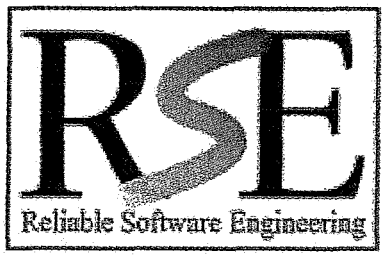


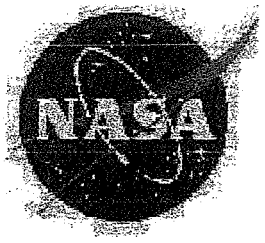
ESAS Project 6G

ESAS Deliverable PS 1.1.2.3

Customer Survey on Code Generators in Safety-critical Applications

Johann Schumann, schumann@email.arc.nasa.gov
Ewen Denney, edenney@email.arc.nasa.gov

March 31, 2006



National Aeronautics and
Space Administration

Contents

List of Tables

List of Figures

1 Executive Summary

This deliverable describes a customer survey which was carried out by the RSE program synthesis team in March 2006. The project team set up an online questionnaire on the use of automated code generators (ACG) in safety-critical application areas and emailed an invitation to participate in the survey to developers, tool vendors, researchers, and managers at NASA, leading aerospace companies, and other related industries, as well as tool providers and research institutions. People who have worked in the area of software development and have extensive experience with code generators were specifically targeted.

We received 23 responses; several colleagues were also contacted directly by telephone – their responses have informed our analyses here, but the statistics are based solely on the survey.

This report presents the objectives of the survey, summarizes the results of the returned questionnaires. The results of the survey indicate that the use of automated code generation is now widespread and being actively used by industry. Moreover, there are many safety-critical applications where ACG has been applied to generate production/flight code. While the most commonly used tools are the integrated modeling, analysis and simulation tool suites provided by commercial vendors (primarily Real-Time Workshop, MatrixX, and SCADE), in-house extensions of these tools are common, particularly to address modeling and verification issues. Indeed, there appears to be a natural synergy of ACG with extensions to support certification activities, namely the production of V&V artifacts, documentation, and traceability information.

In most cases, respondents were generally satisfied with the tools, although substantial effort is typically spent both in learning to use the tools, and in modifying existing development and V&V processes as actual V&V tasks can differ substantially. So, using ACG for a safety-critical application is a long-term decision, as in most cases, benefits won't show up immediately and the V&V process needs to be planned carefully. Although none of the tools were qualified, auto-generated code has itself been successfully certified to meet DO-178B (Level A)[1].

[1] `http://www.rtca.org`

2 Introduction and Objectives

Automated code generators (ACG) are tools that convert a (higher-level) model of a software (sub-)system into executable code without the necessity for a developer to actually implement the code. Although both commercially supported and in-house tools have been used in many industrial applications, little data exists on how these tools are used in *safety-critical* domains (e.g., spacecraft, aircraft, automotive, nuclear).

The aims of the survey, therefore, were threefold:

- to determine if code generation is primarily used as a tool for prototyping, including design exploration and simulation, or for flight/production code
- to determine the verification issues with code generators relating, in particular, to qualification and certification in safety-critical domains
- to determine perceived gaps in functionality of existing tools

In order to obtain this information, we designed the questionnaire[2] around the following major topics:

1. What are the characteristics of the project where an ACG has been used? This includes application domain and area (e.g., control, signal processing, user interfaces, glue code), level of criticality, size of the model and generated code, and the relative percentage of auto-generated code within the entire project.
2. Which tools have been used to generate the code and which other artifacts have been auto-generated (e.g., documentation, test cases)?
3. What was the experience of using ACG? In this section, we asked about benefits of using ACG (in terms of effort and reliability), where the major problems were, and how the use of ACG influenced the software development process.
4. How did you address V&V and certification? Safety-critical code must pass tests, reviews, and certification (depending on the application) in order to demonstrate its quality and reliability. In this part of the

[2]The full questionnaire is shown in Appendix A.

questionnaire, we asked how the quality of the generated code was assured, how the use of ACG impacted verification and validation, and what safety properties are most important.

5. What would the features of an ideal ACG be? In this section we asked about a "wish-list" of features for an ACG.

In the following, we will present statistics on the application domains and participants, paraphrasing participants responses there they permitted this, and then present the detailed results. The charts in this report were produced by the report generator of `http://www.formsite.com`. Most questions allowed the participant to select more than one answer. Thus, unless stated otherwise, percent figures refer to the total number of times, the specific answer was selected.

3 Application Areas and Respondents

The majority of the results in our survey were obtained from participants working in the area of aviation and space (see Figure 1), followed by other safety-critical industries. Other domains included high-integrity distributed real-time systems, design/analysis/visualization tools, compilers/checkers, encoders/decoders, and educational/research.

Most of the participants who responded were from the US, followed by the EU and Japan. Based upon the contact information (where given), roughly 60% of the participants were from industry, 20% from research institutes, and 20% from US government institutions. Figure 2 shows the roles of the respondents within their organizations.

4 ACG application characteristics

This section discusses the survey results on the characteristics projects where code generators have been used, including application area (e.g., control, signal processing, user interfaces, glue code), level of criticality, size of the model and generated code, and the relative percentage of auto-generated code within the entire project.

The vast majority of ACG applications in safety-critical areas are in the area of control (Figure 3. This result is not surprising, because (a) digital

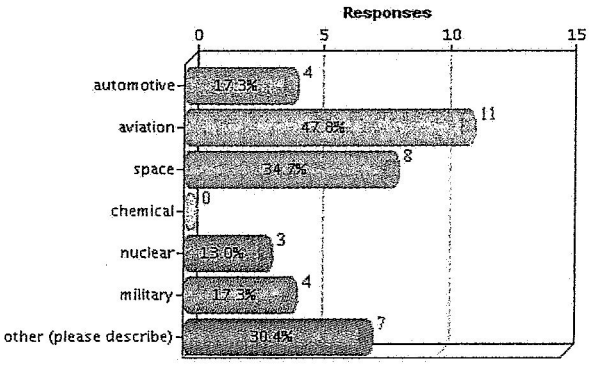


Figure 1: Target domains for code generation

controls comprise a major portion of highly critical code in aircraft, spacecraft or cars, and (b) major commercial modeling tools, like Simulink, MatrixX or SCADE, are traditionally used in the controls domain.

Other applications included mission management, distributed on-board systems, embedded systems, distributed systems, and user interfaces.

A code generator can also be used for various purposes (Figure 4). The most common are to generate production code that is deployed (74%), to quickly produce prototypes (during the design phase, 52%), simulation and modeling (48%), i.e., to speed up modeling and simulation runs, testing (30%), and the generation of glue or interface code (30%). Other, "non-standard" uses of ACG cited were formal verification (generation of verification tasks from models), generation of artifacts for testing of the specification, and for integration and visualization purposes.

In highly critical applications (10 out of 23 responses), the focus on generation of production code was even stronger. 80% of these projects used ACG to generate production code, 70% for design and prototyping, and 60% for simulation and modeling.

Figure 5 shows the level of safety-criticality (mean is 2.5) of the application on a scale from 1 (highly safety-critical) to 5 (not safety-critical). As expected, many applications were highly safety-critical, most originating from Aviation and Space. Some applications, such as the flight-control software for the NASA F-15 ACTIVE within the IFCS project even concern

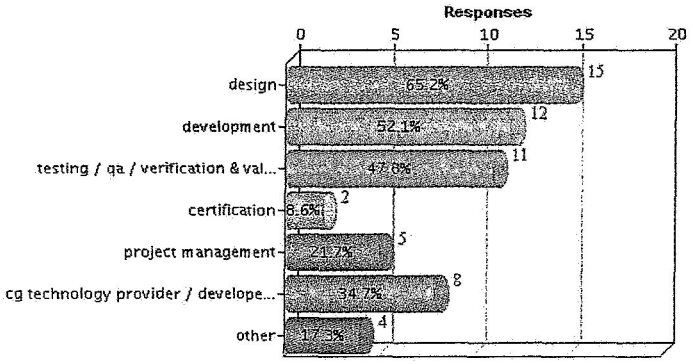


Figure 2: Roles of survey respondents

man-rated systems.

For the most safety-critical projects (level 1), MatrixX, closely followed by Real-Time Workshop and in-house tools were the most common. The most common domain was aviation, and then space, while control was overwhelmingly listed as the most common application. The most common target languages were C and then Ada (see also Section 5.2). Perhaps surprisingly, code generators are reported as being used for production code almost as much as for prototyping.

5 ACG tools

This section of the questionnaire concerns the code generation tools that have been used in the projects and the characteristics of the models and the generated code (e.g., sizes of the model, target systems, generated artifacts).

5.1 Most Common Tools

The most commonly cited commercial code generators (see Figure 6) were:

Real-Time Workshop: This tool[3] is a part of the MathWorks tool suite,

[3] `http://www.mathworks.com`

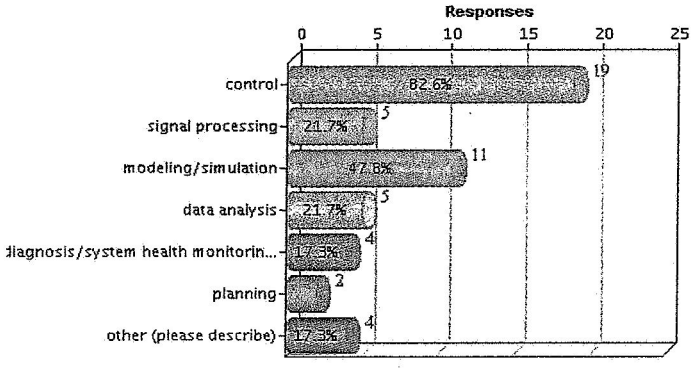


Figure 3: Application areas

based on Matlab and Simulink. Models are developed using the graphical Simulink and Stateflow representation and can be automatically compiled into C code that can be executed on the target platform (e.g., the flight hardware). This tool combines techniques for modeling and simulating continuous components (e.g., PID controllers or signal-processing functions) with discrete mode logic and is usually used for the development of digital control and avionics software.

MatrixX: MatrixX is a tool suite by National Instruments[4] for the model-based design, simulation and code generation, specifically targeting the area of control. Models are developed using a graphical (block-based) modeling environment. The models can be simulated interactively and analyzed. The MatrixX code generator (AutoCode) generates optimized C or Ada code. with automatic code generation.

SCADE: Though not listed specifically in Figure 6, this tool was the third most commonly cited tool, used in 15% of the projects. SCADE[5] is a graphical modeling and design tool suite. Developed by Esterel Technologies, it has been applied to numerous applications in automotive

[4] `http://www.ni.com`

[5] `http://www.esterel-technologies.com`

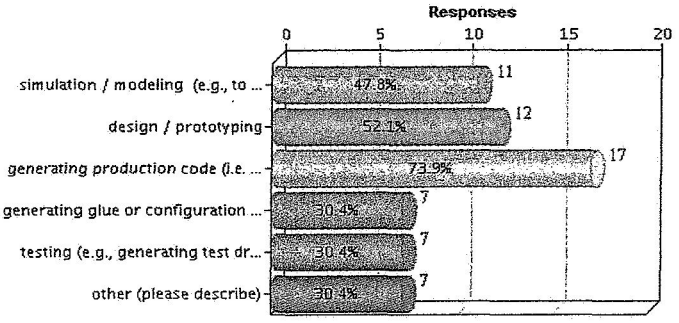


Figure 4: Purpose of ACG

and aerospace[6] industries, mainly in Europe (e.g., Airbus, Eurocopter).

Rose/RT: This tool is a code generator for the Rational Rose UML modeling tool[7]. The software system is designed and modeled using a graphical UML notation (e.g., class diagrams, statecharts, sequence diagrams), from which object-oriented code (C++ or Java) can be generated automatically.

Other tools and tool generators mentioned were Vanderbilt GME[8], Telelogic Tau[9], Semantic Design's DMS Toolkit[10], as well as CASE tools like Ilogix's[11], Rhapsody and Statemate, as well as STOOD/Sildex[12]. Furthermore, generic environments and languages like Eclipse[13], Alloy[14], or Mathematica[15] have also been used in this area.

As the responses showed, a significant number of projects used in-house tools, or combined commercial tools with in-house tools. Often these tools

[6]Although a qualified version (DO-178B, Level A) of this tool is available, it has not been used by the participants of our survey.

[7]`http://www.ibm.com/software/rational`

[8]`http://www.isis.vanderbilt.edu/projects/gme/`

[9]`http://www.telelogic.com`

[10]`http://www.semdesigns.com/Products/DMS/index.htm`

[11]`http://www.ilogix.com`

[12]`http://www.tni-software.fr`

[13]`http://www.eclipse.org`

[14]`http://alloy.mit.edu`

[15]`http://www.wolfram.com`

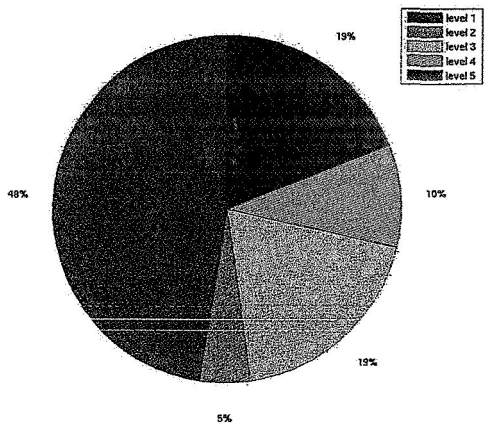


Figure 5: Levels of safety-criticality: % of responses for criticality levels 1 (= most critical) to 5 (= non-critical).

were implemented as scripts (e.g., using Perl) to address specific issues. In-house tools have been used to extract specific model information, to address specific target issues (e.g., to generate assembly code from Simulink models), or to generate non-standard artifacts. An example of this is the generation of code for analysis and verification tools (e.g., model checkers and theorem provers) from Simulink/Stateflow and SCADE models. Also, more elaborate in-house code generators (e.g., from MagicDraw UML diagrams to C, JPL) are being used in safety-critical applications.

5.2 Target Languages, Target Systems, and Generated Artifacts

In the majority of cases, ACG tools generated C, followed by C++ and Ada (Figure 7). Other languages mentioned: OCaml/MetaOCaml/FORTRAN, Relay ladder logic, Mathematica, XML, PARLANSE, as well as various logic-based languages.

For highly safety-critical applications, the most common target languages were C (70%) and Ada (30%).

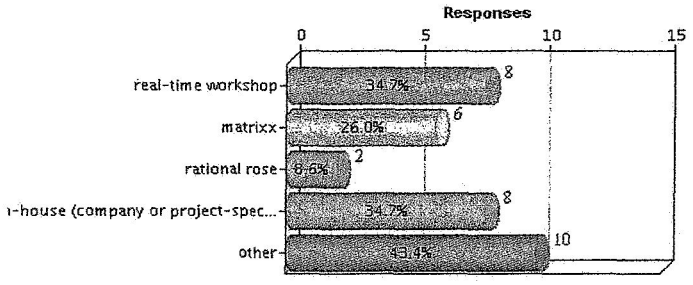


Figure 6: Tools used for ACG

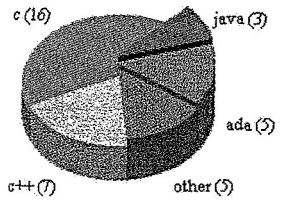


Figure 7: Target languages

The survey shows that code has been generated for a multitude of platforms. Quite often, the same models are used for different platforms (e.g., a desktop machine for simulation and flight hardware for the production code).

Desktop systems used as targets were running under Solaris, Linux, VxWorks, or Windows NT. Dedicated hardware systems are often project- or mission-specific. M68K and PPC processors were most often mentioned for embedded systems, in particular for highly safety-critical projects; the software running on those systems varies, but VxWorks is the most popular choice.

An ACG obviously needs to generate code in the desired target language. Most tools, however, are capable of automatically producing additional artifacts, like documentation, test cases, or installation scripts. Figure 8 shows the results. No specific responses were given for "Other".

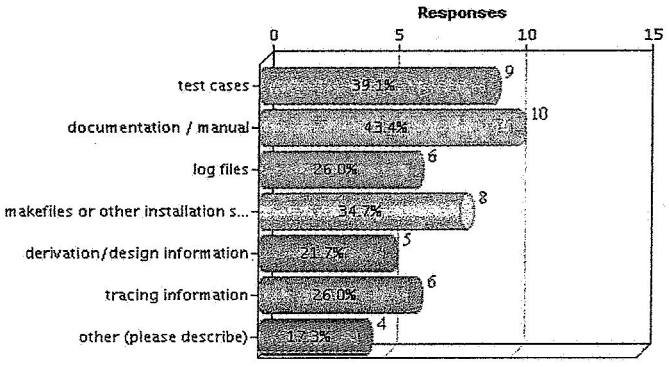


Figure 8: Additional artifacts produced by ACG

5.3 Sizes

The sizes of the models and the generated code varies widely between the different projects. For block-based graphical models (e.g., Simulink or MatrixX), the given sizes ranged from 5 to more than 16,000; for state-based specifications (e.g., statecharts), the number of states ranged from 5 to 1000; text-based specifications had up to 35,000 lines. This is a strong indication that ACG is used not just for very specific components or toy examples, but is used for large and complex models. Therefore, the number of generated lines of code ranges from 300 to 1.6MLoC. No tool-specific model to code ratio could be detected in the survey responses.

5.4 Generated Components within the Project

In a software project, usually only a part of the code is generated automatically. The answers in the survey ranged from 15% to 95% of the total software (with a mean of 50%).

Figure 9 shows which parts of the system in particular were auto-generated. Telemetry, instrumentation code, and user interfaces were other targets listed for automatic code generation.

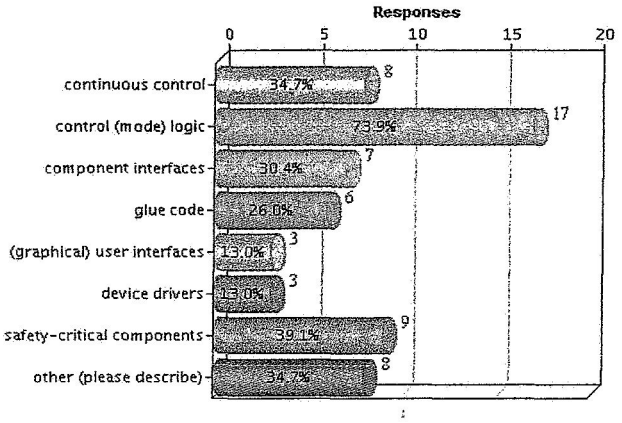


Figure 9: Software parts generated by ACG.

6 Experience with ACG

In this section, we asked the survey participants about the experience with using ACG in safety-critical applications, major benefits and problems encountered.

6.1 Overall Experience and Benefits

As might perhaps be expected, given the targeted nature of the survey, the reported experiences with using ACG were generally very good, even (and particular) for safety-critical projects. An average response of 1.5 (where 1 is the best on a scale of 1 to 5) was obtained.

It appears that those early adopters who are willing to digest weighty product manuals and deal with occasional quality issues to adapt the tools to their project are quite satisfied. Indeed, several users reported that they simply could not carry out some activities without autocoding, such as running simulations or maintaining huge amounts of (auto-generated) code.

The acceptance of autocoding is assisted by an increasing trend towards the use of models in the software development process, and the increasing feasibility of automated verification technology.

Specific benefits listed included quantifiable improvements to the development process (schedule, cost, productivity). Specific aspects of this were a reduced need for testing and a rapid turnaround from design to code.

Improvements in software quality were also reported (reliability, performance, reduced redundancy, and adherence to standards).

A number of advantages mentioned hold more generally for a model-oriented approach, namely a reduced maintenance burden through smaller specifications, ease of understanding, graphical programming, and the ability to detect faults earlier in the development process.

6.2 Quantifiable Benefits

Major benefits for using ACG are *productivity gain over manual development*, *reduction in defects*, and *faster time-to-market* for the project.

Although most participants qualitatively agreed to these benefits, no consistent qualitative answers could be extracted, and some of the data were not revealed due to proprietary concerns. Where actual numbers were given, the productivity gain ranged from 15% to 1000%, where values around 100% seemed to be most common.

Reduction of defects ranged from 5% to 95%. Some participants reported a significant reduction in defect rate or the benefit that no new errors were introduced by ACG use. One participant claimed a reduction of error from 1E-3 (manual) to 1E-5 (ACG) bugs per LOC.

Benefits in schedule and time to market can be substantial; 15% benefits in schedule and 6 month reduction in development time was reported. Although most participants agreed that there are benefits in schedule and time to market, it seems to be very application/project specific and hard to quantify. For some projects, no benefits could be measured due to considerable inherent domain complexity, for others, it would have been "impossible to get to market without ACG".

6.3 Learning to Use ACG

Though code generation is often touted as a push-button technology, in reality there is often a substantial learning curve for a project team to adapt and efficiently use a tool.

We assessed this by asking about difficulties in learning to use generators and in adapting existing processes. With a rating of 2.5 on a scale from

1-5, the use of ACG was not impossible to learn, but there is a substantial learning curve and effort has to be spent on introducing ACG.

Also the software process must change to accommodate ACG use. The impact of this change was rated 1.8, which means that ACG uses causes a substantial impact.

Most difficulties in learning and introducing the ACG technology were caused by the following issues:

- The major problem is using them seems to be the need to adapt existing software development processes to accommodate code generation. This can be an enormous task and is exacerbated when certification must be considered. One respondent stated that iterative customization of the generator should be seen as an integral part of the development process.

- The generators, themselves, typically need to be adapted, both in order to generate acceptable code which existing models often have to be adapted so they will be accepted by the generator. Indeed, actually understanding what can be adapted and what is a fixed "bias" in a tool was mentioned as a specific hurdle.

- Several respondents highlighted the overly complex nature of code generators, and that it is not always clear to which problems the generators are applicable to. In particular, its many features, the necessity to use the modeling/specification language in the right way, to understand the ACG software architecture, and poor documentation makes things difficult.

- There is an implied methodology which can be quite difficult to understand.

- Finding the "right" tool that matches a given problem and customizing it for special needs (e.g., special interfaces, architectures) can require high effort.

- Finding bugs and applying bug-fixes and bug-workarounds in the ACG (especially for new or in-house tools) and dealing with semantic ambiguities of the modeling language can pose major problems.

- The lack of round-trip engineering was also mentioned as a problem.

6.4 ACG Customization and Reliability

In many applications, the ACG could be directly used "out of the box". In roughly a third of the cases, the ACG had to be customized. In 60% of these cases, these bug-fixes and customizations were performed by the tool vendor. Thus, a good relationship with the tool vendor seems to be important.

7 Certification and V&V

Safety-critical code must pass tests, reviews, and certification (depending on the application) in order to demonstrate its quality and reliability. In this part of the questionnaire, we asked how the quality of the generated code was assured, how the use of ACG impacted verification and validation, and what safety properties are most important.

7.1 Bugs in ACG

Automated code generators are complex pieces of software, so it is not surprising that more than 60% of respondents reported that bugs were found in the ACG.

7.2 Validation of generated code

The survey asked if the code generator had been qualified — no respondents claimed that this had been done, though this was mentioned as desirable. Without a qualified code generator, each individual piece of auto-generated code must be validated. Obviously, testing of the generated code plays a major role (90.4%), followed by static analysis[16] (58%). Further trust could be obtained by running the generated code in simulations and comparing the results with those obtained by the model (52%), and (manual) reviews of the generated code (48%). Formal verification and validation or reviews/validation of the innards of the ACG (e.g., templates, transformation rules) is another possibility that, however, seems to be used very seldomly.

[16] It seems that most survey participants understood the term "static analysis" as manually performing analyses (e.g., transient behavior, numerical accuracy, run-time) and manual code inspection, rather than using automated static analysis tools like PolySpace `http://www.polyspace.com` or Coverity `http://www.coverity.com`.

Imp	Safety Property
1.3	Numerical exceptions (e.g. divide-by-zero)
1.4	Overflow/underflow
1.4	Memory safety (array bounds memory allocation)
1.4	Correctness of code with respect to model
1.5	Variable initialization before use
1.9	Correct use of physical units
1.9	Deadlocks, race conditions
2.0	Correct use of coordinate systems
2.1	Numerical accuracy round-off errors
2.2	Transient behavior (i.e. correct and smooth switching of modes)
2.3	Convergence
2.5	Timing, order of execution
2.5	Flight rules (e.g. minimal/maximal values or rates sign rules)
2.6	Termination
2.9	Mathematical invariants (e.g. symmetry of matrices)
–	no extraneous functionality (not called for in the model)
–	fault handling (DFIR, fault tolerance, etc.)
–	compliance/compatibility (e.g., with architecture principles, coding rules, semantics)
–	independently checkable correctness proofs

Table 1: Importance of safety properties

7.3 Important Safety Properties

When analyzing or testing the generated code, testers have to make sure that the code obeys a number of safety properties. Typical safety properties and their relative importance (on a scale of 1 to 5, 1 = most important) are shown in Table 1. The last four features in Table 1 are not rated, because they were given as a response by individual survey participants.

7.4 Software Process and ACG

The use of ACG obviously has a large impact in the overall software development process. Participants of the survey also indicated that ACGs significantly impact the verification and validation (V&V) and certification process (1.9). As expected, highly safety-critical applications had a stronger

influence on the V&V and certification process, than on the overall software development process.

In safety-critical applications, usually standardized software processes have to be followed. Our survey revealed that for the aerospace domain, DO-178B was followed to various levels of criticality and formality. It seems that the use of ACG fits in relatively easy In several applications, in-house processes (e.g., Boeing) are being used. Roughly 30% of the respondents did not use a specific V&V process.

The software processes used did only impose few constraints on the use of ACG. Examples include the restriction of the modeling language (types of modeling blocks) to ensure compliance with DO-178B, direct model to source mapping and traceability, and complexity/performance and interface-compatibility of the generated code. Otherwise, the processes "do not differentiate between automatically and manually generated code".

Despite the fact that software processes seem to be able to handle the use of ACG, a number of problems and obstacles were reported in getting auto-generated code accepted by safety/flight-readiness/certification reviews.

Novelty of the technology and readability of the code are important issues (although one user remarked that ACG-generated code "[...] was preferred over hand-written code used in prior projects". A proven track record and demonstrated improvements of the ACG tools (e.g., bug-fixes, code with smaller footprints, improved traceability, features for demanding users) seem to be suitable to improve acceptance of ACG technology in safety and certification reviews.

8 An ACG "Wishlist"

Current ACGs provide a number of benefits, but they also have shortcomings. In order to identify gaps, which need to be closed to facilitate a widespread use of ACG in safety-critical applications, we asked about a "wish-list" of features for an ideal ACG.

8.1 Important Features

Questions in the sections concerned additional features of an ACG that would improve its usability. Table 2 shows the relative importance (range of 1 to 5, 1 is most important) of a given set of features. The last three features in

Imp	Feature
1.2	Reliability of generated code
1.6	Traceability between code and model
1.6	Support for generating code in desired target platform
1.8	Instrumentation of generated code (e.g. error handling data logging)
1.9	Interface to software libraries
1.9	Support for safety certification and safety cases
1.9	User interface to code generator
2.1	Efficiency of generated code
2.1	Control over algorithms in generated code
2.2	Error handling and feedback during code generation
2.2	Ease of modifying generator target (e.g., for specific processors and operating systems)
2.3	Quality of generated documentation
2.3	Extensibility of generator
2.5	Conformance of generated code to given coding standards
2.6	Readability and clarity of the generated code
3.2	Synchronization between model and manual modifications in generated code
–	Support form model-level debugging
–	User-friendly tool configuration
–	Advanced user interfaces

Table 2: Importance of ACG features

Table 2 are not rated, because they were given as a response by individual users.

Extensibility of an ACG is clearly important and there is a whole spectrum of possibilities, as the answers revealed. Of particular importance seems to be the ability of the ACG tool to be able to integrate legacy code or code produced by other tools. The extension of ACG (and the modeling tools) to increase power and expressiveness of the models have been mentioned by several participants: extensions to incorporate libraries of design knowledge and provide special purpose blocks.

However, as one participant noted, "uncontrolled extensions should be prohibited", as they can lead to severe problems (e.g., with verification, validation, or compatibility).

8.2 Tool Integration

As mentioned earlier, ACGs are often used in combination with other tools, and the survey sought to determine which groups were the most important to integrate with. Figure 10 shows the results.

Tools to simulate or analyze the execution of the generated code and regarded as being the most important. It is also obvious that code generators need to work closely together with regular software engineering tools, like version control and testing environments (e.g., test case generation and test execution).

Other tools in this category include workflow and software process tools, and tools for code integration (e.g., of code generated by other tools or legacy code). The importance of integration of the ACG with project management tools is not as important as expected, though perhaps this reflects the preponderance of software designers and developers among survey respondents.

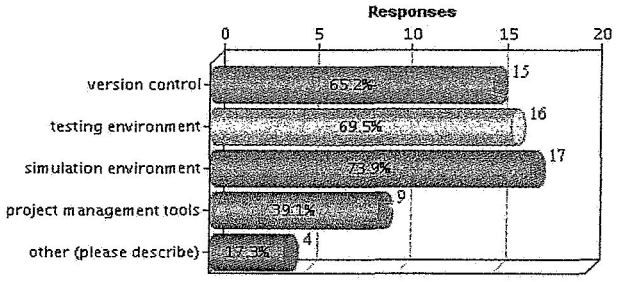


Figure 10: Integration with other tools

8.3 Design Wizards and Analysis Tools

Design wizards are program generation aids which allow the user to interactively set up a "specification" of a problem and then automatically derive or step through a derivation of a solution. For example, Simulink features control design wizard. In contrast to generic ACGs, which can be seen as translators between a high-level specification language and the target language, such wizards can contain knowledge-based systems to actually generate and

instantiate a problem-specific architecture and algorithm. Their importance was ranked as medium (2.7).

The generation of complex components (e.g., filters or control laws) can require the designer (or system) to perform a number of complex mathematical operations. Typical examples include: equation solving, linearization of a non-linear model, discretization, or the symbolic calculation of Jacobians and Hessians, stability analysis (e.g. root-locus, Lyapunov). Some advanced ACG can produce such mathematical operations and deviations in a symbolic manner and document each step. The importance of presenting mathematical derivations was rated very high (1.9).

The relevance of fact that the ACG can perform domain-specific analyses, e.g., on stability, robustness, or convergence, was even rated slightly more important (1.7).

8.4 Documentation and Traceability

For the practical applicability of ACG, the generation of a complete and comprehensive set of documentation is very important. Figure 11 shows the responses on what autogenerated documentation should contain. The most important is design and traceability information (between model and code; see details on traceability below), followed closely by interface descriptions.

Other important information that should be present in the generated documentation includes results of tests and performance analyses (e.g., resource utilization, exceptions, coverage), data from configuration management (in order to be able to "replay" the code generation in exactly the same way). Also, the ability to customize the generated documentation by providing transformation hooks was reported as a desirable feature.

Traceability between artifacts is a major requirement in most software processes, as it allows designers and reviewers to easily switch between the various stages of the software lifecycle. The traditional traceability between requirements and implementation can be generalized to a number of different development artifacts. ACG can and should provide detailed traceability information between each of these. Figure 12 shows a traceability graph between the most important artifacts. Numbers on the edges indicate the relative importance of the links, as given by the survey respondents. Other links are certainly possible but were not mentioned.

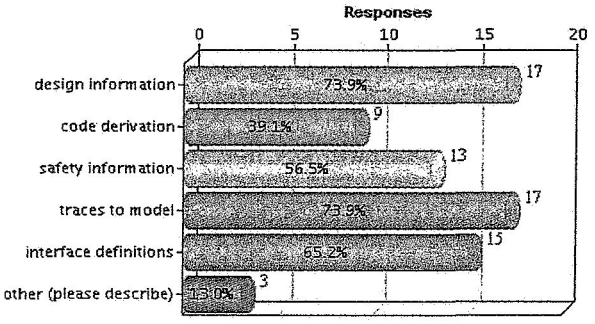


Figure 11: Important ingredients of generated documentation

9 Conclusions

While the targeted nature of this survey should be borne in mind when interpreting its results, it is clear that the use of automated code generation is now widespread and being actively used by industry. Moreover, there are many safety-critical applications where it has been applied, even for production/flight code, in addition to during simulation and prototyping.

The most commonly used tools are the integrated modeling, analysis and simulation tool suites provided by commercial vendors (primarily Real-Time Workshop, MatrixX, and SCADE) with autocoding extensions.

In most cases, respondents were generally satisfied with the tools, although substantial effort is typically spent both in learning to use the tools, and in modifying existing development and V&V processes. Although most V&V processes do not distinguish between manually developed and auto-generated code, actual V&V tasks can differ substantially, so the V&V process needs to be planned carefully. Finally, although none of the tools were qualified, auto-generated code has itself been successfully certified to meet DO-178B.

Interestingly, in-house extensions of these tools are common, particularly to address modeling and verification issues. Indeed, there appears to be a natural synergy with extensions to support certification activities, namely the production of V&V artifacts, documentation, and traceability information.

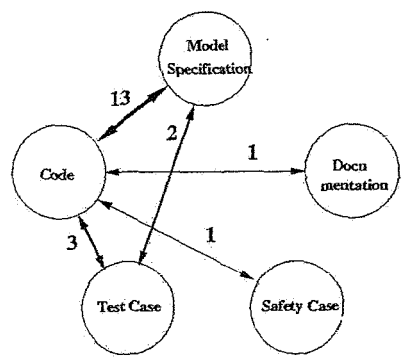


Figure 12: Traceability graph

A Questionnaire

Robust Software Engineering at NASA Ames

Use of code generators in safety-critical applications

Dear Colleague,
We are currently evaluating NASA's needs in the area of high-reliability automated code generation for the Exploration Systems Architecture Study. We believe that you have particular expertise and experience in this area and would like to ask if you would be willing to fill out this questionnaire before

March 23, 2006.

For the purpose of this study we define code generation (also known as autocoding or program synthesis) to be any technology for automatically generating executable code from higher level artifacts, such as models, requirements, or specifications. Typical tools include Matlab Real-Time Workshop, MatrixX, Rational Rose, and DSpace, as well as tools based on template expansion or meta-modeling. We are particularly interested in the application of code generators in safety-critical projects. Please answer whichever questions you are able to and forward this to any colleagues who you think may be interested. We will use all responses in an anonymized form unless you indicate otherwise.

Ewen Denney
Johann Schumann
RIACS, NASA Ames

Have you used a CG in your projects?
C Yes C No

What were the target domains? (select all that apply)
- Automotive
- Aviation
- Space
- Chemical
- Nuclear
- Military
- Other (please describe)

What was the purpose of using CG? (select all that apply)

- Simulation / modeling (e.g., to speed up a simulation)
- Design / prototyping
- Generating production code (i.e. code which runs in the final product)
- Generating glue or configuration code
- Testing (e.g., generating test drivers)
- Other (please describe)

What were the specific application areas? (select all that apply)
- Control
- Signal processing
- Modeling/simulation
- Data analysis
- Diagnosis/system health monitoring
- Planning
- Other (please describe)

What was the level of safety-criticality? (1=highly critical ... 5=non-critical)

Which tools were used?
- Real-Time Workshop
- MatrixX
- Rational Rose
- In-house (company or project-specific) tool(s)
- Other

If you are using your own tool, please give a short description, including: modeling notation, implementation language, background information (e.g., URLs, papers)

What was the generated target language?
- C
- C++
- Ada
- Java
- Other

Was a specific target platform used for the generated code? If yes, please describe (e.g., hardware system, middleware, operating system)

If you have more detailed information about the size of the artifacts in your use of CG, please give a rough estimate *for the model* (input to the CG):

number of atomic blocks (for graphical notations)

number of states (for statechart etc.)

number of lines of specification

Please give a rough estimate *for the number of lines/components of generated code*

What other artifacts were auto-generated? (select all that apply)
- Test cases
- Documentation / Manual
- Log files
- Makefiles or other installation scripts
- Derivation/design information
- Tracing information
- Other (please describe)

What was the rough percentage of auto-generated code with regard to the size of the entire project software (in %)?

Which parts of the system in particular were auto-generated? (select all that apply)
- Continuous control
- Control (mode) logic
- Component interfaces
- Glue code
- (Graphical) user interfaces
- Device drivers
- Safety-critical components
- Other (please describe)

Do you maintain the generated code or the models?

What was your overall experience with CG? (1= highly recommended ... 5= never again)

Please comment:

What were, in your opinion, the main benefits of using CG?

Did the use of CG result in a productivity gain over manual development? If yes, please give a rough estimate (in %)

Did the use of CG result in reduction in defects? If yes, please give a rough estimate

Did the use of CG result in faster time-to-market for the project? If yes, please give a rough estimate

Were substantial customizations or updates of the tool necessary during your project?
C Yes C No

If you answered the previous question with yes, did the tool provider patch or extend the tool for this project?
C Yes C No

Were any bugs found in the CG?

○ Yes ○ No

How did the use of CG change the overall software development process? (1=big impact ... 5=no impact)

How easy did you find it to learn to use a CG? (1=easy ... 5=hard)

What were the main difficulties?

How do you validate the auto-generated code?

- ☐ Static analysis of source code
- ☐ Testing
- ☐ Code reviews
- ☐ Simulation
- ☐ Other (please describe)

How would you rate the relative importance of the following properties of the auto-generated code (where applicable)? (1-5, 1 is most important)

- Numerical exceptions (e.g., divide-by-zero)
- Overflow/underflow
- Variable initialization before use
- Memory safety (array bounds, memory allocation)
- Numerical accuracy, round-off errors
- Timing, order of execution
- Deadlocks, race conditions
- Convergence
- Termination
- Transient behavior (i.e., correct and smooth switching of modes)
- Flight rules (e.g., minimal/maximal values or rates, sign rules)

Are there any artifacts which the generator does or could produce which could support certification?

How would you rate the following features of a CG? (1-5, 1 is most important)

- Traceability between code and model
- Readability and clarity of the generated code
- Efficiency of generated code
- Reliability of generated code
- Quality of generated documentation
- Error handling and feedback during code generation
- Synchronization between model and manual modifications in generated code
- Support for safety certification and safety cases
- Conformance of generated code to given coding standards
- Support for generating code in desired target platform
- Ease of modifying generator target (e.g., for specific processors, and operating systems)
- User interface to code generator
- Control over algorithms in generated code
- Instrumentation of generated code (e.g. error handling, data logging)
- Interface to software libraries

- Correct use of physical units
- Correct use of coordinate systems
- Mathematical invariants (e.g., symmetry of matrices)
- Correctness of code with respect to model

Please list any other properties which you consider to be important:

How did the use of CG affect your verification and validation or certification process? (1=big impact ... 5=no impact)

Did you follow a standardized certification process? If yes, which one(s)?

Did the certification process place specific requirements on the CG? If yes, please explain

Has your code generator been qualified?

○ Yes ○ No

If yes, please give details on how this was achieved and if this gave any advantage for your project.

Have you encountered any particular obstacles to getting auto-generated code accepted by safety/flight-readiness/certification reviews?

- Extensibility of generator

In what ways should the CG be extensible?

Which other features do you consider to be important?

With which other tools should the CG be integrated?

- ☐ Version control
- ☐ Testing environment
- ☐ Simulation environment
- ☐ Project management tools
- ☐ Other (please describe)

How important are the following design support features in connection with CG? (1=most important ... 5=not important

- Design wizards (e.g., for control or filter design)
- Automating mathematical derivations (e.g., equation solving)
- Support for analyses (e.g., stability, robustness, convergence)
- Others (please describe)

What does/should auto-generated documentation contain?

- ☐ Design information
- ☐ Code derivation
- ☐ Safety information
- ☐ Traces to model
- ☐ Interface definitions
- ☐ Other (please describe)

What tracing information should be provided by the CG? Please give up to

three examples in decreasing importance

From To

From To

From To

Please give any further comments or relevant URLS

How would you characterize your role in this project? (select all that apply)

- Design
- Development
- Testing / QA / Verification & Validation
- Certification
- Project management
- CG Technology provider / developer
- Other

If you allow us to quote you, please enter

Name

Institution

Email (opt)

Would you like us to send you the survey results? (requires email address)

Yes No

Submit

B Change Control

This document is released subject to a change management process. Revisions are reviewed, approved and logged. The change management log is available at the project website.

Version#	Date	Change Description	Rationale	Approved by
1.0 (RSE-1.24)	03/31/2006	inital version	N/A	