NASA/TM–2016–219443

Report: Survey on Model-Based Software Engineering and Auto-Generated Code

Katerina Goseva-Popstojanova
West Virginia University, Morgantown, WV 26506, USA

Teme Kahsai
CMU, NASA Ames Research Center, Moffett Field, CA 94035, USA

Matt Knudson
NASA Ames Research Center, Moffett Field, CA 94035, USA

Thomas Kyanko
West Virginia University, Morgantown, WV 26506, USA

Noble Nkwocha
NASA Independent Verification & Validation Facility, Fairmont, WV 26554, USA

Johann Schumann
SGT, Inc., NASA Ames Research Center, Moffett Field, CA 94035, USA

October 2016

NASA/TM–2016–219443

Report: Survey on Model-Based Software Engineering and Auto-Generated Code

Katerina Goseva-Popstojanova
West Virginia University, Morgantown, WV 26506, USA

Teme Kahsai
CMU, NASA Ames Research Center, Moffett Field, CA 94035, USA

Matt Knudson
NASA Ames Research Center, Moffett Field, CA 94035, USA

Thomas Kyanko
West Virginia University, Morgantown, WV 26506, USA

Noble Nkwocha
NASA Independent Verification & Validation Facility, Fairmont, WV 26554, USA

Johann Schumann
SGT, Inc., NASA Ames Research Center, Moffett Field, CA 94035, USA

National Aeronautics and
Space Administration

Ames Research Center, Moffett Field, CA 94035
West Virginia University, Morgantown, WV 26506
NASA IV&V Facility, Fairmont, WV 26554

October 2016

Executive Summary

This report presents the results of our survey, which focused on the use of Model-based Software Engineering (MBSwE) and Auto-generated Code (AGC) at NASA and in different industry domains, worldwide. Our main goals were to: (1) assess the current state-of-the-practice of using MBSwE and AGC, (2) identify and quantify benefits and challenges, and (3) explore the software assurance practice of models and auto-generated code.

The analysis and results presented in this report are based on the answers provided by 114 respondents to the survey who used MBSwE in their projects and answered more than just the first question. The main findings, as they pertain to our goals, are as follows:

1. **State-of-the-practice.** MBSwE and AGC are used mainly in Space, Automotive, and Aeronautics industries, mostly for control systems applications. Besides auto-generated code (most often C or C++), other artifacts such as documentation, test cases, and tracing information were generated.

2. **Benefits and challenges.** Benefits of using MBSwE and AGC included improved productivity, maintainability, and quality, but the transition to and learning to effectively use MBSwE and AGC were mostly considered far from easy.

3. **Software assurance.** Verification and validation (V&V) of models were predominately done by testing and simulation, manual model inspection and reviews, and automated model analysis. V&V of AGC was done by testing, static code analysis, manual code inspection and reviews, and testing with auto-generated test cases. Surprisingly, majority of projects did not use V&V standards and certification, and qualified code generators. Models and AGC were not always synchronized, and AGC was sometimes modified manually. More respondents found bugs in models than in AGC.

We present the results of the survey in detail and conclude the report with providing initial recommendations based on our findings.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Model-based Software Engineering (MBSwE) techniques and auto-generated code (AGC) are increasingly used at NASA to produce actual mission software. However, although MBSwE and AGC have been around for a while, there is only anecdotal knowledge about the different forms of auto-generated code that exist across NASA missions and the spectrum of Software Assurance (SWA) techniques used on these missions.

MBSwE and AGC promise many benefits, including higher productivity and improved quality. The actual situation, however, is not simple and clear cut. On one side, MBSwE and AGC have potential to reduce the SWA effort because models may be tested (using formal verification or simulation) and silly coding mistakes may be avoided. On the other side, auto-generated code needs to be kept in sync with the models and re-verified, and typically is difficult to understand. Obviously, MBSwE and AGC offer opportunities, but also impose challenges for SWA.

This report presents the results of our survey, which was focused on the use of MBSwE and AGC in different industry domains, worldwide. 114 respondents to the survey (out of 216) used MBSwE in their projects and answered more than just the first question. The answers provided by these 114 respondents are analyzed in this report. It should be noted that the respondents were not required to complete the answers to all survey questions. The mean and median number of questions answered by these 114 respondents were 30 and 39, respectively. To assure that respondents' answers are not biased, the survey was anonymous. Among those that voluntarily provided information about their affiliation, many respondents are affiliated with NASA, including GSFC, MSFC, JPL, Stennis Space Center, and NASA IV&V.

Similar to previous surveys, our work explores questions related to the benefits and challenges of MBSwE and AGC [1,2]. In that respect, this work explores the generalizability of the previous findings, which were based on the use of MBSwE and AGC in companies geographically limited to Italy [1,3], multiple European countries [2], and Brazil [4]. Our work is also related to a previous study on the use of AGC [5] that was carried out at NASA ARC in 2006. More importantly, our survey and the results presented in this report aim to fill out the gaps in the current knowledge about the state-of-the-practice of MBSwE and AGC use. Specifically, we explore respondents experiences related to verification and validation, maintenance and synchronization, and types of bugs found in the models and auto-generated code.

The main goals of our survey are as follows:

G1 Assess the current state-of-the-practice of using MBSwE and AGC (i.e., answer the "Where", "What", "How", "Why", and "Who" questions),

G2 Identify and – wherever relevant – quantify the benefits and challenges, and

G3 Assess the current state-of-the-practice and identify the challenges of quality assurance of models and auto-generated code, including verification and validation, synchronization, and types of detected bugs.

The specific research questions that address these goals are described next. Note that research question RQ1 addresses the goal G1, RQ2 addresses the goal G2, and research questions RQ3, RQ4, and RQ5 are aimed at addressing the goal G3.

RQ1 "Where, What, Who, How and Why?"

- **RQ1a** Which areas of industry are using MBSwE and AGC for the software design, development, and deployment?
- **RQ1b** What are the focus-areas (e.g. controllers, drivers, or mode logic) and safety criticality of the applications where MBSwE is being used?
- **RQ1c** What are the respondents roles in the project?
- **RQ1d** Which tool sets and/or approaches are being used?
- **RQ1e** What is/are the purpose(s) of using MBSwE? What are the rough "size" and "complexity" of the developed models? Which are the most popular target languages for AGC and which other artifacts were automatically generated?
- **RQ1f** Which kinds of software development processes are commonly used in conjunction with MBSwE and AGC?
- **RQ1g** Are modeling and coding standards used for model development and coding? Are industry-wide standards and processes for safety-critical systems (e.g., DO-178B/C, ISO 26262, or ISO 61508) being used in conjunction with MBSwE and AGC?

RQ2 Benefits and Challenges

- **RQ2a** How good are the overall experiences with MBSwE and AGC?
- **RQ2b** What are the benefits and challenges of the use of MBSwE and AGC?

RQ3 Verification and Validation

- **RQ3a** How are the models and AGC verified and validated?
- **RQ3b** Are any standards for V&V and certification used?

RQ4 Synchronization and Maintenance

- **RQ4a** Are the models and auto-generated code always synchronized throughout the project life-cycle? If not, how significant are the problems due to lack of synchronization?
- **RQ4b** Are the models and auto-generated code maintained after the development phase? Is the auto-generated code modified manually?

RQ5 Bugs found in Models and Auto-generated Code

> **RQ5a** What are the percentages of respondents who found bugs in the models and in the auto-generated code?
>
> **RQ5b** What are the types of bugs found in models and auto-generated code?

This report is structured as follows. Chapter 2 discusses the related works. Chapter 3 gives an overview of the survey, its design and motivation and describes how the actual survey was carried out. Results of the survey are discussed in detail in Chapter 4. Finally, Chapter 5 provides a summary of the observations and formulates initial recommendations.

Chapter 2

Related Work

Evaluation of MBSwE and AGC is limited, but recently there have been a number of publications addressing this topic. Here we restrict the description to works that primary used survey as a method for data collection and studied the experiences of industrial use of MBSwE and/or AGC. Papers that studied the use of UML, with no association to MBSwE and AGC (e.g., [6]) are not addressed.

We start with a study on the use of AGC, which was carried out at NASA ARC in 2006 [5]. That study focused on tools and approaches to automatically generate production code and other artifacts, asked about the experience with AGC, and performed detailed analysis on the questions of how to V&V and certify such tools and on important tool and process characteristics that respondents put on their wish-list in order to improve efficiency and effectiveness of AGC. This study was based on 23 responses to the survey.

Forward and Lethbridge in [7] presented the results of a survey of 113 software practitioners with a goal to compare the problems and opportunities for model-centric versus code-centric software development. About two thirds of the respondents to this survey were from Canada and United States, and the remaining one third from the rest of the world. Based on the survey responses authors concluded that modeling tools were primarily used for design and to create documentation with a little code generation. In addition, participants believed that model-centric approaches were not very popular as most of them worked in code-centric environments, and the biggest perceived problem with the model-centric approaches was related to keeping the model up to date with the code.

Torchiano et al. [1,3] conducted a survey among Italian software professionals, which was focused on the relevance of software modeling and model driven techniques, the way they are applied, and the benefits of using them. The analysis based on 155 responses to the survey led to a conclusion that model driven techniques are very relevant in the Italian industry. On one side, the adoption of model driven techniques resulted in better design support, improved documentation, better maintenance, higher productivity, and higher software quality. On the other side, model driven techniques required too much effort and had limited usefulness. In addition, their use was prevented by the lack of competencies and supporting tools.

Agner et al. [4] presented the results of a survey of the Brazilian software engineering industry focused on the use of UML and model-driven practices for embedded software. Based on 209 responses, the results showed that most participants were clearly aware of the modeling approach value, even though only 23% knew and used model-driven approaches. Those who used the model-driven approaches perceived the productivity and portability as key advantages.

Whittle et al. [8] presented the observations based on surveying 450 model driven engineering practitioners and interviewing 22 more. This work, in addition to technical factors considered the role of social and organizational factors in the successful use of model driven engineering. Findings of this study suggested that developers rarely use model driven engineering to generate the whole system. Instead model driven engineering was used to develop key parts of a system typically using domain-specific modeling languages. However, there was no consensus about the use of specific modeling techniques and tools.

In a recent paper Liebel et al. [2] presented the results of a survey on the use of model-based engineering in the European embedded software development industry. The 113 respondents answered questions focused on different aspects of model-based engineering, such as the used modeling languages, tools, notations, and shortcomings. The results showed that model-based engineering was mainly used for simulation, code generation, and documentation. Positive effects included higher quality and improved reusability, while the mail shortcomings were interoperability difficulties between model-based engineering tools, high training effort for developers, and usability issues.

Our survey, similar to some of related works, explored questions related to the benefits and challenges of MBSwE and AGC, and in that respect, explored the generalizability of the previous findings. More importantly, our survey and the results presented in this report aimed to fill out the gaps in the current knowledge about the state-of-the-practice of MBSwE and AGC use. Specifically, we explored respondents experiences related software assurance practice of MBSwE and AGC, including verification and validation, maintenance and synchronization, and types of bugs found in the models and auto-generated code.

Chapter 3

Survey Design and Execution

The survey has been conducted following the steps outlined in [9]: (1) define the goals, (2) transform the goals into research questions, (3) design the questionnaire, (4) evaluate of questionnaire using pilot executions, (5) execute the survey, and (6) analyze and package results.

The goals of the survey were defined based on the NASA needs and findings of previously conducted surveys, with a specific focus on establishing the state-of-practice of using MBSwE and AGC (i.e., G1), identifying the benefits and challenges of using MBSwE and AGC (i.e., G2), and especially filling out the gaps related to SWA of models and auto-generated code (i.e., G3).

The goals were then transformed in research questions (given in the Introduction section) and the questionnaire was designed accordingly. The initial version of the questionnaire was evaluated in pilot executions by our colleagues in academia and industry, including members of the NASA's Software Assurance Working Group (SAWG). Based on the feedback, we added several questions and made changes to the questionnaire to improve the validity of the instrument.

The final version of the questionnaire consisted of a total of 46 questions, both multiple choice and free-form. These questions were grouped into the following sections:

1. *Where, What, and Who* section was related to target domain, application area, and safety criticality of the use of MBSwE and AGC, as well as the respondents' role in the project.

2. *Why and How* section focused on tools, the purpose of use of MBSwE and AGC, programming languages used in AGC, other generated artifacts, life-cycle models used in the software development process, and the use of modeling and coding standards.

3. *Benefits and Challenges* section sought respondents' opinions on the effect of the MBSwE and AGC on productivity, maintainability, quality, transition from traditional software engineering development to model-based development, and learning efforts.

4. *Verification and Validation* section was focused on the tools, processes and standards used for verification and validation (V&V) of models and auto-generated code.

5. *Synchronization and Maintenance* section contained questions related to how the models and auto-generated code were synchronized and maintained throughout the development process.

6. *Bugs* section was focused on the type of bugs found in models and auto-generated code, as well as the severity of associated failures.

Our target population was software engineering practitioners and researchers with experience in using MBSwE and/or AGC in industry. For the execution of the survey we tried to reach as many potential respondents as possible using non-probabilistic convenience sampling and snowballing. Techniques included: sending invitation messages to academic and industrial contacts of the research team and to relevant mailing lists, and placing advertisements at related conferences and online forums.

The data were collected by means of an on-line questionnaire created using the Survey Monkey during March and April 2016. Compared to paper-based or email-based questionnaires, Web-based questionnaires allow easier data entry, simpler data collection, and are less error-prone [10].

A total of 216 respondents took the survey, out of which 34% did not use MBSwE and/or AGC in their projects. From the remaining 143 respondents, 29 respondents answered only the first question of the survey, and therefore were excluded from the analysis. Consequently, we analyzed the answers of the 114 respondents that used MBSwE and/or AGC and responded to multiple questions. Because respondents were not required to answer all questions, different survey questions have different number of responses. The mean and median number of questions answered by the 114 respondents were 30 and 39, respectively.

It should be noted that the respondents could stay anonymous, that is, providing their affiliations and other personally identifiable information was voluntary. Among those that voluntarily provided information about their affiliation, many respondents were affiliated with NASA centers, including GSFC, MSFC, JPL, Stennis Space Center, and NASA IV&V. Respondents from industry included AIRBUS, CNRS, Ford, Itemis AG, Liebherr SAS, MITRE Corporation, Saab Aeronautics, Siemens AG, and Softeam, while academic participants were affiliated with Concordia University, Vanderbilt University, Ostfalia University, Universite de Toulouse, University of L'Aquila, and Universita' di Napoli Federico II.

Chapter 4

Detailed Analysis

The analysis presented in this section is based on the responses by the 114 respondents who have used MBSwE and AGC in their projects and answered more than just the first survey question. The survey did not require the respondents to answer all questions. Therefore, the percentages given below are relative to the total number of respondents answering that specific questions.

4.1 RQ1: "Where, What, Who, How and Why?"

4.1.1 RQ1a: Which areas of industry are using MBSwE and AGC for the software design, development, and deployment?

Although there are numerous approaches and tools for Model-based Software Engineering (MBSwE), major application areas seem to focus around *safety-critical* applications. Our survey showed that Space-oriented applications with about 35% seem to be the most heavy users of MBSwE, closely followed by the automotive industry (33%) and aviation (28%), as shown in Figure 4.1.

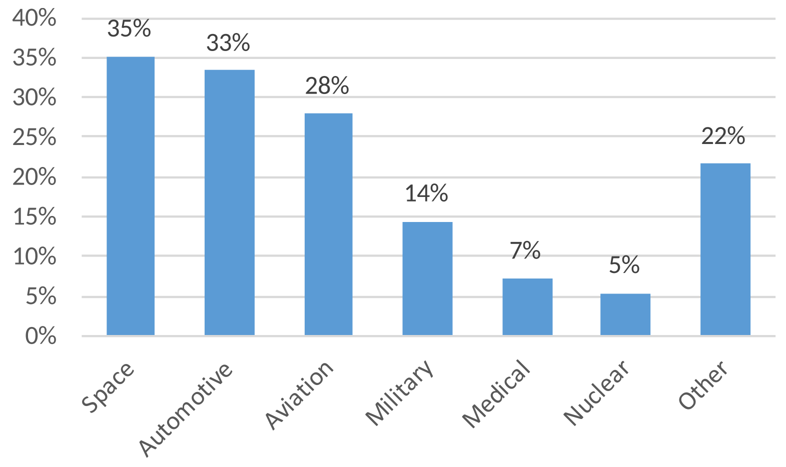


Figure 4.1: Areas of industry that use MBSwE (111 respondents, multiple answers possible)

In comparison to the earlier study conducted by ARC in 2006 [5], an increase of MBSwE use in the Automotive area could be observed (from 17% to 33%). This might be due to different distributions of respondents in the two surveys, but seem to be aligned with the strong interest in MBSwE in this area [11,12]. Figure 4.1, which gives an overview of all the responses provided, shows relatively broad application of MBSwE across many industries and areas. However, areas of media, web-based and cloud-based computing, data analytics, as well as scientific computing, among others, were mentioned only once or were totally missing from this list. It seems that MBSwE is (still) not common in these areas of software development.

4.1.2 RQ1b: What are the focus-areas and safety criticality of the applications where MBSwE is being used?

Figure 4.2 shows the details based upon the answers from 111 respondents. (Note that multiple answers could be selected.) 72% of respondents used MBSwE for control systems. Given the fact that the majority of respondents were from the areas of aerospace and automotive industries, this is hardly surprising. When looking at the components of typical embedded systems, like control systems, signal processing, low-level code/device drivers, the use of MBSwE seems to be very strong. More interesting, but also not surprising, is that MBSwE is being used for development of system architecture and interface code (42%), as well as for simulation (33%).

Application of MBSwE for fault diagnosis / system monitoring, autonomy, and planning / decision making is also notable, with 15%, 11% and 9%, respectively. Participants also listed non-functional analysis of software, databases, performance modeling, ad-hoc networks, system & mission design, command & control, and testing as active application areas, which are grouped under "Other" in Figure 4.2.

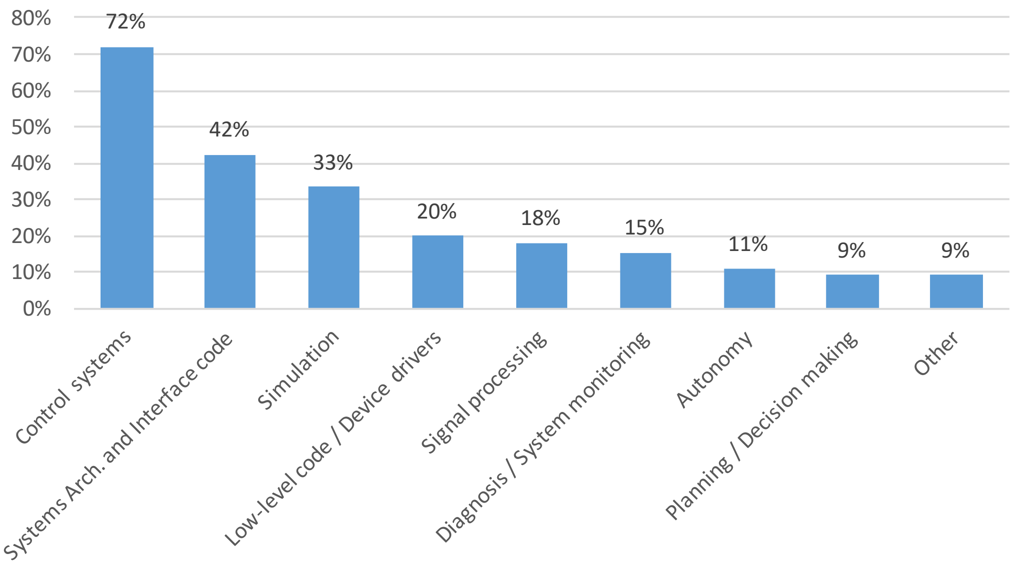


Figure 4.2: Specific focus areas for MBSwE and AGC (111 respondents, multiple selections possible)

Figure 4.3 shows a Venn diagram, which depicts the overlaps of the focus areas usage, where multiple nominations were made. For example, of the 111 respondents, a total of 80 respondents applied MBSwE for control systems. Out of these 80, only 24 respondents selected only controls systems. Others used MBSwE for control systems and one or more

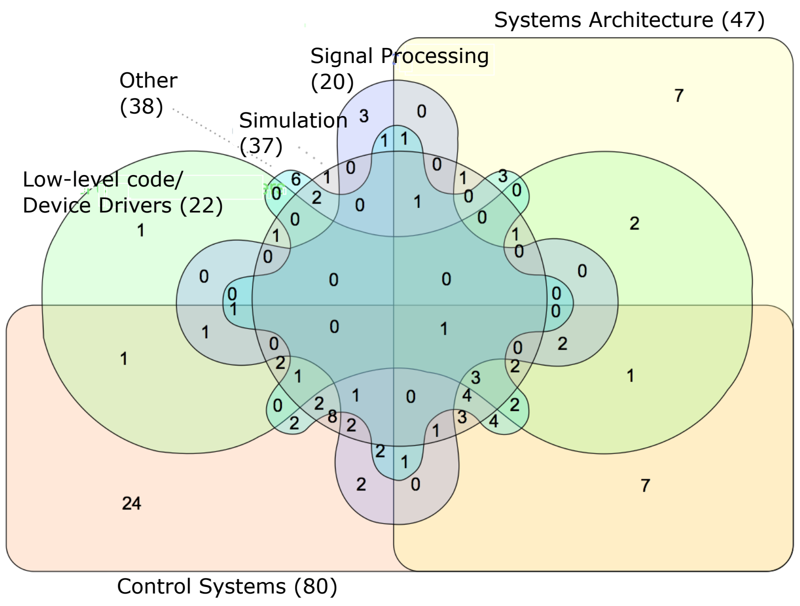


Figure 4.3: Venn diagram which represents the focus areas and their overlaps (111 respondents, multiple selections possible)

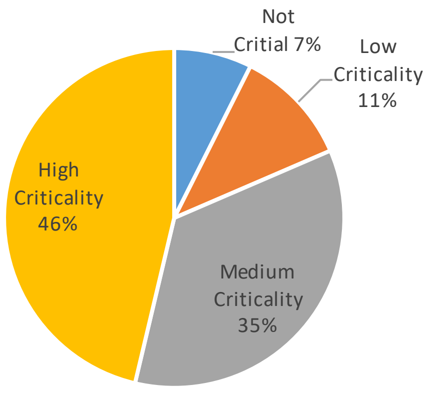


Figure 4.4: Safety-criticality of MBSwE and AGC applications (108 respondents)

other focus areas.

Figure 4.4 represents the distribution of the safety-criticality of the applications. Close to half of the participants responded that their application was highly safety critical (46%). With another 35% of applications with medium criticality, this leaves only 18% for low or non-critical applications. Obviously, MBSwE is being used very actively in areas of high-criticality, which might be due, in part, to the fact that existing software standards for safety-critical software systems (e.g., DO178-B [13] or DO178-C [14]) cover model-based development and automatic code generation. Also a large percentage of respondents are from fields like Aerospace and automotive industry, where much of the developed code is naturally safety-critical.

Our question on “Did the mission/project get an operational state at the intended location (and back again) or known safe state without injuries, death, and/or damage to the equipment and environment?” revealed that the application of MBSwE was usually successful. Only in 8% of cases someone was injured or killed, in 12% equipment, facilities, or vehicles were damaged, and in 16%, the environment was damaged.

The question has to be asked, though, why so few non-critical applications use MBSwE. Is that because of the often high costs of MBSwE tools, the steep learning curve, or additional effort that would need to be spent in order to successfully apply MBSwE? We will try to address some of these aspects in the following sections.

4.1.3 RQ1c: What are the respondents roles in the project?

For any empirical study, including a survey study, it is important to have a representative sample of the population. When it comes to the roles of the respondents to our survey, we observed a roughly even distribution among different roles, as it can be observed in Figure 4.5. Technical roles (such as model development, design, programming, QA/testing) seem to be similarly spread among the respondents as are project management and tool vendor roles. This provides the basis of a well-balanced mix of answers that represent different respondents' roles.

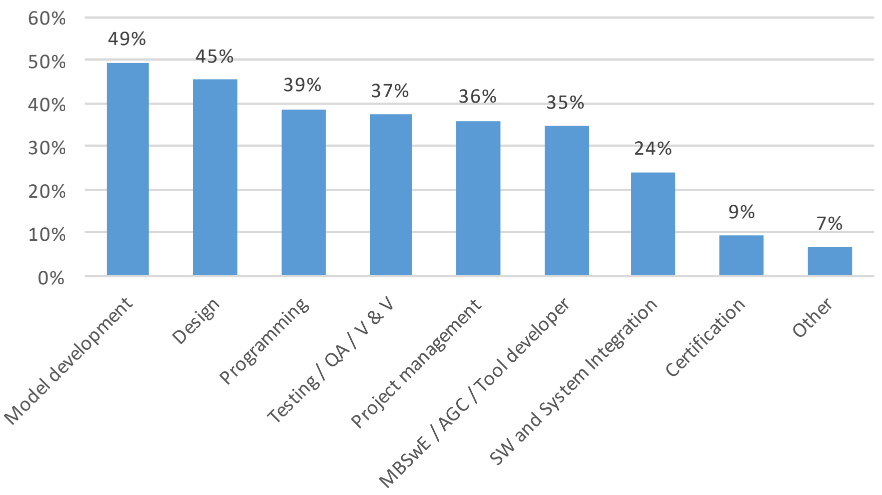


Figure 4.5: Roles of respondents (75 respondents, multiple selections possible)

4.1.4 RQ1d: Which tool sets and/or approaches are being used?

The participants' responses indicated that a significant number of different tools have been used within their applications, as shown in Figure 4.6. Well-known tool chains like Simulink [15] and tools dealing with UML and Eclipse-based tools dominated the tool usage. The large number of tools grouped under "Other" nomination indicates wider spectrum of tools being used that the authors expected. In Appendix 6, we list these tools in alphabetical order and provide some details.

Many respondents indicated that they used a combination of two or more tools / toolchains for their projects. These combinations are represented in the Venn digram shown in Figure 4.7.

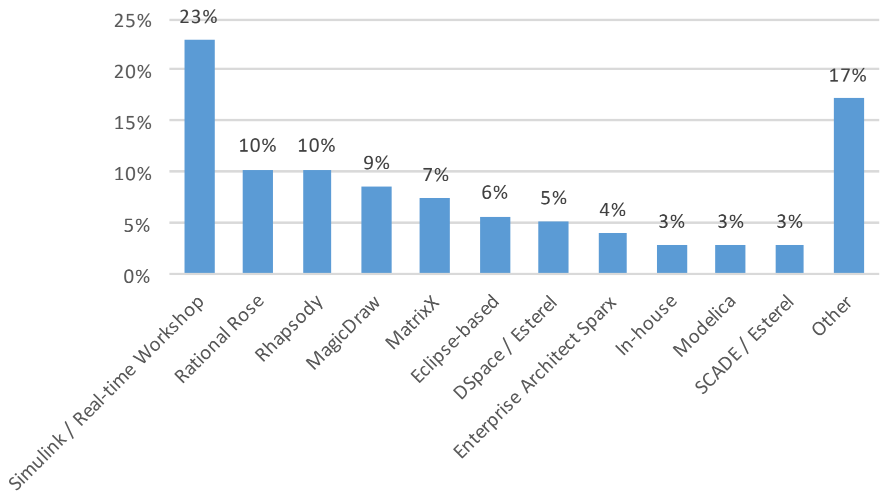


Figure 4.6: Tool usage of MBSwE and AGC tools (92 respondents, multiple selections possible)

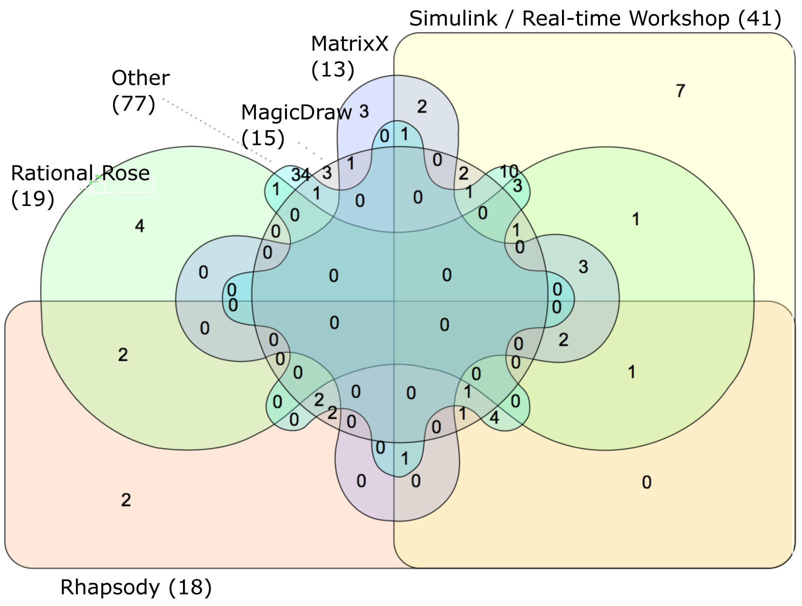


Figure 4.7: Venn diagram which represents the combinations of tool usage (92 respondents, multiple selections possible)

4.1.5 RQ1e: What is/are the purpose(s) of using MBSwE? What are the rough "size" and "complexity" of the developed models? Which are the most popular target languages for AGC and which other artifacts were automatically generated?

We start with the purposes of the use of MBSwE. As it can be seen in Figure 4.8, in overwhelming majority of the cases, MBSwE was used for generation of production code (82%). This is an obvious choice, since many MBSwE tools contain highly sophisticated (and sometimes even certified) automatic code generators. Similarly often, MBSwE was

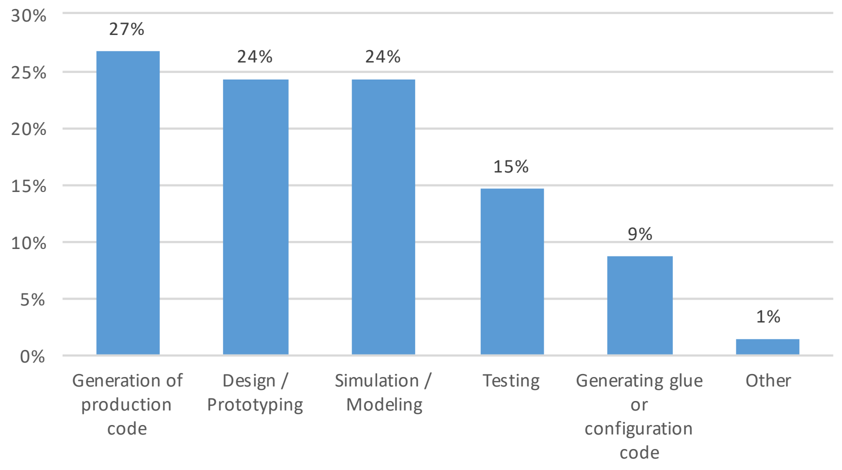


Figure 4.8: Purposes of using MBSwE (74 respondents, multiple selections possible)

used for design/prototyping (74%) and simulation and modeling (74%), supporting the hypothesis that MBSwE can play a prominent role in the early stages of the software process. (One participant specifically used MBSwE to provide a means of standardized communication to others about the design.) The uses of MBSwE in testing (45%) and the generation of glue or configuration code (27%) are visibly lower. Out of the six "Other" responses, three respondents were using MBSwE for V&V purposes (formal verification, safety analysis, artifact generation for analysis, and validation).

With respect to model size and model complexity, Large models and complex models were used in approximately 1/3 of the cases (i.e., 33% and 39%, respectively). Approximately half of the respondents replied with "medium" size and complexity (i.e., 51% and 52%). These answers only can give a very coarse approximation, since metrics on how to gauge model size or complexity were not given in our questions. This decision was made due to the fact that it is almost impossible to come up with metrics of size and complexity that cover the wide of variety of different modeling approaches, types of models, and tools being used.

As expected, the majority of projects used automatic code generators to generate C or C++ code. Specifically, as shown in Figure 4.9, 60% of respondents indicated that they auto-generated C code, followed by 46% C++ code. (Note that multiple selections could be made.) With 18% Ada (and SPARK Ada) was indicated surprisingly often. This is about on the same level as auto-generated code in Java. "Other" target languages included: C#, Python, Verilog, VHDL, SMV, Matlab/Simulink models, XML, nesC, and Scala.

Related to AGC, we also asked a question about the percentage of the code that was auto-generated, compared to the overall project software. The results are shown in Figure 4.10. Interestingly, the portion of the auto-generated code was almost evenly distributed among the four quartiles (i.e., 0-25%, 26-50%, 51-75%, and 76-100%), which indicates that there is no "sweet spot" in the amount of auto-generated software.

MBSwE is being used to automatically generate numerous artifacts ranging from documentation, test cases, and tracing information (most frequently) to coverage and test reports (least frequently). Table 4.1 shows the main types of artifacts generated, roughly ordered by their frequency of occurrence. This table is not meant to gauge the frequencies, but to give an overview of the kinds of artifacts generated with MBSwE.

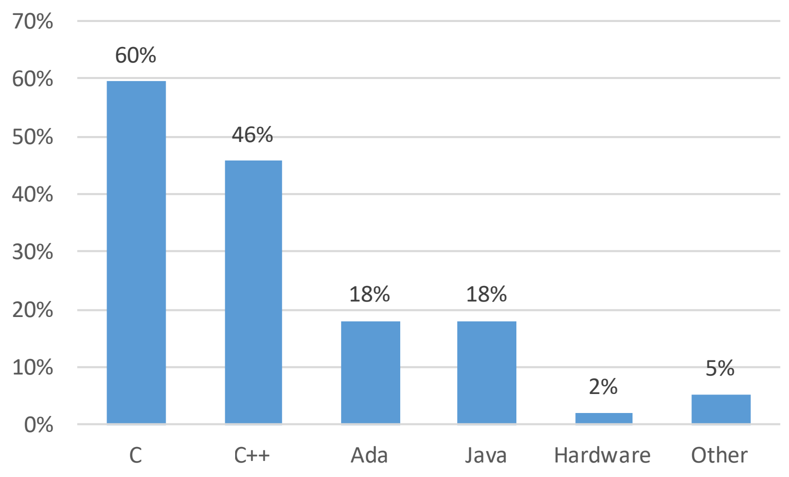


Figure 4.9: Auto-generated code target languages (140 respondents, multiple selections possible)

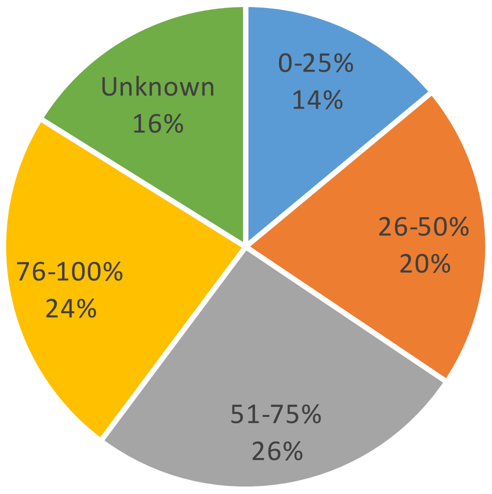


Figure 4.10: Percentage of auto-generated code with regard to the size of the entire project software (93 respondents)

4.1.6 RQ1f: Which life-cycle models are commonly used in conjunction with MBSwE and AGC?

Figure 4.11 shows diagrams of different life-cycle models. The list of the life-cycle models used in the respondents' projects is given in Table 4.2. The Waterfall and Agile models were the most popular, with 34% and 30%, respectively. Less popular seem to be the Rapid Application Development model with 12% and the Spiral model (11%). The V-Model, which is strongly related to the waterfall model and arranges the tasks in such a way that verification and validation can be clearly separated, was used by 8% of respondents. The CENELEC V-Model and the V-Model according to MIL-STD-498 were mentioned specifically. A number of respondents used combinations of life-cycle models, which are not shown in Table 4.2.

Table 4.1: Automatically generated artifacts

Documentation/Manual	FMEA
Test cases	Refactored file
Tracing Information	Performance models
Makefiles/installation scripts	Safety models
Derivation/Design information	Spring-bean configurations ??
Log files	Function block diagram, fault tree
configuration (middleware, initialization)	Formal model for formal verification
	Inputs to other analysis tools
	UML tool-specific adaptors
	Maintenance parts
	Interface generation and checking
	Coverage and test reports

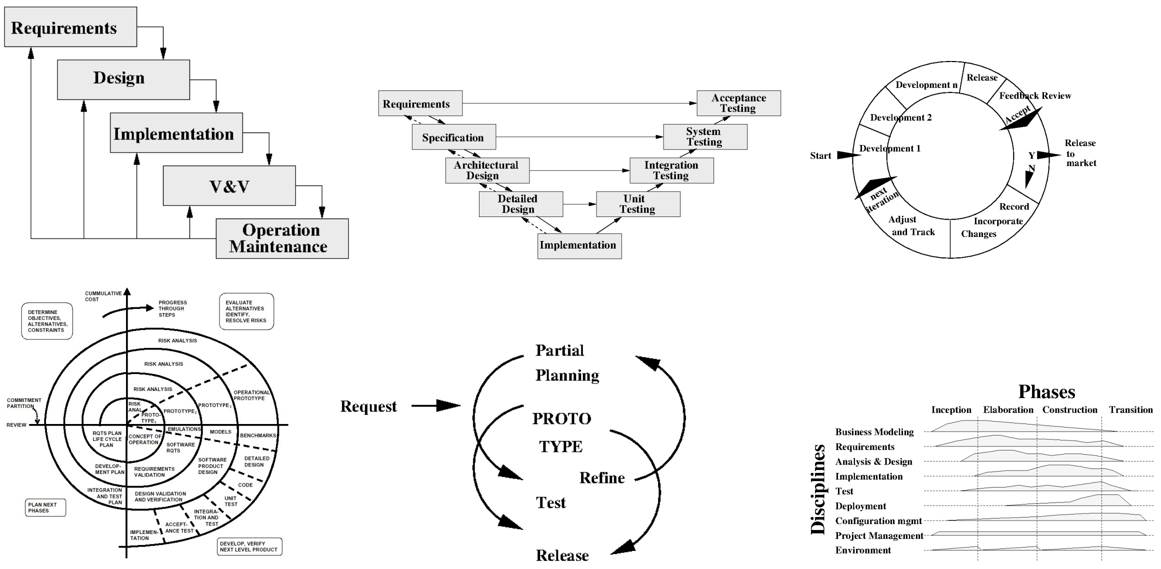


Figure 4.11: Popular life-cycle models

Table 4.2: Life-cycle models used in respondents' projects (95 respondents)

Life-cycle Model	# Responses	Percentage
Waterfall	32	34%
Agile	29	30%
Rapid Application Development	11	12%
Spiral	10	11%
V-Model	8	8%
Rational Unified Process	3	3%
Incremental	1	1%
System Engineering	1	1%

4.1.7 RQ1g: Are modeling and coding standards used for model development and coding? Are industry-wide standards and processes for safety-critical systems used in conjunction with MBSwE and AGC?

For development of any software in general, and for safety-critical areas in particular, the overall procedural approach is of importance. A clear sequence of steps and standards can help to improve productivity, effectiveness of tool use, and reduce the number of bugs. Therefore, in the survey we inquired about modeling and coding standards. (Verification and validation standards are addressed in Section 4.3.)

The breakdown of used modeling standards/guideline is given in Table 4.3. Surprisingly, 39% of respondents indicated that no modeling standards were used, that is, modeling standards were used/enforced in 61% of the cases. Industry-wide guidelines/standards were used by 26% of the respondents. These included MAAB [16] (MISRA, AUTOSAR), SysML [17], and UML. Many projects used some home-grown (25%) or project-specific (7%) modeling guidelines. Other standards and guidelines, which were mentioned only once included: dSPACE Targetlink Guidelines, NASA-STD-7009, Ecore, and EAST-ADL.

Interestingly, the percentage of respondents using coding standard(s) was almost the same (58%) as percentage using modeling standards. Here again, a large majority of projects used their own coding standard(s) and guidelines. The MISRA [18] standard was used in 19% of cases. NASA and JPL coding standards, like NPR-7150.2, NASA SW Development Branch (code 582), as well as coding standards derived from RTC DO-178 were mentioned a few times.

Table 4.3: Used / enforced modeling standards (94 respondents)

Standard Type	# Responses	Percentage
No standard used	37	39%
Industry-wide	24	26%
Home-grown	23	25%
Project-specific	7	7%
Unknown	2	2%
Varies	1	1%

4.2 RQ2: Benefits and Challenges

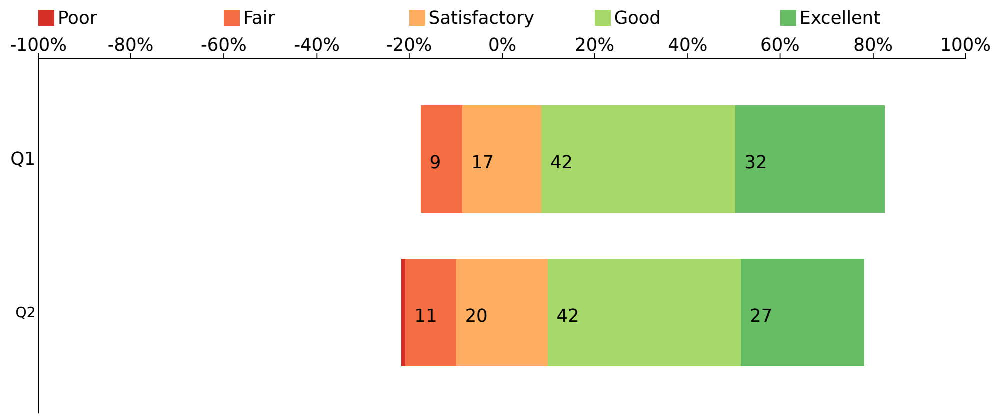
4.2.1 RQ2a: How good are the overall experiences with MBSwE and AGC?

The overall experiences with MBSwE and AGC were rated similarly by the respondents, Respondents could give a rating of poor, fair, satisfactory, good, and excellent. As it can be seen in Figure 4.12, the overall experiences with MBSwE and AGC were rated as either "good" or "excellent" by 74% and 69% of the respondents, respectively. The median in both cases was "good".

4.2.2 RQ2b: What are the benefits and challenges of the use of MBSwE and AGC?

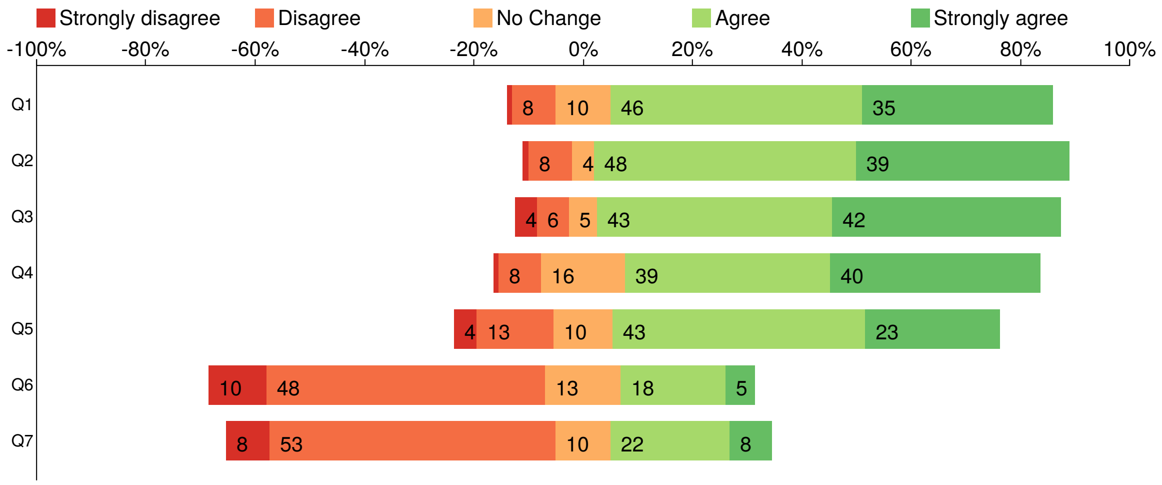
Obviously the use MBSwE and the use of AGC can increase the overall productivity of a software project. However, exact figures are extremely hard to obtain, because that would require controlled experiments. In this survey, we therefore asked to what degree a participant agrees with the statement "MBSwE increased the overall productivity" and the same statement about AGC.

As shown in Figure 4.13, vast majority of respondents either "agreed" or "strongly agreed" than MBSwE and AGC increased the productivity. The specific percentages were



Q1 Your overall experiences with MBSwE (81 responses)
Q2 Your overall experiences with AGC (82 responses)

Figure 4.12: Overall Experience with MBSwE (number of responses varies by question)



Q1 MBSwE increased overall productivity. (79 responses)
Q2 AGC increased overall productivity. (79 responses)
Q3 MBSwE/AGC improved the maintainability of your project(s). (77 responses)
Q4 MBSwE/AGC resulted in better quality (i.e. fewer bugs) compared to manual development. (80 responses)
Q5 Lack of knowledge/familiarity with MBSwE/AGC led to fear/concerns on the use of MBSwE/AGC. (72 responses)
Q6 The transition from traditional Software Engineering Process to MBSwE/AGC was easy. (73 responses)
Q7 Learning the effective use of the MBSwE/AGC tool(s) was easy. (78 responses)

Figure 4.13: Responses to questions regarding benefits and challenges of MBSwE and AGC (number of responses varies by question)

81% and 87%, respectively, with median values of “agree” in both cases.

We also asked about the perceived productivity gain due to MBSwE and AGC. Figure 4.14 shows a distribution of the responses related to MBSwE. Most respondents gave

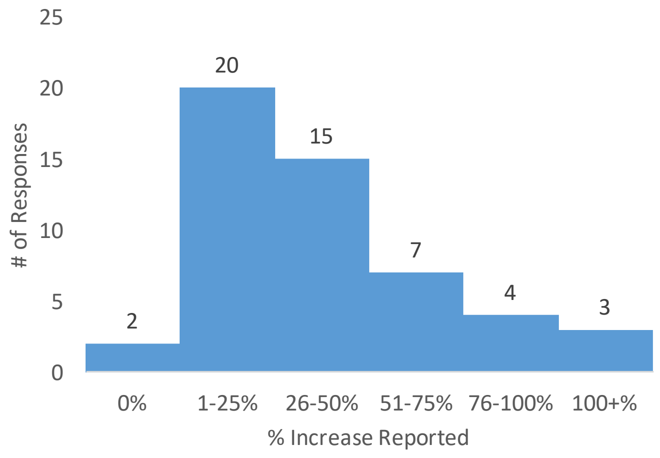


Figure 4.14: Percentage of increased productivity: Distribution of responses

surprisingly low percentages (between 1 and 50%), although responses covered a wide range of up to more than 2,000%. The mean productivity improvement (not taking into account the 2,000% answer) was 48%, while the median was 35%. Similarly, the mean and median improvement of productivity due to use of AGC were 51% and 50%, respectively. It thus seems that in many cases, the overall increase in productivity was not as large as tool vendors and MBSwE and AGC proponents often claim. However, these numbers only reflect the productivity gain as perceived by the respondents to our survey.

On the many "-ilities" of software quality, i.e., the developmental, operational, and support requirements a software must address (e.g., usability, availability, maintainability, reliability, supportability, etc.) a combined total of 80% of the respondents indicated that MBSwE/AGC improved maintainability of their projects. Specifically, 85% and 79% of the respondents either "agreed" or "strongly agreed" with these claims, respectively. The median was "agree" in both cases. One could interpret this, perhaps accurately, to mean that MBSwE with AGC has a substantial impact on the maintainability of the software. This is great, but somehow cloudy. The cloudiness here derives from the phraseology "Project Maintainability" in the survey question. What exactly does "Project Maintainability" mean or imply? More generally, what did or do the terms connote to the respondents? One can attribute this cloudiness to the way and form of the survey question. The expanse in the backgrounds of our survey respondents (Figure 4.5) might possibly also have contributed to how they responded to the question. For example, to a Project Manager, and the general Management respondents, "Project Maintainability", surely encompasses and spans more than just the software. Overall, on the basis of improved maintainability, MBSwE with AGC is a non-lossy venture or investment. Almost everything is beneficial, gainfully beneficial.

On the other hand, MBSwE/AGC introduce challenges, as becomes obvious by the detailed representation of the responses shown in Figure 4.13. Thus, the transition from a traditional software development process to MBSwE/AGC does not appear to be easy; 58% of the respondents either "strongly disagreed" or "disagreed" that the transition was easy, with a median value "disagree". Similarly, majority of respondents either "strongly disagreed" or "disagreed" that learning the effective use of MBSwE/AGC was easy, with a median value "disagree", which indicates that there is a steep learning curve on how to use the technology and tools effectively. 66% of respondents either "agreed" or "strongly agreed" that the lack of knowledge and familiarity with MBSwE/AGC can lead to fear and concerns regarding its use, with a median value "agree".

4.3 RQ3: Verification & Validation

While several previous surveys focused on MBSwE and/or AGC explored the improvements of productivity and maintainability, none of the related works asked questions related to how the models and AGC were verified and validated and whether standards for verification and validation were used. Next, we describe the findings related to these issues.

4.3.1 RQ3a: How are the models and AGC verified and validated?

Verification and Validation (V&V) is concerned with making sure that the software is "doing the right thing" (validation) and "doing things right" (verification). In the realm of V&V and, more generally, Quality Assurance in projects using MBSwE and AGC, we have to address these questions. We have to ask, where or what the V&V/QA focus ought to be when it comes to MBSwE with AGC? The exact plausible options are to focus on: (1) Inputs to the models, (2) the models themselves, (3) the outputs of the models (e.g., the auto-generated code), or (4) all of the above. Within the Safety & Mission Assurance (SMA) sphere of concern, the choice(s) to this question is/are not just fundamental, but crucial. We therefore separated our survey questions into groups dealing with model V&V, and groups about V&V of AGC.

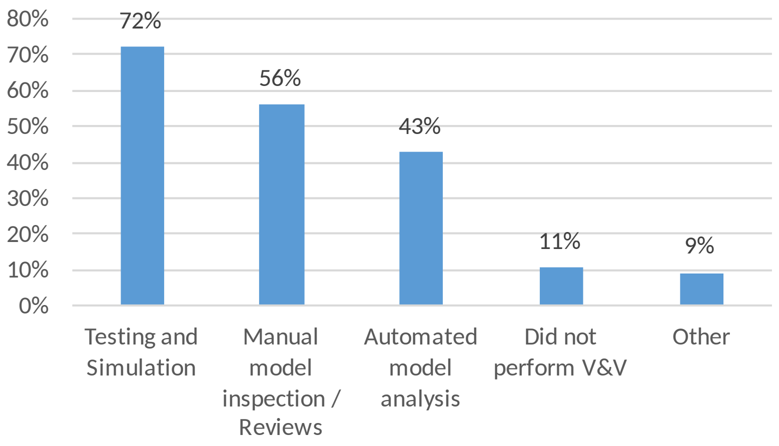


Figure 4.15: V&V methods used to V&V models (75 respondents, multiple selections possible)

The distribution of the responses on the methods used to V&V models are presented in Figure 4.15. The most widely used V&V method for models was "Testing and Simulation", with 72%. This approach to V&V was followed by "Manual model inspection/Reviews" and "Automated Model Analysis", with 56% and 43%, respectively. Around 9% of respondents used other methods to V&V the models, which included formal analysis and build-in tools in the model generators. Interestingly, 11% of respondents to our survey indicated that they did not perform V&V on the models.

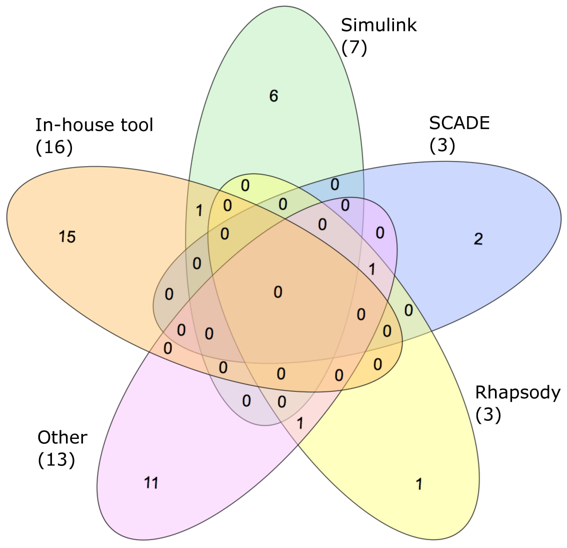


Figure 4.16: Venn diagram which shows the tools used for V&V of models (42 respondents)

64% (out of 74) of respondents used tools to V&V the models. We asked those who answered "Yes" to the question related to using tools to specify the tool(s) they used. Only 42 respondents provided this information. As it can be seen in Figure 4.16, in-house developed tools were used most frequently, followed by Simulink, Rhapsody, and SCADE. Among other tools simulation was mentioned twice and JSim, MagicDraw, Papyrus, Reactis, Scala, SMV, State space analysis, and Colored Petri nets were each mentioned once.

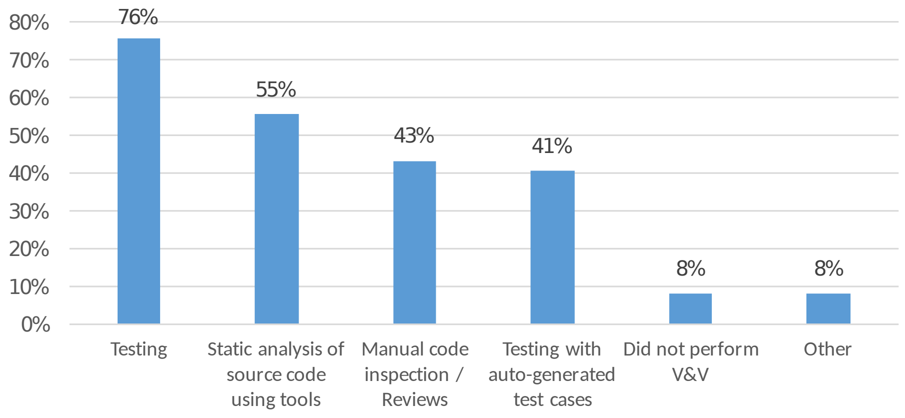


Figure 4.17: V&V methods used for V&V of AGC (74 respondents, multiple selections possible)

Figure 4.17 shows the distribution of responses to the question related to the methods used to V&V the AGC. "Testing", with 76%, was the most frequently used method, followed by "static code analysis using tools" (55%), "manual code inspection and reviews" (43%), and "testing with auto-generated test cases" (41%). 8% of respondents used "Other" methods to test the AGC, which included "simulation" and "analysis of execution traces". Similarly as for V&V of models, 8% of respondents did not perform V&V of the AGC.

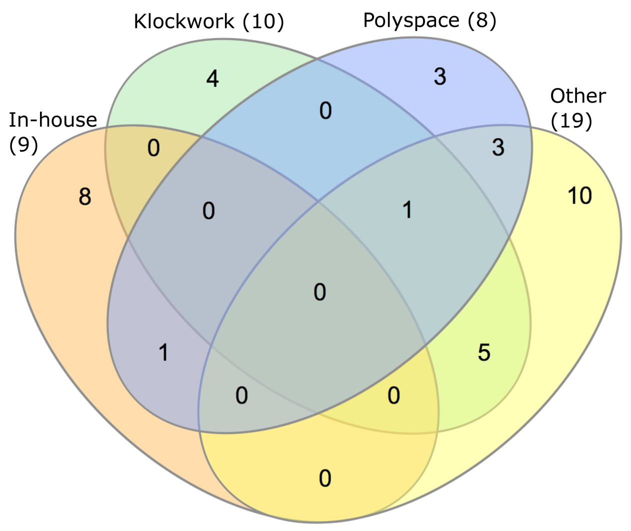


Figure 4.18: Tools used for V&V of AGC (46 responses)

55% of respondents (out of 74 who answered this question) reported using tools to V&V the AGC. Static code analysis tools (one or more than one in combination) were used the most frequently. Klocwork and Polyspace were used much more frequently than other static code analysis tools, such as Coverity, Flexelint, Codesonar, Understand, Astree, QA-C, and PC-Lint. In-house developed tools for V&V of AGC were also used frequently, but details about their functionality were not provided by the respondents. Dynamic testing tools, such as JMeter, Cantata, Rapita, Valgrind, and Tessy (each mentioned only once), were used on their own or in combination with static code analysis tools. Note that, as in case of tools used to V&V the models, more than one tool could be selected. The Venn diagram of tool usage presented in Figure 4.18, for example, shows that Klocwork was used on its own by four respondents, in combination with "other" tools by five respondents, and in combination with Polyspace and one or more "other" tools by one respondent.

We also asked whether the same V&V process was used for AGC as for manually produced code. 69% (out of 72 of respondents who answered this question) used the same V&V process. The 31% of respondents who used different V&V process for AGC provided brief written descriptions of how the V&V processes of AGC and manually produced code differed. Responses included a spectrum of answers, from not using V&V of AGC at all, to not doing manual review/inspection of AGC, using different rules for static code analysis of AGC, or doing less rigorous testing of AGC. Based on these answers, the main reasons behind using different processes for V&V of AGC and manually written code seems to be due to facts that the models were tested, the auto-generation tool was qualified, and that doing manual reviews / inspections on AGC is very inefficient because AGC is essentially unreadable.

Finally, we asked if auto-generated test cases were augmented with additional test cases and 60% of the respondents who answered this question answered "No". The 40% who augmented the auto-generated test cases, did that to increase the test coverage, to test the exception handling, for integration and performance testing, and for stress testing and scenario-based testing.

4.3.2 RQ3b: Are any standards for V&V and certification used?

Only 42% (out of 73) respondents used a standardized V&V process. Note that lower percentage of respondents used V&V standards, compared to those who used modeling or coding standards (see Section 4.1.7). The following V&V standards were used the most often: DO-178B/C, ISO 26262, ISO 61508, and IEEE-1012.

Standardized certification processes (such as RTCA DO-178B/C and its European equivalent EUROCAE ED12B) were followed even less frequently – by only 23% of the respondents (out of 74). It appears that, somewhat surprisingly, standardized V&V and certification are not widely used despite the fact that over 81% of MBSwE and AGC applications were with medium and high criticality.

Only 27% of the respondents (out of 71) used qualified code generator. This seems to be consistent with the fairly low usage of standardization and certification described previously in this section. A majority of tool qualification processes and procedures make use of and/or are based on one or a combination of the software certification standards such as RTCA DO-178B/C, ISO 26262, ISO 61508 and their European equivalents such as the EUROCAE ED12B.

Interestingly, 79% of respondents (out of 70 who provided responses) did not encounter any obstacles to get the AGC accepted by safety / flight-readiness / certification review or any other stakeholder or group. This result is reassuring and shines a bright light on the acceptance level for MBSwE and AGC.

4.4 RQ4: Synchronization and Maintenance

As the software development progresses throughout the life-cycle, it is important to get insights into the project practices used to maintain and synchronize the models and auto-generated code and the observed problems.

4.4.1 RQ4a: Are the models and auto-generated code always synchronized throughout the project life-cycle? If not, how significant are the problems due to lack of synchronization?

We first asked questions focused on the synchronization between the models and auto-generated code. Two thirds of the respondents (i.e., 66%) stated that the models and the auto-generated code were "always" synchronized throughout the project. Of the remaining respondents 26% synchronized the models and auto-generated code "occasionally" and 8% "never" synchronized the models and auto-generated code.

The follow-up question requested the respondents to rate the significance of the problems that were due to the lack of synchronization. For 54% of respondents this question was not applicable. Of the rest, 5% observed significant problems, 13% moderate and 17% minor problems, while 11% did not observe any problems. To collect further information, we asked those that encountered any problems due to lack of synchronization to provide brief description. Respondents stated that "The model is always newer than the code baselines.", "Nonalignment between code and models leads to loose most of advantages of MDE approach.", "... there is often pressure to work on the generated code (presumably a shorter time to fix a problem) rather than do the principled fix on the AGC itself." "Bugs", and "Lost of time in root cause analysis".

4.4.2 RQ4b: Are the models and auto-generated code maintained after the development phase? Is the auto-generated code modified manually?

Next, we explored if the models and auto-generated code were maintained after the development phase of the project. Based on the answers, 84% of the respondents maintained the models and 71% maintained the auto-generated code after the development phase. Interestingly, one third of respondents (33%) answered that the auto-generated code was modified manually after the development phase, while 57% did not modify the auto-generated code manually and for 10% of the respondents the question was not applicable.

4.5 RQ5: Bugs found in models and auto-generated code

4.5.1 RQ5a: What are the percentages of respondents who found bugs in the models and in the auto-generated code?

As shown in Figure 4.19 vast majority of the respondents (i.e., 83%) found bugs in models, while less respondents (i.e., 60%) found bugs in the auto-generated code. This percentages indicate that MBSwE leads to early detection of at least some faults, but there are faults still remaining in the auto-generated code.

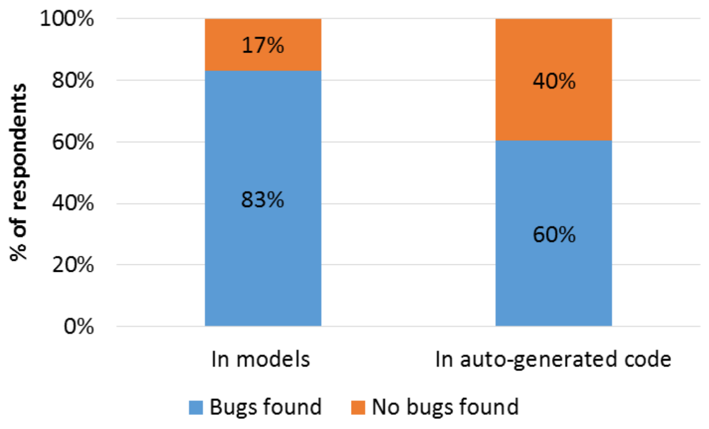


Figure 4.19: Percentages of respondents observing bugs in the models and auto-generated code (71 responses on models, 68 responses on AGC)

4.5.2 RQ5b: What are the types of bugs found in models and auto-generated code?

We also asked a question related to the type of bugs found in the models and auto-generated code. The results shown in Figure 4.20 indicate that significantly more requirements bugs, design bugs, structural logic bugs, syntax bugs, misleading/wrong documentation bugs, interface and integration bugs, and architecture bugs were detected in models than in the auto-generated code. This observation seems to indicate that the MBSwE is serving its purpose of early detection of some type of bugs. As a part of the question related to types of bugs found in the models and auto-generated code, respondents were allowed to provide information on other fault types, not listed in the provided options. Several interesting comments included “Late in the design process, changes have not always been

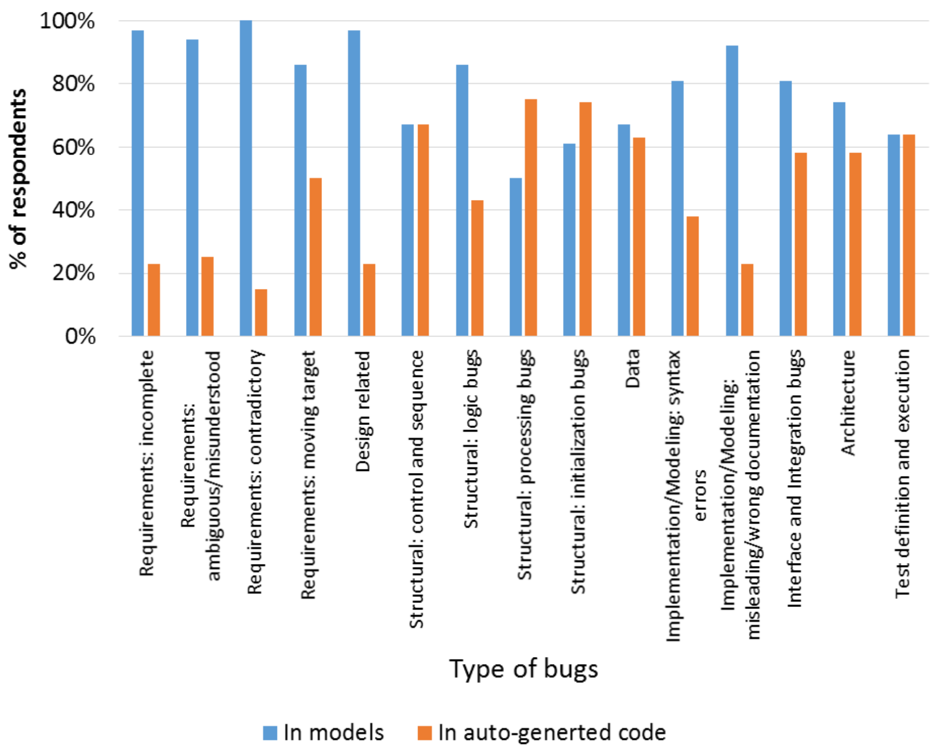


Figure 4.20: Types of bugs observed in the models and auto-generated code (responses to each part of question ranged from 11 to 35)

applied to previously developed models. This becomes a problem in simulators (and models) that are slated for reuse on future missions. The effective use of NASA-STD-7009 is not followed stringently. If asked which input set should be used with which version of the models/databases/simulators, most of the time they are out of sync. AGC requires a certain level of discipline which has not yet made it reliably into the MBSwE environment. Add to it the complexity of Agile processing and you can have a very out of sync environment.", "Code generator had a bug in generating code for linked lists. Was found by tests because of systematic error.", and "Shortcomings related to the processor for which the auto code was generated. There were other compiler/ library issues that need to be worked on for specific targets."

It should be noted, however, that the numbers of responses to the question related to the type of bugs was rather low (i.e., from 11 to 35 respondents). Our future work, which is focused on the root-cause analysis of the bugs found in models and auto-generated code based on data extracted from issue tracking systems of a NASA mission, will further explore the types of bugs and provide additional evidence-based insights.

Many respondents reported that the problems and failures caused by the bugs in models and auto-generated code were of minor or moderate severity. The following description of these problems and failures were provided as responses to the open-ended question: "Problems mainly due to reused (legacy) models that have not been updated to the current version of the tool and/or were assumed to be correct.", "Unreachable safety code inserted by the tool vendor in the older version of the tool.", "Missing implementation of a few functionalities due to ambiguous requirements", "Unreachable safety code inserted by the tool vendor in the older version of the tool", "All severity types and all very hard to troubleshoot", and "Difficult to read code for maintenance".

4.6 Further comments provided by the respondents

At the end of the survey, we asked the respondents if they may have any further comments to provide. We divided the provided answers in three groups: value added, training, and standards and tools. These comments are given in Table 4.4.

Table 4.4: Further comments provided by respondents

Value added	"Engineers developing software should be using MBSwE/AGC whenever possible as it provides a better way of designing, implementing, and testing the software. Additionally, it helps communication about project requirements and functionalities between project teams."
	"Once you have your process in place it has a great deal of value added. Getting to that point needs management support and careful planning."
	"Significantly better than hand coding in C."
	"AGC should not be the main driver for embracing MBSwE. MBSwE is wider in scope than merely development. MBSwE main benefit is in that it allows to reason on a higher abstraction level than programming. It supports early defect detection and better quality."
Training	"It took more time in the beginning of a new project to use MBSwE. You must handle so much complexity in the model, before generating code, that some people lost there patience. It is hard work to maintain model and code and not do "fast fixes" in the code. Many people understand coding, but do not understand MBSwE. You don't educate students in MBSwE, but in coding. So you must teach employees in MBSwE and get them to accept it."
	"Training is essential, both for the tool and the method selected. There is a non-trivial ramp-up time for developers not familiar with the approach. It is also not a panacea!"
Standards and tools	"Very good opinion for the need to apply NASA-STD-7009 (Standard for Models and Simulations). "There should be an effort to consider the impact of Agile on the NASA-STD-7009 standard."
	"Tools like Simulink are not suitable to capture open loop model logic and other computations since the notation doesn't provide modern software abstractions for proper encapsulation, interfaces and design factoring."
	"If the developer had to do it over again, it's unlikely they would have used MatrixX at all."

Chapter 5

Summary of the Results and Recommendations

Obviously, MBSwE and AGC have established role in design, development, and deployment of safety-critical systems in many industrial areas. Table 5.1 summarizes our findings of this survey. In most cases, application of MBSwE and AGC exhibited benefits and brought improvements. However, as our survey revealed, there are many issues with MBSwE and AGC that need to be carefully considered throughout the entire project life (and beyond) and there does not seem to be a unique or easy way to get there. In the following, we list some initial recommendations that are supported by the survey results.

- Getting the model "up and running" and have the production-code generated is only one benefit of using MBSwE/AGC. Many respondents are using MBSwE tools to generate a multitude of artifacts that are necessary for standards/certification, but also, and this seems to be an important ingredient to better quality, use models to facilitate communication between the members of the team or teams.

- Early and detailed planning of the MBSwE endeavor and its implications throughout the development, V&V, and maintenance phases are instrumental in order to be able to use all benefits of such approaches.

- MBSwE and AGC tool(s) or tool chains need to be selected carefully; not all tools work equally well with the selected process(es) and standards. Often, combinations of tools seem to help solve problems in MBSwE and AGC.

- Both, open-source tools and commercial tools are actively being used in MBSwE projects; there does not need to be a clear preference toward one or the other. Only few tools/tool chains are certified for a specific standard (e.g., DO-178B/C), but many respondents reported successful use of MBSwE and AGC with non-certified tools.

- MBSwE does not come for "free" (even using open-source tools); many respondents mentioned a steep learning curve and the necessity of training. College and university education often focuses on coding, but the necessity of knowledge and experience in model development should not be underestimated, in particular for larger projects, where multiple teams need to collaborate in the modeling and design process.

Table 5.1: Summary of the main findings

RQ1: Where, What, Who, How and Why?
Over 70% of the application of MBSwE and AGC are for the design and implementation of control systems; major target industries are space, automotive, and aeronautics.
Most heavily used tools include Simulink/RTW, UML-oriented tools (Rational Rose, Rhapsody, Magic Draw), MatrixX, or DSpace and Esterel. A combined use of different tools is relatively common.
Besides production code (most often C or C++), a multitude of other artifacts are generated (e.g., documentation, test cases, safety- and performance models) demonstrating the versatility of MBSwE and AGC tools.
MBSwE and AGC tools are used with various life-cycle models, with Waterfall and Agile being the most often used.

RQ2: Benefits & Challenges
More than 70% of respondents reported a good or excellent overall experience with MBSwE and AGC.
Benefits included improvement of productivity, maintainability, and quality for 81%, 85%, and 79% of the respondents, respectively.
The estimates for productivity-increase varied widely between none and more that 20-fold; however 68% of the respondents reported a productivity increase between 1 and 50% (mean 48.2%).
The transition toward using MBSwE and AGC was often considered to be far from easy (58%) and often, a steep learning curve was reported (61%)

RQ3: Verification & Validation
Most frequently V&V of models was done by testing & simulation, manual model inspection & reviews, and automated model analysis. 11% of respondents did not V&V the models.
Most frequently V&V of AGC was done testing, static code analysis, manual code inspection & reviews, testing with auto-generated test cases. 8% of respondents did not V&V the AGC.
31% of respondents used different V&V process for AGC than for manually written code.
Standardized V&V and certification were used by only 42% and 23% of respondents, respectively. Only 27% of code generators were qualified.

RQ4: Synchronization & Maintenance
66% of respondents always synchronized models and the auto-generated code, 25% synchronized them only occasionally, and 8% never synchronized them.
35% of respondents observed problems due to lack of synchronization (5% observed significant, 13% observed moderate, and 17% observed minor problems).
84% of the respondents maintained the models and 71% maintained the auto-generated code after the development phase.
33% of respondents stated that the auto-generated code was modified manually.

RQ5: Bugs
83% of respondents found bugs in models, and 60% found bugs in the auto-generated code.
Significantly more requirements bugs, design bugs, structural logic bugs, syntax bugs, misleading/wrong documentation bugs, interface and integration bugs, and architecture bugs were found in models than in auto-generated code.

- For each new project using MBSwE and AGC, time and resources need to be set aside for (a) setting up the model-based environment/tools, (b) training the team on MBSwE and "get them to accept it" (one respondent).

  These activities should be done as early as possible, in particular *prior* to requirements phases, as MBSwE methods provide many advantages during this phase.

- Due to this ramp up and modeling efforts, the generation of first code might take some time, so team members should not "loose their patience".

- In the majority of projects, the models were maintained after deployment phase. This means that the model-based phase should not be over with deployment. Keeping models maintained and current during the entire life-cycle of the product (especially after deployment) and even after end of the project life time can bring substantial benefits: (a) easier maintenance and knowledge transfer to other teams, (b) facilitated model reuse.

  Again, careful and early planning is essential (e.g., extended tool licenses, lifetime of tool versions, etc.) Changes in tool versions during the project can lead to bugs and might require time-consuming and often unnecessary modifications to the models. In many cases, it is advisable to select and then freeze the versions of the tool chains, even if the latest tool features might not be available.

- Modeling and coding standards have been used by many projects to their advantage. Concise modeling standards, like coding standards, facilitate, by enforcing a uniform structure of the models, the readability of the models by other teams/team members, and often reduces the number of modeling errors. Many tools provide some form of automatic checking for modeling standards.

  The early definition and adoption (by everyone!) of a modeling standard is essential. A number of well-accepted and available standards exist. Although less important for AGC, the definition of a coding standard throughout the entire project, i.e., for auto-generated code as well as for handwritten code can improve readability and helps to avoid coding errors and time-consuming nuisances.

  If possible, the code generator should be customized to adhere to the selected coding standard. That might take some effort early on, but will usually pay off.

- MBSwE does not mean you only have to do V&V on the model level. Our survey showed that bugs were found in the model and in the auto-generated code. Thus, V&V activities are necessary on both levels, where they can be tailored toward model or code analysis. Bugs related to requirements and design, architecture, documentation, or structural logic often manifest themselves already on the model level; structural and data (initialization) related bugs tend to show more often on the code level. Again, bug-catching as early as possible and on the higher model level should be preferred. Also note that, in many cases, debugging of auto-generated code can be extremely difficult and time-consuming because AGC is often hard to read.

- A lesson that can be gleaned from these responses on processes in V&V of MBSwE and AGC is that Software "Certification" standards such as RTCA DO-178B/C, ISO 26262, ISO 61508 and their European equivalents such as the EUROCAE ED12B, appear not to be very popular or highly utilized among the spectrum of the survey respondents. Not all of these standards are specifically tailored toward the use of MBSwE and AGC. Therefore, the instantiation of certification standards in a project using MBSwE and AGC is not only difficult and time-consuming, but often this knowledge (including tool customizations) are not sufficiently carried over to other projects.

- Model and AGC should be synchronized often and early on as they tend to diverge over time. Software practices like the "daily build" can, and should, be extended within the MBSwE frameworks and tool chains.
- Once, code has been generated, it should be left alone and not be modified manually. Manual modification, even if done using scripts can be a big source of bugs and cause multiple problems when the models or the tools change.

Acknowledgments

This work was funded by the NASA Software Assurance Research Program (SARP) in FY-2016. The authors would like to thank the members of the NASA's Software Assurance Working Group (SAWG) and other colleagues who provided their feedback on the survey questionnaire. Last but not least, the authors would like to thank all respondents to this survey for sharing their information and opinions.

Bibliography

1. Torchiano, M.; Tomassetti, F.; Ricca, F.; Tiso, A.; and Reggio, G.: Relevance, Benefits, and Problems of Software Modelling and Model Driven techniques-A Survey in the Italian Industry. *Journal of Systems and Software*, vol. 86, no. 8, Aug. 2013, pp. 2110–2126.

2. Liebel, G.; Marko, N.; Tichy, M.; Leitner, A.; and Hansson, J.: Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. *Software & Systems Modeling*, 2016, pp. 1–23.

3. Torchiano, M.; Tomassetti, F.; Ricca, F.; Tiso, A.; and Reggio, G.: Preliminary Findings from a Survey on the MD* State of the Practice. *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, ESEM '11, 2011, pp. 372–375.

4. Agner, L. T. W.; Soares, I. W.; Stadzisz, P. C.; and SimãO, J. M.: A Brazilian Survey on UML and Model-driven Practices for Embedded Software Development. *Journal of Systems and Software*, vol. 86, no. 4, Apr. 2013, pp. 997–1005.

5. Schumann, J.; and Denney, E.: Customer Survey on Code Generators in Safety-Critical Applications. ESAS 6G-PS 1.1.2.3 Report, NASA, 2006.

6. Dobing, B.; and Parsons, J.: How UML is Used. *Commun. ACM*, vol. 49, no. 5, May 2006, pp. 109–113.

7. Forward, A.; and Lethbridge, T. C.: Problems and Opportunities for Model-centric Versus Code-centric Software Development: A Survey of Software Professionals. *Proceedings of the 2008 International Workshop on Models in Software Engineering*, MiSE '08, 2008, pp. 27–32.

8. Whittle, J.; Hutchinson, J.; and Rouncefield, M.: The State of Practice in Model-Driven Engineering. *IEEE Software*, vol. 31, no. 3, 2014, pp. 79–85.

9. Kitchenham, B. A.; and Pfleeger, S. L.: Personal Opinion Surveys. *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. Sjøberg, eds., Springer London, 2008, pp. 63–92.

10. Punter, T.; Ciolkowski, M.; Freimut, B.; and John, I.: Conducting On-line Surveys in Software Engineering. *Proceedings of the International Symposium on Empirical Software Engineering*, ISESE, 80–88, p. 2003.

11. Cabot, J.: Model-based software development in the Automotive industry. URL http://modeling-languages.com/model-based-software-development-automotive-industry/.

12. Schaetz, B.; Broy, M.; Kirstan, S.; and Krcmar, H.: What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry? *Emerging Technologies for the Evolution and Maintenance of Software Models*, IGI Global, 2011.

13. RTCA: DO-178B: Software Considerations in Airborne Systems and Equipment Certification. 1992. URL http://www.rtca.org.

14. RTCA: DO-178C/ED-12C: Software Considerations in Airborne Systems and Equipment Certification. 2012. URL http://www.rtca.org.

15. The Mathworks Company: Matlab, Simulink, Stateflow. URL mathworks.com.

16. Mathworks: The MathWorks Automotive Advisory Board. URL http://www.mathworks.com/solutions/automotive/standards/maab.html.

17. SysML.org: SysML Open Source Specification Project. URL http://sysml.org.

18. MISRA - The Motor Industry Software Reliability Association: MISRA C Guidelines. URL https://www.misra.org.uk.

19. Open Ameos: Open Ameaos Modeling Environment. URL https://www.scopeforge.de/cb/project/8.

20. Ameos Users Group: A Quick Tour of OpenAmeos. URL https://www.scopeforge.de/cb/displayDocument/UMLEVAL.pdf?doc_id=2386.

21. Dassault group: Autosarbuilder. URL http://www.3ds.com/products-services/catia/products/autosarbuilder/.

22. Autosar: URL https://www.autosar.org.

23. Bridgepoint. URL https://xtuml.org/about/.

24. One Fact, Inc: Bridgepoint Pro. URL http://onefact.net/products/bridgepoint-pro/.

25. ISIS: DREMS: Distributed Real-Time Managed Systems. URL http://www.isis.vanderbilt.edu/drems.

26. dSPACE GmbH: URL www.dspace.com.

27. Eclipse.org: Papyrus modeling environment. URL https://eclipse.org/papyrus/.

28. Eclipse.org: Xtend. URL https://marketplace.eclipse.org/content/eclipse-xtend.

29. Eclipse.org: XText. URL http://www.eclipse.org/Xtext/.

30. Sparxsystems.com: Enterprise Architect. URL http://www.sparxsystems.com/products/ea/.

31. ISIS: Generic Modeling Environment (GME). URL `http://www.isis.vanderbilt.edu/projects/gme/`.

32. Abstract Solutions: iUML. URL `http://www.abstractsolutions.co.uk/PRODUCTS/iuml/`.

33. No Magic, Inc: MagicDraw. URL `http://www.nomagic.com/products/magicdraw.html`.

34. National Instruments: NI MATRIXx. URL `http://sine.ni.com/nips/cds/view/p/lang/en/nid/12153`.

35. Mbeddr: Mbeddr tool. URL `http://mbeddr.com`.

36. The Modelica Association: Modelica. URL `www.modelica.org`.

37. Wikipedia: Modelica. URL `https://en.wikipedia.org/wiki/Modelica`.

38. Siemens: LMS Imageine Lab Amesim. URL `https://www.plm.automation.siemens.com/en_us/products/lms/imagine-lab/amesim/`.

39. Dassault Systems: CATIA. URL `http://www.3ds.com/products-services/catia/capabilities/`.

40. Dassault Systems: Dymola. URL `http://www.3ds.com/products-services/catia/products/dymola/key-advantages/`.

41. JModelica: URL `http://www.jmodelica.org`.

42. OpenModelica: URL `www.openmodelica.org`.

43. EaSE Group: Merapi Modeling. URL `https://www.ostfalia.de/cms/de/ivs/ease/Merapi/MerapiModeling.html`.

44. ObjecTime Developer. URL `https://en.wikipedia.org/wiki/ObjecTime_Developer`.

45. Quantum Leaps: QP frameworks. URL `http://www.state-machine.com/products/index.html#QP`.

46. IBM: Rational Rose Modeler. URL `http://www-03.ibm.com/software/products/en/rosemod`.

47. IBM: Rational Rhapsody family. URL `www.ibm.com/software/awdtools/rhapsody/`.

48. Wikipedia: Rational Rhapsody. URL `https://en.wikipedia.org/wiki/Rational_Rhapsody`.

49. ROSlab. URL `http://precise.github.io/ROSLab/`.

50. Esterel-Technologies: Scade Suite. URL `esterel-technologies.com`.

51. UPPAAL. URL `http://www.uppaal.org`.

52. Visual Paradigm: Effective IT System Design, with UML & ERD. URL `https://www.visual-paradigm.com/features/`.

Chapter 6

Appendix A: MBSwE and AGC Tools

These tools have been reported in our survey

Ameos / OpenAmeos is a UML modeling environment. It supports UML2.0 and MDA-based model transformation [19].

Ameos / OpenAmeos is a UML modeling environment that supports UML 2.0 profiles and MDA based model transformation [19]. It provides MDA-based code generation templates for Java, C++, C, Ada95, and C#. OpenAmeos, which is open-source, is based on the Ameos UML tool from Aonix [20]

AUTOSAR Builder This [21] is a modeling and simulation tool for development, simulation, and deployment of embedded systems. It is aimed for the area of automotive systems. The tool suite is based on Eclipse and designed for design and development of embedded systems and software that is AUTOSAR [22] compliant.

Bridgepoint supports model-driven development using executable, translatable UML (xtUML) [23]. This open-source tool provides modeling GUI, a simulated execution environment (Verifier), and model compilers. Commercial versions, like Bridgepoint Pro [24] exist.

DREMS is a software tool for the model-based design, implementation, configuration, deployment and management of distributed real-time software [25]

DSpace TargetLink from dSPACE [26] is a tool that generates C code from Simulink/Stateflow diagrams and is mainly targeted toward the automotive industry.

Eclipse Papyrus UML is a modeling environment for UML based on eclipse [27]. It also supports SysML 1.1 and 1.4 [17].

Eclipse Xtend Xtend is a statically typed programming language that leverages Java's type system and supports lambda expressions, type inference, or operator overloading. It is supported within the Eclipse-IDE [28].

Eclipse Xtext is an eclipse-based framework for the development of domain-specific languages [29].

Enterprise Architect is a commercial UML design and CASE tool [30].

GME The Generic Modeling Environment (GME) has been developed at ISIS, Vanderbilt University and is a configurable toolkit for domain-specific languages and code generators [31].

iUML is a modeling and simulation tool for the construction and testing if system models using executable UML [32].

MagicDraw is a visual modeling tool for UML, SysML, BPMN and UPDM. It features customizations for domain-specific languages, model decomposition, model transformation, refactoring, and decomposition, as well as template-based document generation [33].

Matlab/Simulink Simulink [15] is a model-based design system, which works as a part of Mathworks' Matlab system. Very widely distributed in engineering areas, the hierarchical graphical model features numerous blocks and libraries to design continuous and discrete simulations. Stateflow, an additional package, integrates modeling of hierarchical state machines. Production-level code can be generated using Real-Time Workshop for generic and embedded platform. Mathworks sells additional tools for model analysis ("model analyzer"), testing and testcase generation, as well as static analysis of C/C++ code.

MatrixX is tool chain for control design, consisting of the SystemBuild modeler and AutoCode code generator [34].

Mbeddr is a set of languages and development environment for embedded software engineering [35].

Modelica Modelica [36,37] is an open-source, object-oriented modeling language for the modeling of complex systems. Large free libraries facilitate the design of multi-domain models that, e.g., contain mechanical, electrical, hydraulic, or electric subcomponents. Several commercial (e.g., AMESim [38], CATIA [39], or Dymola [40]) and open-source implementations (e.g., JModelica [41] or Openmodelica [42] exist. Most of these tools provide environments for simulation, optimization, compilation, and model analysis.

merapi Modeling is an eclipse-based UML modeling tool with code generator for C specifically tailored for embedded systems [43].

ObjecTime Developer (OTD) is a software automation tool for generation of production-quality code and is part of Rational [44].

Quantum Leaps – QP Framework is a lightweight software framework for the development of modular real-time systems based upon event-driven actors [45]

Rational Rose is a UML software design tool for visual software modeling and component construction [46].

Rhapsody Rhapsody, a modeling environment based on UML, is a visual development environment for systems engineers and software developers creating real-time or embedded systems and software [47,48].

ROSLab is a high-level programming language for robotic applications [49].

SCADE The SCADE suite [50] is a commercial tool for the model-based design of control software, including software prototyping, verification and validation, and automatic code generation. The code generator has been qualified as development tool for DO-178B/C up to level A and for ASIL level D (ISO 26262).

UPPAAL is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types [51].

Visual Paradigm Eclipse is a commercial UML design tool [52].

Chapter 7

Appendix B: Survey Questionnaire

1. Have you used Model-Based Software Engineering (MBSwE) with Auto-Generated Code (AGC) in your projects?
   - Yes
   - No
2. What are/were the target domains? (select all that apply)
   - Automotive
   - Aviation
   - Chemical
   - Military
   - Nuclear
   - Space
   - Other (please specify)
3. What are/were the specific application areas? (select all that apply)
   - Control systems
   - Signal processing
   - Simulation
   - Systems Architecture and Interface code
   - Low-level code / Device drivers
   - Diagnosis / System health monitoring
   - Planning / Decision making
   - Autonomy
   - Machine learning
   - Other (please specify)
4. To what extent did the mission/project perform what it was supposed to do?
   - N/A
   - Completely
   - Partially
   - Not at all
5. What is/was the level of safety-criticality for the overall application?
   - High criticality

- Medium criticality
- Low criticality
- Not critical

6. Did the mission/project get an operational state at the intended location (and back again) or known safe state

   (a) without killing or injuring anyone
   (b) without damaging equipment, facilities, or vehicles
   (c) without damaging the environment

   - Yes
   - No

7. What is/was the purpose of using MBSwE? (select all that apply)

   - Simulation / Modeling
   - Design / Prototyping
   - Generation of production code (i.e., code which runs on the final product)
   - Generating glue or configuration code
   - Testing (e.g., generation of test cases and test drivers)
   - Other (please specify)

8. Which MBSwE tools are/were used? (select all that apply)

   - Simulink / Real-time Workshop
   - MatrixX
   - Rational Rose
   - Rhapsody
   - MagicDraw
   - DSpace / Esterel
   - Modelica
   - In-house developed tool or Other (please specify and give details)

9. Is/was your project using more than one model (possibly even using different systems like Simulink and Modelica)?

   - No
   - Yes (please specify)

10. How would you characterize the size of the models?

    - Small
    - Medium
    - Large

11. How would you characterize the complexity of the models?

    - Low
    - Medium
    - High

12. What is/was the auto-generated code target language(s)? (please select all that apply)

    - C

- C++
- Ada
- JAVA
- Other (please specify)

13. What is the percentage of auto-generated code with regard to the size of the entire project software?
    - Unknown
    - 0% - 25%
    - 26% - 50%
    - 51% - 75%
    - 76% - 100%
14. Which other artifacts are/were auto-generated? (select all that apply)
    - Test cases
    - Documentation / Manuals
    - Log files
    - Makefiles or other installation scripts
    - Derivation / Design information
    - Tracing information
    - Other (please specify)
15. Which life-cycle model is/was used on your project?
    - Waterfall
    - Agile
    - Spiral
    - Rapid application development
    - Other (please specify)
16. Did you use/enforce any modeling standard? (If yes, please indicate if an industry-wide, home-grown, or project-specific standard had been used)
    - No
    - Yes (please specify)
17. Did you use/enforce any coding standard? (If yes, please indicate if an industry-wide, home-grown, or project-specific standard had been used)
    - No
    - Yes (please specify)
18. Using the adjectives in the columns to the right, please rate the following:
    (a) Your overall experiences with MBSwE
    (b) Your overall experiences with auto-generated code
    - Poor
    - Fair
    - Satisfactory
    - Good
    - Excellent
    - N/A

19. Using the adjectives in the columns to the right, to what extent do you agree with the following statements?

    (a) MBSwE increased overall productivity.
    (b) Automated code generation increased overall productivity.

    - Strongly disagree
    - Disagree
    - No change
    - Agree
    - Strongly agree
    - N/A

20. If you agree to any extent that MBSwE increased overall productivity, please give an estimate (in %) of those gains compared to manual development.

    - FREE RESPONSE

21. If you agree to any extent that automated code generation increased overall productivity, please give an estimate (in %) of those gains compared to manual development.

    - FREE RESPONSE

22. Using the adjectives in the columns to the right, to what extent do you agree with the following statements?

    (a) MBSwE/AGC improved the maintainability of your project(s).
    (b) The use of MBSwE/AGC resulted in better quality (i.e., fewer bugs) compared to manual development.
    (c) The transition from the traditional Software Engineering Process to MBSwE/AGC was easy.
    (d) Learning the effective use of the MBSwE/AGC tool(s) was easy.
    (e) Lack of knowledge / familiarity with MBSwE / AGC led to fear/concerns on the use of MBSwE / AGC

    - Strongly disagree
    - Disagree
    - No change
    - Agree
    - Strongly agree
    - N/A

23. How did you perform V&V of the models? (select all that apply)

    - Did not perform V&V
    - Manual model inspection / Reviews
    - Testing and Simulation
    - Automated model analysis
    - Other (please specify)

24. Did you use tools for V&V of the models? (e.g., Design Verifier or "Simulink Verification and Validation")

    - No

- Yes (please specify)

25. How did you perform V&V of the auto-generated code on the unit and subsystem level? (select all that apply)
    - Did not perform V&V
    - Manual code inspection / Reviews
    - Static analysis of source code using tools
    - Testing
    - Testing with auto-generated test cases
    - Other (please specify)
26. Did you use tools for V&V of the code? (e.g., PolySpace, Klocwork)
    - No
    - Yes (please specify)
27. Did you use the same V&V process for auto-generated code as for manually produced code?
    - No
    - Yes (please specify)
28. Were auto-generated test cases augmented with additional test cases (e.g., stress testing, testing exception handling, etc)?
    - No
    - Yes (please specify)
29. Did you (need to) follow a standardized V&V process? (e.g., DO 178B/C, ISO 26262, ISO 61508, etc)
    - No
    - Yes (please specify which standard was followed)
30. Did you follow a standardized certification process such as those of the RTCA DO-178B/C or its European equivalent, the EUROCAE ED12B?
    - No
    - Yes (please specify)
31. Did the certification process place specific requirements on the MBSwE?
    - No
    - Yes (please explain)
32. Have you encountered any obstacles to getting auto-generated code accepted by safety/flight-readiness/certification review or any other stakeholder or group in the organization?
    - No
    - Yes (please describe briefly)
33. Was your code generator qualified?
    - No
    - Yes (please give details on how this was achieved and if this gave any advantage for your project)

34. Did you (or someone else) maintain the synchronization between the models and AGC throughout the project?
    - Always
    - Occasionally
    - Never
35. If the models and AGC were not always synchronized, did the lack of synchronization lead to problems?
    - N/A
    - Not at all
    - Minor problems
    - Moderate problems
    - Significant problems
36. If you encountered any problems due to lack of synchronization, please provide brief description of these problems.
    - FREE RESPONSE
37. Were the models maintained after the developmental phase of the project?
    - No
    - Yes
38. Was the auto-generated code maintained after the development phase of the project?
    - No
    - Yes
39. Was the auto-generated code modified manually?
    - Yes
    - No
    - N/A
40. Were any bugs found in the models?
    - No
    - Yes
41. Were any bugs found in the auto-generated code?
    - No
    - Yes
42. If any bugs were found, please specify in the columns below the types of bugs found in the models and/or auto-generated code.
    (a) Requirements: incomplete
    (b) Requirements: ambiguous/misunderstood
    (c) Requirements: contradictory
    (d) Requirements: moving target
    (e) Design related
    (f) Structural: control and sequence (e.g., dead code, wrong loops)
    (g) Structural: logic bugs (e.g., in conditions)
    (h) Structural: processing bugs (e.g., arithmetic problems, overflow, underflow)

(i) Structural: initialization bugs (e.g., uninitialized variables)
(j) Data (e.g., data formats, numbers)
(k) Implementation / Modeling: syntax errors
(l) Implementation / Modeling: misleading/wrong generated documentation
(m) Interface and Integration Bugs
(n) Architecture (e.g., problems under stress, incorrect assumptions)
(o) Test Definition and execution (e.g., bugs in generated test drivers)
(p) Other (please specify any other type of bugs that were found and whether they appeared in the models, auto-generated code, or both)

- Models
- Auto-generated code

43. Please describe briefly the severity of problems / failures caused by the bugs you found in the models/AGC.
    - FREE RESPONSE
44. If you have any further comments related to MBSwE and AGC, please provide them here.
    - FREE RESPONSE
45. How would you characterize your role in this project? (select all that apply)
    - Design
    - Model development
    - Programming
    - SW and System Integration
    - Testing / QA / Verification & Validation
    - Certification
    - Project management
    - MBSwE / AGC Technology / Tool provider or developer
    - Other (please specify)
46. This survey is anonymous. Providing any personally identifiable information below is entirely optional.
    - Name
    - Institution
    - Address
    - Address 2
    - City/Town
    - State/Province
    - ZIP/Postal Code
    - Country
    - Email Address
    - Phone Number

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

1. REPORT DATE *(DD-MM-YYYY)* 01-10-2016	**2. REPORT TYPE** Technical Memorandum	**3. DATES COVERED** *(From - To)* 01/2016–05/2016

4. TITLE AND SUBTITLE Report: Survey on Model-Based Software Engineering and Auto-Generated Code	**5a. CONTRACT NUMBER** NNA14AA60C/NNG12SA03C
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER SARP
6. AUTHOR(S) Katerina Goseva-Popstojanova, Teme Kahsai, Matt Knudson, Thomas Kyanko, Noble Nkwocha, Johann Schumann	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Ames Research Center, Moffett Field, CA 94035 West Virginia University, Morgantown, WV 26506 NASA IV&V Facility, Fairmont, WV 26554	**8. PERFORMING ORGANIZATION REPORT NUMBER** L–
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA/TM–2016–219443

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Category 61
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES
An electronic version can be found at http://ntrs.nasa.gov.

14. ABSTRACT

This report presents the results of our survey, which focused on the use of Model-based Software Engineering (MBSwE) and Auto-generated Code (AGC) at NASA and in different industry domains, worldwide. Our main goals were to: (1) assess the current state-of-the-practice of using MBSwE and AGC, (2) identify and quantify benefits and challenges, and (3) explore the software assurance practice of models and auto-generated code. The main findings, as they pertain to our goals, are as follows:
(1) State-of-the-practice. MBSwE and AGC are used mainly in Space, Automotive, and Aeronautics industries, mostly for control systems applications. Besides auto-generated code (most often C or C++), other artifacts such as documentation, test cases, and tracing information were generated.
(2) Benefits and challenges. Benefits of using MBSwE and AGC included improved productivity, maintainability, and quality, but the transition to and learning to effectively use MBSwE and AGC were mostly considered far from easy.
(3) Software assurance. Verification and validation (V&V) of models were predominately done by testing and simulation, manual model inspection and reviews, and automated model analysis. V&V of AGC was done by testing, static code analysis, manual code inspection and reviews, and testing with auto-generated test cases. Surprisingly, majority of projects did not use V&V standards and certification, and qualified code generators. Models and AGC were not always synchronized, and AGC was sometimes modified manually. More respondents found bugs in models than in AGC.
We present the results of the survey in detail and conclude the report with providing initial recommendations based on our findings.

15. SUBJECT TERMS
software engineering, model-based software development, code generation

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Information Desk (help@sti.nasa.gov)
U	U	U	UU		**19b. TELEPHONE NUMBER** *(Include area code)* (757) 864-9658

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18