *taxonomy view* concentrated on three groups of attributes for tool evaluation:

- functional attributes related to "what?" the user wants from the system;
- quality attributes related to "how?" the system fulfils the function regarding such criteria as dependability, performance, security, etc.
- business attributes related to "if?" there is a rationale of using the tool, describing the quality aspects of the software without considering its functionalities.

For assessment of these attributes, a typical procedure can be aplied that involves using *metrics* (measurements units) and *measures* (evaluation procedures) [4].

The *project view* considers how well a software tool fits into the specific project. Several characteristics of the tool are normally considered in the evaluation process, including: language, completeness of code generation, self-documentation, learning curve, communication methods, lifecycle integration, vendor support, etc.

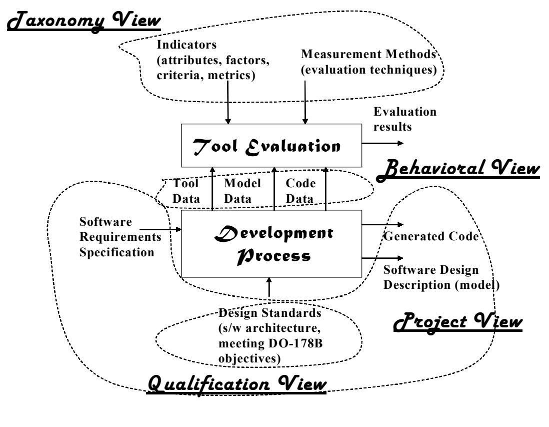
Taxonomy View
Indicators (attributes, factors, criteria, metrics)
Measurement Methods (evaluation techniques)
Evaluation results
Tool Evaluation
Behavioral View
Tool Data
Model Data
Code Data
Software Requirements Specification
Development Process
Generated Code
Software Design Description (model)
Design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

Fig. 1: Tool Evaluation Using Four Different Views.

The *qualification view* considers qualification of a development tool using RTCA DO-178B guidance criteria [1]. Accordingly, the specific concerns (criteria) included in the evaluation process are: traceability, determinism, robustness, correctness, and conformance to standards.

Finally, the *behavioural view* represents the observation that the extent to which the tool is capable of representing the requirements faithfully in the design (not introducing faults into it) is best viewed by observing the tool's behavior in use during the design process. To evaluate the tool in use (i.e., during its operation) several steps should be performed:

(1) adopt a model of a typical application being developed by the tool
(2) develop a model of taking measurements
(3) collect results of developing this application, and
(4) analyze these results.

Our model of the application is based on the Integrated Modular Avionics ARINC Specification 653 [5], and the model of measurements relies on metrics based on rough sets theory, both being currently developed.

Since most of the views include assessing the traceability property, this attribute was selected as a criterion for evaluation experiments.

The third assumption, vital for conducting the experiments, is the adoption of an appropriate evaluation methodology. In this project, we conducted the experiments in the following seven steps:

- Tools and Platform Preparation: acquisition of sample software development tools from the selected category, installation of the tools, and preparation of the experimental platform.
- Experiment Preparation: development of the process and scripts for the subsequent experiments.
- Initial Experiment: conducting the initial experimentation.
- Experiment Improvement: identification of he tool assessment methodology and related assessment mechanisms.
- Controlled Experiment: conducting the controlled experiment and collecting data.
- Data Integration and Data Analysis: analysis of the data and documenting the experimental process and results in a report

In the following sections, we present both the preliminary experiment and the controlled experiment, and discuss the results.

3. Preliminary Experiment

The experiment's objective was an initial evaluation of the selected tools representing real-time 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 to develop a real-time program to be implemented in a VxWorks target environment. VxWorks provides a runtime environment for embedded application development, which comprises the core capabilities supporting a full range of real-time features including multitasking and interrupt handling, along with pre-emptive and round-robin scheduling.

The focus of the experiment was on learning and exploring capabilities of software tools used in the process of developing and implementing a 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 may indicate the software tool's quality. Through the development of the sample system, the developers collected the design artefacts 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 were used to infer on the tool's quality and constituted the base for future controlled experiments.

3.1 Project Description

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. The user from pre-defined menu of options selects the frequency of the moving average computation and which three of over twenty parameters are to be captured. The parameter values and averages would be displayed with a timestamp. Each developer implemented the following requirements: (1) timing requirement; (2) system requirements, and (3) external interface requirements, using a different tool (A, B, C, D):

1) The system shall collect two data packets from the serial port every second (2 Hz), at the same frequency in which the *TestFlight* simulator sends the data; when appropriate, the system shall prioritize this activity in order to fulfil this requirement.
2.1) (a) Upon receiving each data packet, the system shall record the current timestamp. (b) The timestamp shall be presented in the format HH: MM: SS. *Note.* The timestamp should reflect the time at which the data is received.
2.2) (a) The system shall present to the user a menu option to select the parameter(s) for moving average calculation. (b) The system shall allow the user to pick up to three parameters once during the initialization.
2.3) (a) The system shall also present a menu option for the user to select the frequency of the calculation in x data packets/calculation. (b) Each parameter shall be specified with its own calculation frequency. (c) This option shall be given only once during system initialization.
2.4) (a) A timestamp shall be recorded with the moving average results upon the completion of the calculation. (b) The timestamp shall be presented in the format HH:MM:SS.
2.5) (a) The system shall output the selected parameters with their names and timestamps, as well as the moving averages with timestamp to the terminal output, as in Figure 2. (b) The result shall be displayed as a floating-point value with 3 significant digits precision, with each set of data per timestamp on one line, and a set of moving averages on another line of the display.

```
Airspeed: 300 knots; Altitude: 10,000 ft
          Timestamp: 13:12:43
Airspeed: 300 knots; Altitude: 10,004 ft
          Timestamp: 13:12:43
Airspeed: 299 knots; Altitude: 10,009 ft
          Timestamp: 13:12:44
Airspeed: 300 knots; Altitude: 10,014 ft
          Timestamp: 13:12:44
Airspeed: 300 knots; Altitude: 10,018 ft
          Timestamp: 13:12:45
Moving Average of Airspeed: 299.800 knots
          Timestamp: 13:12:45
```

Fig. 2. Sample Output of the System.

3.1) (a) The system shall communicate with TestFlight through the RS-232 port. (b) The configuration of the serial port shall be set to 9600 baud rate, 8 data bits, no parity, and 1 stop bit, and no flow control. (c) A data stream shall consist of the following:
- value 0x55, an unsigned char value of 1 byte
- the number of parameters (N) sent, an unsigned char value of 1 byte
- 1st parameter values in 8 bytes double type
- 2nd parameter values in 8 bytes double type
- …
- Nth parameter values in 8 bytes double type

3.2) Since the starting and stopping of the data stream is controlled within the *TestFlight* system's GUI, the data collection system has to have no on/off control of the data flow.

A process script 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 and Tool Familiarization, (2) Model Creation and Code Generation, (3) Measurement, (4) Postmortem.

3.2 Preliminary Experiment Results

The experiment used two basic methods of evaluation. First, engineering observations were made throughout the development to identify any perceived strengths and weaknesses of the tool, processes used, and any other related concerns. These observations mainly relate to the developer's acceptance of the tool operation, ease of understanding, support of the development methodology, help in development, availability of notations to represent the system, etc.

The second method focused on the quality of tool to properly translate the requirements into design models and subsequently into the target code. All software requirements were traced to specific model components. The created model components were compared and mapped to the code segments generated by the tool (objects methods or function blocks) that represent them. Any component that did not map directly to a section of code was then checked against the generated code to identify any code the 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 purpose of any non-traceable function/code. With this approach, the relationship between the requirements, design, and code was established.

The aggregate results are shown in Table 1. The developers, using well known to them Personal Software Process (PSP) [6], 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 on average about 72 hours to be dedicated to the design and coding phase. An actual average for this phase was less than 39 hours. Automatic code generation reduced the development time at the order of 46%. The average code size was about 1.8 KLOC. The average total time spent on 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 the tool). It is interesting that despite the steep learning curve the total project development was also completed on time. Use of automatic code generation reduced the planned total development time on average by 12%.

Table 1. Preliminary Experiment 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
Development effort			
LOC/hr			12.492

4. Controlled Experiment

The experiment objective was a more detailed evaluation of real-time software design tools with automatic code generation capability. The selected sample included six tools from both structural (object-oriented) and functional (block-oriented) categories. Tools L, M and N are object-oriented, tools P and Q are block-oriented, and tool O crosses the boundary of two categories. Four of the tools used in the Preliminary Experiment (listed there as tools A, B, C, D) were used again for this Controlled Experiment. The following is the tool equivalency: A = L, C = Q, D = P, and D = M. One tool was used only in Preliminary Experiment (B), and two tools were used only in the Controlled Experiment (N and O).

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/three developers who shared the initial training and the final reporting. However, each of them developed the model and implemented code as an individual assignment.

4.1 Project Description

The experiment consisted of developing two models. The first model, a simple hair dryer simulator, was used during the learning phase of the experiment, to facilitate the learning and constituted a capstone for familiarization with the methodology, tool, and the 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 software simulator, was used for the actual design and data collection.

Specification of the first model, hair dryer simulator, consisted of the following requirements:

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

Specification of the second model, simple microwave oven simulator, included the following:

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 from default 1 to 5).
3. The start of cooking shall initiate on 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: the power level of 5 means that the emitter is on all the time, the power level of 1 means that the emitter is on only 1/5th 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)

Suggested interface included:

- Inputs: TIME, POWER, START, RESET, 0-9, DOOR
- Outputs: TIME, POWER, SOUND, LIGHT, MOTOR, EMITTER

As in the preliminary experiment, a process script was given to each developer, which had the four following top-level tasks: (1) Preparation, (2) Model Creation and Code Generation, (3) Measurement, (4) Postmortem. This script is a refinement of the one used in the preliminary experiment, based on feedback from the participants. An overview of the script tasks is presented below.

(1) Preparation.

(1a) Creation of PSP estimates of time and code size for the project.

(1b) Tool selection and becoming familiar with the project requirements.

(1c) Learning to use the tool and identifying available resources that can be used during development.

(1d) The development of the demonstration hair dryer model as a learning aid.

(2) Model Creation and Code Generation.

(2a) The microwave oven model was created according to the specified requirements.

(2b) Each developer was to manually verify that all requirements were covered in the model. If the tool provided verification capabilities, they were to be used if possible.

(2c) The code-generation capabilities of the tool were then to be used to generate C code for the model.

(3) Measurement:

(3a) Decomposition of the design model, which was then analyzed for traceability.

(3b) Decomposition and traceability to the model.

(3c) Decomposition and traceability to the requirements.

(3d) Identification of code that did not have a representation in the model.

(4) Postmortem:

(4a) Completion and analysis of PSP data.

(4b) Assessment of the tool's conformance to traceability.

(4c) Compilation of each developer's individual data into a joint report summarizing their findings.

In addition to completing the process task, the participants were required to complete two questionnaires. The first questionnaire addressed the documentation, manuals, and support of the tool under evaluation. The second questionnaire addressed the code generation capabilities of the tool. The application of each of these methods was described in the process script used for the experiment.

4.2 Controlled Experiment Results

The results of the controlled experiment were collected for the six tools as follows:

a) Size and Effort: The number of lines of code generated by the tool from the user-designed experiment model (microwave oven model only), the time spent in the experiment for each process phase, the overall time spent by each developer, and an average of each measure.
b) Developer Subjective Assessment (on scale 1-5) extracted from questionnaires with the results grouped into the following four categories:
   - tutorial (Tool Questionnaire: Q2 – Q4)
   - user manuals and reference (Tool Questionnaire: Q5 – Q7)
   - readability (Automatic Code Generation Questionnaire: Q1 – Q3)
   - functionality (Automatic Code Generation Questionnaire: Q4 – Q6).
c) Engineering Observations
d) Traceability
e) Questionnaire Comments

It needs to be noted that the presented results are based on a rather small observation sample. As such, the results give only an approximate assessment of the tool and do not have any statistical significance.

From the perspective of the development paradigms used for these tools, i.e. object-oriented or block-oriented, the developers' effort seems to be related to the paradigm itself. Tools L, M and N are all based on object-oriented approaches and, with the exception of tool M, the time spent on development is very similar. The difference in the effort while using tool M was attributed to the learning curve associated with the particular object-oriented methodology used by the tool, slightly different than the more familiar UML. For the functional, block-oriented tools, O, P, and Q, the effort is also similar across the phases, and on average is less than the effort for the object-oriented tools. Tables 2(a) and 2(b) show the productivity resulting from the developers effort and the size of code generated by the tool. It is important to remember that the tools automatically generated the code, with little or no manual coding by the developers.

Across the evaluated tools, the overall ratings were all on the low side (Tables 3 and 4). This indicates that despite the advertised capabilities of these tools, the available resources for developers to make effective use of the tools are not sufficient or are of poor quality. All the developers also mentioned this in their feedback – this happened to be a particular problem during the preparation phase where the need for these materials was the greatest.

Table 2(a): Tool Controlled Experiment – Effort Analysis, Tools L – N (in hours)

LOC	**Tool L**	**339**		**Tool M**	**159**		**Tool N**	**3007**	
	plan	***actual***	**% change**	***plan***	***actual***	**% change**	***plan***	***actual***	**% change**
Preparation	17.5	14.3	-18.40	32.3	50.1	55.44	16.8	17.3	2.99
Model/Code	9.3	17.8	91.89	19.5	25.5	30.77	15.5	14.5	-6.71
Measurement	7.0	4.0	-42.86	8.0	8.3	3.13	9.0	5.7	-37.00
Postmortem	8.0	10.3	28.13	13.5	14.2	4.96	7.5	6.5	-13.33
TOTAL	**41.8**	**46.3**	**10.85**	**73.3**	**98.1**	**33.86**	**48.8**	**43.9**	**-9.99**
Dev Effort									
LOC/hr			***7.325***			***1.622***			***68.528***

Table 2(b): Tool Controlled Experiment – Effort Analysis, Tools O – Q (in hours)

LOC	**Tool O**	**11,227**		**Tool P**	**482**		**Tool Q**	**1,293**	
	plan	***actual***	**% change**	***plan***	***actual***	**% change**	***plan***	***actual***	**% change**
Preparation	4.0	9.0	125.00	12.3	12.7	3.09	8.4	8.8	4.87
Model/Code	11.0	16.5	50.27	6.3	7.6	20.16	2.8	6.0	113.26
Measurement	4.0	3.0	-25.00	4.3	3.3	-22.79	4.7	4.2	-10.71
Postmortem	5.0	7.0	40.00	4.7	5.0	6.42	2.7	3.3	24.72
TOTAL	**24.0**	**35.5**	**48.04**	**27.6**	**28.5**	**3.52**	**18.6**	**22.3**	**20.11**
Dev Effort									
LOC/hr			***315.986***			***16.889***			***58.034***

Table 3. Tool Controlled Experiment – Tool and Auto Code Generation Questionnaire Results

	L	**M**	**N**	**O**	**P**	**Q**
Tutorial	3.2	0.3	3.3	2.3	1.7	2.1
User Manuals & Reference	2.8	2.7	1.7	2.7	3.0	3.0
Readability	3.2	2.7	4.0	2.0	3.4	1.3
Functionality	3.5	3.0	3.3	1.7	2.2	1.8
AVERAGE	3.2	2.2	3.1	2.2	2.6	2.1

4.3 Some Lessons Learned

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 mindset was already focused on the problem at hand and the task then became fitting the tool into the problem solution.

Table 4. Tool Controlled Experiment – Average Questionnaire Results

	Average
Tutorial	2.16
User Manuals & Reference	2.64
Readability	2.77
Functionality	2.59
AVERAGE	**2.54**

Learning was also an issue due to the lack of sufficient reference materials from tool vendors. This reflects poorly on the state of the tool industry, as this was a problem for all the developers regardless of the tool being used. In engineering observations and questionnaire responses, developers noted that insufficient materials and support might prevent them from using the tool again. Almost all developers recommended that improvements in this area were necessary for future releases and that verification of documentation accuracy should become a priority for tool vendors. It needs to be noted that the tool vendors typically support the tool by offering the purchasing organization, as a part of the package, a hands-on 3-7 days intensive training for the developers. Such approach may alleviate some of the above-mentioned problems and is perhaps the reason why the quality of documentation is of lower priority for the tool vendor.

The process used for the controlled experiment was a refinement of that used in the preliminary experiment. As before, areas of the process that are still difficult to implement will be subject to process improvement in the future experiments. This effort ensures that future revisions will enable developers to perform more objective evaluations and yield more useful results. This statement also applies to the questionnaires used in the evaluation process. Improvement of these questionnaires is vital if they are to be useful in making determinations about the development tools under assessment.

5. Conclusions

Software development tools play an important part in the development of safety-critical software artefacts leading, ultimately, to executable code. A significant issue is to provide assurance that the translation of designs to code, typically achieved with assistance of software development tools, does not introduce faults and accurately creates the executable software according to specifications.

The purpose of the present study has been to address some of the problems and opportunities in evaluation of a certain category of software development tools, that is, real-time design tools with code generation capability. Software design tools supporting model-based development with an automatic code generation capability are the fastest growing category of development tools and are extremely popular in the software development community. This study focused specifically on this category of tools because of the growing interest and use of these tools in the aviation industry. The research work made it possible to gain valuable experience in the application of selected development tools and led to observations about software development practices.

The objective of evaluating the software design tool with respect to safety is to check how precisely the design developed with a tool can represent the requirements and assist with creating the correct software product, without introducing faults on its own. Since the design process and its tools cannot necessarily detect or resolve incorrect requirements, validation of the requirements is typically covered by another set of tools and was, therefore, outside the scope of this project.

The first practical step in the tool evaluation was composed of the following five activities:

- identification of sample criteria to be evaluated (e.g. traceability),
- development of a method to evaluate each criterion,
- selection and acquisition of real-time design tools suitable for evaluation,
- application of each tool to the development of a small project, and
- collection of results.

The tool evaluation experiments used rather simplistic projects of developing embedded software. The traceability assessment included in the preliminary experiment was an activity to manually trace the line(s) or section(s) of the code that fulfil a particular requirement, and to evaluate 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 will be too time-consuming for practical purposes.

The developers were also collecting engineering observations and data on effort and product size. A common observation was about an excessive amount of time required to learn how to use the tool, and the awareness that there are still a large number of features that have not been learned or mastered. Although some of the developers had to deal with the tool software abruptly crashing or with degradation in performance as a result of memory leaks, they were satisfied, in general, with the capability of the selected tools in helping them in developing the target software and accomplishing the process of traceability.

The data collected from both the preliminary and the controlled experiments show that the selected software design tools with code generation capability can significantly assist developers in their work. The data collected by the group of researchers familiar with metrics of the Personal Software Process (PSP) show that tool use reduces the development effort. At the same time, the property of traceability can be shown to support claims about the tool validity.

Further research work will focus on:

- integrating the currently developed tool evaluation criteria into a coherent set of metrics (in sense of IEEE definitions [4]),
- 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.

There is an evident risk in the proposed approach due to the fact that such an evaluation methodology is currently non-existent. Moreover, there are no well-defined criteria for evaluating software tools (or any software, for that matter) with respect to safety. On the other hand, the need for procedures for tool evaluation is such that developing a practical handbook for use by industry and certification authorities on actual tool projects, with focus on code generation tools, may offset these risks. The need for more detailed evaluation experiments and collection of data on the concerns and characteristics defined in the taxonomy research has been addressed in a recent Tool Forum [7].

Acknowledgement

The authors would like to acknowledge the support of the Aviation Airworthiness Center of Excellence program and the FAA personnel: Leanna Rierson, Barbara Lindberg, and Charles Kilgore. The paper resulted from a team effort of faculty and graduate students of ERAU Software Engineering program: N. Brixius, J.P. Linardon, J. Labbe, L. Crawford, K. Hall, H. Lau, D. Hearn, C. Sanouillet, S. Lakha, J. Poole, T. Osako, A. Abou-Rahma, K. Madler, B. Kim, O. Rettig, J. Erwing, R. Atwood, A. Moore, P. Hoden, and T. Islam.

References

[1] RTCA, *Software Considerations in Airborne Systems and Equipment Certification*, Report RTCA/DO-178B, Washington, DC, 1992

[2] 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 2003, pp. 2-7.

[3] Kornecki A., J. Zalewski, "Design Tool Assessment for Safety-Critical Software Development", *Proc. SEW-28 2003 Annual NASA Software Engineering Workshop*, Greenbelt, MB, December 3-4, 2003, pp. 105-113.

[4] IEEE Std. 1061, *Software Quality Metrics Methodology*, IEEE, New York, 1998

[5] ARINC Specification 653 - *Avionics Application Software Standard Interface*, ARINC Inc., Baltimore, MD, 1997

[6] Humphrey W. *Introduction to the Personal Software Process.* Addison-Wesley, 1997

[7] *Software Tools Forum*, Embry-Riddle Aeronautical University, Daytona Beach, FL, May 18-19, 2004, http://www.erau.edu/db/campus/softwaretoolsforum.html